



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Thuật toán ứng dụng

Nguyễn Khánh Phương

Computer Science department
School of Information and Communication technology
E-mail: phuongnk@soict.hust.edu.vn

Nội dung khóa học

Chương 1. Các cấu trúc dữ liệu và thư viện

Chương 2. Kỹ thuật đệ quy và nhánh cận

Chương 3. Chia để trị

Chương 4. Thuật toán tham lam

Chương 5. Quy hoạch động

Chương 6. Các thuật toán trên đồ thị và ứng dụng

Chương 7. Các thuật toán xử lý xâu và ứng dụng

Chương 8. Lớp bài toán NP-đầy đủ

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - DHBKHN

Nội dung

1. Sơ đồ chung của thuật toán
2. Các ví dụ minh họa
 - 2.1. Dãy con lớn nhất
 - 2.2. Dãy con chung dài nhất
 - 2.3. Dãy con tăng dài nhất
 - 2.4. Bài toán đổi tiền
 - 2.5. Bài toán người du lịch
 - 2.6. Bài toán cái túi

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - DHBKHN

Nội dung

- 1. Sơ đồ chung của thuật toán**
2. Các ví dụ minh họa
 - 2.1. Dãy con lớn nhất
 - 2.2. Dãy con chung dài nhất
 - 2.3. Dãy con tăng dài nhất
 - 2.4. Bài toán đổi tiền
 - 2.5. Bài toán người du lịch
 - 2.6. Bài toán cái túi

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - DHBKHN

1. Sơ đồ chung của thuật toán quy hoạch động

Việc phát triển thuật toán dựa trên quy hoạch động bao gồm 3 giai đoạn:

- **Phân rã:**

- Chia bài toán cần giải thành những bài toán con nhỏ hơn có cùng dạng với bài toán ban đầu sao cho bài toán con kích thước nhỏ nhất có thể giải một cách trực tiếp.
- Bản thân bài toán xuất phát có thể coi là bài toán con có kích thước lớn nhất trong họ các bài toán con này.

- **Ghi nhận lời giải:**

- Lưu trữ lời giải của các bài toán con vào một bảng. Việc làm này là cần thiết vì lời giải của những bài toán con thường được sử dụng lại rất nhiều lần, và điều đó nâng cao hiệu quả của giải thuật do không phải giải lặp lại cùng một bài toán nhiều lần.

- **Tổng hợp lời giải:**

- Lần lượt từ lời giải của các bài toán con kích thước nhỏ hơn tìm cách xây dựng lời giải của bài toán kích thước lớn hơn, cho đến khi thu được lời giải của bài toán xuất phát (là bài toán con có kích thước lớn nhất).

5

1. Sơ đồ chung của thuật toán quy hoạch động

Có nhiều điểm tương đồng với phương pháp Chia để trị (Divide and Conquer):

- **Chia để trị:**

- Chia bài toán cha thành các bài toán con độc lập
- Giải từng bài toán con (bằng đệ qui)
- Kết hợp lời giải các bài toán con lại thành lời giải của bài toán cha.

- **Quy hoạch động:**

- Chia bài toán cha thành các bài toán con **gối nhau**
- Giải từng bài toán con (bằng đệ qui)
- Kết hợp lời giải các bài toán con lại thành lời giải của bài toán cha.
- Mỗi bài toán con chỉ giải duy nhất 1 lần bằng cách lưu lời giải vào 1 bảng và khi bài toán con đó được gọi lại thì chỉ cần tra bảng.

NGUYỄN KHÁNH PHƯƠNG 6
KHMT – SOICT - ĐHBKHN

1. Sơ đồ chung của thuật toán quy hoạch động

Kỹ thuật giải các bài toán con của quy hoạch động là quá trình đi từ dưới lên (bottom-up): bài toán con có kích thước nhỏ được giải trước, sau đó dùng lời giải những bài toán con này để xây dựng lời giải của bài toán lớn hơn;

Phương pháp chia để trị: bài toán lớn được chia nhỏ thành các bài toán con, và các bài toán con được trị một cách đệ quy (top-down).

NGUYỄN KHÁNH PHƯƠNG 7
KHMT – SOICT - ĐHBKHN

Công thức quy hoạch động

1. Tìm công thức quy hoạch động cho bài toán dựa trên các bài toán con
2. Cài đặt công thức quy hoạch động: chuyển công thức thành hàm đệ quy
3. Lưu trữ kết quả của các hàm tính toán

Nhận xét: Bước 1 là khó và quan trọng nhất. Bước 2 và 3 có thể áp dụng sơ đồ chung sau đây để thực hiện

```
map<problem, value> memory;

value DP(problem P) {
    if (is_base_case(P)) {
        return base_case_value(P);
    }

    if (memory.find(P) != memory.end()) {
        return memory[P];
    }

    value result = some value;
    for (problem Q in subproblems(P)) {
        result = combine(result, DP(Q));
    }

    memory[P] = result;
    return result;
}
```

NGUYỄN KHÁNH PHƯƠNG 8
KHMT – SOICT - ĐHBKHN

Ví dụ: Dãy Fibonacci

Hai số đầu tiên của dãy Fibonacci là 1 và 1. Các số còn lại của dãy được tính bằng tổng của hai số ngay trước nó trong dãy.

Yêu cầu: Tim số Fibonacci thứ n .

