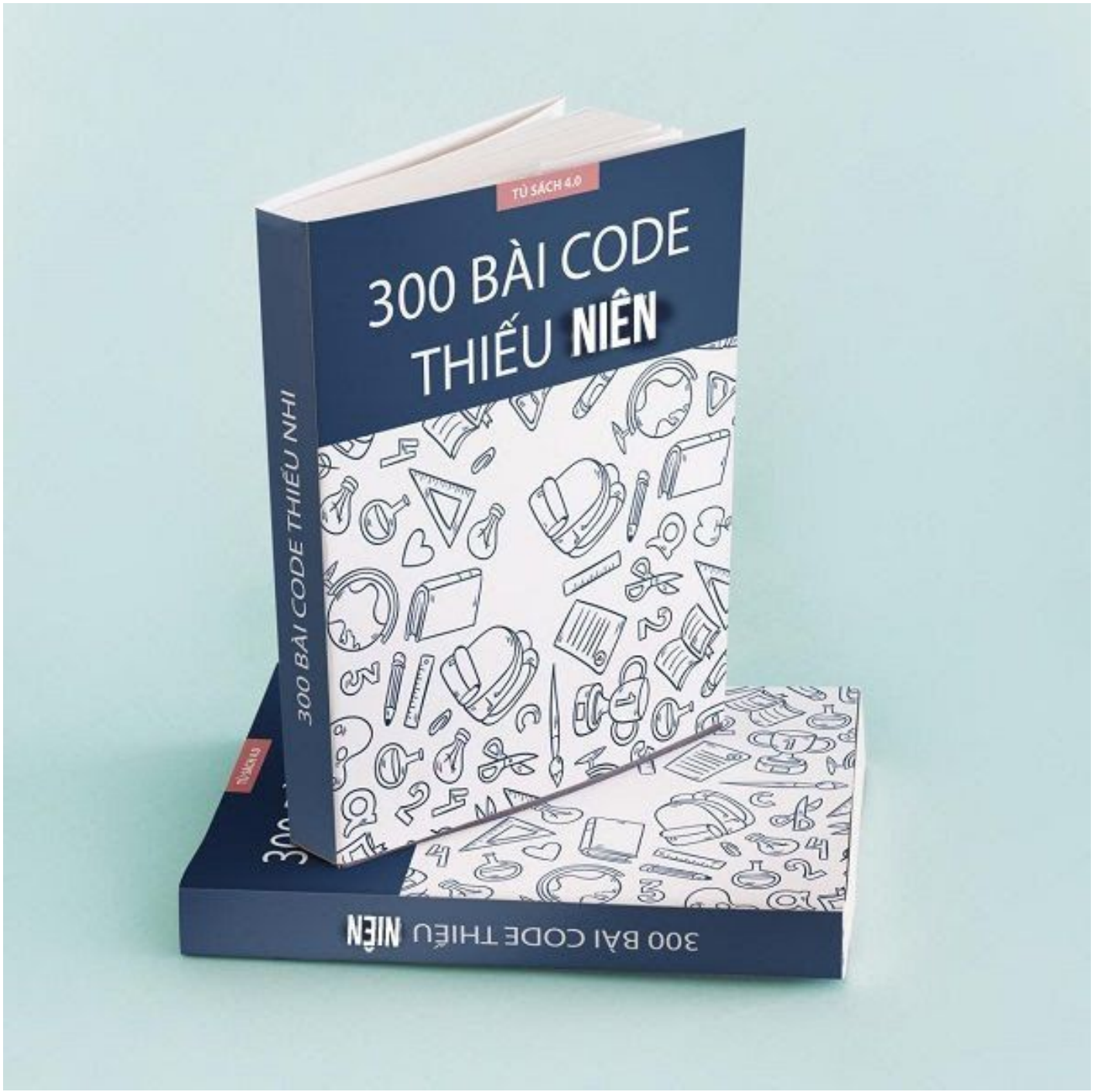

TỦ SÁCH 4.0
300 BÀI CODE
THIẾU NIÊN
300 BÀI CODE THIẾU NHI
300 BÀI CODE THIẾU NIÊN

Review từ những người đã học hỏi và đổi đời từ ebook

1. Bà hàng xóm nhà mình đây, trước đi buôn đồng nát, tình cờ có thằng sinh viên khoa cntt nó bán đống sách cũ. Thế nào mà bà ấy mua "300 bài code thiếu nhi" cầm về đọc.
Rồi sáng đi mua đồng nát, tối về đọc sách, cuối tuần ra quán net thực hành.
Sáu tháng sau bà ấy khăn gói lên HN đi phỏng vấn, cũng nhờ code trên giấy nhiều mà mấy bài "white board" bà ấy làm ngon ơ. Cũng 5 năm rồi, giờ đang làm lead ở một công ty khá lớn.
Đúng là cái nghề này mang lại cơ hội đổi đời cho nhiều người.
2. Quê tôi miền biển, có gia đình cạnh nhà làm nghề chài lưới. Bữa đi kéo lưới, thấy gì nặng nặng tưởng được mẻ cá to, ai ngờ toàn sách là sách. Nào là "300 bài code thiếu nhi", "Lập trình căn bản", " Machine Learning", "Deep learning", "AI"...
A định vất đi nhưng nhà mấy đời k biết mặt chữ là gì nên quyết tâm cầm về gối đầu giường. Đi học bổ túc văn hóa ban đêm phổ cập con chữ.
Ấy thế mà bẵng đi 6 tháng tôi từ thủ đô về thăm A khoe giờ ở nhà làm freelancer cho cty gì ở Mỹ ấy, to lắm, lương xấp xỉ 1 củ Trump/năm.
3. Gần nhà mình có ông tầm gần 30. bảo làm cà phê, tiêu mệt quá.
Thế là khăn gói xuống tp học 1 khóa lập trình pithon dip leaning gì đó, xong rồi làm 300 bài code thiếu nhi luyện tập. Bữa mới nói chuyện khoe đang làm lương cũng 1 2k gì đó!
4. Bác họ tui 46 rồi, chạy ba gác hoài mệt quá đi học lớp code cấp tốc. Học hết 2 tháng với làm hết bài trong cuốn 300 bài code thiếu nhi, xong apply vào công ty kia làm mảng data science mỗi tháng lương net hơn 400tr.
5. ông xe ôm xóm mình sinh năm 82, hôm trước chạy xe lớ ngớ thế nào rớt xuống cống, rồi nhặt được cuốn lập trình "code thiếu nhi" gì gì đó, về đọc đâu hơn 1 tháng rồi ra HN làm cho công ty trí tuệ nhân tạo to lắm, mới làm 1 năm mua được nhà HN, mua được thêm con mazda 6 rồi.
6. Giống tôi, trước đẩy xe hủ tiếu ngoài đường vô tình bắt gặp nhà nọ mở thời sự về blocktrain, thế là cứ ngày đi bán tối về coi NTN với tranh thủ coi clocktrain 1 xíu.
Về nhà tối nào tôi cũng làm 3 bài trong "300 bài code thiếu nhi", mà sau 3 tháng cũng apply được công ty về tiền ảo. Giờ tôi đánh sang cả mảng AI nữa, mới viết app di dộng auto deepfake có người trả 300k$ chưa bán.
7. Năm ngoái đi fuho sml ngoài công trình vô tình nhặt dc cuốn sách 300 bài lập trình dành cho thiếu nhi về nhà luyện tập theo sau 3 tháng tự tin apply 1 cty chuyên về ai, ml ở quận 1 lương 3k chưa thưởng hay phụ cấp đây.
8. 6 tháng trước làm nhà hàng vất vả quá, trong lúc thái thịt đọc lướt được cuốn java căn bản với 300 bài code thiếu nhi. Bây giờ dev full stack lương 2,2k
9. Mẹ thằng chú tôi làm sales sim số đẹp cho viettel đợt rồi đói ăn quá nên vất cho cuốn lập trình code thiếu nhi gì đấy, 6 tháng sau vào làm dev cứng FPT rồi , nghe bảo lương 2k vì ngành này đang hot
10. Ông già tôi làm công nhân than đuối quá, muốn kiếm việc gì nhẹ hơn. Tôi giới thiệu ng khoá học HTML CSS PHP trên ucademy với làm bài tập trong cuốn 300 bài code thiếu nhi.
Sau 6 tháng, giờ ổng ở nhà làm freelancer rung đùi hàng tháng tài khoản cứ bắn vào mấy ngàn $. Đang tính bảo bà già khỏi đi làm nữa ở nhà mà xài tiền.

Có sách cái l ý tin người vkl
Lừa bạn 1 lần, là lỗi của tui
Còn để mình tự bị lừa 2 lần, là lỗi của bạn ok!

LÊ MINH HOÀNG

ΦΦΦ✷φφφ

Bài giảng chuyên đề

Đại học Sư phạm Hà Nội, 1999-2002

Lời cảm ơn

Tôi muốn bày tỏ lòng biết ơn đối với những người thầy đã chỉ dạy tận tình trong những năm tháng đầy khó khăn khi tôi mới bước vào học tin học và lập trình. Sự hiểu biết và lòng nhiệt tình của các thầy không những đã cung cấp cho tôi những kiến thức quý báu mà còn là tấm gương sáng cho tôi noi theo khi tôi đứng trên bục giảng cũng với tư cách là một người thầy.

Cuốn tài liệu này được viết dựa trên những tài liệu thu thập được từ nhiều nguồn khác nhau, bởi công sức của nhiều thế hệ thầy trò đã từng giảng dạy và học tập tại Khối Phổ thông chuyên Toán-Tin, Đại học Sư phạm Hà Nội, còn tôi chỉ là người tổng hợp lại. Qua đây, tôi muốn gửi lời cảm ơn tới các đồng nghiệp đã đọc và đóng góp những ý kiến quí báu, cảm ơn các bạn học sinh - những con người đã trực tiếp làm nên cuốn sách này.

Do thời gian hạn hẹp, một số chuyên đề tuy đã có nhưng chưa kịp chỉnh sửa và đưa vào tài liệu. Bạn đọc có thể tham khảo thêm trong phần tra cứu. Rất mong nhận được những lời nhận xét và góp ý của các bạn để hoàn thiện cuốn sách này.

Tokyo, 28 tháng 4 năm 2003

Lê Minh Hoàng

Φ i φ

MỤC LỤC

PHẦN 1. BÀI TOÁN LIỆT KÊ 1

§1. NHẮC LẠI MỘT SỐ KIẾN THỨC ĐẠI SỐ TỔ HỢP.....2

1.1. CHỈNH HỢP LẶP.....2

1.2. CHỈNH HỢP KHÔNG LẶP.....2

1.3. HOÁN VỊ.....2

1.4. TỔ HỢP.....3

§2. PHƯƠNG PHÁP SINH (GENERATION) 4

2.1. SINH CÁC DÃY NHỊ PHÂN ĐỘ DÀI N 5

2.2. LIỆT KÊ CÁC TẬP CON K PHẦN TỬ.....6

2.3. LIỆT KÊ CÁC HOÁN VỊ 8

§3. THUẬT TOÁN QUAY LUI 12

3.1. LIỆT KÊ CÁC DÃY NHỊ PHÂN ĐỘ DÀI N.....12

3.2. LIỆT KÊ CÁC TẬP CON K PHẦN TỬ.....13

3.3. LIỆT KÊ CÁC CHỈNH HỢP KHÔNG LẶP CHẬP K 15

3.4. BÀI TOÁN PHÂN TÍCH SỐ.....16

3.5. BÀI TOÁN XẾP HẬU.....18

§4. KỸ THUẬT NHÁNH CẬN.....24

4.1. BÀI TOÁN TỐI ƯU.....24

4.2. SỰ BÙNG NỔ TỔ HỢP 24

4.3. MÔ HÌNH KỸ THUẬT NHÁNH CẬN.....24

4.4. BÀI TOÁN NGƯỜI DU LỊCH 25

4.5. DÃY ABC 28

PHẦN 2. CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 33

§1. CÁC BƯỚC CƠ BẢN KHI TIẾN HÀNH GIẢI CÁC BÀI TOÁN TIN HỌC.....34

1.1. XÁC ĐỊNH BÀI TOÁN.....34

1.2. TÌM CẤU TRÚC DỮ LIỆU BIỂU DIỄN BÀI TOÁN 34

1.3. TÌM THUẬT TOÁN 35

1.4. LẬP TRÌNH 37

1.5. KIỂM THỬ 37

1.6. TỐI ƯU CHƯƠNG TRÌNH 38

§2. PHÂN TÍCH THỜI GIAN THỰC HIỆN GIẢI THUẬT.....40

2.1. ĐỘ PHỨC TẠP TÍNH TOÁN CỦA GIẢI THUẬT 40

2.2. XÁC ĐỊNH ĐỘ PHỨC TẠP TÍNH TOÁN CỦA GIẢI THUẬT....40
2.3. ĐỘ PHỨC TẠP TÍNH TOÁN VỚI TÌNH TRẠNG DỮ LIỆU VÀO....43
2.4. CHI PHÍ THỰC HIỆN THUẬT TOÁN....43

Φ ii φ

§3. ĐỆ QUY VÀ GIẢI THUẬT ĐỆ QUY 45
3.1. KHÁI NIỆM VỀ ĐỆ QUY 45
3.2. GIẢI THUẬT ĐỆ QUY 45
3.3. VÍ DỤ VỀ GIẢI THUẬT ĐỆ QUY 46
3.4. HIỆU LỰC CỦA ĐỆ QUY 50

§4. CẤU TRÚC DỮ LIỆU BIỂU DIỄN DANH SÁCH.... 52
4.1. KHÁI NIỆM DANH SÁCH 52
4.2. BIỂU DIỄN DANH SÁCH TRONG MÁY TÍNH 52

§5. NGĂN XẾP VÀ HÀNG ĐỢI.... 58
5.1. NGĂN XẾP (STACK).... 58
5.2. HÀNG ĐỢI (QUEUE).... 60

§6. CÂY (TREE).... 64
6.1. ĐỊNH NGHĨA 64
6.2. CÂY NHỊ PHÂN (BINARY TREE) 65
6.3. BIỂU DIỄN CÂY NHỊ PHÂN 67
6.4. PHÉP DUYỆT CÂY NHỊ PHÂN.... 69
6.5. CÂY K_PHÂN 70
6.6. CÂY TỔNG QUÁT.... 71

§7. KÝ PHÁP TIỀN TỐ, TRUNG TỐ VÀ HẬU TỐ.... 74
7.1. BIỂU THỨC DƯỚI DẠNG CÂY NHỊ PHÂN 74
7.2. CÁC KÝ PHÁP CHO CÙNG MỘT BIỂU THỨC.... 74
7.3. CÁCH TÍNH GIÁ TRỊ BIỂU THỨC 75
7.4. CHUYỂN TỪ DẠNG TRUNG TỐ SANG DẠNG HẬU TỐ.... 78
7.5. XÂY DỰNG CÂY NHỊ PHÂN BIỂU DIỄN BIỂU THỨC.... 80

§8. SẮP XẾP (SORTING) 82
8.1. BÀI TOÁN SẮP XẾP.... 82
8.2. THUẬT TOÁN SẮP XẾP KIỂU CHỌN (SELECTIONSORT).... 84
8.3. THUẬT TOÁN SẮP XẾP NỔI BỌT (BUBBLESORT).... 85
8.4. THUẬT TOÁN SẮP XẾP KIỂU CHÈN.... 85
8.5. SHELLSORT.... 87

8.6. THUẬT TOÁN SẮP XẾP KIỂU PHÂN ĐOẠN (QUICKSORT)..... 88
8.7. THUẬT TOÁN SẮP XẾP KIỂU VUN ĐỐNG (HEAPSORT) 92
8.8. SẮP XẾP BẰNG PHÉP ĐẾM PHÂN PHỐI (DISTRIBUTION COUNTING) 95
8.9. TÍNH ỔN ĐỊNH CỦA THUẬT TOÁN SẮP XẾP (STABILITY) 96
8.10. THUẬT TOÁN SẮP XẾP BẰNG CƠ SỐ (RADIXSORT)..... 97
8.11. THUẬT TOÁN SẮP XẾP TRỘN (MERGESORT)..... 102
8.12. CÀI ĐẶT 105
8.13. ĐÁNH GIÁ, NHẬN XÉT..... 112

§9. TÌM KIẾM (SEARCHING) 116

9.1. BÀI TOÁN TÌM KIẾM.....116
9.2. TÌM KIẾM TUẦN TỰ (SEQUENTIAL SEARCH) 116
9.3. TÌM KIẾM NHỊ PHÂN (BINARY SEARCH) 116
9.4. CÂY NHỊ PHÂN TÌM KIẾM (BINARY SEARCH TREE - BST) 117
9.5. PHÉP BĂM (HASH).....122
9.6. KHOÁ SỐ VỚI BÀI TOÁN TÌM KIẾM 122
9.7. CÂY TÌM KIẾM SỐ HỌC (DIGITAL SEARCH TREE - DST).....123
9.8. CÂY TÌM KIẾM CƠ SỐ (RADIX SEARCH TREE - RST) 126
9.9. NHỮNG NHẬN XÉT CUỐI CÙNG 131

PHẦN 3. QUY HOẠCH ĐỘNG 133

§1. CÔNG THỨC TRUY HỒI.....134
1.1. VÍ DỤ.....134
1.2. CẢI TIẾN THỨ NHẤT.....135
1.3. CẢI TIẾN THỨ HAI.....137
1.4. CÀI ĐẶT ĐỆ QUY.....137

§2. PHƯƠNG PHÁP QUY HOẠCH ĐỘNG.....139
2.1. BÀI TOÁN QUY HOẠCH 139
2.2. PHƯƠNG PHÁP QUY HOẠCH ĐỘNG.....139

§3. MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG 143
3.1. DÃY CON ĐƠN ĐIỆU TĂNG DÀI NHẤT.....143
3.2. BÀI TOÁN CÁI TÚI.....148
3.3. BIẾN ĐỔI XÂU 150
3.4. DÃY CON CÓ TỔNG CHIA HẾT CHO K.....154
3.5. PHÉP NHÂN TỔ HỢP DÃY MA TRẬN.....159

3.6. BÀI TẬP LUYỆN TẬP....163

PHẦN 4. CÁC THUẬT TOÁN TRÊN ĐỒ THỊ 169

§1. CÁC KHÁI NIỆM CƠ BẢN....170

1.1. ĐỊNH NGHĨA ĐỒ THỊ (GRAPH)....170

1.2. CÁC KHÁI NIỆM....171

§2. BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH....173

2.1. MA TRẬN LIỀN KỀ (MA TRẬN KỀ) 173

2.2. DANH SÁCH CẠNH....174

2.3. DANH SÁCH KỀ....175

2.4. NHẬN XÉT....176

§3. CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ....177

3.1. BÀI TOÁN 177

3.2. THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU (DEPTH FIRST SEARCH)....178

3.3. THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG (BREADTH FIRST SEARCH) 184

Φ iv φ 3.4. ĐỘ PHỨC TẠP TÍNH TOÁN CỦA BFS VÀ DFS 189

§4. TÍNH LIÊN THÔNG CỦA ĐỒ THỊ..... 190

4.1. ĐỊNH NGHĨA 190

4.2. TÍNH LIÊN THÔNG TRONG ĐỒ THỊ VÔ HƯỚNG..... 191

4.3. ĐỒ THỊ ĐẦY ĐỦ VÀ THUẬT TOÁN WARSHALL 191

4.4. CÁC THÀNH PHẦN LIÊN THÔNG MẠNH 195

§5. VÀI ỨNG DỤNG CỦA CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ 205

5.1. XÂY DỰNG CÂY KHUNG CỦA ĐỒ THỊ 205

5.2. TẬP CÁC CHU TRÌNH CƠ BẢN CỦA ĐỒ THỊ 208

5.3. ĐỊNH CHIỀU ĐỒ THỊ VÀ BÀI TOÁN LIỆT KÊ CẦU 208

5.4. LIỆT KÊ KHỚP 214

§6. CHU TRÌNH EULER, ĐƯỜNG ĐI EULER, ĐỒ THỊ EULER..... 218

6.1. BÀI TOÁN 7 CÁI CẦU 218

6.2. ĐỊNH NGHĨA 218

6.3. ĐỊNH LÝ..... 218

6.4. THUẬT TOÁN FLEURY TÌM CHU TRÌNH EULER..... 219

6.5. CÀI ĐẶT 220

6.6. THUẬT TOÁN TỐT HƠN 222

§7. CHU TRÌNH HAMILTON, ĐƯỜNG ĐI HAMILTON, ĐỒ THỊ HAMILTON 225

7.1. ĐỊNH NGHĨA 225
7.2. ĐỊNH LÝ 225
7.3. CÀI ĐẶT 226

§8. BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT 230

8.1. ĐỒ THỊ CÓ TRỌNG SỐ 230
8.2. BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT 230
8.3. TRƯỜNG HỢP ĐỒ THỊ KHÔNG CÓ CHU TRÌNH ÂM - THUẬT TOÁN FORD BELLMAN 232
8.4. TRƯỜNG HỢP TRỌNG SỐ TRÊN CÁC CUNG KHÔNG ÂM - THUẬT TOÁN DIJKSTRA 234
8.5. THUẬT TOÁN DIJKSTRA VÀ CẤU TRÚC HEAP 237
8.6. TRƯỜNG HỢP ĐỒ THỊ KHÔNG CÓ CHU TRÌNH - THỨ TỰ TÔ PÔ 240
8.7. ĐƯỜNG ĐI NGẮN NHẤT GIỮA MỌI CẶP ĐỈNH - THUẬT TOÁN FLOYD 242
8.8. NHẬN XÉT 245

§9. BÀI TOÁN CÂY KHUNG NHỎ NHẤT 247

9.1. BÀI TOÁN CÂY KHUNG NHỎ NHẤT 247
9.2. THUẬT TOÁN KRUSKAL (JOSEPH KRUSKAL - 1956) 247
9.3. THUẬT TOÁN PRIM (ROBERT PRIM - 1957) 252

§10. BÀI TOÁN LUỒNG CỰC ĐẠI TRÊN MẠNG 256

10.1. BÀI TOÁN 256
10.2. LÁT CẮT, ĐƯỜNG TĂNG LUỒNG, ĐỊNH LÝ FORD - FULKERSON 256
10.3. CÀI ĐẶT 258

10.4. THUẬT TOÁN FORD - FULKERSON (L.R.FORD & D.R.FULKERSON - 1962) 262

§11. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI TRÊN ĐỒ THỊ HAI PHÍA 266

11.1. ĐỒ THỊ HAI PHÍA (BIPARTITE GRAPH) 266
11.2. BÀI TOÁN GHÉP ĐÔI KHÔNG TRỌNG VÀ CÁC KHÁI NIỆM 266
11.3. THUẬT TOÁN ĐƯỜNG MỞ 267
11.4. CÀI ĐẶT 268

§12. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI VỚI TRỌNG SỐ CỰC TIỂU TRÊN ĐỒ THỊ HAI PHÍA - THUẬT TOÁN HUNGARI 273

12.1. BÀI TOÁN PHÂN CÔNG 273
12.2. PHÂN TÍCH 273
12.3. THUẬT TOÁN 274
12.4. CÀI ĐẶT 278
12.5. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI VỚI TRỌNG SỐ CỰC ĐẠI TRÊN ĐỒ THỊ HAI PHÍA 284

12.6. NÂNG CẤP....284

§13. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI TRÊN ĐỒ THỊ 290

13.1. CÁC KHÁI NIỆM....290

13.2. THUẬT TOÁN EDMONDS (1965) 291

13.3. PHƯƠNG PHÁP LAWLER (1973)....293

13.4. CÀI ĐẶT....295

13.5. ĐỘ PHỨC TẠP TÍNH TOÁN 299

TÀI LIỆU ĐỌC THÊM 301

Φ vi φ

HÌNH VẼ

Hình 1: Cây tìm kiếm quay lui trong bài toán liệt kê dãy nhị phân.... 13

Hình 2: Xếp 8 quân hậu trên bàn cờ 8x8 19

Hình 3: Đường chéo ĐB-TN mang chỉ số 10 và đường chéo ĐN-TB mang chỉ số 0 19

Hình 4: Lưu đồ thuật giải (Flowchart).... 36

Hình 5: Tháp Hà Nội.... 49

Hình 6: Cấu trúc nút của danh sách nối đơn.... 53

Hình 7: Danh sách nối đơn.... 53

Hình 8: Cấu trúc nút của danh sách nối kép 55

Hình 9: Danh sách nối kép 55

Hình 10: Danh sách nối vòng một hướng.... 55

Hình 11: Danh sách nối vòng hai hướng 56

Hình 12: Dùng danh sách vòng mô tả Queue.... 61

Hình 13: Di chuyển toa tàu.... 63

Hình 14: Di chuyển toa tàu (2).... 63

Hình 15: Cây 64

Hình 16: Mức của các nút trên cây.... 65

Hình 17: Cây biểu diễn biểu thức.... 65

Hình 18: Các dạng cây nhị phân suy biến 66

Hình 19: Cây nhị phân hoàn chỉnh và cây nhị phân đầy đủ 66

Hình 20: Đánh số các nút của cây nhị phân đầy đủ để biểu diễn bằng mảng.... 67

Hình 21: Nhược điểm của phương pháp biểu diễn cây bằng mảng.... 68

Hình 22: Cấu trúc nút của cây nhị phân 68

Hình 23: Biểu diễn cây bằng cấu trúc liên kết.... 69

Hình 24: Đánh số các nút của cây 3_phân để biểu diễn bằng mảng..... 71
Hình 25: Biểu diễn cây tổng quát bằng mảng 72
Hình 26: Cấu trúc nút của cây tổng quát 73
Hình 27: Biểu thức dưới dạng cây nhị phân..... 74
Hình 28: Vòng lặp trong của QuickSort..... 89
Hình 29: Trạng thái trước khi gọi đệ quy 90
Hình 30: Heap 92
Hình 31: Vun đống..... 93
Hình 32: Đảo giá trị k_1 cho k^n và xét phần còn lại..... 93 Hình 33: Vun phần còn lại thành đống rồi lại đảo trị k_1 cho k^{n-1}..... 94 Hình 34: Đánh số các bit 97
Hình 35: Thuật toán sắp xếp trộn 102
Hình 36: Cài đặt các thuật toán sắp xếp với dữ liệu lớn..... 114
Hình 37: Cây nhị phân tìm kiếm 118
Hình 38: Xóa nút lá ở cây BST 119
Hình 39. Xóa nút chỉ có một nhánh con trên cây BST..... 120