1. Tim công thức quy hoạch động:

$$fibonacci(1) = 1$$

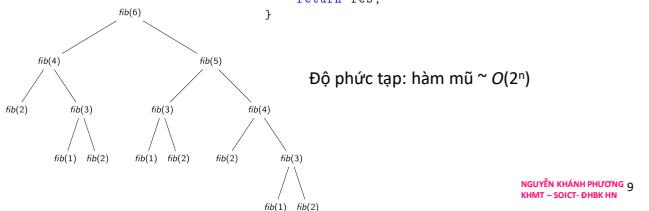
$$fibonacci(2) = 1$$

$$fibonacci(n) = fibonacci(n - 2) + fibonacci(n - 1)$$

2. Cài đặt công thức quy hoạch động:

```
Công thức quy hoạch động
1. Tim công thức quy hoạch động cho bài toán dựa trên các bài toán con
2. Cài đặt công thức quy hoạch động: chuyển công thức thành hàm đệ quy
3. Lưu trữ kết quả của các bài toán toán
```

```
int fibonacci(int n) {
    if (n <= 2) return 1;
    int res = fibonacci(n - 2) + fibonacci(n - 1);
    return res;
}
```



Độ phức tạp: hàm mũ $\sim O(2^n)$

NGUYỄN KHÁNH PHƯƠNG 9
KHMT – SOICT- ĐHBK HN

Ví dụ: Dãy Fibonacci

3. Lưu trữ kết quả của các bài toán toán

```
map<int, int> mem;
int fibonacci(int n) {
    if (n <= 2) return 1;
    if (mem.find(n) != mem.end()) return mem[n];
    int res = fibonacci(n - 2) + fibonacci(n - 1);
    mem[n] = res;
    return res;
}
```

```
int mem[1000];
for (int i = 0; i < 1000; i++) mem[i] = -1;
int fibonacci(int n) {
    if (n <= 2) return 1;
    if (mem[n] != -1) return mem[n];
    int res = fibonacci(n - 2) + fibonacci(n - 1);
    mem[n] = res;
    return res;
}
```

NGUYỄN KHÁNH PHƯƠNG 10
KHMT – SOICT- ĐHBK HN

Ví dụ: Dãy Fibonacci

```
int mem[1000];
for (int i = 0; i < 1000; i++) mem[i] = -1;
int fibonacci(int n) {
    if (n <= 2) return 1;
    if (mem[n] != -1) return mem[n];
    int res = fibonacci(n - 2) + fibonacci(n - 1);
    mem[n] = res;
    return res;
}
```

Độ phức tạp tính toán lúc này bằng bao nhiêu?

Khi gọi `Fibonacci (n)`: có n khả năng input cho hàm: $1, 2, \dots, n$

Với mỗi input:

- Hoặc là kết quả được lấy luôn từ bộ nhớ nếu như trước đây đã từng được tính: $O(1)$ nếu giả thiết lấy trực tiếp từ bộ nhớ chỉ mất $O(1)$
- Hoặc là kết quả được tính và lưu lại: $O(1)$ nếu giả thiết mỗi phép gọi đệ quy và tính tổng chỉ tốn một lượng hằng số

Mỗi input sẽ được tính tối đa 1 lần

Thời gian tính tổng cộng là $O(n)$

NGUYỄN KHÁNH PHƯƠNG 11
KHMT – SOICT- ĐHBK HN

1. Sơ đồ chung của thuật toán quy hoạch động

Để việc áp dụng quy hoạch động dẫn đến thuật toán hiệu quả, bài toán cần có 2 tính chất:

- Cấu trúc con tối ưu:** (còn được gọi là tiêu chuẩn tối ưu) Để giải được bài toán đặt ra một cách tối ưu, mỗi bài toán con cũng phải được giải một cách tối ưu. Mặc dù sự kiện này có vẻ là hiển nhiên, nhưng nó thường không được thỏa mãn do các bài toán con là giao nhau.
- Số lượng các bài toán con phải không quá lớn.** Chính xác hơn là tổng số các bài toán con cần giải cùng lầm phải bị chặn bởi một đa thức của kích thước dữ liệu vào.

NGUYỄN KHÁNH PHƯƠNG 12
KHMT – SOICT- ĐHBK HN

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Dãy con lớn nhất

- 2.2. Dãy con chung dài nhất
- 2.3. Dãy con tăng dài nhất
- 2.4. Bài toán đổi tiền
- 2.5. Bài toán người du lịch
- 2.6. Bài toán cái túi

NGUYỄN KHÁNH PHƯƠNG 13
KHMT – SOICT- ĐHBKHN

Dãy con lớn nhất (The maximum subarray problem)

- Cho dãy số gồm n số:

$$a_1, a_2, \dots, a_n$$

Dãy gồm liên tiếp các số a_i, a_{i+1}, \dots, a_j với $1 \leq i \leq j \leq n$ được gọi là **dãy con** của dãy đã cho và $\sum_{k=i}^j a_k$ được gọi là **trọng lượng** của dãy con này

Bài toán đặt ra là: Hãy tìm trọng lượng lớn nhất của các dãy con, tức là tìm cực đại giá trị $\sum_{k=i}^j a_k$. Ta gọi dãy con có trọng lượng lớn nhất là **dãy con lớn nhất**.

Ví dụ: Cho dãy số -2, 11, -4, 13, -5, 2 thì cần đưa ra câu trả lời là 20 (dãy con lớn nhất là 11, -4, 13 với giá trị = $11 + (-4) + 13 = 20$)

NGUYỄN KHÁNH PHƯƠNG 14
KHMT – SOICT- ĐHBKHN

Dãy con lớn nhất (The maximum subarray problem)

1. Duyệt toàn bộ (Brute force)

$$\frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3}$$

2. Duyệt toàn bộ có cải tiến

$$\frac{n^2}{2} + \frac{n}{2}$$

3. Thuật toán đệ quy (Recursive algorithm)

$$n \log n$$

4. Thuật toán Quy hoạch động (Dynamic programming)

$$n$$

NGUYỄN KHÁNH PHƯƠNG 15
KHMT – SOICT- ĐHBKHN

Thuật toán quy hoạch động giải bài toán dãy con lớn nhất

Công thức quy hoạch động

1. Tìm công thức quy hoạch động cho bài toán dựa trên các bài toán con
2. Cải đặt công thức quy hoạch động: chuyển công thức thành hàm đệ quy
3. Lưu trữ kết quả của các hàm tính toán

Bước 1. Tìm công thức quy hoạch động

- Gọi $\max_sum(i)$ là trọng lượng của dãy con lớn nhất của dãy $a_1, a_2, \dots, a_i, i = 1, \dots, n$.
- Neo đề quy: $\max_sum(1) = \max(0, a[1])$
- Xây dựng công thức đệ quy tính $\max_sum(i) = ?$
 - Liên hệ gì với $\max_sum(i-1)$
 - Liệu có thể xây dựng lời giải của bài toán kích thước i từ lời giải của các bài toán nhỏ hơn i ?



Gọi $\max_sum(i)$ là trọng lượng của dãy con lớn nhất của dãy a_1, a_2, \dots, a_i , và **dãy con này kết thúc tại a_i**

NGUYỄN KHÁNH PHƯƠNG 16
KHMT – SOICT- ĐHBKHN

Thuật toán quy hoạch động giải bài toán dãy con lớn nhất

Công thức quy hoạch động

1. Tìm công thức quy hoạch động cho bài toán dựa trên các bài toán con
2. Cài đặt công thức quy hoạch động: chuyển công thức thành hàm đệ quy
3. Lưu trữ kết quả của các hàm tính toán

Bước 1. Tìm công thức quy hoạch động

- Gọi $\max_sum(i)$ là trọng lượng của dãy con lớn nhất của dãy a_1, a_2, \dots, a_i mà **dãy con này kết thúc tại a_i**
 - Neo đệ quy: $\max_sum(1) = a[1]$
 - Công thức đệ quy tính $\max_sum(i)$:
- $$\max_sum(i) = \max(a[i], a[i] + \max_sum(i-1))$$
- ⇒ Lời giải của bài toán: $\max_{1 \leq i \leq n} \max_sum(i)$

NGUYỄN KHÁNH PHƯƠNG 17
KHMT – SOICT - ĐHBK HN

Thuật toán quy hoạch động giải bài toán dãy con lớn nhất

Công thức quy hoạch động

1. Tìm công thức quy hoạch động cho bài toán dựa trên các bài toán con
2. Cài đặt công thức quy hoạch động: chuyển công thức thành hàm đệ quy
3. Lưu trữ kết quả của các hàm tính toán

Bước 2. Cài đặt công thức quy hoạch động

Neo đệ quy: $\max_sum(1) = a[1]$
 $\max_sum(i) = \max(a[i], a[i] + \max_sum(i-1))$

```
int a[1000];
int max_sum(int i) {
    if (i == 1) return a[i];
    int res = max(a[i], a[i] + max_sum(i - 1));
    return res;
}
```

NGUYỄN KHÁNH PHƯƠNG 18
KHMT – SOICT - ĐHBK HN

Thuật toán quy hoạch động giải bài toán dãy con lớn nhất

Công thức quy hoạch động

1. Tìm công thức quy hoạch động cho bài toán dựa trên các bài toán con
2. Cài đặt công thức quy hoạch động: chuyển công thức thành hàm đệ quy
3. Lưu trữ kết quả của các hàm tính toán

Bước 3. Lưu trữ kết quả của các hàm tính toán

```
int a[1000];
int mem[1000];
bool comp[1000];
memset(comp, 0, sizeof(comp));

int max_sum(int i) {
    if (i == 1) return a[i];
    if (comp[i]) return mem[i];
    int res = max(a[i], a[i] + max_sum(i - 1));
    mem[i] = res;
    comp[i] = true;
    return res;
}
```

NGUYỄN KHÁNH PHƯƠNG 19
KHMT – SOICT - ĐHBK HN

Dãy con lớn nhất (The maximum subarray problem)

Trong thủ tục chính, chỉ cần gọi $\max_sum(n)$; hàm này sẽ tính toàn bộ các giá trị của $\max_sum(i)$ với $1 \leq i \leq n$

Khi đó, kết quả của bài toán là giá trị lớn nhất trong các giá trị $\max_sum(i)$ đã được lưu trong mảng $mem[i]$

```
int maximum = 0;
for (int i = 1; i <= n; i++) {
    maximum = max(maximum, mem[i]);
}
maximum = *max_element(mem+1, mem+n+1);
printf("%d\n", maximum);
```

NGUYỄN KHÁNH PHƯƠNG 20
KHMT – SOICT - ĐHBK HN

Thuật toán quy hoạch động giải bài toán dãy con lớn nhất

```
int a[1000];
int mem[1000];
bool comp[1000];
memset(comp, 0, sizeof(comp));

int max_sum(int i) {
    if (i == 1) return a[i];
    if (comp[i]) return mem[i];
    int res = max(a[i], a[i] + max_sum(i - 1));
    mem[i] = res;
    comp[i] = true;
    return res;
}
```

Độ phức tạp ?

Khi gọi $\max_sum(n)$: có n khả năng input cho hàm: $1, 2, \dots, n$

Với mỗi input:

- hoặc là kết quả được lấy luôn từ bộ nhớ nếu như trước đây đã từng được tính: $O(1)$ nếu giả thiết lấy trực tiếp từ bộ nhớ chỉ mất $O(1)$
- hoặc là kết quả được tính và lưu lại: $O(1)$ nếu giả thiết mỗi phép gọi đệ quy và tính max tồn tại lượng hằng số

Mỗi input sẽ được tính tối đa 1 lần

Thời gian tính tổng cộng là $O(n)$

NGUYỄN KHÁNH PHƯƠNG 21
KHMT – SOICT- ĐHQGHN

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

 2.1. Dãy con lớn nhất

2.2. Dãy con chung dài nhất

 2.3. Dãy con tăng dài nhất

 2.4. Bài toán đổi tiền

 2.5. Bài toán người du lịch

 2.6. Bài toán cái túi

NGUYỄN KHÁNH PHƯƠNG 23
KHMT – SOICT- ĐHQGHN

Dãy con lớn nhất: Truy vết bằng đệ quy

```
int a[1000];
int mem[1000];
bool comp[1000];
memset(comp, 0, sizeof(comp));

int max_sum(int i) {
    if (i == 1) return a[i];
    if (comp[i]) return mem[i];
    int res = max(a[i], a[i] + max_sum(i - 1));
    mem[i] = res;
    comp[i] = true;
    return res;
}
```

Làm thế nào để biết dãy con có trọng lượng lớn nhất gồm những phần tử nào?