Hình 40: Xóa nút có cả hai nhánh con trên cây BST thay bằng nút cực phải của cây con trái 120
Hình 41: Xóa nút có cả hai nhánh con trên cây BST thay bằng nút cực trái của cây con phải 120
Hình 42: Đánh số các bit.....123
Hình 43: Cây tìm kiếm số học.....124
Hình 44: Cây tìm kiếm cơ số 126
Hình 45: Với độ dài dãy bit z = 3, cây tìm kiếm cơ số gồm các khoá 2, 4, 5 và sau khi thêm giá trị 7 127
Hình 46: RST chứa các khoá 2, 4, 5, 7 và RST sau khi loại bỏ giá trị 7.....128
Hình 47: Cây tìm kiếm cơ số a) và Trie tìm kiếm cơ số b) 130
Hình 48: Hàm đệ quy tính số Fibonacci.....141
Hình 49: Tính toán và truy vết 144
Hình 50: Truy vết.....153
Hình 51: Ví dụ về mô hình đồ thị 170
Hình 52: Phân loại đồ thị 171
Hình 53 174
Hình 54 175
Hình 55: Đồ thị và đường đi 177
Hình 56: Cây DFS.....180
Hình 57: Cây BFS.....184
Hình 58: Thuật toán loang 187

Hình 59: Đồ thị G và các thành phần liên thông G1, G2, G3 của nó 190
Hình 60: Khớp và cầu 190
Hình 61: Liên thông mạnh và liên thông yếu.....191
Hình 62: Đồ thị đầy đủ.....192
Hình 63: Đơn đồ thị vô hướng và bao đóng của nó 192
Hình 64: Ba dạng cung ngoài cây DFS.....196
Hình 65: Thuật toán Tarjan "bẻ" cây DFS 198
Hình 66: Đánh số lại, đảo chiều các cung và duyệt BFS với cách chọn các đỉnh xuất phát ngược lại với thứ tự duyệt xong (thứ tự 11, 10... 3, 2, 1).....204
Hình 67: Đồ thị G và một số ví dụ cây khung T1, T2, T3 của nó 207
Hình 68: Cây khung DFS (a) và cây khung BFS (b) (Mũi tên chỉ chiều đi thăm các đỉnh).....207
Hình 69: Phép định chiều DFS.....210
Hình 70: Phép đánh số và ghi nhận cung ngược lên cao nhất.....212
Hình 71 Duyệt DFS, xác định cây DFS và các cung ngược.....215
Hình 72: Mô hình đồ thị của bài toán bảy cái cầu.....218
Hình 73 219
Hình 74 219
Hình 75 225
Hình 76: Phép đánh lại chỉ số theo thứ tự tôpô 240
Hình 77: Hai cây gốc r_1 và r_2 và cây mới khi hợp nhất chúng .....248 Hình 78: Mạng với các khả năng thông qua (1 phát, 6 thu) và một luồng của nó với giá trị 7 256
Hình 79: Mạng G, luồng trên các cung (1 phát, 6 thu) và đồ thị tăng luồng tương ứng.....257
Hình 80: Luồng trên mạng G trước và sau khi tăng.....258

Hình 81: Đồ thị hai phía..... 266
Hình 82: Đồ thị hai phía và bộ ghép M 267
Hình 83: Mô hình luồng của bài toán tìm bộ ghép cực đại trên đồ thị hai phía 271
Hình 84: Phép xoay trọng số cạnh..... 274
Hình 85: Thuật toán Hungari..... 277
Hình 86: Cây pha "mọc" lớn hơn sau mỗi lần xoay trọng số cạnh và tìm đường..... 285
Hình 87: Đồ thị G và một bộ ghép M..... 290
Hình 88: Phép chập Blossom 292
Hình 89: Nở Blossom để dò đường xuyên qua Blossom..... 292

CHƯƠNG TRÌNH

P_1_02_1.PAS * Thuật toán sinh liệt kê các dãy nhị phân độ dài n 6
P_1_02_2.PAS * Thuật toán sinh liệt kê các tập con k phần tử....8
P_1_02_3.PAS * Thuật toán sinh liệt kê hoán vị....9
P_1_03_1.PAS * Thuật toán quay lui liệt kê các dãy nhị phân độ dài n....12
P_1_03_2.PAS * Thuật toán quay lui liệt kê các tập con k phần tử....14
P_1_03_3.PAS * Thuật toán quay lui liệt kê các chỉnh hợp không lặp chập k....15
P_1_03_4.PAS * Thuật toán quay lui liệt kê các cách phân tích số....17
P_1_03_5.PAS * Thuật toán quay lui giải bài toán xếp hậu....21
P_1_04_1.PAS * Kỹ thuật nhánh cận dùng cho bài toán người du lịch....26
P_1_04_2.PAS * Dãy ABC 28
P_2_07_1.PAS * Tính giá trị biểu thức RPN....76
P_2_07_2.PAS * Chuyển biểu thức trung tố sang dạng RPN....79
P_2_08_1.PAS * Các thuật toán săp xếp 105
P_3_01_1.PAS * Đếm số cách phân tích số n 135
P_3_01_2.PAS * Đếm số cách phân tích số n 136
P_3_01_3.PAS * Đếm số cách phân tích số n 136
P_3_01_4.PAS * Đếm số cách phân tích số n 137
P_3_01_5.PAS * Đếm số cách phân tích số n dùng đệ quy....137
P_3_01_6.PAS * Đếm số cách phân tích số n dùng đệ quy....138
P_3_03_1.PAS * Tìm dãy con đơn điệu tăng dài nhất....144
P_3_03_2.PAS * Cải tiến thuật toán tìm dãy con đơn điệu tăng dài nhất....146
P_3_03_3.PAS * Bài toán cái túi....149
P_3_03_4.PAS * Biến đổi xâu....153
P_3_03_5.PAS * Dãy con có tổng chia hết cho k....156
P_3_03_6.PAS * Dãy con có tổng chia hết cho k....158
P_3_03_7.PAS * Nhân tối ưu dãy ma trận 162
P_4_03_1.PAS * Thuật toán tìm kiếm theo chiều sâu 178
P_4_03_2.PAS * Thuật toán tìm kiếm theo chiều sâu không đệ quy 181
P_4_03_3.PAS * Thuật toán tìm kiếm theo chiều rộng dùng hàng đợi 185
P_4_03_4.PAS * Thuật toán tìm kiếm theo chiều rộng dùng phương pháp loang 187
P_4_04_1.PAS * Thuật toán Warshall liệt kê các thành phần liên thông 194
P_4_04_2.PAS * Thuật toán Tarjan liệt kê các thành phần liên thông mạnh 201
P_4_05_1.PAS * Phép định chiều DFS và liệt kê cầu 213
P_4_05_2.PAS * Liệt kê các khớp của đồ thị....216
P_4_06_1.PAS * Thuật toán Fleury tìm chu trình Euler....220
P_4_06_2.PAS * Thuật toán hiệu quả tìm chu trình Euler 223

P_4_07_1.PAS * Thuật toán quay lui liệt kê chu trình Hamilton 226
P_4_08_1.PAS * Thuật toán Ford-Bellman.....233
P_4_08_2.PAS * Thuật toán Dijkstra 235
P_4_08_3.PAS * Thuật toán Dijkstra và cấu trúc Heap 237
Φ x φ

P_4_08_4.PAS * Đường đi ngắn nhất trên đồ thị không có chu trình 241
P_4_08_5.PAS * Thuật toán Floyd 243
P_4_09_1.PAS * Thuật toán Kruskal..... 249
P_4_09_2.PAS * Thuật toán Prim..... 252
P_4_10_1.PAS * Thuật toán tìm luồng cực đại trên mạng 259
P_4_10_2.PAS * Thuật toán Ford-Fulkerson..... 262
P_4_11_1.PAS * Thuật toán đường mở tìm bộ ghép cực đại 269
P_4_12_1.PAS * Thuật toán Hungari ..... 280 P_4_12_2.PAS * Cài đặt phương pháp Kuhn-Munkres $O(n^3)$ 286
P_4_13_1.PAS * Phương pháp Lawler áp dụng cho thuật toán Edmonds..... 296

PHẦN 1. BÀI TOÁN LIỆT KÊ

Có một số bài toán trên thực tế yêu cầu chỉ rõ: trong một tập các đối tượng cho trước có bao nhiêu đối tượng thoả mãn những điều kiện nhất định. Bài toán đó gọi là **bài toán đếm**.

Trong lớp các bài toán đếm, có những bài toán còn yêu cầu chỉ rõ những cấu hình tìm được thoả mãn điều kiện đã cho là những cấu hình nào. Bài toán yêu cầu đưa ra danh sách các cấu hình có thể có gọi là **bài toán liệt kê**.

Để giải bài toán liệt kê, cần phải xác định được một **thuật toán** để có thể theo đó lần lượt xây dựng được tất cả các cấu hình đang quan tâm.

Có nhiều phương pháp liệt kê, nhưng chúng cần phải đáp ứng được hai yêu cầu dưới đây:

- Không được lặp lại một cấu hình

• Không được bỏ sót một cấu hình

Có thể nói rằng, phương pháp liệt kê là phương kế cuối cùng để giải được một số bài toán tổ hợp hiện nay. Khó khăn chính của phương pháp này chính là sự bùng nổ tổ hợp dẫn tới sự đòi hỏi lớn về không gian và thời gian thực hiện chương trình. Tuy nhiên cùng với sự phát triển của máy tính điện tử, bằng phương pháp liệt kê, nhiều bài toán tổ hợp đã tìm thấy lời giải. Qua đó, ta cũng nên biết rằng **chỉ nên dùng phương pháp liệt kê khi không còn một phương pháp nào khác** tìm ra lời giải. Chính những nỗ lực giải quyết các bài toán thực tế không dùng phương pháp liệt kê đã thúc đẩy sự phát triển của nhiều ngành toán học.

Chuyên Φ 2 φ
đề

§1. NHẮC LẠI MỘT SỐ KIẾN THỨC ĐẠI SỐ TỔ HỢP

Cho S là một tập hữu hạn gồm n phần tử và k là một số tự nhiên.
Gọi X là tập các số nguyên dương từ 1 đến k: X = {1, 2, ..., k}

1.1. CHỈNH HỢP LẶP

Mỗi ánh xạ f: X → S. Cho tương ứng với mỗi i ∈ X, một và chỉ một phần tử f(i) ∈ S.
Được gọi là một **chỉnh hợp lặp** chập k của S.
Nhưng do X là tập hữu hạn (k phần tử) nên ánh xạ f có thể xác định qua bảng các giá trị f(1), f(2), ..., f(k).
Ví dụ: S = {A, B, C, D, E, F}; k = 3. Một ánh xạ f có thể cho như sau:

i 1 2 3 f(i) E C E Vậy có thể **đồng nhất f với dãy giá trị (f(1), f(2), ..., f(k))** và coi dãy giá trị này cũng là một chỉnh hợp lặp chập k của S. Như ví dụ trên (E, C, E) là một chỉnh hợp lặp chập 3 của S. Dễ dàng chứng minh được kết quả sau bằng quy nạp hoặc bằng phương pháp đánh giá khả năng lựa chọn:

Số chỉnh hợp lặp chập k của tập gồm n phần tử:

nA $^{k}_{n}$ **=** $_{k}$

1.2. CHỈNH HỢP KHÔNG LẶP

Khi f là đơn ánh có nghĩa là với ∀i, j ∈ X ta có f(i) = f(j) ⇔ i = j. Nói một cách dễ hiểu, khi dãy giá trị f(1), f(2), ..., f(k) gồm các phần tử thuộc S **khác nhau đôi một** thì f được gọi là một **chỉnh hợp không lặp chập k** của S. *Ví dụ một chỉnh hợp không lặp (C, A, E):*

i 1 2 3 f(i) C A E Số chỉnh hợp không lặp chập k của tập gồm n phần tử:

)1kn)...(2n)(1n(nA $^{k}_{n}$ **=**

− − − + =)!kn(**_** !n

1.3. HOÁN VỊ

Khi k = n. Một chỉnh hợp không lặp chập n của S được gọi là một **hoán vị** các phần tử của S.
Ví dụ: một hoán vị: (A, D, C, E, B, F) của S = {A, B, C, D, E, F}

i 1 2 3 4 5 6 f(i) A D C E B F Để ý rằng khi k = n thì số phần tử của tập X = {1, 2, ..., n} đúng bằng số phần tử của S. Do tính

chất đôi một khác nhau nên dãy f(1), f(2), ..., f(n) sẽ liệt kê được hết các phần tử trong S. Như vậy f là toàn ánh. Mặt khác do giả thiết f là chỉnh hợp không lặp nên f là đơn ánh. Ta có tương ứng 1-1

giữa các phần tử của X và S, do đó f là song ánh. Vậy nên ta có thể định nghĩa một hoán vị của S là một song ánh giữa {1, 2, ..., n} và S.

Số hoán vị của tập gồm n phần tử = số chỉnh hợp không lặp chập n:

$!nP_n$ **=**

1.4. TỔ HỢP

Một tập con gồm k phần tử của S được gọi là một **tổ hợp chập k** của S.

Lấy một tập con k phần tử của S, xét tất cả k! hoán vị của tập con này. Dễ thấy rằng các hoán vị đó là các chỉnh hợp không lặp chập k của S. Ví dụ lấy tập {A, B, C} là tập con của tập S trong ví dụ trên thì: (A, B, C), (C, A, B), (B, C, A), ... là các chỉnh hợp không lặp chập 3 của S. Điều đó tức là khi liệt kê tất cả các chỉnh hợp không lặp chập k thì mỗi tổ hợp chập k sẽ được tính k! lần. Vậy:

Số tổ hợp chập k của tập gồm n phần tử:

)!kn(!k

C

$_{kn}$ **=**

A_{k_n} !k**=** !n **_ Số tập con của tập n phần tử:**

2C...CC $_{0_n}$ **+**

^{1_n}

+ + $_{n_n}$

= $_n$

Chuyên Φ 4 φ

đề **(GENERATION)**

§2. PHƯƠNG PHÁP SINH

Phương pháp sinh có thể áp dụng để giải bài toán liệt kê tổ hợp đặt ra nếu như hai điều kiện sau thoả mãn:

- ***Có thể xác định được một thứ tự trên tập các cấu hình tổ hợp cần liệt kê. Từ đó có thể biết được cấu hình đầu tiên và cấu hình cuối cùng trong thứ tự đó.***
- ***Xây dựng được thuật toán từ một cấu hình chưa phải cấu hình cuối, sinh ra được cấu hình kế tiếp nó.***

Phương pháp sinh có thể mô tả như sau:

<Xây dựng cấu hình đầu tiên>; repeat
<Đưa ra cấu hình đang có>; <Từ cấu hình đang có sinh ra cấu hình kế tiếp nếu còn>; until <hết cấu hình>;

Thứ tự từ điển

Trên các kiểu dữ liệu đơn giản chuẩn, người ta thường nói tới khái niệm thứ tự. Ví dụ trên kiểu số thì có quan hệ: $1 < 2$; $2 < 3$; $3 < 10$; ..., trên kiểu ký tự Char thì cũng có quan hệ 'A' < 'B'; 'C' < 'c'...

Xét quan hệ thứ tự toàn phần "nhỏ hơn hoặc bằng" ký hiệu "≤" trên một tập hợp S, là quan hệ hai ngôi thoả mãn bốn tính chất:

Với $\forall a, b, c \in S$

Tính phổ biến: Hoặc là $a \leq b$, hoặc $b \leq a$;

Tính phản xạ: $a \leq a$

Tính phản đối xứng: Nếu $a \leq b$ và $b \leq a$ thì bắt buộc $a = b$.

Tính bắc cầu: Nếu có $a \leq b$ và $b \leq c$ thì $a \leq c$.

Trong trường hợp $a \leq b$ và $a \neq b$, ta dùng ký hiệu "<" cho gọn, (ta ngầm hiểu các ký hiệu như ≥, >, khỏi phải định nghĩa)

Ví dụ như quan hệ "≤" trên các số nguyên cũng như trên các kiểu vô hướng, liệt kê là quan hệ thứ tự toàn phần.

Trên các dãy hữu hạn, người ta cũng xác định một quan hệ thứ tự:

Xét $a = (a_1, a^2, ..., a^n)$ và $b = (b^1, b^2, ..., b^n)$; trên các phần tử của $a^1, ..., a^n, b^1, ..., b^n$ đã có quan hệ thứ tự "≤". Khi đó $a \leq b$ nếu như

Hoặc $a_i = b^i$ với $\forall i: 1 \leq i \leq n$.

Hoặc tồn tại một số nguyên dương k: $1 \leq k < n$ để:

$a_1 = b^1$

$a_2 = b^2$

$\cdots a^{k-1} = b^{k-1}$

$a_k = b^k$

$a_{k+1} < b^{k+1}$ Trong trường hợp này, ta có thể viết $a < b$.

Thứ tự đó gọi là thứ tự từ điển trên các dãy độ dài n.

Khi độ dài hai dãy a và b không bằng nhau, người ta cũng xác định được thứ tự từ điển. Bằng cách thêm vào cuối dãy a hoặc dãy b những phần tử đặc biệt gọi là phần tử ∅ để độ dài của a và b bằng nhau, và coi những phần tử ∅ này nhỏ hơn tất cả các phần tử khác, ta lại đưa về xác định thứ tự từ điển của hai dãy cùng độ dài. Ví dụ:

(1, 2, 3, 4) < (5, 6)

(a, b, c) < (a, b, c, d)

'calculator' < 'computer'

2.1. SINH CÁC DÃY NHỊ PHÂN ĐỘ DÀI N

Một dãy nhị phân độ dài n là một dãy $x = x_1x_2...x_n$ trong đó $x_i \in \{0, 1\}$ $(\forall i : 1 \le i \le n)$. Dễ thấy: một dãy nhị phân x độ dài n là biểu diễn nhị phân của một giá trị nguyên p(x) nào đó nằm trong đoạn $[0, 2^n - 1]$. Số các dãy nhị phân độ dài n = số các số nguyên $\in [0, 2^n - 1] = 2^n$. Ta sẽ lập chương trình liệt kê các dãy nhị phân theo thứ tự từ điển có nghĩa là sẽ liệt kê lần lượt các dãy nhị phân biểu diễn các số nguyên theo thứ tự 0, 1,..., 2^n-1.

Ví dụ: Khi n = 3, các dãy nhị phân độ dài 3 được liệt kê như sau:

p(x) 0 1 2 3 4 5 6 7

x 000 001 010 011 100 101 110 111 Như vậy dãy đầu tiên sẽ là 00...0 và dãy cuối cùng sẽ là 11...1.

Nhận xét rằng nếu dãy $x = (x_1, x_2, ..., x_n)$ là dãy đang có và không phải dãy cuối cùng thì dãy kế tiếp sẽ nhận được bằng cách cộng thêm 1 (theo cơ số 2 có nhớ) vào dãy hiện tại.

Ví dụ khi n = 8:

Dãy đang có: 10010000 Dãy đang có: 10010111 cộng thêm 1: + 1 cộng thêm 1: + 1 _____ _____ Dãy mới: 10010001 Dãy mới: 10011000

Như vậy kỹ thuật sinh cấu hình kế tiếp từ cấu hình hiện tại có thể mô tả như sau: Xét từ cuối dãy về đầu (xét từ hàng đơn vị lên), gặp số 0 đầu tiên thì thay nó bằng số 1 và đặt tất cả các phần tử phía sau vị trí đó bằng 0.

i := n; while (i > 0) and (x_i = 1) do i := i - 1; if i > 0 then
begin

Lê Minh Hoàng
Chuyên Φ 6 φ
đề

xfor i := 1;

j := i + 1 to n do $x_j := 0$; end; Dữ liệu vào (**Input**): nhập từ file văn bản BSTR.INP chứa số nguyên dương n ≤ 30
Kết quả ra (**Output**)**:** ghi ra file văn bản BSTR.OUT các dãy nhị phân độ dài n.
BSTR.INP 3
Đại học Sư phạm Hà Nội, 1999-2002 **BSTR.OUT 000 001 010 011 100 101 110 111**
P_1_02_1.PAS * Thuật toán sinh liệt kê các dãy nhị phân độ dài n program Binary_Strings; const
InputFile = 'BSTR.INP'; OutputFile = 'BSTR.OUT'; max = 30; var
x: array[1..max] of Integer; n, i: Integer; f: Text; begin
Assign(f, InputFile); Reset(f); ReadLn(f, n); Close(f); Assign(f, OutputFile); Rewrite(f); FillChar(x, repeat SizeOf(x),
{Thuật toán sinh}

0); {Cấu hình ban đầu $x_1 = x^2 = ... = x^n := 0$}

for i := 1 to n do Write(f, x[i]); {In ra cấu hình hiện tại} **WriteLn(f); i := n;** {x_i là phần tử cuối dãy, lùi dần i cho tới khi gặp số 0

hoặc khi i = 0 thì dừng} **while (i > 0) and (x[i] = 1) do Dec(i); if i > 0 then** {Chưa gặp phải cấu hình 11...1}
begin

x[i] := FillChar(x[i 1; {Thay **+ 1],** x_i **(n** bằng **-** số 1}

i) * SizeOf(x[1]), 0); {Đặt $x_{i+1} = x^{i+2} = ... = x^n := 0$} **end; until i = 0;** {Đã hết cấu hình} **Close(f); end.** **2.2. LIỆT KÊ**
CÁC TẬP CON K PHẦN TỬ
Ta sẽ lập chương trình liệt kê các tập con k phần tử của tập {1, 2, ..., n} theo thứ tự từ điển
Ví dụ: với n = 5, k = 3, ta phải liệt kê đủ 10 tập con:

1.{1, 2, 3} 2.{1, 2, 4} 3.{1, 2, 5} 4.{1, 3, 4} 5.{1, 3, 5} 6.{1, 4, 5} 7.{2, 3, 4} 8.{2, 3, 5} 9.{2, 4, 5} 10.{3, 4, 5} Như vậy tập con đầu tiên (cấu hình khởi tạo) là {1, 2, ..., k}.
Cấu hình kết thúc là {n - k + 1, n - k + 2, ..., n}.
Nhận xét: Ta sẽ in ra tập con bằng cách in ra lần lượt các phần tử của nó theo thứ tự tăng dần. Từ đó,

ta có nhận xét nếu x = $\{x_1, x^2, ..., x^k\}$ và $x^1 < x^2 < ... < x^k$ thì giới hạn trên (giá trị lớn nhất có thể nhận) của x_k

là n, của x^{k-1} là n - 1, của x^{k-2} là n - 2...

Cụ thể: **giới hạn trên của x_i = n - k + i;** Còn tất nhiên, **giới hạn dưới của x_i (giá trị nhỏ nhất x^i có thể nhận) là $x^{i-1} + 1$.** Như vậy nếu ta đang có một dãy x đại diện cho một tập con, nếu x là cấu hình kết thúc có nghĩa là

tất cả các phần tử trong x đều đã đạt tới giới hạn trên thì quá trình sinh kết thúc, nếu không thì ta

phải sinh ra một dãy x mới tăng dần thoả mãn **vừa đủ lớn hơn dãy cũ** theo nghĩa không có một tập

con k phần tử nào chen giữa chúng khi sắp thứ tự từ điển.