- Cách 1: Truy vết bằng đệ quy

```
void trace(int i) {
    if (i != 1 && mem[i] == a[i] + mem[i-1])
        trace(i - 1);
    printf("%d ", a[i]);
}
```

- Cách 2: Truy vết bằng vòng lặp

```
int maximum = 0, pos = -1;
for (int i = 1; i <= n; i++) {
    maximum = max(maximum, mem[i]);
    if (maximum == mem[i]) pos = i;
}
printf("%d\n", maximum);
int L = pos, R = pos, sum = a[L];
while (sum != maximum){
    --L;
    sum += a[L];
}
printf("%d %d", L, R);
```

Dãy con chung dài nhất (Longest common subsequence - LCS)

Cho dãy gồm n phần tử $A = \langle a_0, a_1, \dots, a_{n-1} \rangle$

Định nghĩa dãy con: nếu xóa đi 0 phần tử hoặc một số phần tử của mảng A thì sẽ thu được một dãy con của A .

- Ví dụ: $A = [5, 1, 8, 1, 9, 2]$

– Một số dãy con của A :

- [5, 1, 8, 1, 9, 2]
- [5, 8, 9]
- [1, 1]
- []

– Không phải dãy con của A :

- [2, 5]
- [10]

Cho hai dãy A và B , ta nói dãy Z là **dãy con chung** của hai dãy A và B nếu Z là dãy con của cả hai dãy này.

Bài toán LCS: Cho hai dãy

$$A = \langle a_0, a_1, \dots, a_{m-1} \rangle$$

$$B = \langle b_0, b_1, \dots, b_{n-1} \rangle$$

Hãy tìm dãy con chung dài nhất của A và B .

Ứng dụng: Trong sinh học, khi khảo sát gen

NGUYỄN KHÁNH PHƯƠNG 24
KHMT – SOICT- ĐHQGHN

Dãy con chung dài nhất (Longest common subsequence - LCS)

Ví dụ:

$$A = \langle K, J, C, D, E, F, G \rangle$$

$$B = \langle C, C, E, D, E, G, F \rangle$$

- Dãy $Z = \langle C, D, F \rangle$ là dãy con chung.
- Dãy $\langle B, F, G \rangle$ không là dãy con chung.
- Dãy $\langle C, D, E, G \rangle$ là dãy con chung dài nhất vì không tìm được dãy con chung có độ dài 5.

NGUYỄN KHÁNH PHƯƠNG 25
KHMT – SOICT-ĐHQGHN

Dãy con chung dài nhất (Longest common subsequence - LCS)

Bài toán LCS: Cho hai dãy

$$A = \langle a_0, a_1, \dots, a_{m-1} \rangle,$$

$$B = \langle b_0, b_1, \dots, b_{n-1} \rangle.$$

Hãy tìm dãy con chung dài nhất của A và B .

• Thuật toán trực tiếp:

- Duyệt tất cả các dãy con có thể của $A \rightarrow$ số dãy con = $(2^m$ dãy con)
- Với mỗi dãy con của A ta kiểm tra xem nó có là dãy con của B . Thời gian: $O(n)$
- \rightarrow Thời gian của thuật toán trực tiếp: $O(n.2^m)$

• Quy hoạch động:

Bước 1. Tìm công thức quy hoạch động

- Gọi $lcs(i, j)$ là độ dài dãy con chung dài nhất của $A_i = \langle a_0, a_1, \dots, a_i \rangle$ và $B_j = \langle b_0, b_1, \dots, b_j \rangle$ với $-1 \leq i \leq m-1$ và $-1 \leq j \leq n-1$
- Neo đê quy: $lsc(i, -1) = 0$ và $lsc(-1, j) = 0$
- Công thức đê quy tính $lcs(i, j)$?

26

Dãy con chung dài nhất: Thuật toán quy hoạch động

• Rõ ràng

$$lcs(i, -1) = 0, \quad i = -1, 0, 1, \dots, m-1$$

$$lcs(-1, j) = 0, \quad j = -1, 0, 1, \dots, n-1.$$

• Giả sử $i \geq 0, j \geq 0$, ta cần tính $lcs(i, j)$ là độ dài của LCS của hai dãy $A_i = \langle a_0, a_1, \dots, a_i \rangle$ và $B_j = \langle b_0, b_1, \dots, b_j \rangle$. Có hai tình huống:

– Nếu $a_i = b_j$:

- LCS của A_i và B_j sẽ thu được bằng việc bổ sung a_i vào LCS của hai dãy A_{i-1} và $B_{j-1} \Rightarrow lcs(i, j) = lcs(i-1, j-1) + 1$

– Nếu $a_i \neq b_j$:

- LCS của A_i và B_j sẽ là dãy con dài nhất trong hai LCS của $(A_i \text{ và } B_{j-1})$ và $(A_{i-1} \text{ và } B_j) \Rightarrow lcs(i, j) = \max\{lcs(i, j-1), lcs(i-1, j)\}$

• Từ đó ta có công thức sau để tính $lcs(i, j)$:

$$lcs(i, j) = \begin{cases} 0, & \text{nếu } i = -1 \text{ hoặc } j = -1, \\ lcs(i-1, j-1) + 1, & \text{nếu } i, j \geq 0 \text{ và } a_i = b_j \\ \max\{lcs(i, j-1), lcs(i-1, j)\}, & \text{nếu } i, j \geq 0 \text{ và } a_i \neq b_j. \end{cases}$$

27

Thuật toán quy hoạch động: cài đặt đệ quy

$$lcs(i, j) = \begin{cases} 0, & \text{nếu } i = -1 \text{ hoặc } j = -1, \\ lcs(i-1, j-1) + 1, & \text{nếu } i, j \geq 0 \text{ và } a_i = b_j \\ \max\{lcs(i, j-1), lcs(i-1, j)\}, & \text{nếu } i, j \geq 0 \text{ và } a_i \neq b_j. \end{cases}$$

```
string a = "bananinn", b = "kaninan";
int mem[1000][1000];
memset(mem, -1, sizeof(mem));
int lcs(int i, int j) {
    if (i == -1 || j == -1) return 0;
    if (mem[i][j] != -1) return mem[i][j];
    int res = 0;
    if (a[i] == b[j]) {
        res = max(res, 1 + lcs(i - 1, j - 1));
    } else {
        res = max(res, lcs(i, j - 1));
        res = max(res, lcs(i - 1, j));
    }
    mem[i][j] = res;
    return res;
}
a = "bananinn"
b = "kaninan"
LCS = "aninn"
```