*Ví dụ: n = 9, k = 6. Cấu hình đang có x = {1, 2, 6, 7, 8, 9}. Các phần tử x_3 đến x^6 đã đạt tới giới hạn trên nên để sinh cấu hình mới ta không thể sinh bằng cách tăng một phần tử trong số các x_6, x^5, x_4, x^3 lên được, ta phải tăng x^2 = 2 lên thành x^2 = 3. Được cấu hình mới là x = {1, 3, 6, 7, 8, 9}. Cấu hình này đã thoả mãn lớn hơn cấu hình trước nhưng chưa thoả mãn tính chất **vừa đủ lớn** muốn vậy*

ta lại thay x_3, x^4, x^5, x^6 bằng các giới hạn dưới của nó. Tức là:

• $x_3 := x^2 + 1 = 4$

• $x_4 := x^3 + 1 = 5$

• $x_5 := x^4 + 1 = 6$

• $x_6 := x^5 + 1 = 7$ *Ta được cấu hình mới x = {1, **3**, 4, 5, 6, 7} là cấu hình kế tiếp. Nếu muốn tìm tiếp, ta lại nhận thấy*

*rằng $x_6 = 7$ chưa đạt giới hạn trên, như vậy chỉ cần tăng x_6 lên 1 là được x = {1, 3, 4, 5, 6, **8**}.* ***Vậy kỹ thuật sinh tập con kế tiếp từ tập đã có x có thể xây dựng như sau:***

Tìm từ cuối dãy lên đầu cho tới khi gặp một phần tử x_i chưa đạt giới hạn trên n - k + i.

i := n; while (i > 0) and (x_i = n - k + i) do i := i - 1; ***(1, 2, 6, 7, 8, 9);***

Nếu tìm thấy:

if i > 0 then Tăng x_i đó lên 1.

$x_i := x^i + 1$;

(1, 3, 6, 7, 8, 9)

Đặt tất cả các phần tử phía sau x_i bằng giới hạn dưới:

for j := i + 1 to k do $x_j := x^{j-1} + 1$;

(1, 3, 4, 5, 6, 7)

Input: file văn bản SUBSET.INP chứa hai số nguyên dương n, k (1 ≤ k ≤ n ≤ 30) cách nhau ít nhất

một dấu cách

Output: file văn bản SUBSET.OUT các tập con k phần tử của tập {1, 2, ..., n}

Lê Minh Hoàng
Chuyên Φ 8 φ
đề
SUBSET.INP 5 3
Đại học Sư phạm Hà Nội, 1999-2002 **SUBSET.OUT {1, 2, 3} {1, 2, 4} {1, 2, 5} {1, 3, 4} {1, 3, 5} {1, 4, 5} {2, 3, 4} {2, 3, 5} {2, 4, 5} {3, 4, 5}**
P_1_02_2.PAS * Thuật toán sinh liệt kê các tập con k phần tử program Combination; const InputFile = 'SUBSET.INP'; OutputFile = 'SUBSET.OUT'; max = 30; var

```
x: array[1..max] of Integer; n, k, i, j: Integer; f: Text; begin
Assign(f, InputFile); Reset(f); ReadLn(f, n, k); Close(f); Assign(f, OutputFile); Rewrite(f); for i repeat
:= 1 to k do x[i] := i; {x1 := 1; x2 := 2; ... ; x3 := k (Cấu hình khởi tạo)}
{In ra cấu hình hiện tại} Write(f, '{'); for i := 1 to k - 1 do Write(f, x[i], ', '); WriteLn(f, x[k], '}'); {Sinh tiếp} i := while k; (i
{x> i 0) là and phần (x[i] tử cuối = dãy, n - k lùi dần + i) do i Dec(i);
cho tới khi gặp một xi chưa đạt giới hạn trên n - k + i}
if i > 0 then {Nếu chưa lùi đến 0 có nghĩa là chưa phải cấu hình kết thúc}
begin
Inc(x[i]); {Tăng xi lên 1, Đặt các phần tử đứng sau xi bằng giới hạn dưới của nó} for j := i + 1 to k do x[j] := x[j - 1] + 1; end;
until i = 0; {Lùi đến tận 0 có nghĩa là tất cả các phần tử đã đạt giới hạn trên - hết cấu hình} Close(f); end.
```

2.3. LIỆT KÊ CÁC HOÁN VỊ

Ta sẽ lập chương trình liệt kê các hoán vị của {1, 2, ..., n} theo thứ tự từ điển.

Ví dụ với n = 4, ta phải liệt kê đủ 24 hoán vị:

1.1234 2.1243 3.1324 4.1342 5.1423 6.1432 7.2134 8.2143 9.2314 10.2341 11.2413 12.2431 13.3124 14.3142 15.3214 16.3241 17.3412 18.3421 19.4123 20.4132 21.4213 22.4231 23.4312 24.4321 Như vậy hoán vị đầu tiên sẽ là (1, 2, ..., n). Hoán vị cuối cùng là (n, n-1, ... , 1).

Hoán vị sẽ sinh ra phải lớn hơn hoán vị hiện tại, hơn thế nữa phải là hoán vị vừa đủ lớn hơn hoán vị hiện tại theo nghĩa không thể có một hoán vị nào khác chen giữa chúng khi sắp thứ tự.

Giả sử hoán vị hiện tại là x = (3, 2, 6, 5, 4, 1), xét 4 phần tử cuối cùng, ta thấy chúng được xếp giảm dần, điều đó có nghĩa là cho dù ta có hoán vị 4 phần tử này thế nào, ta cũng được một hoán vị bé

hơn hoán vị hiện tại!. Như vậy ta phải xét đến x2 = 2, thay nó bằng một giá trị khác. Ta sẽ thay bằng giá trị nào?, không thể là 1 bởi nếu vậy sẽ được hoán vị nhỏ hơn, không thể là 3 vì đã có x1 = 3 rồi (phần tử sau không được chọn vào những giá trị mà phần tử trước đã chọn). Còn lại các giá trị 4, 5, 6. Vì cần một hoán vị **vừa đủ lớn hơn hiện tại** nên ta chọn $x_2 = 4$. Còn các giá trị (x_3, x_4, x_5, x_6) sẽ lấy trong tập {2, 6, 5, 1}. Cũng vì tính vừa đủ lớn nên ta sẽ tìm biểu diễn nhỏ nhất của 4 số này gán cho x_3, x_4, x_5, x_6 tức là (1, 2, 5, 6). Vậy hoán vị mới sẽ là (3, 4, 1, 2, 5, 6).

(3, ***2***, 6, 5, 4, 1) → (3, **4, 1, 2, 5, 6**).

Ta có nhận xét gì qua ví dụ này: Đoạn cuối của hoán vị được xếp giảm dần, số $x_5 = 4$ là số nhỏ nhất trong đoạn cuối giảm dần thoả mãn điều kiện lớn hơn $x_2 = 2$. Nếu đổi chỗ x_5 cho x_2 thì ta sẽ được $x_2 = 4$ và đoạn cuối **vẫn được sắp xếp giảm dần**. Khi đó muốn biểu diễn nhỏ nhất cho các giá trị trong đoạn cuối thì ta chỉ cần đảo ngược đoạn cuối.

Trong trường hợp hoán vị hiện tại là (2, 1, 3, 4) thì hoán vị kế tiếp sẽ là (2, 1, 4, 3). Ta cũng có thể coi hoán vị (2, 1, 3, 4) có đoạn cuối giảm dần, đoạn cuối này chỉ gồm 1 phần tử (4)

Vậy kỹ thuật sinh hoán vị kế tiếp từ hoán vị hiện tại có thể xây dựng như sau:

• *Xác định đoạn cuối giảm dần dài nhất, tìm chỉ số i của phần tử x_i đứng liền trước đoạn cuối đó. Điều này đồng nghĩa với việc tìm từ vị trí sát cuối dãy lên đầu, gặp chỉ số i đầu tiên thỏa mãn $x_i < x_{i+1}$. Nếu toàn dãy đã là giảm dần, thì đó là cấu hình cuối.*

i := n - 1; while (i > 0) and ($x_i > x_{i+1}$) do i := i - 1;

• *Trong đoạn cuối giảm dần, tìm phần tử x_k nhỏ nhất thoả mãn điều kiện $x_k > x_i$. Do đoạn cuối giảm dần, điều này thực hiện bằng cách tìm từ cuối dãy lên đầu gặp chỉ số k đầu tiên thoả mãn $x_k > x_i$ (có thể dùng tìm kiếm nhị phân).*

k := n; while $x_k < x_i$ do k := k - 1;

• *Đổi chỗ x_k và x_i, lật ngược thứ tự đoạn cuối giảm dần (từ x_{i+1} đến x_k) trở thành tăng dần.*

Input: file văn bản PERMUTE.INP chứa số nguyên dương n ≤ 12

Output: file văn bản PERMUTE.OUT các hoán vị của dãy (1, 2, ..., n)

PERMUTE.INP 3	**PERMUTE.OUT 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1**

Lê Minh Hoàng

P_1_02_3.PAS * Thuật toán sinh liệt kê hoán vị program

Permutation; const
InputFile = 'PERMUTE.INP'; OutputFile
= 'PERMUTE.OUT'; max = 12; var
n, i, k, a, b: Integer; x: array[1..max] of
Integer; f: Text;
Chuyên Φ 10 φ
đề
procedure Swap(var X, Y: Integer); {Thủ tục đảo giá trị hai tham biến X, Y} **var**
Temp: Integer; begin
Temp := X; X := Y; Y := Temp; end;
begin
Assign(f, InputFile); Reset(f); ReadLn(f, n); Close(f); Assign(f, OutputFile); Rewrite(f); for i := 1 to n do x[i] := i; {Khởi tạo cấu hình đầu: **repeat**

$x_1 := 1; x^2 := 2; ..., x^n := n\}$
for i := 1 to n do Write(f, x[i], ' '); {In ra cấu hình hoán vị hiện tại} **WriteLn(f); i := n - 1; while (i > 0) and (x[i] > x[i + 1]) do Dec(i); if i > 0 then** {Chưa gặp phải hoán vị cuối (n, n-1, ... ,1)}
begin

k := n; {x_k là phần tử cuối dãy} **while Swap(x[k], a := x[k] i + < x[i] x[i]); 1; b := do Dec(k);** {Lùi dần k **n;** {Đổi {Lật chỗ

ngược x_k và đoạn x^i}

cuối để tìm gặp x_k đầu tiên lớn hơn x^i }
giảm dần, a: đầu đoạn, b: cuối đoạn} **while a < b do**
begin

Swap(x[a], Inc(a); x[b]); {Tiến a {Đổi và lùi chỗ b, đổi x_a và chỗ x^b}
tiếp cho tới khi a, b chạm nhau} **Dec(b); end; end; until i = 0;** {Toàn dãy là dãy giảm dần - không sinh tiếp được - hết cấu hình}
Close(f); end. **Bài tập:**

Bài 1

Các chương trình trên xử lý không tốt trong trường hợp tầm thường, đó là trường hợp n = 0 đối với chương trình liệt kê dãy nhị phân cũng như trong chương trình liệt kê hoán vị, trường hợp k = 0 đối với chương trình liệt kê tổ hợp, hãy khắc phục điều đó.

Bài 2

Liệt kê các dãy nhị phân độ dài n có thể coi là liệt kê các chỉnh hợp lặp chập n của tập 2 phần tử {0, 1}. Hãy lập chương trình:

Nhập vào hai số n và k, liệt kê các chỉnh hợp lặp chập k của {0, 1, ..., n -1}.

Gợi ý: thay hệ cơ số 2 bằng hệ cơ số n.

Bài 3

Hãy liệt kê các dãy nhị phân độ dài n mà trong đó cụm chữ số "01" xuất hiện đúng 2 lần.

Bài 4.

Nhập vào một danh sách n tên người. Liệt kê tất cả các cách chọn ra đúng k người trong số n người đó.

Gợi ý: xây dựng một ánh xạ từ tập {1, 2, ..., n} đến tập các tên người. Ví dụ xây dựng một mảng Tên: Tên[1] := 'Nguyễn văn A'; Tên[2] := 'Trần thị B';.... sau đó liệt kê tất cả các tập con k phần tử của tập {1, 2, ..., n}. Chỉ có điều khi in tập con, ta không in giá trị số {1, 3, 5} mà thay vào đó sẽ in ra {Tên[1], Tên [3], Tên[5]}. Tức là in ra ảnh của các giá trị tìm được qua ánh xạ

Bài 5

Liệt kê tất cả các tập con của tập {1, 2, ..., n}. Có thể dùng phương pháp liệt kê tập con như trên hoặc dùng phương pháp liệt kê tất cả các dãy nhị phân. Mỗi số 1 trong dãy nhị phân tương ứng với một phần tử được chọn trong tập. Ví dụ với tập {**1**, 2, **3**, 4} thì dãy nhị phân 1010 sẽ tương ứng với tập con {1, 3}. Hãy lập chương trình in ra tất cả các tập con của {1, 2, ..., n} theo hai phương pháp.

Bài 6

Nhập vào danh sách tên n người, in ra tất cả các cách xếp n người đó vào một bàn

Bài 7

Nhập vào danh sách n bạn nam và n bạn nữ, in ra tất cả các cách xếp 2n người đó vào một bàn tròn, mỗi bạn nam tiếp đến một bạn nữ.

Bài 8

Người ta có thể dùng phương pháp sinh để liệt kê các chỉnh hợp không lặp chập k. Tuy nhiên có một cách khác là liệt kê tất cả các tập con k phần tử của tập hợp, sau đó in ra đủ k! hoán vị của nó. Hãy viết chương trình liệt kê các chỉnh hợp không lặp chập k của {1, 2, ..., n} theo cả hai cách.

Bài 9

Liệt kê tất cả các hoán vị chữ cái trong từ MISSISSIPPI theo thứ tự từ điển.

Bài 10

Liệt kê tất cả các cách phân tích số nguyên dương n thành tổng các số nguyên dương, hai cách phân tích là hoán vị của nhau chỉ tính là một cách.

Cuối cùng, ta có nhận xét, mỗi phương pháp liệt kê đều có ưu, nhược điểm riêng và phương pháp sinh cũng không nằm ngoài nhận xét đó. Phương pháp sinh **không thể sinh ra được cấu hình thứ p** nếu như chưa có cấu hình thứ p - 1, chứng tỏ rằng phương pháp sinh tỏ ra ưu điểm trong trường hợp liệt kê toàn bộ một **số lượng nhỏ cấu hình trong một bộ dữ liệu lớn** thì lại có nhược điểm và ít tính phổ dụng trong những thuật toán **duyệt hạn chế.** Hơn thế nữa, không phải cấu hình ban đầu lúc nào cũng dễ tìm được, không phải kỹ thuật sinh cấu hình kế tiếp cho mọi bài toán đều đơn giản như trên (Sinh các chỉnh hợp không lặp chập k theo thứ tự từ điển chẳng hạn). Ta sang một chuyên mục sau nói đến một phương pháp liệt kê có tính phổ dụng cao hơn, để giải các bài toán liệt kê phức tạp hơn đó là: Thuật toán quay lui (Back tracking).

Lê Minh Hoàng
Chuyên Φ 12 φ
đề

§3. THUẬT TOÁN QUAY LUI

Thuật toán quay lui dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng cách xây dựng từng phần tử, mỗi phần tử được chọn bằng cách thử tất cả các khả năng.

Giả thiết cấu hình cần liệt kê có dạng $(x_1, x^2,..., x^n)$. Khi đó thuật toán quay lui thực hiện qua các bước sau:

1) Xét tất cả các giá trị x_1 có thể nhận, thử cho x_1 nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho x_1 ta sẽ:

2) Xét tất cả các giá trị x_2 có thể nhận, lại thử cho x_2 nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho x_2 lại xét tiếp các khả năng chọn x_3 ... cứ tiếp tục như vậy đến bước:

⋯ n) Xét tất cả các giá trị x_n có thể nhận, thử cho x_n nhận lần lượt các giá trị đó, thông báo cấu hình tìm được $(x_1, x_2, ..., x_n)$. Trên phương diện quy nạp, có thể nói rằng thuật toán quay lui liệt kê các cấu hình n phần tử dạng $(x_1, x_2, .., x_n)$ bằng cách thử cho x_1 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_1 lại liệt kê tiếp cấu hình n - 1 phần tử $(x_2, x_3, ..., x_n)$. **Mô hình của thuật toán quay lui có thể mô tả như sau:**

{Thủ tục **procedure** này thử **Try(i:** cho **Integer);**

x_i nhận lần lượt các giá trị mà nó có thể nhận}
begin
for begin

(mọi giá trị V có thể gán cho x_i) do

<Thử if (xcho $_i$ <Thông là x phần $_i$:= V>;

tử cuối cùng trong cấu hình) then báo cấu hình tìm được> else

begin

<Ghi Try(i <Nếu nhận + cần, 1); việc {Gọi **cho** đệ x_i quy **nhận** để **giá** chọn **trị** tiếp **bỏ ghi nhận việc thử x_i V (Nếu cần)>;**

:= V, x_{i+1}}

để thử giá trị khác>; end; end; end; ***Thuật toán quay lui sẽ bắt đầu bằng lời gọi Try(1)***

3.1. LIỆT KÊ CÁC DÃY NHỊ PHÂN ĐỘ DÀI N

Input/Output với khuôn dạng như trong P_1_02_1.PAS

Biểu diễn dãy nhị phân độ dài N dưới dạng $(x_1, x_2, ..., x_n)$. Ta sẽ liệt kê các dãy này bằng cách thử dùng các giá trị {0, 1} gán cho x_i. Với mỗi giá trị thử gán cho x_i lại thử các giá trị có thể gán cho x_{i+1}.Chương trình liệt kê bằng thuật toán quay lui có thể viết:

P_1_03_1.PAS * Thuật toán quay lui liệt kê các dãy nhị phân độ dài n program BinaryStrings; const
Đại học Sư phạm Hà Nội, 1999-2002
Φ Bài toán liệt kê
13 φ
InputFile = 'BSTR.INP'; OutputFile = 'BSTR.OUT'; max = 30; var
x: array[1..max] of Integer; n: Integer; f: Text;
procedure PrintResult; {In cấu hình tìm được, do thủ tục tìm đệ quy Try gọi khi tìm ra một cấu hình} **var**
i: Integer; begin
for i := 1 to n do Write(f, x[i]); WriteLn(f); end;

procedure Try(i: Integer); {Thử các cách chọn x_i} **var**

j: Integer; begin
for begin

j := 0 to 1 do {Xét các giá trị có thể gán cho x_i, với mỗi giá trị đó}

x[i] if i := = n j; then {Thử **PrintResult** đặt x_i}

{Nếu i = n thì in kết quả}
end;

else Try(i + 1); {Nếu i chưa phải là phần tử cuối thì tìm tiếp x_{i+1}}

end;

begin

Assign(f, InputFile); Reset(f); ReadLn(f, n); {Nhập dữ liệu} **Close(f); Assign(f, OutputFile); Rewrite(f); Try(1); Close(f);**

{Thử các cách chọn giá trị x_1}

end. ***Ví dụ: Khi n = 3, cây tìm kiếm quay lui như sau:***

Lê Minh Hoàng

$X_1=0$ $X_1=1$

$X_2=0$ $X_2=1$ $X_2=0$ $X_2=1$

$X_3=0$ $X_3=1$ $X_3=0$ $X_3=1$ $X_3=0$ $X_3=1$ $X_3=0$ $X_3=1$

Hình 1: Cây tìm kiếm quay lui trong bài toán liệt kê dãy nhị phân

3.2. LIỆT KÊ CÁC TẬP CON K PHẦN TỬ

Input/Output có khuôn dạng như trong P_1_02_2.PAS

000
Try(3)
001
Try(2)
010
Try(3)
Try(1)
011
100
Try(3)
Try(2)
101
Try(3)
110
111 Result

Chuyên Φ 14 φ đề

Để liệt kê các tập con k phần tử của tập S = {1, 2, ..., n} ta có thể đưa về liệt kê các cấu hình (x_1, x_2, ..., x_k) ở đây các $x_i \in S$ và $x_1 < x_2 < ... < x_k$. Ta có nhận xét:

$x_k \le n$

$x_{k-1} \le x_k - 1 \le n - 1$

$\cdots x_i \le n - k + i$

$\cdots x_1 \le n - k + 1.$

Từ đó suy ra $x_{i-1} + 1 \le x_i \le n - k + i$ ($1 \le i \le k$) ở đây ta giả thiết có thêm một số $x_0 = 0$ khi xét i = 1.

Như vậy ta sẽ xét tất cả các cách chọn x_1 từ 1 ($=x_0 + 1$) đến n - k + 1, với mỗi giá trị đó, xét tiếp tất cả các cách chọn x_2 từ $x_1 + 1$ đến n - k + 2,... cứ như vậy khi chọn được đến x_k thì ta có một cấu hình cần liệt kê. Chương trình liệt kê bằng thuật toán quay lui như sau:

P_1_03_2.PAS * Thuật toán quay lui liệt kê các tập con k phần tử

program Combination; const

```
InputFile = 'SUBSET.INP'; OutputFile =
'SUBSET.OUT'; max = 30; var
  x: array[0..max] of Integer; n, k: Integer;
  f: Text;

procedure
var

        i: Integer; begin
Write(f, '{'); for i := 1 to k - 1 do Write(f, x[i], ', ');
WriteLn(f, x[k], '}'); end;

procedure Try(i: Integer); {Thử các cách chọn giá trị cho x[i]} var
        j: Integer; begin
  for j := x[i - 1] + 1 to n - k + i do
    begin
x[i] := j; if i = k then PrintResult else Try(i +
1); end; end;

begin
Assign(f, InputFile); Reset(F); ReadLn(f, n, k);
Close(f); Assign(f, OutputFile); Rewrite(f); x[0] :=
0; Try(1); Close(f); end.
```


Nếu để ý chương trình trên và chương trình liệt kê dãy nhị phân độ dài n, ta thấy về cơ bản chúng

chỉ khác nhau ở thủ tục Try(i) - chọn thử các giá trị cho x_i, ở chương trình liệt kê dãy nhị phân ta thử chọn các giá

trị 0 hoặc 1 còn ở chương trình liệt kê các tập con k phần tử ta thử chọn x_i là một trong các giá trị nguyên từ x_{i-1} +

1 đến n - k + i. Qua đó ta có thể thấy tính phổ dụng của thuật toán quay lui: mô hình cài đặt có thể thích hợp cho nhiều bài toán, khác với phương pháp sinh tuần tự,

với mỗi bài toán lại phải có một thuật toán sinh kế tiếp riêng làm cho việc cài đặt mỗi bài một khác,

bên cạnh đó, không phải thuật toán sinh kế tiếp nào cũng dễ cài đặt.

3.3. LIỆT KÊ CÁC CHỈNH HỢP KHÔNG LẶP CHẬP K

Để liệt kê các chỉnh hợp không lặp chập k của tập S = {1, 2, ..., n} ta có thể đưa về liệt kê các cấu

hình (x_1, x^2, ..., x^k) ở đây các $x^i \in$ S và khác nhau đôi một. Như vậy thủ tục Try(i) - xét tất cả các khả năng chọn x_i

- sẽ thử hết các giá trị từ 1 đến n, mà các giá trị này chưa bị các phần tử đứng trước chọn. Muốn xem các giá trị nào chưa được chọn ta sử dụng

kỹ thuật dùng mảng đánh dấu:

Khởi tạo một mảng c_1, c^2, ..., c^n mang kiểu logic. Ở đây c^i cho biết giá trị i có còn tự do hay đã bị chọn rồi. Ban đầu

khởi tạo tất cả các phần tử mảng c là TRUE có nghĩa là các phần tử từ 1 đến n

đều tự do.

Tại bước chọn các giá trị có thể của x_i ta chỉ xét những giá trị j có c_j = TRUE có nghĩa là **chỉ chọn những giá trị tự do.**

Trước khi gọi đệ quy tìm xi+1: ta đặt giá trị j vừa gán cho xi là **đã bị chọn** có nghĩa là đặt c_j := FALSE để các thủ tục Try(i + 1), Try(i + 2)... gọi sau này không chọn phải giá trị j đó nữa

Sau khi gọi đệ quy tìm x_{i+1}: có nghĩa là sắp tới ta sẽ thử gán một **giá trị khác** cho xi thì ta sẽ đặt giá trị j vừa thử đó thành **tự do** (cj := TRUE), bởi khi xi đã nhận một giá trị khác rồi thì các phần tử

đứng sau: xi+1, xi+2 ... hoàn toàn có thể nhận lại giá trị j đó. Điều này hoàn toàn hợp lý trong phép

xây dựng chỉnh hợp không lặp: x1 có n cách chọn, x2 có n - 1 cách chọn, ...Lưu ý rằng khi thủ tục

Try(i) có i = k thì ta không cần phải đánh dấu gì cả vì tiếp theo chỉ có in kết quả chứ không cần phải

chọn thêm phần tử nào nữa.

Input: file văn bản ARRANGE.INP chứa hai số nguyên dương n, k (1 ≤ k ≤ n ≤ 20) cách nhau ít

nhất một dấu cách

Output: file văn bản ARRANGE.OUT ghi các chỉnh hợp không lặp chập k của tập {1, 2, ..., n}

ARRANGE.INP	ARRANGE.OUT
3 2	1 2 1 3 2 1 2 3 3 1 3 2

Lê Minh Hoàng

P_1_03_3.PAS * Thuật toán quay lui liệt kê các chỉnh hợp không lặp chập k

Chuyên Φ 16 φ
đề

```
program Arrangement; const
InputFile = 'ARRANGES.INP'; OutputFile = 'ARRANGES.OUT'; max = 20; var
x: array[1..max] of Integer; c: array[1..max] of Boolean; n, k: Integer; f: Text;
procedure PrintResult; {Thủ tục in cấu hình tìm được} var
i: Integer; begin
for i := 1 to k do Write(f, x[i],' '); WriteLn(f); end;
procedure var
Try(i: Integer); {Thử các cách chọn x_i}
j: Integer; begin
for j := 1 to n do
if c[j] then {Chỉ xét những giá trị j còn tự do}
begin
x[i] := j; if i = k then PrintResult {Nếu đã chọn được đến xk thì chỉ việc in kết quả} else
begin
```

c[j] := False; {Đánh dấu: j đã bị chọn} **Try(i + 1);** {Thủ tục này chỉ xét những giá trị còn tự do gán cho x_{i+1}, tức là sẽ không chọn phải j}
end;
c[j] := True; {Bỏ đánh dấu: j lại là tự do, bởi sắp tới sẽ thử một cách chọn khác của x_i}
end; end;
begin
Assign(f, InputFile); Reset(f); ReadLn(f, n, k); Assign(f, OutputFile); Rewrite(f); FillChar(c, SizeOf(c), True); {Tất cả các số đều chưa bị chọn} **Try(1); Close(f);**
{Thử các cách chọn giá trị của x_1}
end.

Nhận xét: khi k = n thì đây là chương trình liệt kê hoán vị

3.4. BÀI TOÁN PHÂN TÍCH SỐ

3.4.1. Bài toán

Cho một số nguyên dương n ≤ 30, hãy tìm tất cả các cách phân tích số n thành tổng của các số nguyên dương, các cách phân tích là hoán vị của nhau chỉ tính là 1 cách.

3.4.2. Cách làm:

Ta sẽ lưu nghiệm trong mảng x, ngoài ra có một mảng t. Mảng t xây dựng như sau: t_i sẽ là tổng các phần tử trong mảng x từ x_1 đến x_i: $t_i := x_1 + x_2 + ... + x_i$.

Khi liệt kê các dãy x có tổng các phần tử đúng bằng n, để tránh sự trùng lặp ta đưa thêm ràng buộc $x_{i-1} \le x_i$. Vì số phần tử thực sự của mảng x là không cố định nên thủ tục PrintResult dùng để in ra 1 cách phân tích phải có thêm tham số cho biết sẽ in ra bao nhiêu phần tử.

Thủ tục đệ quy Try(i) sẽ thử các giá trị có thể nhận của x_i $(x_i \ge x_{i-1})$ Khi nào thì in kết quả và khi nào thì gọi đệ quy tìm tiếp ?

Lưu ý rằng t_{i-1} là tổng của tất cả các phần tử từ x_1 đến x_{i-1} do đó Khi $t_i = n$ tức là $(x_i = n - t_{i-1})$ thì in kết quả Khi tìm tiếp, x_{i+1} sẽ phải lớn hơn hoặc bằng x_i. Mặt khác t_{i+1} là tổng của các số từ x_1 tới x_{i+1} không được vượt quá n. Vậy ta có $t_{i+1} \le n \Leftrightarrow t_{i-1} + x_i + x_{i+1} \le n \Leftrightarrow x_i + x_{i+1} \le n - t_{i-1}$ tức là $x_i \le (n - t_{i-1})/2$. Ví dụ đơn giản khi n = 10 thì chọn $x_1 = 6, 7, 8, 9$ là việc làm vô nghĩa vì như vậy cũng không ra nghiệm mà cũng không chọn tiếp x_2 được nữa.

Một cách dễ hiểu ta gọi đệ quy tìm tiếp khi giá trị x_i được chọn còn cho phép chọn thêm một phần tử khác lớn hơn hoặc bằng nó mà không làm tổng vượt quá n. Còn ta in kết quả chỉ khi

x_i mang giá trị đúng bằng số thiếu hụt của tổng i-1 phần tử đầu so với n. Vậy thủ tục Try(i) thử các giá trị cho

x_i có thể mô tả như sau: (để tổng quát cho i = 1, ta đặt $x_0 = 1$

và $t_0 = 0$).

Xét các giá trị của x_i từ x_{i-1} đến (n - t_{i-1}) div 2, cập nhật $t_i := t_{i-1} + x_i$ và gọi đệ quy tìm tiếp. Cuối cùng xét

giá trị $x_i = n - t_{i-1}$ và in kết quả từ x_1 đến x_i.

Input: file văn bản ANALYSE.INP chứa số nguyên dương n ≤ 30

Output: file văn bản ANALYSE.OUT ghi các cách phân tích số n.

ANALYSE.INP	ANALYSE.OUT
6	6 = 1+1+1+1+1+1 6 = 1+1+1+1+2 6 = 1+1+1+3 6 = 1+1+2+2 6 = 1+1+4 6 = 1+2+3 6 = 1+5 6 = 2+2+2 6 = 2+4 6 = 3+3 6 = 6

Lê Minh Hoàng

P_1_03_4.PAS * Thuật toán quay lui liệt kê các cách phân tích số program

```
Analyses; const
InputFile = 'ANALYSE.INP'; OutputFile =
'ANALYSE.OUT'; max = 30; var
  n: Integer; x: array[0..max] of Integer; t:
  array[0..max] of Integer; f: Text;
```

Chuyên Φ 18 φ

đề

```
procedure Init; {Khởi tạo} begin
Assign(f, InputFile); Reset(f); ReadLn(f, n); Close(f); x[0] := 1; t[0] := 0; end;
procedure PrintResult(k: Integer); var
i: Integer; begin
Write(f, n, ' = '); for i := 1 to k - 1 do Write(f, x[i], '+'); WriteLn(f, x[k]); end;
procedure Try(i: Integer); var
j: Integer; begin
for j := x[i - 1] to (n - T[i - 1]) div 2 do begin
```

{Trường hợp còn chọn tiếp x_{i+1}}

```
x[i] := j; t[i] := t[i - 1] + j; Try(i + 1); end; x[i] := PrintResult(i);
```

n - T[i - 1]; {Nếu x_i là phần tử cuối thì nó bắt buộc phải là ... và in kết quả}
end;
begin
Init; Assign(f, OutputFile); Rewrite(f); Try(1); Close(f); end.

Bây giờ ta xét tiếp một ví dụ kinh điển của thuật toán quay lui:

3.5. BÀI TOÁN XẾP HẬU

3.5.1. Bài toán

Xét bàn cờ tổng quát kích thước nxn. Một quân hậu trên bàn cờ có thể ăn được các quân khác nằm tại các ô cùng hàng, cùng cột hoặc cùng đường chéo. Hãy tìm các xếp n quân hậu trên bàn cờ sao cho không quân nào ăn quân nào.

Ví dụ một cách xếp với n = 8:

Đại học Sư phạm Hà Nội, 1999-2002
Bài toán liệt kê
Lê Minh Hoàng

Hình 2: Xếp 8 quân hậu trên bàn cờ 8x8

3.5.2. Phân tích

Rõ ràng n quân hậu sẽ được đặt mỗi con một hàng vì hậu ăn được ngang, ta gọi quân hậu sẽ đặt ở hàng 1 là quân hậu 1, quân hậu ở hàng 2 là quân hậu 2... quân hậu ở hàng n là quân hậu n. Vậy một nghiệm của bài toán sẽ được biết khi ta tìm ra được **vị trí cột của những quân hậu**.

Nếu ta định hướng Đông (Phải), Tây (Trái), Nam (Dưới), Bắc (Trên) thì ta nhận thấy rằng:

• Một đường chéo theo hướng Đông Bắc - Tây Nam (ĐB-TN) bất kỳ sẽ đi qua một số ô, các ô đó có tính chất: Hàng + Cột = C (Const). Với mỗi đường chéo ĐB-TN ta có 1 hằng số C và với một hằng số C: $2 \le C \le 2n$ xác định duy nhất 1 đường chéo ĐB-TN vì vậy ta có thể đánh chỉ số cho các đường chéo ĐB- TN từ 2 đến 2n

• Một đường chéo theo hướng Đông Nam - Tây Bắc (ĐN-TB) bất kỳ sẽ đi qua một số ô, các ô đó có tính chất: Hàng - Cột = C (Const). Với mỗi đường chéo ĐN-TB ta có 1 hằng số C và với một hằng số C: $1 - n \le C \le n - 1$ xác định duy nhất 1 đường chéo ĐN-TB vì vậy ta có thể đánh chỉ số cho các đường chéo ĐN- TB từ 1 - n đến n - 1.

W W E E

Hình 3: Đường chéo ĐB-TN mang chỉ số 10 và đường chéo ĐN-TB mang chỉ số 0

Cài đặt:

NN S S
1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8

1 2345678
Φ 19 φ
Chuyên Φ 20 φ
đề

Ta có 3 mảng logic để đánh dấu:

• Mảng a[1..n]. a_i = TRUE nếu như cột i còn tự do, a_i = FALSE nếu như cột i đã bị một quân hậu khống chế

• Mảng b[2..2n]. b_i = TRUE nếu như đường chéo ĐB-TN thứ i còn tự do, b_i = FALSE nếu như đường chéo đó đã bị một quân hậu khống chế.

• Mảng c[1 - n..n - 1]. c_i = TRUE nếu như đường chéo ĐN-TB thứ i còn tự do, c_i = FALSE nếu như đường chéo đó đã bị một quân hậu khống chế.

Ban đầu cả 3 mảng đánh dấu đều mang giá trị TRUE. (Các cột và đường chéo đều tự do)

Thuật toán quay lui:

- Xét tất cả các cột, thử đặt quân hậu 1 vào một cột, với mỗi cách đặt như vậy, xét tất cả các cách đặt quân hậu 2 không bị quân hậu 1 ăn, lại thử 1 cách đặt và xét tiếp các cách đặt quân hậu 3...Mỗi cách đặt được đến quân hậu n cho ta 1 nghiệm
- Khi chọn vị trí cột j cho quân hậu thứ i, thì ta phải chọn ô(i, j) không bị các quân hậu đặt trước đó ăn, tức là phải chọn cột j còn tự do, đường chéo ĐB-TN (i+j) còn tự do, đường chéo ĐN-TB(i-j) còn tự do. Điều này có thể kiểm tra ($a_j = b_{i+j} = c_{i-j} = TRUE$)
- Khi thử đặt được quân hậu thứ i vào cột j, nếu đó là quân hậu cuối cùng (i = n) thì ta có một nghiệm. Nếu không:
  - **Trước khi gọi** đệ quy tìm cách đặt quân hậu thứ i + 1, ta đánh dấu cột và 2 đường chéo bị quân hậu vừa đặt khống chế ($a_j = b_{i+j} = c_{i-j} := FALSE$) để các lần gọi đệ quy tiếp sau chọn cách đặt các quân hậu kế tiếp sẽ không chọn vào những ô nằm trên cột j và những đường chéo này nữa.
  - **Sau khi gọi** đệ quy tìm cách đặt quân hậu thứ i + 1, có nghĩa là sắp tới ta lại thử một cách đặt khác cho quân hậu thứ i, ta bỏ đánh dấu cột và 2 đường chéo bị quân hậu vừa thử đặt khống chế ($a_j = b_{i+j} = c_{i-j} := TRUE$) tức là cột và 2 đường chéo đó lại thành tự do, bởi khi đã đặt quân hậu i sang vị trí khác rồi thì cột và 2 đường chéo đó hoàn toàn có thể gán cho một quân hậu khác

Hãy xem lại trong các chương trình liệt kê chỉnh hợp không lặp và hoán vị về kỹ thuật đánh dấu. Ở đây chỉ khác với liệt kê hoán vị là: liệt kê hoán vị chỉ cần một mảng đánh dấu xem giá trị có tự do không, còn bài toán xếp hậu thì cần phải đánh dấu cả 3 thành phần: Cột, đường chéo ĐB-TN, đường chéo ĐN- TB. Trường hợp đơn giản hơn: Yêu cầu liệt kê các cách đặt n quân xe lên bàn cờ nxn sao cho không quân nào ăn quân nào chính là bài toán liệt kê hoán vị

- **Input:** file văn bản QUEENS.INP chứa số nguyên dương n ≤ 12
- **Output:** file văn bản QUEENS.OUT, mỗi dòng ghi một cách đặt n quân hậu

QUEENS.IN
P 5

OUT (1, 1); (2, 3); (3, 5); (4,
1, 1); (2, 4); (3, 2); (4, 5); (5,
2, 4); (3, 1); (4, 3); (5, 5); (1,
3, 3); (4, 1); (5, 4); (1, 3); (2,
4, 2); (5, 5); (1, 3); (2, 5); (3,
5, 1); (1, 4); (2, 1); (3, 3); (4,
1, 4); (2, 2); (3, 5); (4, 3); (5,
2, 2); (3, 4); (4, 1); (5, 3); (1,
3, 1); (4, 4); (5, 2);

```
P_1_03_5.PAS * Thuật toán quay lui giải bài toán xếp hậu program
n_Queens; const
InputFile = 'QUEENS.INP'; OutputFile =
'QUEENS.OUT'; max = 12; var
  n: Integer; x: array[1..max] of Integer; a:
  array[1..max] of Boolean; b: array[2..2 * max] of
  Boolean; c: array[1 - max..max - 1] of Boolean; f:
  Text;

procedure Init; begin
Assign(f, InputFile); Reset(f); ReadLn(f, n); Close(f); FillChar(a, SizeOf(a), True); {Mọi
cột đều tự do} FillChar(b, SizeOf(b), True); {Mọi đường chéo Đông Bắc - Tây Nam đều tự
do} FillChar(c, SizeOf(c), True); {Mọi đường chéo Đông Nam - Tây Bắc đều tự do} end;

procedure PrintResult; var
  i: Integer; begin
for i := 1 to n do Write(f, '(', i, ', ', x[i], '); '); WriteLn(f); end;

procedure Try(i: Integer); {Thử các cách đặt quân hậu thứ i vào hàng i} var
  j: Integer; begin
  for j := 1 to n do
    if a[j] and b[i + j] and c[i - j] then {Chỉ xét những cột j mà ô (i, j) chưa bị khống chế}
      begin
        x[i] := j; {Thử đặt quân hậu i vào cột j} if i = n then
        PrintResult else
          begin
a[j] := False; b[i + j] := False; c[i - j] := False; {Đánh dấu} Try(i + 1); {Tìm các cách đặt
quân hậu thứ i + 1} a[j] := True; b[i + j] := True; c[i - j] := True; {Bỏ đánh dấu} end;
end; end;

begin
Chuyên Φ 22 φ                                    ewrite(f); Try(1); Close(f); end.
                                                 đề

Init; Assign(f, OutputFile);
```

Tên gọi thuật toán quay lui, đứng trên phương diện cài đặt có thể nên gọi là kỹ thuật vét cạn bằng quay lui thì chính xác hơn, tuy nhiên đứng trên phương diện bài toán, nếu như ta coi công việc giải bài toán bằng cách xét tất cả các khả năng cũng là 1 cách giải thì tên gọi Thuật toán quay lui cũng không có gì trái logic. Xét hoạt động của chương trình trên cây tìm kiếm quay lui ta thấy tại bước thử chọn x_i nó sẽ gọi đệ quy để tìm tiếp x_{i+1} có nghĩa là quá trình sẽ duyệt tiến sâu xuống phía dưới đến tận nút lá, sau khi đã duyệt hết các nhánh, tiến trình lùi lại thử áp đặt một giá trị khác cho x_i, đó chính là nguồn gốc của tên gọi "thuật toán quay lui"

Bài tập:

Bài 1

Một số chương trình trên xử lý không tốt trong trường hợp tầm thường (n = 0 hoặc k = 0), hãy khắc phục các lỗi đó

Bài 2

Viết chương trình liệt kê các chỉnh hợp lặp chập k của n phần tử

Bài 3

Cho hai số nguyên dương l, n. Hãy liệt kê các xâu nhị phân độ dài n có tính chất, bất kỳ hai xâu con nào độ dài l liền nhau đều khác nhau.

Bài 4

Với n = 5, k = 3, vẽ cây tìm kiếm quay lui của chương trình liệt kê tổ hợp chập k của tập {1, 2, ..., n}

Bài 5

Liệt kê tất cả các tập con của tập S gồm n số nguyên $\{S_1, S^2, ..., S^n\}$ nhập vào từ bàn phím Bài 6

Tương tự như bài 5 nhưng chỉ liệt kê các tập con có max - min ≤ T (T cho trước).

Bài 7

Một dãy $(x_1, x^2, ..., x^n)$ gọi là một hoán vị hoàn toàn của tập {1, 2,..., n} nếu nó là một hoán vị và thoả mãn $x_i \neq i$ với $\forall i$: $1 \leq i \leq n$. Hãy viết chương trình liệt kê tất cả các hoán vị hoàn toàn của tập trên (n vào từ bàn phím).

Bài 8

Sửa lại thủ tục in kết quả (PrintResult) trong bài xếp hậu để có thể vẽ hình bàn cờ và các cách đặt hậu ra màn hình.

Bài 9

Mã đi tuần: Cho bàn cờ tổng quát kích thước nxn và một quân Mã, hãy chỉ ra một hành trình của quân Mã xuất phát từ ô đang đứng đi qua tất cả các ô còn lại của bàn cờ, mỗi ô đúng 1 lần.

Bài 10

Chuyển tất cả các bài tập trong bài trước đang viết bằng sinh tuần tự sang quay lui.

Bài 11

Xét sơ đồ giao thông gồm n nút giao thông đánh số từ 1 tới n và m đoạn đường nối chúng, mỗi đoạn

đường nối 2 nút giao thông. Hãy nhập dữ liệu về mạng lưới giao thông đó, nhập số hiệu hai nút giao thông s và d. Hãy in ra tất cả các cách đi từ s tới d mà mỗi cách đi không được qua nút giao thông nào quá một lần.

Lê Minh Hoàng
Chuyên Φ 24 φ
đề

§4. KỸ THUẬT NHÁNH CẬN

4.1. BÀI TOÁN TỐI ƯU

Một trong những bài toán đặt ra trong thực tế là việc tìm ra **một** nghiệm thoả mãn một số điều kiện nào đó, và nghiệm đó là **tốt nhất** theo một chỉ tiêu cụ thể, nghiên cứu lời giải các lớp bài toán tối ưu

thuộc về lĩnh vực quy hoạch toán học. Tuy nhiên cũng cần phải nói rằng trong nhiều trường hợp chúng ta chưa thể xây dựng một thuật toán nào thực sự hữu hiệu để giải bài toán, mà cho tới nay việc tìm nghiệm của chúng vẫn phải dựa trên mô hình **liệt kê** toàn bộ các cấu hình có thể và đánh giá, tìm ra cấu hình tốt nhất. Việc liệt kê cấu hình có thể cài đặt bằng các phương pháp liệt kê: Sinh tuần tự và tìm kiếm quay lui. Dưới đây ta sẽ tìm hiểu phương pháp liệt kê bằng thuật toán quay lui để tìm nghiệm của bài toán tối ưu.

4.2. SỰ BÙNG NỔ TỔ HỢP

Mô hình thuật toán quay lui là tìm kiếm trên 1 cây phân cấp. Nếu giả thiết rằng ứng với mỗi nút tương ứng với một giá trị được chọn cho x_i sẽ ứng với chỉ 2 nút tương ứng với 2 giá trị mà x_{i+1} có thể nhận thì cây n cấp sẽ có tới 2^n nút lá, con số này lớn hơn rất nhiều lần so với dữ liệu đầu vào n.

Chính vì vậy mà nếu như ta có thao tác thừa trong việc chọn x_i thì sẽ phải trả giá rất lớn về chi phí thực thi thuật toán bởi quá trình tìm kiếm lòng vòng vô nghĩa trong các bước chọn kế tiếp x_{i+1}, x_{i+2}, ... Khi đó, một vấn đề đặt ra là trong quá trình liệt kê lời giải ta cần tận dụng những thông tin đã tìm được để loại bỏ sớm những phương án chắc chắn không phải tối ưu. Kỹ thuật đó gọi là kỹ thuật đánh giá nhánh cận trong tiến trình quay lui.

4.3. MÔ HÌNH KỸ THUẬT NHÁNH CẬN

Dựa trên mô hình thuật toán quay lui, ta xây dựng mô hình sau:

procedure Init; begin
<Khởi tạo một cấu hình bất kỳ BESTCONFIG>; end;
{**Thủ tục procedure này thử Try(i: chọn cho Integer);**

x_i tất cả các giá trị nó có thể nhận}
begin
for begin

(Mọi giá trị V có thể gán cho x_i) do

<Thử if cho (Việc xthử $_i$:= V>;

trên vẫn còn hi vọng tìm ra cấu hình tốt hơn BESTCONFIG) then if <Cập (x_i là phần tử cuối cùng trong cấu hình) then
nhật BESTCONFIG> else
begin <Ghi Try(i <Bỏ end;

nhận + ghi 1); việc {Gọi **thử** đệ **x_i** quy, **= V nếu** chọn **nhận việc thử cho x_i cần>;**

= tiếp **V (nếu x_{i+1} cần)>;**

}

Đại học Sư phạm Hà Nội, 1999-2002

end;

Init; Try(1); <Thông báo cấu hình tối ưu BESTCONFIG> end. Kỹ thuật nhánh cận thêm vào cho thuật toán quay lui khả năng đánh giá theo từng bước, nếu tại bước thứ i, giá trị thử gán cho x_i không có hi vọng tìm thấy cấu hình tốt hơn cấu hình BESTCONFIG thì thử giá trị khác ngay mà không cần phải gọi đệ quy tìm tiếp hay ghi nhận kết

quả làm gì. Nghiệm của bài toán sẽ được làm tốt dần, bởi khi tìm ra một cấu hình mới (tốt hơn BESTCONFIG - tất nhiên), ta không in kết quả ngay mà sẽ cập nhật BESTCONFIG bằng cấu hình mới vừa tìm được

4.4. BÀI TOÁN NGƯỜI DU LỊCH

4.4.1. Bài toán

Cho n thành phố đánh số từ 1 đến n và m tuyến đường giao thông hai chiều giữa chúng, mạng lưới giao thông này được cho bởi bảng C cấp nxn, ở đây $C_{ij} = C_{ji}$ = Chi phí đi đoạn đường trực tiếp từ thành phố i đến thành phố j. Giả thiết rằng $C_{ii} = 0$ với $\forall$ i, $C_{ij} = +\infty$ nếu không có đường trực tiếp từ thành phố i đến thành phố j.

Một người du lịch xuất phát từ thành phố 1, muốn đi thăm tất cả các thành phố còn lại mỗi thành phố đúng 1 lần và cuối cùng quay lại thành phố 1. Hãy chỉ ra cho người đó hành trình với chi phí ít nhất. Bài toán đó gọi là bài toán người du lịch hay bài toán hành trình của một thương gia (Traveling Salesman)

4.4.2. Cách giải

Hành trình cần tìm có dạng ($x_1 = 1, x_2, ..., x_n, x_{n+1} = 1$) ở đây giữa x_i và x_{i+1}: hai thành phố liên tiếp trong hành trình phải có đường đi trực tiếp ($C_{ij} \neq +\infty$) và ngoại trừ thành phố 1, không thành phố nào được lặp lại hai lần. Có nghĩa là dãy ($x_1, x_2, ..., x_n$) lập thành 1 hoán vị của (1, 2, ..., n). Duyệt quay lui: x_2 có thể chọn một trong các thành phố mà x_1 có đường đi tới (trực tiếp), với mỗi cách thử chọn x_2 như vậy thì x_3 có thể chọn một trong các thành phố mà x_2 có đường đi tới (ngoài x_1). Tổng quát: x_i có thể chọn 1 trong các thành phố **chưa đi qua** mà **từ x_{i-1} có đường đi trực tiếp tới ($1 \leq i \leq n$).**

Nhánh cận: Khởi tạo cấu hình BestConfig có chi phí = $+\infty$. Với mỗi bước thử chọn x_i xem chi phí đường đi cho tới lúc đó có < Chi phí của cấu hình BestConfig?, nếu không nhỏ hơn thì thử giá trị khác ngay bởi có đi tiếp cũng chỉ tốn thêm. Khi thử được một giá trị x_n ta kiểm tra xem x_n có đường đi trực tiếp về 1 không ? Nếu có đánh giá chi phí đi từ thành phố 1 đến thành phố x_n cộng với chi

phí từ x_n đi trực tiếp về 1, nếu nhỏ hơn chi phí của đường đi BestConfig thì cập nhật lại BestConfig bằng cách đi mới.

Sau thủ tục tìm kiếm quay lui mà chi phí của BestConfig vẫn bằng +∞ thì có nghĩa là nó không tìm thấy một hành trình nào thoả mãn điều kiện đề bài để cập nhật BestConfig, bài toán không có lời giải, còn nếu chi phí của BestConfig < +∞ thì in ra cấu hình BestConfig - đó là hành trình ít tốn kém nhất tìm được

Input: file văn bản TOURISM.INP

• Dòng 1: Chứa số thành phố n (1 ≤ n ≤ 20) và số tuyến đường m trong mạng lưới giao thông

• m dòng tiếp theo, mỗi dòng ghi số hiệu hai thành phố có đường đi trực tiếp và chi phí đi trên quãng đường đó (chi phí này là số nguyên dương ≤ 100)

Output: file văn bản TOURISM.OUT, ghi hành trình tìm được.