NGUYỄN KHÁNH PHƯƠNG 28
KHMT – SOICT-ĐHQGHN

Dãy con chung dài nhất: Thuật toán quy hoạch động

Trong thủ tục chính, chỉ cần gọi `lcs(a.length()-1, b.length()-1);`

```
int main()
{
    int m = a.length();
    int n = b.length();
    int answer = lcs(m-1,n-1);
    printf ("%d \n", answer);
    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG 29
KHMT – SOICT – ĐHQGHN

Dãy con chung dài nhất: Thuật toán quy hoạch động

```
string a = "bananinn", b = "kaninan";
int mem[1000][1000];
memset(mem, -1, sizeof(mem));

int lcs( int i , int j ) {
    if (i == -1 || j == -1) return 0;
    if (mem[i][j] != -1) return mem[i][j];

    int res = 0;
    if (a[i] == b[j]) {
        res = max(res , 1 + lcs(i - 1 , j - 1));
    }
    else {
        res = max(res , lcs(i - 1 , j));
        res = max(res , lcs(i , j - 1));
    }
    mem[i][j] = res;
    return res;
}
```

Độ phức tạp ?

Khi gọi `lcs(len(a)-1, len(b)-1)`: có $m*n$ khả năng input cho hàm

Với mỗi input:

- hoặc là kết quả được lấy luôn từ bộ nhớ nếu như trước đây đã từng được tính: $O(1)$ nếu giả thiết lấy trục tiếp từ bộ nhớ chỉ mất $O(1)$
- hoặc là kết quả được tính và lưu lại: $O(1)$ nếu giả thiết mỗi lần gọi đệ quy chỉ mất $O(1)$

Mỗi input sẽ được tính tối đa 1 lần

Thời gian tính tổng cộng là $O(m*n)$

30

LCS: Cài đặt không đệ quy

$$lcs(i, j) = \begin{cases} 0, & \text{nếu } i = 0 \text{ hoặc } j = 0, \\ lcs(i-1, j-1)+1, & \text{nếu } i, j > 0 \text{ và } a_i = b_j, \\ \max\{lcs(i, j-1), lcs(i-1, j)\}, & \text{nếu } i, j > 0 \text{ và } a_i \neq b_j. \end{cases}$$

```
string a = "bananinn", b = "kaninan";
int mem[1000][1000];
int lcs(string a, string b) {
    int m=a.length();
    int n=b.length();

    for (int i=0; i<m; i++) mem[i][0]=0;
    for (int j = 0; j<n; j++) mem[0][j] = 0;
    for (int i = 1; i<m; i++)
        for (int j=i; j<n; j++)
            if (a[i-1] == b[j-1]) mem[i][j] = mem[i-1][j-1]+1;
            else
                mem[i][j] = max (mem[i][j-1], mem[i-1][j]);
    return mem[m][n];
}

int main(){
    memset (mem , -1 , sizeof (mem));
    int k = lcs(a, b);
    cout<<k<<endl;
    return 0;
}
```

Câu hỏi: làm thế nào để đưa ra được dãy con chung dài nhất gồm những phần tử nào ?

$O(m*n)$

Dãy con chung dài nhất: Truy vết bằng đệ quy

Làm thế nào để đưa ra được dãy con chung dài nhất gồm những phần tử nào ?

$$lcs(i, j) = \begin{cases} 0, & \text{nếu } i = -1 \text{ hoặc } j = -1, \\ lcs(i-1, j-1)+1, & \text{nếu } i, j \geq 0 \text{ và } a_i = b_j, \\ \max\{lcs(i, j-1), lcs(i-1, j)\}, & \text{nếu } i, j \geq 0 \text{ và } a_i \neq b_j. \end{cases}$$

Cách 1: Truy vết bằng đệ quy

```
void trace(int i , int j) {
    if (i == -1 || j == -1) return;
    if (a[i] == b[j]) {
        trace(i-1, j);
        return;
    }
    if (mem[i][j] == mem[i][j-1]){
        trace(i, j-1);
        return;
    }
    if (a[i] == b[j] && mem[i][j] == 1 + mem[i-1][j-1]){
        trace(i-1, j-1);
        printf("%c", a[i]);
        return;
    }
}
```

```
int lcs( int i , int j ){
    if (i == -1 || j == -1) return 0;
    if (mem[i][j] != -1) return mem[i][j];

    int res = 0;
    if (a[i] == b[j]) {
        res = max(res , 1 + lcs(i - 1 , j - 1));
    }
    else {
        res = max(res , lcs(i - 1 , j));
        res = max(res , lcs(i , j - 1));
    }
    mem[i][j] = res;
    return res;
}
```

32

Dãy con chung dài nhất: Truy vết bằng đệ quy

Cách 1: Truy vết bằng đệ quy

```

int lcs( int i , int j ) {
    if ( i == -1 || j == -1) return 0;
    if ( mem[i][j] != -1) return mem[i][j];

    int res = 0;
    if ( a[i] == b[j] )
        res = max(res , 1 + lcs(i - 1 , j - 1));
    else
    {
        res = max(res , lcs( i , j - 1));
        res = max(res , lcs( i - 1 , j));
    }
    mem [i][j] = res;
    return res;
}

string a = "bananinn", b = "kaninan";
int mem [1000][1000];

```

5
Xau con chung dai nhat: aninn

NGUYỄN KHÁNH PHƯƠNG 33
KHMT – SOICT- ĐH BKHN

Dãy con chung dài nhất: Truy vết bằng vòng lặp

Làm thế nào để đưa ra được dãy con chung dài nhất gồm những phần tử nào ?