TOURISM.OUT 1->3->2->4->1 Cost: 6

P_1_04_1.PAS * Kỹ thuật nhánh cận dùng cho bài toán người du lịch program TravellingSalesman; const InputFile = 'TOURISM.INP'; OutputFile = 'TOURISM.OUT'; max = 20; maxC = 20 * 100 + 1;{+∞} **var C: array[1..max, 1..max] of Integer;** {Ma trận chi phí} **X, BestWay: array[1..max + 1] of Integer;** {X để thử các khả năng, BestWay để ghi nhận nghiệm} **T: array[1..max Free: array[1..max] m, n: Integer;**

+ 1] of of Boolean; Integer; {Free {T_i để lưu chi phí đi từ để đánh dấu, $Free_i$= XTrue 1 đến nếu X_i}

chưa đi qua tp i}
MinSpending: Integer; {Chi phí hành trình tối ưu}
procedure Enter; var
i, j, k: Integer; f: Text; begin
Assign(f, InputFile); Reset(f); ReadLn(f, n, m); for i := 1 to n do {Khởi tạo bảng chi phí ban đầu}
for j := 1 to n do
if i = j then C[i, j] := 0 else C[i, j] := maxC; for k := 1 to m do
begin
ReadLn(f, i, j, C[i, j]); C[j, i] := C[i, j]; {Chi phí như nhau trên 2 chiều} **end; Close(f); end;**
procedure Init; {Khởi tạo}
2

3_4
1
$4^{1\ 2}$ 1
2
3

TOURISM.INP 4 6 1 2 3 1 3 2 1 4 1 2 3 1 2 4 2 3 4 4

begin
FillChar(Free, n, True); Free[1] := False; {Các thành phố là chưa đi qua ngoại trừ thành phố 1} **X[1] := 1;** {Xuất phát từ thành phố 1} **T[1] := 0;** {Chi phí tại thành phố xuất phát là 0} **MinSpending := maxC; end;**
procedure Try(i: Integer); {Thử các cách chọn xi} **var**
j: Integer; begin
for j := 2 to n do {Thử các thành phố từ 2 đến n} **if Free[j] then** {Nếu gặp thành phố chưa đi qua}
begin
X[i] := j; {Thử đi} **T[i] := T[i - 1] + C[x[i - 1], j];** {Chi phí := Chi phí bước trước + chi phí đường đi trực tiếp} **if T[i] < MinSpending then** {Hiển nhiên nếu có điều này thì C[x[i - 1], j] < +∞ rồi}
if begin

i < n then {Nếu chưa đến được x_n}

Free[j] := False; {Đánh dấu thành phố vừa thử} **Try(i Free[j] + 1); := True;** {Tìm {Bỏ các đánh khả năng dấu}

chọn x_{i+1}}

end else

if T[n] + C[x[n], 1] < MinSpending then {Từ x_n quay lại 1 vẫn tốn chi phí ít hơn trước}

begin {Cập nhật BestConfig}
BestWay := X; MinSpending := T[n] + C[x[n], 1]; end; end; end;
procedure PrintResult; var
i: Integer; f: Text; begin
Assign(f, OutputFile); Rewrite(f); if MinSpending = maxC then WriteLn(f, 'NO SOLUTION') else
for i := 1 to n do Write(f, BestWay[i], '->'); WriteLn(f, 1); WriteLn(f, 'Cost: ', MinSpending); Close(f); end;
begin
Enter; Init; Try(2); PrintResult; end. Trên đây là một giải pháp nhánh cận còn rất thô sơ giải bài toán người du lịch, trên thực tế người ta
còn có nhiều cách đánh giá nhánh cận chặt hơn nữa. Hãy tham khảo các tài liệu khác để tìm hiểu về những phương pháp đó.

Lê Minh Hoàng
Chuyên Φ 28 φ
đề

4.5. DÃY ABC

Cho trước một số nguyên dương N (N ≤ 100), hãy tìm một xâu chỉ gồm các ký tự A, B, C thoả mãn 3 điều kiện:
Có độ dài N
Hai đoạn con bất kỳ liền nhau đều khác nhau (đoạn con là một dãy ký tự liên tiếp của xâu)
Có ít ký tự C nhất.
Cách giải:
Không trình bày, đề nghị tự xem chương trình để hiểu, chỉ chú thích kỹ thuật nhánh cận như sau:

Nếu dãy $X_1X_2...X_n$ thoả mãn 2 đoạn con bất kỳ liền nhau đều khác nhau, thì trong 4 ký tự liên tiếp bất kỳ bao giờ cũng phải có 1 ký tự "C". Như vậy với một dãy con gồm k ký tự liên tiếp của dãy X thì số ký tự C trong dãy con đó bắt buộc phải ≥ k div 4.

Tại bước thử chọn X_i, nếu ta đã có T_i ký tự "C" trong đoạn đã chọn từ X_1 đến X_i, thì cho dù các bước đệ quy tiếp sau làm tốt như thế nào chăng nữa, số ký tự "C" sẽ phải chọn thêm bao giờ cũng ≥ (n - i) div 4. Tức là nếu theo phương án chọn X_i như thế này thì số ký tự "C" trong dãy kết quả (khi chọn đến X_n) cho dù có làm tốt đến đâu cũng ≥ **T_i + (n - i) div 4**. Ta dùng con số này để đánh giá nhánh cận, nếu nó nhiều hơn số ký tự "C" trong BestConfig thì chắc chắn có làm tiếp cũng chỉ được một cấu hình tồi tệ hơn, ta bỏ qua ngay cách chọn này và thử phương án khác.

Input: file văn bản ABC.INP chứa số nguyên dương n ≤ 100

Output: file văn bản ABC.OUT ghi xâu tìm được

ABC.INP 10
Đại học Sư phạm Hà Nội, 1999-2002 **ABC.OUT ABACABCBAB "C" Letter Count : 2**
P_1_04_2.PAS * Dãy ABC program ABC_STRING; const
InputFile = 'ABC.INP'; OutputFile = 'ABC.OUT'; max = 100; var
N, MinC: Integer; X, Best: array[1..max] of 'A'..'C'; T: array[0..max] f: Text;

of Integer; {T_i cho biết số ký tự "C" trong đoạn từ X_1 đến X_i}
{Hàm Same(i, l) cho **function Same(i, l:** biết xâu **Integer):** gồm l **Boolean;**

ký tự kết thúc tại X_i có trùng với xâu l ký tự liền trước nó không ?}

```
var
j, k: Integer; begin
j := i - l; {j là vị trí cuối đoạn liền trước đoạn đó} for k := 0 to l - 1 do
if X[i - k] <> X[j - k] then
begin
Same := False; Exit; end; Same := True;
```

Φ Bài toán liệt kê

29 φ

```
end;
```

{Hàm Check(i) cho **function Check(i:** biết **Integer):** X_i có làm **Boolean;**

hỏng tính không lặp của dãy $X^1X^2 ... X^i$ hay không}

```
var
l: Integer; begin
for l := 1 to i div 2 do {Thử các độ dài l}
if Same(i, begin
```

l) then {Nếu có xâu độ dài l kết thúc bởi X_i bị trùng với xâu liền trước}

```
Check := False; Exit; end; Check := True; end;
```

{Giữ lại kết quả vừa tìm được vào BestConfig (MinC và mảng Best)} **procedure KeepResult; begin**

```
MinC := T[N]; Best := X; end;
```

{Thuật toán quay lui có nhánh cận} **procedure var**

Try(i: Integer); {Thử các giá trị có thể của X_i}

```
j: 'A'..'C'; begin
for j := 'A' to 'C' do {Xét tất cả các giá trị}
begin
X[i] := j; if Check(i) then {Nếu thêm giá trị đó vào không làm hỏng tính không lặp }
begin
if j else = 'C' then T[i] T[i] := T[i - := 1];
```

T[i - 1] + 1 {Tính T_i qua T^{i-1}}

```
if T[i] + (N - i) div 4 < MinC then {Đánh giá nhánh cận}
if i = N then KeepResult else Try(i + 1); end; end; end;
procedure PrintResult; var
i: Integer; begin
for i := 1 to N do Write(f, Best[i]); WriteLn(f); WriteLn(f, '"C" Letter Count : ', MinC); end;
begin
Assign(f, InputFile); Reset(f); ReadLn(f, N); Close(f); Assign(f, OutputFile); Rewrite(f); T[0] := 0; MinC := N; {Khởi tạo
```

cấu hình BestConfig ban đầu rất tồi} **Try(1); PrintResult; Close(f); end.** Nếu ta thay bài toán là tìm xâu ít ký tự 'B' nhất mà vẫn viết chương trình tương tự như trên thì

chương trình sẽ chạy chậm hơn chút ít. Lý do: thủ tục Try ở trên sẽ thử lần lượt các giá trị 'A', 'B',

rồi mới đến 'C'. Có nghĩa ngay trong cách tìm, nó đã tiết kiệm sử dụng ký tự 'C' nhất nên trong phần lớn các bộ dữ liệu nó nhanh chóng tìm ra lời giải hơn so với bài toán tương ứng tìm xâu ít ký tự 'B' nhất. Chính vì vậy mà nếu như đề bài yêu cầu ít ký tự 'B' nhất ta cứ lập chương trình làm yêu cầu ít ký tự 'C' nhất, chỉ có điều khi in kết quả, ta đổi vai trò 'B', 'C' cho nhau. Đây là một ví dụ cho thấy sức mạnh của thuật toán quay lui khi kết hợp với kỹ thuật nhánh cận, nếu viết quay lui thuần tuý

hoặc đánh giá nhánh cận không tốt thì với N = 100, tôi cũng không đủ kiên nhẫn để đợi chương trình cho kết quả (chỉ biết rằng > 3 giờ). Trong khi đó khi N = 100, với chương trình trên chỉ chạy hết hơn 3 giây cho kết quả là xâu 27 ký tự 'C'.

Nói chung, ít khi ta gặp bài toán mà chỉ cần sử dụng một thuật toán, một mô hình kỹ thuật cài đặt là có thể giải được. Thông thường các bài toán thực tế đòi hỏi phải có sự tổng hợp, pha trộn nhiều thuật toán, nhiều kỹ thuật mới có được một lời giải tốt. Không được lạm dụng một kỹ thuật nào và cũng không xem thường một phương pháp nào khi bắt tay vào giải một bài toán tin học. Thuật toán quay lui cũng không phải là ngoại lệ, ta phải biết phối hợp một cách uyển chuyển với các thuật toán khác thì khi đó nó mới thực sự là một công cụ mạnh.

Bài tập:

Bài 1

Một dãy dấu ngoặc hợp lệ là một dãy các ký tự "(" và ")" được định nghĩa như sau:

i. Dãy rỗng là một dãy dấu ngoặc hợp lệ độ sâu 0

ii. Nếu A là dãy dấu ngoặc hợp lệ độ sâu k thì (A) là dãy dấu ngoặc hợp lệ độ sâu k + 1

iii. Nếu A và B là hay dãy dấu ngoặc hợp lệ với độ sâu lần lượt là p và q thì AB là dãy dấu ngoặc hợp lệ độ sâu là max(p, q)

Độ dài của một dãy ngoặc là tổng số ký tự "(" và ")"

Ví dụ: Có 5 dãy dấu ngoặc hợp lệ độ dài 8 và độ sâu 3:

1. ((()())) 2. ((())()) 3. ((()))() 4. (()(())) 5. ()((())) ***Bài toán đặt ra là khi cho biết trước hai số nguyên dương n và k. Hãy liệt kê hết các dãy ngoặc hợp lệ có độ dài là n và độ sâu là k (làm được với n càng lớn càng tốt).***

Bài 2

Cho một bãi mìn kích thước mxn ô vuông, trên một ô có thể có chứa một quả mìn hoặc không, để biểu diễn bản đồ mìn đó, người ta có hai cách:

Cách 1: dùng bản đồ đánh dấu: sử dụng một lưới ô vuông kích thước mxn, trên đó tại ô (i, j) ghi số 1 nếu ô đó có mìn, ghi số 0 nếu ô đó không có mìn

Cách 2: dùng bản đồ mật độ: sử dụng một lưới ô vuông kích thước mxn, trên đó tại ô (i, j) ghi một số trong khoảng từ 0 đến 8 cho biết tổng số mìn trong các ô lân cận với ô (i, j) (ô lân cận với ô (i, j) là ô có chung với ô (i, j) ít nhất 1 đỉnh).

Giả thiết rằng hai bản đồ được ghi chính xác theo tình trạng mìn trên hiện trường.

Về nguyên tắc, lúc cài bãi mìn phải vẽ cả bản đồ đánh dấu và bản đồ mật độ, tuy nhiên sau một thời gian dài, khi người ta muốn gỡ mìn ra khỏi bãi thì vấn đề hết sức khó khăn bởi bản đồ đánh dấu đã bị thất lạc !!.

Công việc của các lập trình viên là: Từ bản đồ mật độ, hãy tái tạo lại bản đồ đánh dấu của bãi mìn. **Dữ liệu: Vào từ file văn bản MINE.INP, các số trên 1 dòng cách nhau ít nhất 1 dấu cách**

• Dòng 1: Ghi 2 số nguyên dương m, n (2 ≤ m, n ≤ 30)

• m dòng tiếp theo, dòng thứ i ghi n số trên hàng i của bản đồ mật độ theo đúng thứ tự từ trái qua phải.

Kết quả: Ghi ra file văn bản MINE.OUT, các số trên 1 dòng ghi cách nhau ít nhất 1 dấu cách

• Dòng 1: Ghi tổng số lượng mìn trong bãi

• m dòng tiếp theo, dòng thứ i ghi n số trên hàng i của bản đồ đánh dấu theo đúng thứ tự từ trái qua phải.

Ví dụ:

MINE.INP MINE.OUT 10 15 0 3 2 3 3 3 5 3 4 4 5 4 4 4 3 1 4 3 5 5 4 5 4 7 7 7 5 6 6 5 1 4 3 5 4 3 5 4 4 4 4 3 4 5 5 1 4 2 4 4 5 4 2 4 4 3 2 3 5 4 1 3 2 5 4 4 2 2 3 2 3 3 2 5 2 2 3 2 3 3 5 3 2 4 4 3 4 2 4 1 2 3 2 4 3 3 2 3 4 6 6 5 3 3 1 2 6 4 5 2 4 1 3 3 5 5 5 6 4 3 4 6 5 7 3 5 3 5 5 6 5 4 4 4 3 2 4 4 4 2 3 1 2 2 2 3 3 3 4 2

Lê Minh Hoàng

80 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 0 1

PHẦN 2. CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Hạt nhân của các chương trình máy tính là sự lưu trữ và xử lý thông tin. Việc tổ chức dữ liệu như thế nào có ảnh hưởng rất lớn đến cách thức xử lý dữ liệu đó cũng như tốc độ thực thi và sự chiếm dụng bộ nhớ của chương trình. Việc đặc tả bằng các cấu trúc tổng quát (generic structures) và các kiểu dữ liệu trừu tượng (abstract data types) còn cho phép người lập trình có thể dễ dàng hình dung ra các công việc cụ thể và giảm bớt công sức trong việc chỉnh sửa, nâng cấp và sử dụng lại các thiết kế đã có.

Mục đích của phần này là cung cấp những hiểu biết nền tảng trong việc thiết kế một chương trình máy tính, để thấy rõ được sự cần thiết của việc phân tích, lựa chọn cấu trúc dữ liệu phù hợp cho từng bài toán cụ thể; đồng thời khảo sát một số cấu trúc dữ liệu và thuật toán kinh điển mà lập trình viên nào cũng cần phải nắm vững.

Chuyên Φ 34 φ đề

§1. CÁC BƯỚC CƠ BẢN KHI TIẾN HÀNH GIẢI CÁC BÀI TOÁN TIN HỌC

1.1. XÁC ĐỊNH BÀI TOÁN Input → Process → Output

(Dữ liệu vào → Xử lý → Kết quả ra)

Việc xác định bài toán tức là phải xác định xem ta phải giải quyết vấn đề gì?, với giả thiết nào đã cho và lời giải cần phải đạt những yêu cầu gì. Khác với bài toán thuần tuý toán học chỉ cần xác định rõ giả thiết và kết luận chứ không cần xác định yêu cầu về lời giải, đôi khi những bài toán tin học ứng dụng trong thực tế chỉ cần tìm lời giải tốt tới mức nào đó, thậm chí là tồi ở mức chấp nhận được. Bởi lời giải tốt nhất đòi hỏi quá nhiều thời gian và chi phí.

Ví dụ:

Khi cài đặt các hàm số phức tạp trên máy tính. Nếu tính bằng cách khai triển chuỗi vô hạn thì độ chính xác cao hơn nhưng thời gian chậm hơn hàng tỉ lần so với phương pháp xấp xỉ. Trên thực tế việc tính toán luôn luôn cho phép chấp nhận một sai số nào đó nên các hàm số trong máy tính đều được tính bằng phương pháp xấp xỉ của giải tích số

Xác định đúng yêu cầu bài toán là rất quan trọng bởi nó ảnh hưởng tới cách thức giải quyết và chất lượng của lời giải. Một bài toán thực tế thường cho bởi những thông tin khá mơ hồ và hình thức, ta phải phát biểu lại một cách chính xác và chặt chẽ để hiểu đúng bài toán.

Ví dụ:

- *Bài toán: Một dự án có n người tham gia thảo luận, họ muốn chia thành các nhóm và mỗi nhóm thảo luận riêng về một phần của dự án. Nhóm có bao nhiêu người thì được trình lên*

bấy nhiêu ý kiến. Nếu lấy ở mỗi nhóm một ý kiến đem ghép lại thì được một bộ ý kiến triển khai dự án. Hãy tìm cách chia để số bộ ý kiến cuối cùng thu được là lớn nhất.

- *Phát biểu lại: Cho một số nguyên dương n, tìm các phân tích n thành tổng các số nguyên dương sao cho tích của các số đó là lớn nhất.*

Trên thực tế, ta nên xét một vài trường hợp cụ thể để thông qua đó hiểu được bài toán rõ hơn và thấy được các thao tác cần phải tiến hành. Đối với những bài toán đơn giản, đôi khi chỉ cần qua ví dụ là ta đã có thể đưa về một bài toán quen thuộc để giải.

1.2. TÌM CẤU TRÚC DỮ LIỆU BIỂU DIỄN BÀI TOÁN

Khi giải một bài toán, ta cần phải định nghĩa tập hợp dữ liệu để biểu diễn tình trạng cụ thể. Việc lựa chọn này tuỳ thuộc vào vấn đề cần giải quyết và những thao tác sẽ tiến hành trên dữ liệu vào. Có những thuật toán chỉ thích ứng với một cách tổ chức dữ liệu nhất định, đối với

những cách tổ chức dữ liệu khác thì sẽ kém hiệu quả hoặc không thể thực hiện được. Chính vì vậy nên bước xây dựng cấu trúc dữ liệu không thể tách rời bước tìm kiếm thuật toán giải quyết vấn đề.

Các tiêu chuẩn khi lựa chọn cấu trúc dữ liệu

Cấu trúc dữ liệu trước hết phải biểu diễn được đầy đủ các thông tin nhập và xuất của bài toán

Cấu trúc dữ liệu phải phù hợp với các thao tác của thuật toán mà ta lựa chọn để giải quyết bài toán.

Cấu trúc dữ liệu phải cài đặt được trên máy tính với ngôn ngữ lập trình đang sử dụng

Đối với một số bài toán, trước khi tổ chức dữ liệu ta phải viết một đoạn chương trình nhỏ để **khảo sát** xem dữ liệu cần lưu trữ lớn tới mức độ nào.

1.3. TÌM THUẬT TOÁN

Thuật toán là một hệ thống chặt chẽ và rõ ràng các quy tắc nhằm xác định một dãy thao tác trên cấu trúc dữ liệu sao cho: Với một bộ dữ liệu vào, sau một số hữu hạn bước thực hiện các thao tác đã chỉ ra, ta đạt được mục tiêu đã định.

Các đặc trưng của thuật toán

1.3.1. Tính đơn định

Ở mỗi bước của thuật toán, các thao tác phải hết sức rõ ràng, không gây nên sự nhập nhằng, lộn xộn, tuỳ tiện, đa nghĩa. Thực hiện đúng các bước của thuật toán thì với một dữ liệu vào, chỉ cho duy nhất một kết quả ra.

1.3.2. Tính dừng

Thuật toán không được rơi vào quá trình vô hạn, phải dừng lại và cho kết quả sau một số hữu hạn bước.

1.3.3. Tính đúng

Sau khi thực hiện tất cả các bước của thuật toán theo đúng quá trình đã định, ta phải được kết quả mong muốn với mọi bộ dữ liệu đầu vào. Kết quả đó được kiểm chứng bằng yêu cầu bài toán.

1.3.4. Tính phổ dụng

Thuật toán phải dễ sửa đổi để thích ứng được với bất kỳ bài toán nào trong một lớp các bài toán và có thể làm việc trên các dữ liệu khác nhau.

1.3.5. Tính khả thi

a) Kích thước phải đủ nhỏ: Ví dụ: Một thuật toán sẽ có tính hiệu quả bằng 0 nếu lượng bộ nhớ mà nó yêu cầu vượt quá khả năng lưu trữ của hệ thống máy tính.
b) Thuật toán phải được máy tính thực hiện trong thời gian cho phép, điều này khác với lời giải toán (Chỉ cần chứng minh là kết thúc sau hữu hạn bước). Ví dụ như xếp thời khoá biểu cho một học kỳ thì không thể cho máy tính chạy tới học kỳ sau mới ra được.
c) Phải dễ hiểu và dễ cài đặt.

Ví dụ:

Input: 2 số nguyên tự nhiên a và b không đồng thời bằng 0

Output: Ước số chung lớn nhất của a và b

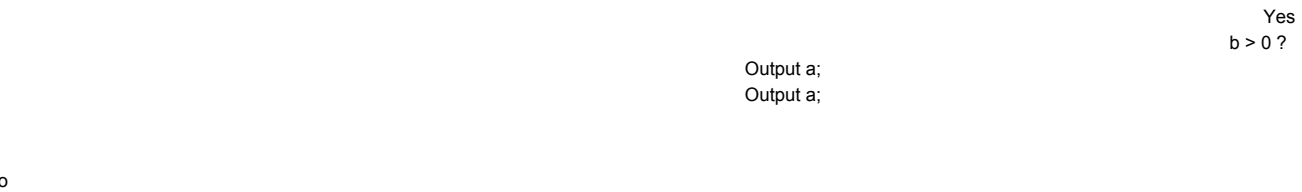
Thuật toán sẽ tiến hành được mô tả như sau: (Thuật toán Euclide)

Bước 1 (Input): Nhập a và b: Số tự nhiên

Bước 2: Nếu b ≠ 0 thì chuyển sang bước 3, nếu không thì bỏ qua bước 3, đi làm bước 4

Bước 3: Đặt r := a mod b; Đặt a := b; Đặt b := r; Quay trở lại bước 2.

Bước 4 (Output): Kết luận ước số chung lớn nhất phải tìm là giá trị của a. Kết thúc thuật toán.

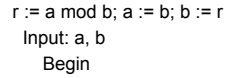


Hình 4: Lưu đồ thuật giải (Flowchart)

Khi mô tả thuật toán bằng ngôn ngữ tự nhiên, ta không cần phải quá chi tiết các bước và tiến trình thực hiện mà chỉ cần mô tả một cách hình thức đủ để chuyển thành ngôn ngữ lập trình. Viết sơ đồ các thuật toán đệ quy là một ví dụ.

Đối với những thuật toán phức tạp và nặng về tính toán, các bước và các công thức nên mô tả một cách tường minh và chú thích rõ ràng để khi lập trình ta có thể nhanh chóng tra cứu.

Đối với những thuật toán kinh điển thì phải thuộc. Khi giải một bài toán lớn trong một thời gian giới hạn, ta chỉ phải thiết kế tổng thể còn những chỗ đã thuộc thì cứ việc lắp ráp vào. Tính đúng đắn của những mô-đun đã thuộc ta không cần phải quan tâm nữa mà tập trung giải quyết các phần khác.



End

1.4. LẬP TRÌNH

Sau khi đã có thuật toán, ta phải tiến hành lập trình thể hiện thuật toán đó. Muốn lập trình đạt hiệu quả cao, cần phải có kỹ thuật lập trình tốt. Kỹ thuật lập trình tốt thể hiện ở kỹ năng viết chương trình, khả năng gỡ rối và thao tác nhanh. Lập trình tốt không phải chỉ cần nắm vững ngôn ngữ lập trình là đủ, phải biết cách viết chương trình uyển chuyển, khôn khéo và phát triển dần dần để chuyển các ý tưởng ra thành chương trình hoàn chỉnh. Kinh nghiệm cho thấy một thuật toán hay nhưng do cài đặt vụng về nên khi chạy lại cho kết quả sai hoặc tốc độ chậm.