```

int lcs( int i , int j ) {
    if ( i == -1 || j == -1) return 0;
    if ( mem[i][j] != -1) return mem[i][j];

    int res = 0;
    if ( a[i] == b[j] )
        res = max(res , 1 + lcs(i - 1 , j - 1));
    else
    {
        res = max(res , lcs( i , j - 1));
        res = max(res , lcs( i - 1 , j));
    }
    mem [i][j] = res;
    return res;
}

```

Cách 2: Truy vết bằng vòng lặp

```

int answer = lcs(m-1, n-1);
printf("%d\n", answer);
stack<int> s;
for (int i = m-1, j = n-1, k = 0; k < answer; ++k) {
    if (a[i] == b[j] && mem[i][j] == 1 + mem[i-1][j-1]){
        s.push(a[i]);
        --i;
        --j; continue;
    }
    if (mem[i][j] == mem[i-1][j]){
        --i; continue;
    }
    if (mem[i][j] == mem[i][j-1]){
        --j; continue;
    }
}
while (!s.empty())
{
    printf("%c", s.back());
    s.pop();
}

```

NGUYỄN KHÁNH PHƯƠNG 34
KHMT – SOICT- ĐH BKHN

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Dãy con lớn nhất

2.2. Dãy con chung dài nhất

2.3. Dãy con tăng dài nhất

2.4. Bài toán đổi tiền

2.5. Bài toán người du lịch

2.6. Bài toán cái túi

NGUYỄN KHÁNH PHƯƠNG 35
KHMT – SOICT- ĐH BKHN

Dãy con tăng dài nhất

Phát biểu bài toán: Cho một mảng n số nguyên: $a[0], a[1], \dots, a[n-1]$. Hãy tìm độ dài của dãy con tăng dài nhất.

Định nghĩa dãy con: nếu xóa đi 0 phần tử hoặc một số phần tử của mảng a thì sẽ thu được một dãy con của a .

- Ví dụ: $A = [5, 1, 8, 1, 9, 2]$

Một số dãy con của A :

- [5, 1, 8, 1, 9, 2]
- [5, 8, 9]
- [1, 1]
- []

Không phải dãy con của A :

- [2, 5]
- [10]

NGUYỄN KHÁNH PHƯƠNG 36
KHMT – SOICT- ĐH BKHN

Dãy con tăng dài nhất

Phát biểu bài toán: Cho một mảng n số nguyên: $a[0], a[1], \dots, a[n-1]$. Hãy tìm độ dài của dãy con tăng dài nhất.

Định nghĩa dãy con: nếu xóa đi 0 phần tử hoặc một số phần tử của mảng a thì sẽ thu được một dãy con của a .

Định nghĩa dãy con tăng của mảng a : là một dãy con của a sao cho các phần tử là tăng chất từ trái sang phải.

Ví dụ: $[5, 8, 9]$ và $[1, 8, 9]$ là hai dãy con tăng của mảng $a = [5, 1, 8, 1, 9, 2]$

NGUYỄN KHÁNH PHƯƠNG 37
KHMT – SOICT - ĐHQGHN

Dãy con tăng dài nhất

Phát biểu bài toán: Cho một mảng n số nguyên: $a[0], a[1], \dots, a[n-1]$. Hãy tìm độ dài của dãy con tăng dài nhất.

- **Duyệt toàn bộ:** có 2^n dãy con

→ độ phức tạp $O(n^2^n)$, chỉ có thể chạy nhanh được ra kết quả với $n \leq 23$.

- **Quy hoạch động:**

Bước 1. Tìm công thức quy hoạch động

– Gọi $\text{lis}(i)$ là độ dài dãy con tăng dài nhất của mảng $a[0], a[1], \dots, a[i]$

– Neo đề quy: $\text{list}(0) = 1$

– Công thức đề quy tính $\text{list}(i)$?



– Gọi $\text{lis}(i)$ là độ dài dãy con tăng dài nhất của mảng $a[0], a[1], \dots, a[i]$ mà kết thúc tại $a[i]$

– Neo đề quy: không cần thiết

– Công thức đề quy tính $\text{list}(i) = \max(1, \max_{j \text{ s.t. } a[j] < a[i]} \{1 + \text{lis}(j)\})$

Dãy con tăng dài nhất: Thuật toán quy hoạch động

Phát biểu bài toán: Cho một mảng n số nguyên: $a[0], a[1], \dots, a[n-1]$. Hãy tìm độ dài của dãy con tăng dài nhất.

• Quy hoạch động:

Gọi $\text{lis}(i)$ là độ dài dãy con tăng dài nhất của mảng $a[0], a[1], \dots, a[i]$ mà kết thúc tại $a[i]$

$\text{list}(i) = \max(1, \max_{j \text{ s.t. } a[j] < a[i]} \{1 + \text{lis}(j)\})$

```
int a[1000];
int mem[1000];
memset(mem, -1, sizeof(mem));

int lis(int i) {
    if (mem[i] != -1) return mem[i];
    int res = 1;
    for (int j = 0; j < i; j++) {
        if (a[j] < a[i]) res = max(res, 1 + lis(j));
    }
    mem[i] = res;
    return res;
}
```

39

Dãy con tăng dài nhất: Thuật toán quy hoạch động

Trong thủ tục chính, chỉ cần gọi $\text{lis}(n-1)$; hàm này sẽ tính toàn bộ các giá trị của $\text{list}(i)$ với $0 \leq i \leq n-1$

Khi đó, kết quả của bài toán là giá trị lớn nhất trong các giá trị $\text{lis}(i)$ đã được lưu trong mảng $\text{mem}[i]$

```
int maximum = 0;
for (int i = 0; i < n; i++) {
    maximum = max(maximum, mem[i]);
}
maximum = *max_element(mem, mem+n);

printf("%d\n", maximum);
```

NGUYỄN KHÁNH PHƯƠNG 40
KHMT – SOICT - ĐHQGHN

Dãy con tăng dài nhất: Thuật toán quy hoạch động