Thông thường, ta không nên cụ thể hoá ngay toàn bộ chương trình mà nên tiến hành theo phương pháp tinh chế từng bước (Stepwise refinement):

Ban đầu, chương trình được thể hiện bằng ngôn ngữ tự nhiên, thể hiện thuật toán với các bước

tổng thể, mỗi bước nêu lên một công việc phải thực hiện.

Một công việc đơn giản hoặc là một đoạn chương trình đã được học thuộc thì ta tiến hành viết mã lệnh ngay bằng ngôn ngữ lập trình.

Một công việc phức tạp thì ta lại chia ra thành những công việc nhỏ hơn để lại tiếp tục với những công việc nhỏ hơn đó.

Trong quá trình tinh chế từng bước, ta phải đưa ra những biểu diễn dữ liệu. Như vậy cùng với sự tinh chế các công việc, dữ liệu cũng được tinh chế dần, có cấu trúc hơn, thể hiện rõ hơn mối liên hệ giữa các dữ liệu.

Phương pháp tinh chế từng bước là một thể hiện của tư duy giải quyết vấn đề từ trên xuống, giúp cho người lập trình có được một định hướng thể hiện trong phong cách viết chương trình. Tránh việc mò mẫm, xoá đi viết lại nhiều lần, biến chương trình thành tờ giấy nháp.

1.5. KIỂM THỬ

1.5.1. Chạy thử và tìm lỗi

Chương trình là do con người viết ra, mà đã là con người thì ai cũng có thể nhầm lẫn. Một chương trình viết xong chưa chắc đã chạy được ngay trên máy tính để cho ra kết quả mong muốn. Kỹ năng tìm lỗi, sửa lỗi, điều chỉnh lại chương trình cũng là một kỹ năng quan trọng của người lập trình. Kỹ năng này chỉ có được bằng kinh nghiệm tìm và sửa chữa lỗi của chính mình.

Có ba loại lỗi:

Lỗi cú pháp: Lỗi này hay gặp nhất nhưng lại dễ sửa nhất, chỉ cần nắm vững ngôn ngữ lập trình là đủ. Một người được coi là không biết lập trình nếu không biết sửa lỗi cú pháp.

Lỗi cài đặt: Việc cài đặt thể hiện không đúng thuật toán đã định, đối với lỗi này thì phải xem lại tổng thể chương trình, kết hợp với các chức năng gỡ rối để sửa lại cho đúng.

Lỗi thuật toán: Lỗi này ít gặp nhất nhưng nguy hiểm nhất, nếu nhẹ thì phải điều chỉnh lại thuật toán, nếu nặng thì có khi phải loại bỏ hoàn toàn thuật toán sai và làm lại từ đầu.

1.5.2. Xây dựng các bộ test

Có nhiều chương trình rất khó kiểm tra tính đúng đắn. Nhất là khi ta không biết kết quả đúng là thế nào?. Vì vậy nếu như chương trình vẫn chạy ra kết quả (không biết đúng sai thế nào) thì việc tìm lỗi rất khó khăn. Khi đó ta nên làm các bộ test để thử chương trình của mình.

Các bộ test nên đặt trong các file văn bản, bởi việc tạo một file văn bản rất nhanh và mỗi lần chạy thử chỉ cần thay tên file dữ liệu vào là xong, không cần gõ lại bộ test từ bàn phím. Kinh nghiệm làm các bộ test là:

Bắt đầu với một bộ test nhỏ, đơn giản, làm bằng tay cũng có được đáp số để so sánh với kết quả chương trình chạy ra.

Tiếp theo vẫn là các bộ test nhỏ, nhưng chứa các giá trị đặc biệt hoặc tầm thường. Kinh nghiệm cho thấy đây là những test dễ sai nhất.

Các bộ test phải đa dạng, tránh sự lặp đi lặp lại các bộ test tương tự.

Có một vài test lớn chỉ để kiểm tra tính chịu đựng của chương trình mà thôi. Kết quả có đúng hay không thì trong đa số trường hợp, ta không thể kiểm chứng được với test này.

Lưu ý rằng chương trình chạy qua được hết các test không có nghĩa là chương trình đó đã đúng. Bởi có thể ta chưa xây dựng được bộ test làm cho chương trình chạy sai. Vì vậy nếu có thể, ta nên tìm cách chứng minh tính đúng đắn của thuật toán và chương trình, điều này thường rất khó.

1.6. TỐI ƯU CHƯƠNG TRÌNH

Một chương trình đã chạy đúng không có nghĩa là việc lập trình đã xong, ta phải sửa đổi lại một vài chi tiết để chương trình có thể chạy nhanh hơn, hiệu quả hơn. Thông thường, trước khi kiểm thử thì ta nên đặt mục tiêu viết chương trình sao cho đơn giản, **miễn sao chạy ra kết quả đúng** là được, sau đó khi tối ưu chương trình, ta xem lại những chỗ nào viết chưa tốt thì tối ưu lại mã lệnh để chương trình ngắn hơn, chạy nhanh hơn. Không nên viết tới đâu tối ưu mã đến đó, bởi chương trình có mã lệnh tối ưu thường phức tạp và khó kiểm soát.

Việc tối ưu chương trình nên dựa trên các tiêu chuẩn sau:

1.6.1. Tính tin cậy

Chương trình phải chạy đúng như dự định, mô tả đúng một giải thuật đúng. Thông thường khi viết chương trình, ta luôn có thói quen kiểm tra tính đúng đắn của các bước mỗi khi có thể.

1.6.2. Tính uyển chuyển

Chương trình phải dễ sửa đổi. Bởi ít có chương trình nào viết ra đã hoàn hảo ngay được mà vẫn cần phải sửa đổi lại. Chương trình viết dễ sửa đổi sẽ làm giảm bớt công sức của lập trình viên khi phát triển chương trình.

1.6.3. Tính trong sáng

Chương trình viết ra phải dễ đọc dễ hiểu, để sau một thời gian dài, khi đọc lại còn hiểu mình làm cái gì?. Để nếu có điều kiện thì còn có thể sửa sai (nếu phát hiện lỗi mới), cải tiến hay biến đổi để được chương trình giải quyết bài toán khác. Tính trong sáng của chương trình phụ thuộc rất nhiều vào công cụ lập trình và phong cách lập trình.

1.6.4. Tính hữu hiệu

Chương trình phải chạy nhanh và ít tốn bộ nhớ, tức là tiết kiệm được cả về không gian và thời gian. Để có một chương trình hữu hiệu, cần phải có giải thuật tốt và những tiểu xảo khi lập trình. Tuy nhiên, việc áp dụng quá nhiều tiểu xảo có thể khiến chương trình trở nên rối rắm, khó hiểu khi sửa đổi. Tiêu chuẩn hữu hiệu nên dừng lại ở mức chấp nhận được, không quan trọng bằng ba tiêu chuẩn trên. Bởi phần cứng phát triển rất nhanh, yêu cầu hữu hiệu không cần phải đặt ra quá nặng.

Từ những phân tích ở trên, chúng ta nhận thấy rằng việc làm ra một chương trình đòi hỏi rất nhiều công đoạn và tiêu tốn khá nhiều công sức. Chỉ một công đoạn không hợp lý sẽ làm tăng chi phí viết chương trình. Nghĩ ra cách giải quyết vấn đề đã khó, biến ý tưởng đó thành hiện thực cũng không dễ chút nào.

Những cấu trúc dữ liệu và giải thuật đề cập tới trong chuyên đề này là những kiến thức rất phổ thông, một người học lập trình không sớm thì muộn cũng phải biết tới. Chỉ hy vọng rằng khi học xong chuyên đề này, qua những cấu trúc dữ liệu và giải thuật hết sức mẫu mực, chúng ta rút ra được bài học kinh nghiệm: **Đừng bao giờ viết chương trình khi mà chưa suy xét kỹ về giải thuật và những dữ liệu cần thao tác**, bởi như vậy ta dễ mắc phải hai sai lầm trầm trọng: hoặc là sai về giải thuật, hoặc là giải thuật không thể triển khai nổi trên một cấu trúc dữ liệu không phù hợp. Chỉ cần mắc một trong hai lỗi đó thôi thì nguy cơ sụp đổ toàn bộ chương trình là hoàn toàn có thể, càng cố chữa càng bị rối, khả năng hầu như chắc chắn là phải làm lại

từ đầu(*).

(*) Tất nhiên, cẩn thận đến đâu thì cũng có xác suất rủi ro nhất định, ta hiểu được mức độ tai hại của hai lỗi này để hạn chế nó càng nhiều càng tốt

Chuyên Φ 40 φ

đề

2.1. ĐỘ PHỨC TẠP TÍNH TOÁN CỦA GIẢI THUẬT

§2. PHÂN TÍCH THỜI GIAN THỰC HIỆN GIẢI THUẬT

Với một bài toán không chỉ có một giải thuật. Chọn một giải thuật đưa tới kết quả nhanh nhất là một đòi hỏi thực tế. Như vậy cần có một căn cứ nào đó để nói rằng giải thuật này nhanh hơn giải thuật kia ?.

Thời gian thực hiện một giải thuật bằng chương trình máy tính phụ thuộc vào rất nhiều yếu tố. Một yếu tố cần chú ý nhất đó là kích thước của dữ liệu đưa vào. Dữ liệu càng lớn thì thời gian xử lý càng chậm, chẳng hạn như thời gian sắp xếp một dãy số phải chịu ảnh hưởng của số lượng các số thuộc dãy số đó. Nếu gọi n là kích thước dữ liệu đưa vào thì thời gian thực hiện của một giải thuật có thể biểu diễn một cách tương đối như một hàm của n: T(n).

Phần cứng máy tính, ngôn ngữ viết chương trình và chương trình dịch ngôn ngữ ấy đều ảnh hưởng tới thời gian thực hiện. Những yếu tố này không giống nhau trên các loại máy, vì vậy không thể dựa vào chúng khi xác định T(n). Tức là T(n) không thể biểu diễn bằng đơn vị thời gian giờ, phút, giây được. Tuy nhiên, không phải vì thế mà không thể so sánh được các giải thuật về mặt tốc độ. Nếu như thời gian thực hiện một giải thuật là $T_1(n) = n^2$ và thời gian thực hiện của một giải thuật khác là $T_2(n) = 100n$ thì khi n đủ lớn, thời gian thực hiện của giải thuật T_2 rõ ràng nhanh hơn giải thuật T_1. Khi đó, nếu nói rằng thời gian thực hiện giải thuật tỉ lệ thuận với n hay tỉ lệ thuận với n^2 cũng cho ta một cách đánh giá tương đối về tốc độ thực hiện của giải thuật đó khi n khá lớn. Cách đánh giá thời gian thực hiện giải thuật độc lập với máy tính và các yếu tố liên quan tới máy tính như vậy sẽ dẫn tới khái niệm gọi là **độ phức tạp tính toán của giải thuật**.

Cho f và g là hai hàm xác định dương với mọi n. Hàm f(n) được gọi là O(g(n)) nếu tồn tại một hằng số $c > 0$ và một giá trị n_0 sao cho:

$$f(n) \le c.g(n) \text{ với } \forall\ n \ge n_0$$

Nghĩa là nếu xét những giá trị $n \ge n_0$ thì hàm f(n) sẽ bị chặn trên bởi một hằng số nhân với g(n). Khi đó, nếu f(n) là thời gian thực hiện của một giải thuật thì ta nói giải thuật đó có cấp là g(n), ký hiệu: $O(g(n))^{(*)}$ hay $\Theta(g(n))$.

2.2. XÁC ĐỊNH ĐỘ PHỨC TẠP TÍNH TOÁN CỦA GIẢI THUẬT

Việc xác định độ phức tạp tính toán của một giải thuật bất kỳ có thể rất phức tạp. Tuy nhiên,

trong thực tế, đối với một số giải thuật ta có thể phân tích bằng một số quy tắc đơn giản:

(*) K ý pháp O(.) được gọi là ký pháp chữ O lớn (big O notation)

2.2.1. Quy tắc cộng

Nếu đoạn chương trình P_1 có thời gian thực hiện $T_1(n) = O(f(n))$ và đoạn chương trình P_2 có thời gian thực hiện là $T_2(n) = O(g(n))$ thì thời gian thực hiện P_1 rồi đến P_2 tiếp theo sẽ là

$$T_1(n) + T_2(n) = O(\max(f(n), g(n)))$$ Chứng minh:

$T_1(n) = O(f(n))$ nên $\exists\ n_1$ và c_1 để $T_1(n) \le c_1.f(n)$ với $\forall\ n \ge n_1$.

$T_2(n) = O(g(n))$ nên $\exists\ n_2$ và c_2 để $T_2(n) \le c_2.g(n)$ với $\forall\ n \ge n_2$. Chọn $n_0 =$

$\max(n_1, n_2)$ và $c = \max(c_1, c_2)$ ta có:

Với $\forall\ n \ge n_0$:

$T_1(n) + T_2(n) \le c_1.f(n) + c_2.g(n) \le c.f(n) + c.g(n) \le c.(f(n) + g(n)) \le 2c.(\max(f(n), g(n)))$. Vậy $T_1(n) +$

$T_2(n) = O(\max(f(n), g(n)))$.

2.2.2. Quy tắc nhân

Nếu đoạn chương trình P có thời gian thực hiện là $T(n) = O(f(n))$. Khi đó, nếu thực hiện k(n)

lần đoạn chương trình P với $k(n) = O(g(n))$ thì độ phức tạp tính toán sẽ là $O(g(n).f(n))$

Chứng minh:

Thời gian thực hiện k(n) lần đoạn chương trình P sẽ là k(n)T(n). Theo định nghĩa:

$\exists$ $c_k \geq 0$ và n_k để $k(n) \leq c^k(g(n))$ với $\forall$ $n \geq n_k$

$\exists$ $c_T \geq 0$ và n_T để $T(n) \leq c^T(f(n))$ với $\forall$ $n \geq n_T$

Vậy với $\forall$ $n \geq \max(n_T, n_k)$ ta có $k(n).T(n) \leq c^T.c^k(g(n).f(n))$

2.2.3. Một số tính chất

Theo định nghĩa về độ phức tạp tính toán ta có một số tính chất: a) Với P(n) là một đa thức bậc k thì $O(P(n)) = O(n^k)$. Vì thế, một thuật toán có độ phức tạp cấp đa thức, người ta thường ký hiệu là $O(n^k)$

b) Với a và b là hai cơ số tuỳ ý và f(n) là một hàm dương thì $\log_a f(n) = \log_a b.\log_b f(n)$. Tức là: $O(\log_a f(n)) = O(\log_b f(n))$. Vậy với một thuật toán có độ phức tạp cấp logarit của f(n), người ta ký hiệu là O(logf(n)) mà không cần ghi cơ số của logarit.

c) Nếu một thuật toán có độ phức tạp là hằng số, tức là thời gian thực hiện không phụ thuộc

vào kích thước dữ liệu vào thì ta ký hiệu độ phức tạp tính toán của thuật toán đó là O(1). d) Một giải thuật có cấp là các hàm như 2^n, n!, n^n được gọi là một giải thuật có độ phức tạp

hàm mũ. Những giải thuật như vậy trên thực tế thường có tốc độ rất chậm. Các giải thuật có

cấp là các hàm đa thức hoặc nhỏ hơn hàm đa thức thì thường chấp nhận được. e) Không phải lúc nào một giải thuật cấp $O(n^2)$ cũng tốt hơn giải thuật cấp $O(n^3)$. Bởi nếu như giải thuật cấp $O(n^2)$ có thời gian thực hiện là $1000n^2$, còn giải thuật cấp $O(n^3)$ lại chỉ cần thời

gian thực hiện là n^3, thì với $n < 1000$, rõ ràng giải thuật $O(n^3)$ tốt hơn giải thuật $O(n^2)$. Trên

đây là xét trên phương diện tính toán lý thuyết để định nghĩa giải thuật này "tốt" hơn giải

thuật kia, khi chọn một thuật toán để giải một bài toán thực tế phải có một sự mềm dẻo nhất

định.

f) Cũng theo định nghĩa về độ phức tạp tính toán

Một thuật toán có cấp O(1) cũng có thể viết là O(logn)

Một thuật toán có cấp O(logn) cũng có thể viết là O(n)

Một thuật toán có cấp O(n) cũng có thể viết là O(n.logn) Một thuật toán có cấp O(n.logn) cũng có thể viết là $O(n^2)$ Một thuật toán có cấp $O(n^2)$ cũng có thể viết là $O(n^3)$ Một thuật toán có cấp $O(n^3)$ cũng có thể viết là $O(2^n)$

Vậy độ phức tạp tính toán của một thuật toán có nhiều cách ký hiệu, thông thường người ta chọn cấp thấp nhất có thể, tức là chọn ký pháp O(f(n)) với f(n) là một hàm tăng chậm nhất theo n.

Dưới đây là một số hàm số hay dùng để ký hiệu độ phức tạp tính toán và bảng giá trị của chúng để tiện theo dõi sự tăng của hàm theo đối số n.

$\log_2 n$ n $n\log_2 n$ n^2 n^3 2^n 0 1 0 1 1 2 1 2 2 4 8 4 2 4 8 16 64 16 3 8 24 64 512 256 4 16 64 256 4096 65536 5 32 160 1024 32768 2147483648 Ví dụ:

Thuật toán tính tổng các số từ 1 tới n:

Nếu viết theo sơ đồ như sau:

Input n; S := 0; for i := 1 to n do S := S + i; Output S; Các đoạn chương trình ở các dòng 1, 2 và 4 có độ phức tạp tính toán là O(1).

Vòng lặp ở dòng 3 lặp n lần phép gán S := S + i, nên thời gian tính toán tỉ lệ thuận với n. Tức là độ phức tạp tính toán là O(n).

Vậy độ phức tạp tính toán của thuật toán trên là O(n).

Còn nếu viết theo sơ đồ như sau:

Input n; S := n * (n - 1) div 2; Output S; Thì độ phức tạp tính toán của thuật toán trên là O(1), thời gian tính toán không phụ thuộc vào

n.

2.2.4. Phép toán tích cực

Dựa vào những nhận xét đã nêu ở trên về các quy tắc khi đánh giá thời gian thực hiện giải thuật, ta chỉ cần chú ý đến một phép toán mà ta gọi là phép toán tích cực trong một đoạn chương trình. Đó là **một phép toán trong một đoạn chương trình mà số lần thực hiện không ít hơn các phép toán khác**. Xét hai đoạn chương trình tính e^x bằng công thức gần đúng:

$$\approx + + + + = \sum_{=} \textit{Lê Minh Hoàng}\ _2\ ^1\ !2!1\ ^{\cdots}\ !\ !$$ với x và n cho trước.

{Chương trình 1: Tính riêng từng hạng tử rồi cộng lại} **program Exp1; var i, j, n: Integer;**

x, p, S: Real; begin Write('x, n = '); ReadLn(x, n); S := 0; for i := 0 to n do
begin

p := 1; for j := 1 to i do p := p * x / j; S := S + p; end; WriteLn('exp(', x:1:4, ') = ', S:1:4); end.

e^x

$xx\ x_{n}$

n

$x_{i\ n_i}$

0

i {Tính hạng tử sau qua hạng tử trước} **program Exp2; var i, n: Integer; x, p, S: Real; begin Write('x, n = '); ReadLn(x, n); S := 1; p := 1; for i := 1 to n do**
begin

p := p * x / i; S := S + p; end; WriteLn('exp(', x:1:4, ') = ', S:1:4); end.

Ta có thể coi phép toán tích cực ở đây là p := p * x / j; Số lần thực hiện phép toán này là: 0 + 1 + 2 + ... + n = n(n - 1)/2 lần. Vậy độ phức tạp tính toán của thuật toán là $O(n^2)$

Ta có thể coi phép toán tích cực ở đây là phép p := p * x / i. Số lần thực hiện phép toán này là n. Vậy độ phức tạp tính toán của thuật toán là O(n).

2.3. ĐỘ PHỨC TẠP TÍNH TOÁN VỚI TÌNH TRẠNG DỮ LIỆU VÀO

Có nhiều trường hợp, thời gian thực hiện giải thuật không phải chỉ phụ thuộc vào kích thước dữ liệu mà còn phụ thuộc vào tình trạng của dữ liệu đó nữa. Chẳng hạn thời gian sắp xếp một dãy số theo thứ tự tăng dần mà dãy đưa vào chưa có thứ tự sẽ khác với thời gian sắp xếp một dãy số đã sắp xếp rồi hoặc đã sắp xếp theo thứ tự ngược lại. Lúc này, khi phân tích thời gian thực hiện giải thuật ta sẽ phải xét tới trường hợp tốt nhất, trường hợp trung bình và trường hợp xấu nhất. Khi khó khăn trong việc xác định độ phức tạp tính toán trong trường hợp trung bình (bởi việc xác định T(n) trung bình thường phải dùng tới những công cụ toán phức tạp), người ta thường chỉ đánh giá độ phức tạp tính toán trong trường hợp xấu nhất.

2.4. CHI PHÍ THỰC HIỆN THUẬT TOÁN

Khái niệm độ phức tạp tính toán đặt ra là để đánh giá chi phí thực hiện một giải thuật về mặt thời gian. Nhưng chi phí thực hiện giải thuật còn có rất nhiều yếu tố khác nữa: không gian bộ nhớ phải sử dụng là một ví dụ. Tuy nhiên, trên phương diện phân tích lý thuyết, ta chỉ có thể

xét tới vấn đề thời gian bởi việc xác định các chi phí khác nhiều khi rất mơ hồ và phức tạp.

Đối với người lập trình thì khác, một thuật toán với độ phức tạp dù rất thấp cũng sẽ là vô dụng nếu như không thể cài đặt được trên máy tính, chính vì vậy khi bắt tay cài đặt một thuật toán, ta phải biết cách tổ chức dữ liệu một cách khoa học, tránh lãng phí bộ nhớ không cần thiết. Có một quy luật tương đối khi tổ chức dữ liệu: Tiết kiệm được bộ nhớ thì thời gian thực hiện thường sẽ chậm hơn và ngược lại. Biết cân đối, dung hoà hai yếu tố đó là một kỹ năng cần thiết của người lập trình.

Bài tập

Bài 1

Chứng minh một cách chặt chẽ: Tại sao với P(n) là đa thức bậc k thì một giải thuật cấp O(P(n)) cũng có thể coi là cấp $O(n^k)$

Bài 2

Xác định độ phức tạp tính toán của những giải thuật sau bằng ký pháp chữ O lớn:

a) Đoạn chương trình tính tổng hai đa thức: $P(x) = a_m x^m + a_{m-1}x^{m-1} + ... + a_1 x + a_0$ và $Q(x) = b_n x^n + a_{n-1}x^{n-1} + ... + b_1 x + b_0$ Để được đa thức $R(x) = cpxp + cp_{-1}xp^{-1} + ... + c_1 x + c_0$

```
if m < n then p := m else p := n; {p = min(m, n)} for i := 0 to p
do c[i] := a[i] + b[i]; if p < m then
  for i := p + 1 to m do c[i] := a[i] else
  for i := p + 1 to n do c[i] := b[i]; while (p > 0) and (c[p] = 0) do p := p - 1;
```

b) Đoạn chương trình tính tích hai đa thức: $P(x) = a_m x^m + a_{m-1}x^{m-1} + ... + a_1 x + a_0$ và $Q(x) = b_n x^n + a_{n-1}x^{n-1} + ... + b_1 x + b_0$ Để được đa thức $R(x) = c_p x^p + c_{p-1}x^{p-1} + ... + c_1 x + c_0$

```
p := m + n; for i := 0 to p do c[i] := 0; for i
:= 0 to m do
  for j := 0 to n do
    c[i + j] := c[i + j] + a[i] * b[j];
```


§3. ĐỆ QUY VÀ GIẢI THUẬT ĐỆ QUY

3.1. KHÁI NIỆM VỀ ĐỆ QUY

Ta nói một đối tượng là đệ quy nếu nó được định nghĩa qua chính nó hoặc một đối tượng khác cùng dạng với chính nó bằng quy nạp.

Ví dụ: Đặt hai chiếc gương cầu đối diện nhau. Trong chiếc gương thứ nhất chứa hình chiếc gương thứ hai. Chiếc gương thứ hai lại chứa hình chiếc gương thứ nhất nên tất nhiên nó chứa lại hình ảnh của chính nó trong chiếc gương thứ nhất... Ở một góc nhìn hợp lý, ta có thể thấy một dãy ảnh vô hạn của cả hai chiếc gương.

Một ví dụ khác là nếu người ta phát hình trực tiếp phát thanh viên ngồi bên máy vô tuyến truyền hình, trên màn hình của máy này lại có chính hình ảnh của phát thanh viên đó ngồi bên

máy vô tuyến truyền hình và cứ như thế...

Trong toán học, ta cũng hay gặp các định nghĩa đệ quy:

Giai thừa của n (n!): Nếu n = 0 thì n! = 1; nếu n > 0 thì n! = n.(n-1)!

Ký hiệu số phần tử của một tập hợp hữu hạn S là |S|: Nếu S = ∅ thì |S| = 0; Nếu S ≠ ∅ thì tất có một phần tử x ∈ S, khi đó |S| = |S\{x}| + 1. Đây là phương pháp định nghĩa tập các số tự nhiên.

3.2. GIẢI THUẬT ĐỆ QUY

Nếu lời giải của một bài toán P được thực hiện bằng lời giải của bài toán P' có dạng giống như P thì đó là một lời giải đệ quy. Giải thuật tương ứng với lời giải như vậy gọi là giải thuật đệ quy. Mới nghe thì có vẻ hơi lạ nhưng điểm mấu chốt cần lưu ý là: P' tuy có dạng giống như P, nhưng theo một nghĩa nào đó, nó phải "nhỏ" hơn P, dễ giải hơn P và việc giải nó không cần dùng đến P.

Trong Pascal, ta đã thấy nhiều ví dụ của các hàm và thủ tục có chứa lời gọi đệ quy tới chính nó, bây giờ, ta tóm tắt lại các phép đệ quy trực tiếp và tương hỗ được viết như thế nào:

Định nghĩa một hàm đệ quy hay thủ tục đệ quy gồm hai phần:

Phần neo (anchor): Phần này được thực hiện khi mà công việc quá đơn giản, có thể giải trực tiếp chứ không cần phải nhờ đến một bài toán con nào cả.

Phần đệ quy: Trong trường hợp bài toán chưa thể giải được bằng phần neo, ta xác định những bài toán con và gọi đệ quy giải những bài toán con đó. Khi đã có lời giải (đáp số) của những bài toán con rồi thì phối hợp chúng lại để giải bài toán đang quan tâm.

Phần đệ quy thể hiện tính "quy nạp" của lời giải. Phần neo cũng rất quan trọng bởi nó quyết định tới tính hữu hạn dừng của lời giải.

Lê Minh Hoàng
Chuyên Φ 46 φ

đề

function Factorial(n: Integer): Integer; {Nhận vào số tự nhiên n và trả về n!}
begin

3.3. VÍ DỤ VỀ GIẢI THUẬT ĐỆ QUY

3.3.1. Hàm tính giai thừa

if n = 0 then Factorial := 1 {Phần neo} **else Factorial := n * Factorial(n - 1);** {Phần đệ quy} **end;** Ở đây, phần neo định nghĩa kết quả hàm tại n = 0, còn phần đệ quy (ứng với n > 0) sẽ định

nghĩa kết quả hàm qua giá trị của n và giai thừa của n - 1.

Ví dụ: Dùng hàm này để tính 3!, trước hết nó phải đi tính 2! bởi 3! được tính bằng tích của 3 * 2!. Tương tự để tính 2!, nó lại đi tính 1! bởi 2! được tính bằng 2 * 1!. Áp dụng bước quy nạp này thêm một lần nữa, 1! = 1 * 0!, và ta đạt tới trường hợp của phần neo, đến đây từ giá trị 1 của 0!, nó tính được 1! = 1*1 = 1; từ giá trị của 1! nó tính được 2!; từ giá trị của 2! nó tính được 3!; cuối cùng cho kết quả là 6:

3! = 3 * 2! ↓ 2! = 2 * 1! ↓ 1! = 1 * 0! ↓ 0! = 1

3.3.2. Dãy số Fibonacci

Dãy số Fibonacci bắt nguồn từ bài toán cổ về việc sinh sản của các cặp thỏ. Bài toán đặt ra như sau:

1) Các con thỏ không bao giờ chết

2) Hai tháng sau khi ra đời, mỗi cặp thỏ mới sẽ sinh ra một cặp thỏ con (một đực, một cái)

3) Khi đã sinh con rồi thì cứ mỗi tháng tiếp theo chúng lại sinh được một cặp con mới

Giả sử từ đầu tháng 1 có một cặp mới ra đời thì đến giữa tháng thứ n sẽ có bao nhiêu cặp.

Ví dụ, n = 5, ta thấy:

Giữa tháng thứ 1: 1 cặp (ab) (cặp ban đầu)

Giữa tháng thứ 2: 1 cặp (ab) (cặp ban đầu vẫn chưa đẻ)

Giữa tháng thứ 3: 2 cặp (AB)(cd) (cặp ban đầu đẻ ra thêm 1 cặp con)

Giữa tháng thứ 4: 3 cặp (AB)(cd)(ef) (cặp ban đầu tiếp tục đẻ)

Giữa tháng thứ 5: 5 cặp (AB)(CD)(ef)(gh)(ik) (cả cặp (AB) và (CD) cùng đẻ)

Bây giờ, ta xét tới việc tính số cặp thỏ ở tháng thứ n: F(n)

Nếu mỗi cặp thỏ ở tháng thứ n - 1 đều sinh ra một cặp thỏ con thì số cặp thỏ ở tháng thứ n sẽ là:

$$F(n) = 2 * F(n - 1)$$

Nhưng vấn đề không phải như vậy, trong các cặp thỏ ở tháng thứ n - 1, chỉ có những cặp thỏ đã có ở tháng thứ n - 2 mới sinh con ở tháng thứ n được thôi. Do đó F(n) = F(n - 1) + F(n - 2) (= số cũ + số sinh ra). Vậy có thể tính được F(n) theo công thức sau:

F(n) = 1 nếu n ≤ 2

F(n) = F(n - 1) + F(n - 2) nếu n > 2

```
function F(n: Integer): Integer; {Tính số cặp thỏ ở tháng thứ n} begin
if n ≤ 2 then F := 1 {Phần neo} else F := F(n - 1) + F(n - 2);
{Phần đệ quy} end;
```

3.3.3. Giả thuyết của Collatz

Collatz đưa ra giả thuyết rằng: với một số nguyên dương X, nếu X chẵn thì ta gán X := X div 2; nếu X lẻ thì ta gán X := X * 3 + 1. Thì sau một số hữu hạn bước, ta sẽ có X = 1.

Ví du: X = 10, các bước tiến hành như sau:

1. X = 10 (chẵn) ⇒ X := 10 div 2; (5) 2. X = 5 (lẻ) ⇒ X := 5 * 3 + 1; (16) 3. X = 16 (chẵn) ⇒ X := 16 div 2; (8) 4. X = 8 (chẵn) ⇒ X := 8 div 2 (4) 5. X = 4 (chẵn) ⇒ X := 4 div 2 (2) 6. X = 2 (chẵn) ⇒ X := 2 div 2 (1) Cứ cho giả thuyết Collatz là đúng đắn, vấn đề đặt ra là: Cho trước số 1 cùng với hai phép toán * 2 và div 3, hãy sử dụng một cách hợp lý hai phép toán đó để biến số 1 thành một giá trị nguyên dương X cho trước.

Ví dụ: X = 10 ta có 1 * 2 * 2 * 2 * 2 div 3 * 2 = 10.

Dễ thấy rằng lời giải của bài toán gần như thứ tự ngược của phép biến đổi Collatz: Để biểu diễn số X > 1 bằng một biểu thức bắt đầu bằng số 1 và hai phép toán "* 2", "div 3". Ta chia hai trường hợp:

Nếu X chẵn, thì ta tìm cách biểu diễn số X div 2 và viết thêm phép toán * 2 vào cuối

Nếu X lẻ, thì ta tìm cách biểu diễn số X * 3 + 1 và viết thêm phép toán div 3 vào cuối

```
procedure Solve(X: Integer); {In ra cách biểu diễn số X} begin
  if X = 1 then Write(X) {Phần neo} else {Phần đệ
  quy}
    if X mod 2 = 0 then {X chẵn}
      begin
    Solve(X div 2); {Tìm cách biểu diễn số X div 2} Write(' * 2');
    {Sau đó viết thêm phép toán * 2} end else {X lẻ}
      begin
Solve(X * 3 + 1); {Tìm cách biểu diễn số X * 3 + 1} Write(' div 3'); {Sau đó viết thêm phép toán div 3} end; end;
```

Trên đây là cách viết đệ quy trực tiếp, còn có một cách viết đệ quy tương hỗ như sau:

```
procedure Solve(X: Integer); forward; {Thủ tục tìm cách biểu diễn số X: Khai báo trước, đặc tả sau}
```

chẵn} **begin**
Solve(X div 2);
Write(' div 2'); end;

procedure SolveOdd(X: Integer); {Thủ tục tìm cách biểu diễn số X trong trường hợp X lẻ} **begin**
Solve(X * 3 + 1);
Write(' * 3 + 1'); end;

procedure Solve(X: Integer); {Phần đặc tả của thủ tục Solve đã khai báo trước
begin
if X = 1 then Write(X)
else

procedure SolveEven(X: Integer); {Thủ tục tìm cách biểu diễn số X trong trường hợp X chẵn}
if X mod 2 = 1 then SolveOdd(X) else SolveEven(X); end; Trong cả hai cách viết, để tìm biểu diễn số X theo yêu cầu chỉ cần gọi Solve(X) là xong. Tuy

nhiên trong cách viết đệ quy trực tiếp, thủ tục Solve có lời gọi tới chính nó, còn trong cách viết đệ quy tương hỗ, thủ tục Solve chứa lời gọi tới thủ tục SolveOdd và SolveEven, hai thủ tục này lại chứa trong nó lời gọi ngược về thủ tục Solve.

Đối với những bài toán nêu trên, việc thiết kế các giải thuật đệ quy tương ứng khá thuận lợi vì cả hai đều thuộc dạng tính giá trị hàm mà định nghĩa quy nạp của hàm đó được xác định dễ dàng.

Nhưng không phải lúc nào phép giải đệ quy cũng có thể nhìn nhận và thiết kế dễ dàng như vậy. Thế thì vấn đề gì cần lưu tâm trong phép giải đệ quy?. Có thể tìm thấy câu trả lời qua việc giải đáp các câu hỏi sau:

1. Có thể định nghĩa được bài toán dưới dạng phối hợp của những bài toán cùng loại nhưng nhỏ hơn hay không ? Khái niệm "nhỏ hơn" là thế nào ?
2. Trường hợp đặc biệt nào của bài toán sẽ được coi là trường hợp tầm thường và có thể giải ngay được để đưa vào phần neo của phép giải đệ quy

3.3.4. Bài toán Tháp Hà Nội

Đây là một bài toán mang tính chất một trò chơi, tương truyền rằng tại ngôi đền Benares có ba cái cọc kim cương. Khi khai sinh ra thế giới, thượng đế đặt n cái đĩa bằng vàng chồng lên nhau theo thứ tự giảm dần của đường kính tính từ dưới lên, đĩa to nhất được đặt trên một chiếc cọc.

Hình 5: Tháp Hà Nội

Các nhà sư lần lượt chuyển các đĩa sang cọc khác theo luật:

- Khi di chuyển một đĩa, phải đặt nó vào một trong ba cọ
- Mỗi lần chỉ có thể chuyển một đĩa và phải là đĩa ở trên
- Tại một vị trí, đĩa nào mới chuyển đến sẽ phải đặt lên tr

Φ 49 φ

- Đĩa lớn hơn không bao giờ được phép đặt lên trên đĩa nhỏ hơn (hay nói cách khác: một đĩa chỉ được đặt trên cọc hoặc đặt trên một đĩa lớn hơn)

Ngày tận thế sẽ đến khi toàn bộ chồng đĩa được chuyển sang một cọc khác.

Trong trường hợp có 2 đĩa, cách làm có thể mô tả như sau:

Chuyển đĩa nhỏ sang cọc 3, đĩa lớn sang cọc 2 rồi chuyển đĩa nhỏ từ cọc 3 sang cọc 2.

Những người mới bắt đầu có thể giải quyết bài toán một cách dễ dàng khi số đĩa là ít, nhưng họ sẽ gặp rất nhiều khó khăn khi số các đĩa nhiều hơn. Tuy nhiên, với tư duy quy nạp toán học và một máy tính thì công việc trở nên khá dễ dàng:

Có n đĩa.

- Nếu n = 1 thì ta chuyển đĩa duy nhất đó từ cọc 1 sang cọc 2 là xong.
- Giả sử rằng ta có phương pháp chuyển được n - 1 đĩa từ cọc 1 sang cọc 2, thì cách chuyển n - 1 đĩa từ cọc x sang cọc y (1 ≤ x, y ≤ 3) cũng tương tự.
- Giả sử ràng ta có phương pháp chuyển được n - 1 đĩa giữa hai cọc bất kỳ. Để chuyển n đĩa từ cọc x sang cọc y, ta gọi cọc còn lại là z (=6 - x - y). Coi đĩa to nhất là ... cọc, chuyển n - 1 đĩa còn lại từ cọc x sang cọc z, sau đó chuyển đĩa to nhất đó sang cọc y và cuối cùng lại coi đĩa to nhất đó là cọc, chuyển n - 1 đĩa còn lại đang ở cọc z sang cọc y chồng lên đĩa to nhất.

Cách làm đó được thể hiện trong thủ tục đệ quy dưới đây:

```
procedure Move(n, x, y: Integer); {Thủ tục chuyển n đĩa từ cọc x sang cọc y} begin
  if n = 1 then WriteLn('Chuyển 1 đĩa từ ', x, ' sang ', y) else {Để chuyển n > 1 đĩa từ
  cọc x sang cọc y, ta chia làm 3 công đoạn}
    begin
Move(n - 1, x, 6 - x - y); {Chuyển n - 1 đĩa từ cọc x sang cọc trung gian} Move(1, x, y); {Chuyển đĩa to nhất từ x sang y}
```

Move(n - 1, 6 - x - y, y); {Chuyển n - 1 đĩa từ cọc trung gian sang cọc y} **end; end;** Chương trình chính rất đơn giản, chỉ gồm có 2 việc: Nhập vào số n và gọi Move(n, 1, 2).

Chuyên Φ 50 φ

đề

3.4. HIỆU LỰC CỦA ĐỆ QUY

Qua các ví dụ trên, ta có thể thấy đệ quy là một công cụ mạnh để giải các bài toán. Có những bài toán mà bên cạnh giải thuật đệ quy vẫn có những giải thuật lặp khá đơn giản và hữu hiệu. Chẳng hạn bài toán tính giai thừa hay tính số Fibonacci. Tuy vậy, đệ quy vẫn có vai trò xứng đáng của nó, có nhiều bài toán mà việc thiết kế giải thuật đệ quy đơn giản hơn nhiều so với lời giải lặp và trong một số trường hợp chương trình đệ quy hoạt động nhanh hơn chương trình viết không có đệ quy. Giải thuật cho bài Tháp Hà Nội và thuật toán sắp xếp kiểu phân đoạn (QuickSort) mà ta sẽ nói tới trong các bài sau là những ví dụ.

Có một mối quan hệ khăng khít giữa đệ quy và quy nạp toán học. Cách giải đệ quy cho một bài toán dựa trên việc định rõ lời giải cho trường hợp suy biến (neo) rồi thiết kế làm sao để lời giải của bài toán được suy ra từ lời giải của bài toán nhỏ hơn cùng loại như thế. Tương tự như vậy, quy nạp toán học chứng minh một tính chất nào đó ứng với số tự nhiên cũng bằng cách chứng minh tính chất đó đúng với một số trường hợp cơ sở (thường người ta chứng minh nó đúng với 0 hay đúng với 1) và sau đó chứng minh tính chất đó sẽ đúng với n bất kỳ nếu nó đã đúng với mọi số tự nhiên nhỏ hơn n.

Do đó ta không lấy làm ngạc nhiên khi thấy quy nạp toán học được dùng để chứng minh các tính chất có liên quan tới giải thuật đệ quy. Chẳng hạn: Chứng minh số phép chuyển đĩa để giải bài toán Tháp Hà Nội với n đĩa là 2^n-1: Rõ ràng là tính chất này đúng với n = 1, bởi ta cần 2^1 - 1 = 1 lần chuyển đĩa để thực hiện yêu

cầu Với n > 1; Giả sử rằng để chuyển n - 1 đĩa giữa hai cọc ta cần 2^{n-1} - 1 phép chuyển đĩa, khi đó để chuyển n đĩa từ cọc x sang cọc y, nhìn vào giải thuật đệ quy ta có thể thấy rằng trong trường hợp này nó cần $(2^{n-1} - 1) + 1 + (2^{n-1} - 1) = 2^n - 1$ phép chuyển đĩa. Tính chất được chứng minh đúng với n

Vậy thì công thức này sẽ đúng với mọi n.

Thật đáng tiếc nếu như chúng ta phải lập trình với một công cụ không cho phép đệ quy, nhưng như vậy không có nghĩa là ta bó tay trước một bài toán mang tính đệ quy. Mọi giải

thuật đệ quy đều có cách thay thế bằng một giải thuật không đệ quy (khử đệ quy), có thể nói được như vậy bởi tất cả các chương trình con đệ quy sẽ đều được trình dịch chuyển thành những mã lệnh không đệ quy trước khi giao cho máy tính thực hiện.

Việc tìm hiểu cách khử đệ quy một cách "máy móc" như các chương trình dịch thì chỉ cần hiểu rõ cơ chế xếp chồng của các thủ tục trong một dây chuyền gọi đệ quy là có thể làm được. Nhưng muốn khử đệ quy một cách tinh tế thì phải tuỳ thuộc vào từng bài toán mà khử đệ quy cho khéo. Không phải tìm đâu xa, những kỹ thuật giải công thức truy hồi bằng quy hoạch

động là ví dụ cho thấy tính nghệ thuật trong những cách tiếp cận bài toán mang bản chất đệ quy để tìm ra một giải thuật không đệ quy đầy hiệu quả.

Bài tập

Bài 1

Viết một hàm đệ quy tính ước số chung lớn nhất của hai số tự nhiên a, b không đồng thời bằng 0, chỉ rõ đâu là phần neo, đâu là phần đệ quy.

Bài 2

Viết một hàm đệ quy tính C_n^k theo công thức truy hồi sau:

$C_{n0} = 1$

$C_{nn} = 1$ Với $0 < k < n$: $C_{kn} = C_{k-1n-1} + C_{kn-1}$

Chứng minh rằng hàm đó cho ra đúng giá trị $C_n^k = n! / (k!(n-k)!)$

Lê Minh Hoàng

Bài 3

Nêu rõ các bước thực hiện của giải thuật cho bài Tháp Hà Nội trong trường hợp n = 3.

Viết chương trình giải bài toán Tháp Hà Nội không đệ quy

§4. CẤU TRÚC DỮ LIỆU BIỂU DIỄN DANH SÁCH

Danh sách là một tập sắp thứ tự các phần tử cùng một kiểu. Đối với danh sách, người ta có một số thao tác: Tìm một phần tử trong danh sách, chèn một phần tử vào danh sách, xoá một phần tử khỏi danh sách, sắp xếp lại các phần tử trong danh sách theo một trật tự nào đó v.v...

4.2. BIỂU DIỄN DANH SÁCH TRONG MÁY TÍNH

Việc cài đặt một danh sách trong máy tính tức là tìm một cấu trúc dữ liệu cụ thể mà máy tính hiểu được để lưu các phần tử của danh sách đồng thời viết các đoạn chương trình con mô tả các thao tác cần thiết đối với danh sách.

4.2.1. Cài đặt bằng mảng một chiều

Khi cài đặt danh sách bằng một mảng, thì có một biến nguyên n lưu số phần tử hiện có trong danh sách. Nếu mảng được đánh số bắt đầu từ 1 thì các phần tử trong danh sách được cất giữ trong mảng bằng các phần tử được đánh số từ 1 tới n.

Chèn phần tử vào mảng:

Mảng ban đầu:

p

Nếu muốn chèn một phần tử V vào mảng tại vị trí p, ta phải:

Dồn tất cả các phần tử từ vị trí p tới tới vị trí n về sau một vị trí:

L

A

Đặt giá trị V vào vị trí p:

Tăng n lên 1

Xoá phần tử khỏi mảng

Mảng ban đầu:

p

Muốn xoá phần tử thứ p của mảng mà vẫn giữ nguyên thứ tự các phần tử còn lại, ta phải:

Dồn tất cả các phần tử từ vị trí p + 1 tới vị trí n lên trước một vị trí:

A

A

p

K
K
K
K
K
K
L
L
L
L
L
L
L
L
L
L
L
L
L
L
L
L
L
L
L
L
L
p

p

Giảm n đi 1

p
Φ 53 φ

Trong trường hợp cần xóa một phần tử mà không cần duy trì thứ tự của các phần tử khác, ta chỉ cần đảo giá trị của phần tử cần xóa cho phần tử cuối cùng rồi giảm số phần tử của mảng (n) đi 1.

4.2.2. Cài đặt bằng danh sách nối đơn

Danh sách nối đơn gồm các nút được nối với nhau theo một chiều. Mỗi nút là một bản ghi (record) gồm hai trường:

Trường thứ nhất chứa giá trị lưu trong nút đó

Trường thứ hai chứa liên kết (con trỏ) tới nút kế tiếp, tức là chứa một thông tin đủ để biết nút kế tiếp nút đó trong danh sách là nút nào, trong trường hợp là nút cuối cùng (không có nút kế tiếp), trường liên kết này được gán một giá trị đặc biệt.

Hình 6: Cấu trúc nút của danh sách nối đơn

Nút đầu tiên trong danh sách được gọi là chốt của danh sách nối đơn (Head). Để duyệt danh sách nối đơn, ta bắt đầu từ chốt, dựa vào trường liên kết để đi sang nút kế tiếp, đến khi gặp giá trị đặc biệt (duyệt qua nút cuối) thì dừng lại

Hình 7: Danh sách nối đơn

Chèn phần tử vào danh sách nối đơn:

Danh sách ban đầu:

Head

q p

Muốn chèn thêm một nút chứa giá trị V vào vị trí của nút p, ta phải:

A

A

A

A

J
J
J
J
J
J

K
K
K
K
K
K
K
K
K

K
K
K
K
K
K
K
K
K

C
C

C
C

J
J
J
J
J
J
J
J

J
J

D
D

D

D
D
D

L
L
L
L
L
L
L
L
L
L

L
L
L
L
L
L
L
L
L
L

E
E
E
E

E
E
E
E

Φ 54 φ

a) Tạo ra một nút mới NewNode chứa giá trị V:

A

D
D
D
D

E

b_2) Nếu không có nút đứng trước nút p trong danh sách thì tức là p = Head, ta chỉnh lại liên kết: NewNode liên kết tới Head (cũ) và đặt lại Head = NewNode

Xoá phần tử khỏi danh sách nối đơn:

Danh sách ban đầu:

Head

q p

Muốn huỷ nút p khỏi danh sách nối đơn, ta phải:

Tìm nút q là nút đứng liền trước nút p trong danh sách (nút có liên kết tới p)

Nếu tìm thấy thì chỉnh lại liên kết: q liên kết thẳng tới nút liền sau p, khi đó quá trình duyệt danh sách bắt đầu từ Head khi duyệt tới q sẽ nhảy qua không duyệt p nữa. Trên thực tế khi cài đặt bằng các biến động và con trỏ, ta nên có thao tác giải phóng bộ nhớ đã cấp cho nút p

Head

q p

Nếu không có nút đứng trước nút p trong danh sách thì tức là p = Head, ta chỉ việc đặt lại Head bằng nút đứng kế tiếp Head (cũ) trong danh sách. Sau đó có thể giải phóng bộ nhớ cấp cho nút p (Head cũ)

4.2.3. Cài đặt bằng danh sách nối kép

Danh sách nối kép gồm các nút được nối với nhau theo hai chiều. Mỗi nút là một bản ghi

(record) gồm ba trường:

b) Tìm nút q là nút đứng trước nút p trong danh sách (nút có liên kết tới p).

b_1) Nếu tìm thấy thì chỉnh lại liên kết: q liên kết tới NewNode, NewNode liên kết tới p

Head

A

V

A

V

q

D
D
D

E
E
E
E

D
D
D

E
E
E
E

• Trường thứ nhất chứa giá trị lưu trong nút đó

p

• Trường thứ hai (Next) chứa liên kết (con trỏ) tới nút kế tiếp, tức là chứa một thông tin đủ để biết nút kế tiếp nút đó là nút nào, trong trường hợp là nút cuối cùng (không có nút kế tiếp), trường liên kết này được gán một giá tị đặc biệt.

• Trường thứ ba (Prev) chứa liên kết (con trỏ) tới nút liền trước, tức là chứa một thông tin đủ để biết nút đứng trước nút đó trong danh sách là nút nào, trong trường hợp là

nút đầu tiên (không có nút liền trước) trường này được gán một giá trị đặc biệt.

Liên kết sau

Hình 8: Cấu trúc nút của danh sách nối kép

Khác với danh sách nối đơn, danh sách nối kép có hai chốt: Nút đầu tiên trong danh sách

được gọi là First, nút cuối cùng trong danh sách được gọi là Last. Để duyệt danh sách nối kép, ta có hai cách: Hoặc bắt đầu từ First, dựa vào liên kết Next để đi sang nút kế tiếp, đến khi gặp giá trị đặc biệt (duyệt qua nút cuối) thì dừng lại. Hoặc bắt đầu từ Last, dựa vào liên kết Prev để đi sang nút liền trước, đến khi gặp giá trị đặc biệt (duyệt qua nút đầu) thì dừng lại

Hình 9: Danh sách nối kép

Việc chèn / xoá vào danh sách nối kép cũng đơn giản chỉ là kỹ thuật chỉnh lại các mối liên kết giữa các nút cho hợp lý, ta coi như bài tập.

4.2.4. Cài đặt bằng danh sách nối vòng một hướng

Trong danh sách nối đơn, phần tử cuối cùng trong danh sách có trường liên kết được gán một giá trị đặc biệt (thường sử dụng nhất là giá trị nil). Nếu ta cho trường liên kết của phần tử cuối cùng trỏ thẳng về phần tử đầu tiên của danh sách thì ta sẽ được một kiểu danh sách mới gọi là danh sách nối vòng một hướng.

Hình 10: Danh sách nối vòng một hướng

D
D
D
D
D
D
E
E
E
E
Data
Giá trị
Liên kết trước
Φ 56 φ
Chuyên đề
D

Đối với danh sách nối vòng, ta chỉ cần biết một nút bất kỳ của danh sách là ta có thể duyệt được hết các nút trong danh sách bằng cách đi theo hướng của các liên kết. Chính vì lý do này, khi chèn xoá vào danh sách nối vòng, ta không phải xử lý các trường hợp riêng khi chèn xoá tại vị trí của chốt

4.2.5. Cài đặt bằng danh sách nối vòng hai hướng

Danh sách nối vòng một hướng chỉ cho ta duyệt các nút của danh sách theo một chiều, nếu cài đặt bằng danh sách nối vòng hai hướng thì ta có thể duyệt các nút của danh sách cả theo chiều ngược lại nữa. Danh sách nối vòng hai hướng có thể tạo thành từ danh sách nối kép nếu ta cho trường Prev của nút First trỏ thẳng tới nút Last còn trường Next của nút Last thì trỏ thẳng về nút First.

Hình 11: Danh sách nối vòng hai hướng

Bài tập

Bài 1

Lập chương trình quản lý danh sách học sinh, tuỳ chọn loại danh sách cho phù hợp, chương

trình có những chức năng sau: (Hồ sơ một học sinh giả sử có: Tên, lớp, số điện thoại, điểm TB ...)

Cho phép nhập danh sách học sinh từ bàn phím hay từ file.

Cho phép in ra danh sách học sinh gồm có tên và xếp loại

Cho phép in ra danh sách học sinh gồm các thông tin đầy đủ

Cho phép nhập vào từ bàn phím một tên học sinh và một tên lớp, tìm xem có học sinh có tên nhập vào trong lớp đó không ?. Nếu có thì in ra số điện thoại của học sinh đó

Cho phép vào một hồ sơ học sinh mới từ bàn phím, bổ sung học sinh đó vào danh sách học sinh, in ra danh sách mới.

Cho phép nhập vào từ bàn phím tên một lớp, loại bỏ tất cả các học sinh của lớp đó khỏi danh sách, in ra danh sách mới.

Có chức năng sắp xếp danh sách học sinh theo thứ tự giảm dần của điểm trung bình

Cho phép nhập vào hồ sơ một học sinh mới từ bàn phím, chèn học sinh đó vào danh sách mà không làm thay đổi thứ tự đã sắp xếp, in ra danh sách mới.

Cho phép lưu trữ lại trên đĩa danh sách học sinh khi đã thay đổi.

A

D
D
D

Bài 2

Đại học Sư phạm Hà Nội, 1999-2002

Có n người đánh số từ 1 tới n ngồi quanh một vòng tròn (n ≤ 10000), cùng chơi một trò chơi: Một người nào đó đếm 1, người kế tiếp, theo chiều kim đồng hồ đếm 2... cứ như vậy cho tới người đếm đến một số nguyên tố thì phải ra khỏi vòng tròn, người kế tiếp lại đếm bắt đầu từ 1: Hãy lập chương trình

Nhập vào 2 số n và S từ bàn phím

- Cho biết nếu người thứ nhất là người đếm 1 thì người còn lại cuối cùng trong vòng tròn là người thứ mấy
- Cho biết nếu người còn lại cuối cùng trong vòng tròn là người thứ k thì người đếm 1 là người nào?.

Giải quyết hai yêu cầu trên trong trường hợp: đầu tiên trò chơi được đếm theo chiều kim đồng hồ, khi có một người bị ra khỏi cuộc chơi thì vẫn là người kế tiếp đếm 1 nhưng quá trình đếm

ngược lại (tức là ngược chiều kim đồng hồ)

đề

5.1. NGĂN XẾP (STACK)

§5. NGĂN XẾP VÀ HÀNG ĐỢI

Ngăn xếp là một kiểu danh sách được trang bị hai phép toán **bổ sung một phần tử** vào cuối danh sách và **loại bỏ một phần tử** cũng ở cuối danh sách.

Có thể hình dung ngăn xếp như hình ảnh một chồng đĩa, đĩa nào được đặt vào chồng sau cùng sẽ nằm trên tất cả các đĩa khác và sẽ được lấy ra đầu tiên. Vì nguyên tắc"vào sau ra trước" đó, Stack còn có tên gọi là danh sách kiểu LIFO (Last In First Out) và vị trí cuối danh sách được gọi là đỉnh (Top) của Stack.

5.1.1. Mô tả Stack bằng mảng

Khi mô tả Stack bằng mảng:

Việc bổ sung một phần tử vào Stack tương đương với việc thêm một phần tử vào cuối mảng.

Việc loại bỏ một phần tử khỏi Stack tương đương với việc loại bỏ một phần tử ở cuối mảng.

Stack bị tràn khi bổ sung vào mảng đã đầy

Stack là rỗng khi số phần tử thực sự đang chứa trong mảng = 0.

```
program StackByArray; const
      max = 10000; var
  Stack: array[1..max] of Integer; Last: Integer;

procedure StackInit; {Khởi tạo Stack rỗng} begin
      Last := 0; end;

procedure Push(V: Integer); {Đẩy một giá trị V vào Stack} begin
  if Last = max then WriteLn('Stack is full') {Nếu Stack đã đầy thì không đẩy được thêm vào nữa} else
    begin
      Inc(Last); Stack[Last] := V; {Nếu không thì thêm một phần tử vào cuối mảng} end; end;

function Pop: Integer; {Lấy một giá trị ra khỏi Stack, trả về trong kết quả hàm} begin
  if Last = 0 then WriteLn('Stack is empty') {Stack đang rỗng thì không lấy được} else
    begin
      Pop := Stack[Last]; Dec(Last); {Lấy phần tử cuối ra khỏi mảng} end; end;

begin
  StackInit; <Test>; {Đưa một vài lệnh để kiểm tra hoạt động của Stack}
```


end. Khi cài đặt bằng mảng, tuy các thao tác đối với Stack viết hết sức đơn giản nhưng ở đây ta vẫn chia thành các chương trình con, mỗi chương trình con mô tả một thao tác, để từ đó về sau, ta chỉ cần biết rằng chương trình của ta có một cấu trúc Stack, còn ta mô phỏng cụ thể như thế nào thì không cần phải quan tâm nữa, và khi cài đặt Stack bằng các cấu trúc dữ liệu khác, chỉ cần sửa lại các thủ tục StackInit, Push và Pop mà thôi.

5.1.2. Mô tả Stack bằng danh sách nối đơn kiểu LIFO

Khi cài đặt Stack bằng danh sách nối đơn kiểu LIFO, thì Stack bị tràn khi vùng không gian nhớ dùng cho các biến động không còn đủ để thêm một phần tử mới. Tuy nhiên, việc kiểm tra điều này rất khó bởi nó phụ thuộc vào máy tính và ngôn ngữ lập trình. Ví dụ như đối với

Turbo Pascal, khi Heap còn trống 80 Bytes thì cũng chỉ đủ chỗ cho 10 biến, mỗi biến 6 Bytes mà thôi. Mặt khác, không gian bộ nhớ dùng cho các biến động thường rất lớn nên cài đặt dưới đây ta bỏ qua việc kiểm tra Stack tràn.

```
program StackByLinkedList; type
  PNode = ^TNode; {Con trỏ tới một nút của danh sách} TNode =
  record {Cấu trúc một nút của danh sách}
Value: Integer; Link: PNode;
end; var
  Last: PNode; {Con trỏ đỉnh Stack}

procedure StackInit; {Khởi tạo Stack rỗng} begin
Last := nil; end;

procedure Push(V: Integer); {Đẩy giá trị V vào Stack ⇔ thêm nút mới chứa V và nối nút đó vào danh sách} var
P: PNode; begin
New(P); P^.Value := V; {Tạo ra một nút mới} P^.Link := Last; Last
:= P; {Móc nút đó vào danh sách} end;

function Pop: Integer; {Lấy một giá trị ra khỏi Stack, trả về trong kết quả hàm} var
P: PNode; begin
  if Last = nil then WriteLn('Stack is empty') else
    begin
Pop := Last^.Value; {Gán kết quả hàm} P := Last^.Link; {Giữ lại nút tiếp theo last^ (nút được đẩy vào danh
sách trước nút Last^)} Dispose(Last); Last := P; {Giải phóng bộ nhớ cấp cho Last^, cập nhật lại Last mới} end;
end;

begin
  StackInit; <Test>; {Đưa một vài lệnh để kiểm tra hoạt động của Stack}
```

Lê Minh Hoàng

Chuyên Φ 60 φ

(QUEUE)

đề

end.

5.2. HÀNG ĐỢI

Hàng đợi là một kiểu danh sách được trang bị hai phép toán **bổ sung một phần tử** vào cuối danh sách (Rear) và **loại bỏ một phần tử** ở đầu danh sách (Front).

Có thể hình dung hàng đợi như một đoàn người xếp hàng mua vé: Người nào xếp hàng trước sẽ được mua vé trước. Vì nguyên tắc"vào trước ra trước" đó, Queue còn có tên gọi là danh sách kiểu FIFO (First In First Out).

5.2.1. Mô tả Queue bằng mảng

Khi mô tả Queue bằng mảng, ta có hai chỉ số First và Last, First lưu chỉ số phần tử đầu Queue còn Last lưu chỉ số cuối Queue, khởi tạo Queue rỗng: First := 1 và Last := 0;

Để thêm một phần tử vào Queue, ta tăng Last lên 1 và đưa giá trị đó vào phần tử thứ Last.

Để loại một phần tử khỏi Queue, ta lấy giá trị ở vị trí First và tăng First lên 1.

Khi Last tăng lên hết khoảng chỉ số của mảng thì mảng đã đầy, không thể đẩy thêm phần tử vào nữa.

Khi First > Last thì tức là Queue đang rỗng

Như vậy chỉ một phần của mảng từ vị trí First tới Last được sử dụng làm Queue.

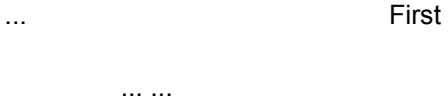
```
program QueueByArray; const
      max = 10000; var
  Queue: array[1..max] of Integer; First, Last:
  Integer;

procedure QueueInit; {Khởi tạo một hàng đợi rỗng} begin
      First := 1; Last := 0; end;

procedure Push(V: Integer); {Đẩy V vào hàng đợi} begin
  if Last = max then WriteLn('Overflow') else
    begin
Inc(Last); Queue[Last] := V; end;
end;

function Pop: Integer; {Lấy một giá trị khỏi hàng đợi, trả về trong kết quả hàm} begin
  if First > Last then WriteLn('Queue is Empty') else
    begin
Pop := Queue[First]; Inc(First); end;
end;

begin
```


QueueInit; <Test>; {Đưa một vài lệnh để kiểm tra hoạt động của Queue} **end.** Xem lại chương trình cài đặt Stack bằng một mảng kích thước tối đa 10000 phần tử, ta thấy

rằng nếu như ta làm 6000 lần Push rồi 6000 lần Pop rồi lại 6000 lần Push thì vẫn không có vấn đề gì xảy ra. Lý do là vì chỉ số Last lưu đỉnh của Stack sẽ được tăng lên 6000 rồi lại giảm đến 0 rồi lại tăng trở lại lên 6000. Nhưng đối với cách cài đặt Queue như trên thì sẽ gặp thông báo lỗi tràn mảng, bởi mỗi lần Push, chỉ số cuối hàng đợi Last cũng tăng lên và không bao giờ bị giảm đi cả. Đó chính là nhược điểm mà ta nói tới khi cài đặt: Chỉ có các phần tử từ vị trí First tới Last là thuộc Queue, các phần tử từ vị trí 1 tới First - 1 là vô nghĩa.

Để khắc phục điều này, ta mô tả Queue bằng một danh sách vòng (biểu diễn bằng mảng hoặc cấu trúc liên kết), coi như các phần tử của mảng được xếp quanh vòng theo một hướng nào đó.

Các phần tử nằm trên phần cung tròn từ vị trí First tới vị trí Last là các phần tử của Queue. Có thêm một biến n lưu số phần tử trong Queue. Việc thêm một phần tử vào Queue tương đương với việc ta dịch chỉ số Last theo vòng một vị trí rồi đặt giá trị mới vào đó. Việc loại bỏ một phần tử trong Queue tương đương với việc lấy ra phần tử tại vị trí First rồi dịch chỉ số First theo vòng.



Hình 12: Dùng danh sách vòng mô tả Queue

Lưu ý là trong thao tác Push và Pop phải kiểm tra Queue tràn hay Queue cạn nên phải cập nhật lại biến n. (Ở đây dùng thêm biến n cho dễ hiểu còn trên thực tế chỉ cần hai biến First và Last là ta có thể kiểm tra được Queue tràn hay cạn rồi)

```
program QueueByCList; const
max = 10000; var
  Queue: array[0..max - 1] of Integer; i, n, First,
  Last: Integer;

procedure QueueInit; {Khởi tạo Queue rỗng} begin
First := 0; Last := max - 1; n := 0; end;

procedure Push(V: Integer); {Đẩy giá trị V vào Queue} begin
  if n = max then WriteLn('Queue is Full') else
```

Last

đề

```
    begin
Last := (Last + 1) mod max; {Last chạy theo vòng tròn}
Queue[Last] := V; Inc(n); end; end;

function Pop: Integer; {Lấy một phần tử khỏi Queue, trả về trong kết quả
hàm} begin
  if n = 0 then WriteLn('Queue is
  Empty') else
    begin
Pop := Queue[First]; First := (First + 1) mod max; {First chạy
theo vòng tròn} Dec(n); end; end;

begin
QueueInit; <Test>; {Đưa một vài lệnh để kiểm tra hoạt động
của Queue} end.
```

5.2.2. Mô tả Queue bằng danh sách nối đơn kiểu FIFO

Tương tự như cài đặt Stack bằng danh sách nối đơn kiểu LIFO, ta cũng không kiểm tra Queue tràn trong trường hợp mô tả Queue bằng danh sách nối đơn kiểu FIFO.

```
program QueueByLinkedList; type
  PNode = ^TNode; {Kiểu con trỏ tới một nút của danh sách} TNode =
  record {Cấu trúc một nút của danh sách}
Value: Integer; Link: PNode; end;
var
  First, Last: PNode; {Hai con trỏ tới nút đầu và nút cuối của danh sách}

procedure QueueInit; {Khởi tạo Queue rỗng} begin
      First := nil; end;

procedure Push(V: Integer); {Đẩy giá trị V vào Queue} var
      P: PNode; begin
New(P); P^.Value := V; {Tạo ra một nút mới} P^.Link := nil; if First = nil then
First := P {Móc nút đó vào danh sách} else Last^.Link := P; Last := P; {Nút mới
trở thành nút cuối, cập nhật lại con trỏ Last} end;

function Pop: Integer; {Lấy giá trị khỏi Queue, trả về trong kết quả hàm} var
      P: PNode; begin
  if First = nil then WriteLn('Queue is empty') else
    begin
      Pop := First^.Value; {Gán kết quả hàm} P := First^.Link; {Giữ lại nút tiếp theo First^ (Nút được đẩy vào
      danh sách ngay sau First^)}
      Dispose(First); First := P; {Giải phóng bộ nhớ cấp cho First^, cập nhật lạ
      end;
```

begin
QueueInit; <Test>; {Đưa một vài lệnh để kiểm tra hoạt động của Queue} **end.**

Bài 1.

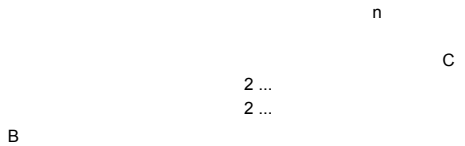
Bài tập

Tìm hiểu cơ chế xếp chồng của thủ tục đệ quy, phương pháp dùng ngăn xếp để khử đệ quy.

Viết chương trình mô tả cách đổi cơ số từ hệ thập phân sang hệ cơ số R dùng ngăn xếp

Bài 3

Hình 13 là cơ cấu đường tàu tại một ga xe lửa



Hình 13: Di chuyển toa tàu

Ban đầu ở đường ray A chứa các toa tàu đánh số từ 1 tới n theo thứ tự từ trái qua phải, người ta muốn chuyển các toa đó sang đường ray C để được một thứ tự mới là một hoán vị của (1, 2, ..., n) theo quy tắc: chỉ được đưa các toa tàu chạy theo đường ray theo hướng mũi tên, có thể dùng đoạn đường ray B để chứa tạm các toa tàu trong quá trình di chuyển.

a) Hãy nhập vào hoán vị cần có, cho biết có phương án chuyển hay không, và nếu có hãy đưa ra cách chuyển:

Ví dụ: n = 4; Thứ tự cần có (1, 4, 3, 2)

1)A → C; 2)A → B; 3)A → B; 4)A → C; 5)B → C; 6)B → C b) Những hoán vị nào của thứ tự các toa là có thể tạo thành trên đoạn đường ray C với luật di chuyển như trên

Bài 4

Tương tự như bài 3, nhưng với sơ đồ đường ray sau:

Hình 14: Di chuyển toa tàu (2)

§6. CÂY (TREE)

6.1. ĐỊNH NGHĨA

Cấu trúc dữ liệu trừu tượng ta quan tâm tới trong mục này là cấu trúc cây. Cây là một cấu trúc dữ liệu gồm một tập hữu hạn các nút, giữa các nút có một quan hệ phân cấp gọi là quan hệ "cha - con". Có một nút đặc biệt gọi là gốc (root).

Có thể định nghĩa cây bằng các đệ quy như sau:

- Mỗi nút là một cây, nút đó cũng là gốc của cây ấy
- Nếu n là một nút và n_1, n^2, ..., n^k lần lượt là gốc của các cây T^1, T^2, ..., T^k; các cây này đôi một không có nút chung. Thì nếu cho nút n trở thành cha của các nút n_1, n^2, ..., n_k ta sẽ được một cây mới T. Cây này có nút n là gốc còn các cây T^1, T^2, ..., T^k trở thành các cây con (subtree) của gốc.

Để tiện, người ta còn cho phép tồn tại một cây không có nút nào mà ta gọi là cây rỗng (null tree).

Xét cây trong Hình 15:

Hình 15: Cây

A là cha của B, C, D, còn G, H, I là con của D

J

K

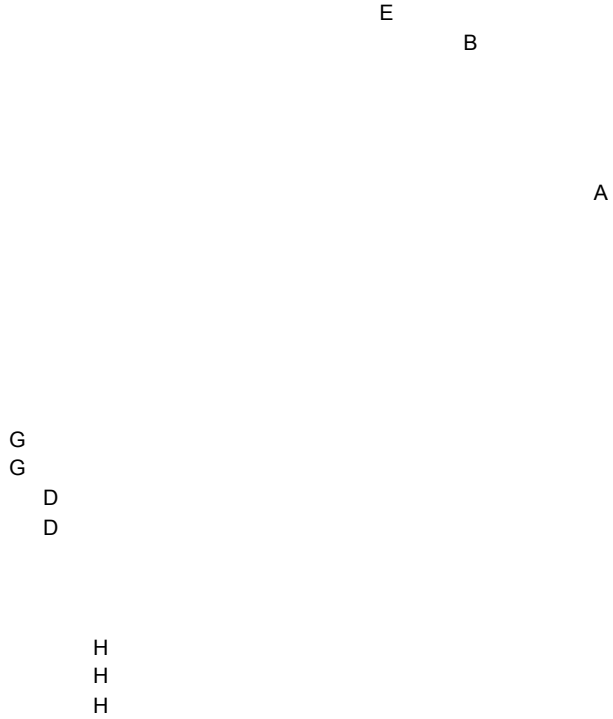
Số các con của một nút được gọi là **cấp của nút** đó, ví dụ cấp của A là 3, cấp của B là 2, cấp của C là 0.

Nút có cấp bằng 0 được gọi là **nút lá** (leaf) hay nút tận cùng. Ví dụ như ở trên, các nút E, F, C, G, J, K và I là các nút là. Những nút không phải là lá được gọi là **nút nhánh** (branch)

Cấp cao nhất của một nút trên cây gọi là **cấp của cây** đó, cây ở hình trên là cây cấp 3.

Gốc của cây người ta gán cho số mức là 1, nếu nút cha có mức là i thì nút con sẽ có mức là i +

1. Mức của cây trong Hình 15 được chỉ ra trong Hình 16:



1_{234}

E

B

A

G
G
G

J

D
D
D

H
H
H
H

K
K

I
I
I
I
I

Φ 65 φ

Hình 16: Mức của các nút trên cây

Chiều cao (height) hay **chiều sâu** (depth) của một cây là số mức lớn nhất của nút có trên cây đó. Cây ở trên có chiều cao là 4

Một tập hợp các cây phân biệt được gọi là **rừng** (forest), một cây cũng là một rừng. Nếu bỏ nút gốc trên cây thì sẽ tạo thành một rừng các cây con.

Ví dụ:

- Mục lục của một cuốn sách với phần, chương, bài, mục v.v... có cấu trúc của cây
- Cấu trúc thư mục trên đĩa cũng có cấu trúc cây, thư mục gốc có thể coi là gốc của cây đó với các cây con là các thư mục con và tệp nằm trên thư mục gốc.
- Gia phả của một họ tộc cũng có cấu trúc cây.
- Một biểu thức số học gồm các phép toán cộng, trừ, nhân, chia cũng có thể lưu trữ trong một cây mà các toán hạng được lưu trữ ở các nút lá, các toán tử được lưu trữ ở các nút nhánh, mỗi nhánh là một biểu thức con.

Hình 17: Cây biểu diễn biểu thú

A

+ C) * (D - E)

6.2. CÂY NHỊ PHÂN (BINARY TREE)

Cây nhị phân là một dạng quan trọng của cấu trúc cây. Nó có đặc điểm là mọi nút trên cây chỉ

có tối đa hai nhánh con. Với một nút thì người ta cũng phân biệt cây con trái và cây con phải của nút đó. Cây nhị phân là cây có tính đến thứ tự của các nhánh con.

Cần chú ý tới một số dạng đặc biệt của cây nhị phân

/ D

\+ -

C E

* E

E

Φ 66 φ

Chuyên đề

D

Các cây nhị phân trong Hình 18**Error! Reference source not found.** được gọi là **cây nhị phân suy biến** (degenerate binary tree), các nút không phải là lá chỉ có một nhánh con. Cây a) được gọi là cây lệch phải, cây b) được gọi là cây lệch trái, cây c) và d) được gọi là cây zíc-zắc.

5

5

5

5

5

a) b) c) d)

Hình 18: Các dạng cây nhị phân suy biến

Các cây trong Hình 19 được gọi là **cây nhị phân hoàn chỉnh** (complete binary tree): Nếu chiều cao của cây là h thì mọi nút có mức < h - 1 đều có đúng 2 nút con. Còn nếu mọi nút có mức ≤ h - 1 đều có đúng 2 nút con như trường hợp cây f) ở trên thì cây đó được gọi là **cây nhị phân đầy đủ** (full binary tree). Cây nhị phân đầy đủ là trường hợp riêng của cây nhị phân hoàn chỉnh.

1

1

e) f)

Hình 19: Cây nhị phân hoàn chỉnh và cây nhị phân đầy đủ

Ta có thể thấy ngay những tính chất sau bằng phép chứng minh quy nạp:

Trong các cây nhị phân có cùng số lượng nút như nhau thì cây nhị phân suy biến có chiều cao lớn nhất, còn cây nhị phân hoàn chỉnh thì có chiều cao nhỏ nhất.

Số lượng tối đa các nút trên mức i của cây nhị phân là 2^{i-1}, tối thiểu là 1 ($i \geq 1$).

Số lượng tối đa các nút trên một cây nhị phân có chiều cao h là 2^h-1, tối thiểu là h ($h \geq 1$).

Cây nhị phân hoàn chỉnh, không đầy đủ, có n nút thì chiều cao của nó là $h = [\log_2(n + 1)] + 1$.

1

2

3

4

4

2

2
2

4
4
4
4
1
1

3
3

Đại h

5
5
5
5
5
2
2

4


6
6
6
6
6
6
6
3
3
3
2
2
2
4
4
4
7
7
7
7
7
7
7
3
3
3
1
1
1