```

int a[1000];
int mem[1000];
memset(mem, -1, sizeof(mem));

int lis(int i) {
    if (mem[i] != -1) return mem[i];
    int res = 1;
    for (int j = 0; j < i; j++) {
        if (a[j] < a[i]) res = max(res, 1 + lis(j));
    }
    mem[i] = res;
    return res;
}

```

Độ phức tạp?

Khi gọi `list(n-1)`: có n khả năng input cho hàm: $0, 1, \dots, n-1$

Với mỗi input:

- hoặc là kết quả được lấy luôn từ bộ nhớ nếu như trước đây đã từng được tính: $O(1)$ nếu giả thiết lấy trực tiếp từ bộ nhớ chỉ mất $O(1)$
- hoặc là kết quả được tính và lưu lại: $O(n)$

Mỗi input sẽ được tính tối đa 1 lần

Thời gian tính tổng cộng là $O(n^2)$

➔ Có thể chạy được đến $n \leq 10\,000$, tốt hơn rất nhiều so với duyệt toàn bộ

41

Dãy con tăng dài nhất: Truy vết bằng đệ quy

```

int a[1000];
int mem[1000];
memset(mem, -1, sizeof(mem));

int lis(int i) {
    if (mem[i] != -1) return mem[i];
    int res = 1;
    for (int j = 0; j < i; j++) {
        if (a[j] < a[i]) res = max(res, 1 + lis(j));
    }
    mem[i] = res;
    return res;
}

```

Làm thế nào để biết dãy con tăng dài nhất gồm những phần tử nào?

- Cách 1: Truy vết bằng đệ quy

```

void trace(int i) {
    int res = 1;
    for (int j = 0; j < i; j++) {
        if (a[j] < a[i] && mem[i] == 1 + mem[j]) {
            trace(j);
            break;
        }
    }
    printf("%d ", i);
}

```

42

Dãy con tăng dài nhất: Truy vết bằng vòng lặp

Làm thế nào để biết dãy con tăng dài nhất gồm những phần tử nào?

- Cách 2: Truy vết bằng vòng lặp

```

for (int i = 0; i < n; i++) {
    mx = max(mx, mem[i]);
    if (mx == mem[i]) pos = i;
}

printf("%d\n", mx);
stack<int> s;
for (int i = pos, k = 0; k < mx; ++k) {
    s.push(i);
    for (int j = 0; j < i; ++j){
        if (a[j] < a[i] && mem[j]+1 == mem[i]){
            i = j;
            break;
        }
    }
}
while (!s.empty()){
    printf("%d ", s.back());
    s.pop();
}

```

43

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Dãy con lớn nhất

2.2. Dãy con chung dài nhất

2.3. Dãy con tăng dài nhất

2.4. Bài toán đổi tiền

2.5. Bài toán người du lịch

2.6. Bài toán cái túi

Bài toán đổi tiền

Cho các mệnh giá tiền xu: 1, 5, 10, 25, 100 cent.

Yêu cầu: hãy tìm cách trả X cent tiền thừa cho khách sao cho số lượng đồng xu đưa khách là ít nhất.



NGUYỄN KHÁNH PHƯƠNG 45
KHMT – SOICT - ĐHBKHN

Bài toán đổi tiền

Thuật toán tham lam:

- Ở mỗi bước lắp: thêm đồng tiền với mệnh giá lớn nhất mà không vượt quá số tiền phải trả



Ví dụ: Trả khách hàng số tiền X = 289 cent

46

Bài toán đổi tiền

Thuật toán tham lam Cashiers:

- Ở mỗi bước lắp: thêm đồng tiền với mệnh giá lớn nhất mà không vượt quá số tiền phải trả

procedure CASHIERS(x , c_1 , c_2 , ..., c_n)

//đổi X cent dùng các mệnh giá $c_1 < c_2 < \dots < c_r$

begin

```

begin
  S ← Ø //tập chứa các đồng xu đem trả cho khách
  while ( X > 0)
    begin
      k ← đồng xu có mệnh giá lớn nhất  $c_k$  sao cho  $c_k \leq X$ 
      if (không tìm được k) return "no solution"
      else
        X ← X -  $c_k$ 
        S ← S ∪ {k}
    end;
  end;

```

Thuật toán Cashier có phải tối ưu?

47

Bài toán đổi tiền: Phân tích thuật toán tham lam

Câu hỏi. Thuật toán tham lam Cashier có thể áp dụng cho bất kỳ tập ménh gián傲?

Trả lời: KHÔNG

- Hàm số: KHÔNG

 - Thuật toán tham lam không cho lời giải tối ưu trên tập mệnh giá 1, 10, 21, 34, 70, 100, 350, 1225, 1500.
 - Ví dụ: đổi 140 cent
 - Thuật toán tham lam cho lời giải $140 = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1$
 - Tối ưu: $140 = 70 + 70$
 - Thuật toán tham lam thậm chí không tìm được lời giải nếu $c_1 > 1$
 - Ví dụ: tập 3 mệnh giá 7, 8, 9 cent. Cần đổi 15 cent
 - Tham lam: $15 = 9 + ???$
 - Tối ưu: $15 = 7 + 8$

NGUYỄN KHÁNH PHƯƠNG 48

Bài toán đổi tiền

Phát biểu bài toán: Cho trước một tập n các đồng mệnh giá d_0, d_1, \dots, d_{n-1} và một mệnh giá X . Hãy tìm số lượng ít nhất các đồng tiền để đổi cho mệnh giá X .

- Thuật toán tham lam không hề chắc chắn đưa ra lời giải tối ưu, thậm chí trong nhiều trường hợp còn không đưa ra được lời giải
- Có thể sử dụng phương pháp Quy hoạch động?

NGUYỄN KHÁNH PHƯƠNG 49
KHMT – SOICT - DHBKHN

Bài toán đổi tiền: Quy hoạch động

Bước 1: Xây dựng công thức quy hoạch động

- Gọi $opt(i, X)$ là số lượng đồng tiền ít nhất cần để đổi mệnh giá X nếu chỉ được sử dụng các đồng mệnh giá d_0, d_1, \dots, d_i

• Neo đệ quy:

$$\begin{aligned} opt(i, 0) &= 0 \\ opt(-1, X) &= +\infty \end{aligned}$$

• Công thức đệ quy tính $opt(i, X) = ???$

$$opt(i, X) = \min \begin{cases} 1 + opt(i, X - d_i) \\ opt(i - 1, X) \end{cases}$$

NGUYỄN KHÁNH PHƯƠNG 50
KHMT – SOICT - DHBKHN

Bài toán đổi tiền: Quy hoạch động

Phát biểu bài toán: Cho trước một tập n các đồng mệnh giá d_0, d_1, \dots, d_{n-1} và một mệnh giá X . Hãy tìm số lượng ít nhất các đồng tiền để đổi cho mệnh giá X .

Quy hoạch động:

```
int INF = 100000;
int d[10];
int mem[10][100000];
memset(mem, -1, sizeof(mem));

int opt(int i, int x) {
    if (x < 0) return INF;
    if (x == 0) return 0;
    if (i == -1) return INF;

    if (mem[i][x] != -1) return mem[i][x];

    int res = INF;
    res = min(res, 1 + opt(i, x - d[i]));
    res = min(res, opt(i - 1, x));

    mem[i][x] = res;
    return res;
}
```

51

Bài toán đổi tiền: Quy hoạch động

Độ phức tạp ?
Khi gọi $opt(n-1, X)$: có $n*X$ khả năng input cho hàm
Với mỗi input:

```
int INF = 100000;
int d[10];
int mem[10][100000];
memset(mem, -1, sizeof(mem));

int opt(int i, int x) {
    if (x < 0) return INF;
    if (x == 0) return 0;
    if (i == -1) return INF;

    if (mem[i][x] != -1) return mem[i][x];

    int res = INF;
    res = min(res, 1 + opt(i, x - d[i]));
    res = min(res, opt(i - 1, x));

    mem[i][x] = res;
    return res;
}
```

- hoặc là kết quả được lấy luôn từ bộ nhớ nếu như trước đây đã từng được tính: $O(1)$ nếu giả thiết lấy trực tiếp từ bộ nhớ chỉ mất $O(1)$
- hoặc là kết quả được tính và lưu lại: $O(1)$, nếu giả thiết mỗi lần gọi đệ quy mất $O(1)$

Mỗi input sẽ được tính tối đa 1 lần \Rightarrow Thời gian tính tổng cộng là $O(n*X)$

Hỏi làm thế nào để xác định phương án tối ưu gồm:

- bao nhiêu đồng tiền
- Số lượng mỗi loại đồng tiền ?

52

Cách 1: Truy vết đệ quy

```

int INF = 100000;
int d[10];
int mem[10][10000];
memset(mem, -1, sizeof(mem));

int opt(int i, int x) {
    if (x < 0) return INF;
    if (x == 0) return 0;
    if (i == -1) return INF;

    if (mem[i][x] != -1) return mem[i][x];

    int res = INF;
    res = min(res, 1 + opt(i, x - d[i]));
    res = min(res, opt(i - 1, x));
}

void trace(int i, int x) {
    if (x < 0) return;
    if (x == 0) return;
    if (i == -1) return;

    int res = INF;
    if (mem[i][x] == 1 + mem[i][x - d[i]]) {
        printf("%d ", d[i]);
        trace(i, x - d[i]);
    } else {
        trace(i-1, x);
    }
}

```

53

Cách 2: Truy vết bằng vòng lặp

```

int INF = 100000;
int d[10];
int mem[10][10000];
memset(mem, -1, sizeof(mem));

int opt(int i, int x) {
    if (x < 0) return INF;
    if (x == 0) return 0;
    if (i == -1) return INF;

    if (mem[i][x] != -1) return mem[i][x];

    int res = INF;
    res = min(res, 1 + opt(i, x - d[i]));
    res = min(res, opt(i - 1, x));
}

int answer = mem[n-1][x];
printf("%d\n", answer);
for (int i = n-1, k = 0; k < answer; ++k) {
    if (mem[i][x] == 1 + mem[i][x-d[i]]) {
        printf("%d ", d[i]);
        x -= d[i];
    } else {
        --i;
    }
}

```

54

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

- 2.1. Dãy con lớn nhất
- 2.2. Dãy con chung dài nhất
- 2.3. Dãy con tăng dài nhất
- 2.4. Bài toán đổi tiền

2.5. Bài toán người du lịch

2.6. Bài toán cái túi

Bài toán người du lịch (Traveling Salesman Problem – TSP)

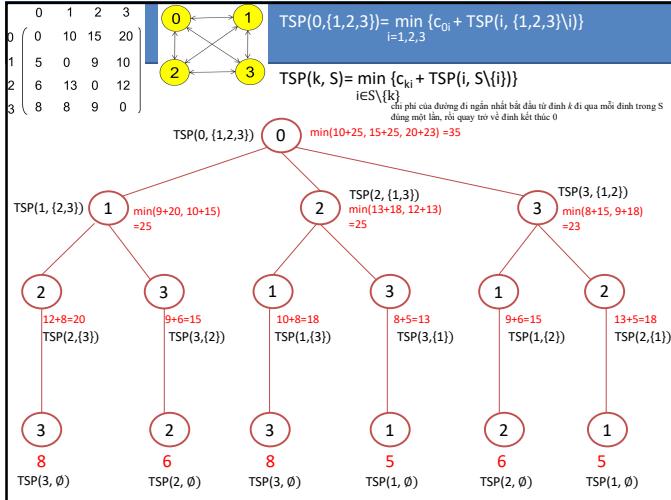
- Có n thành phố $0, 1, \dots, n-1$. Biết ma trận chi phí đi lại giữa các thành phố là

$$D = (c_{ij}; i, j = 0, 1, \dots, n-1).$$

- Người du lịch xuất phát từ một thành phố bất kỳ muốn đi qua tất cả các thành phố, mỗi thành phố đúng một lần rồi lại quay về thành phố xuất phát. Cách đi như vậy được gọi là một *hành trình*. Chi phí của hành trình được tính như là tổng các chi phí của các đoạn đường của nó.

Cần tìm hành trình với chi phí nhỏ nhất.

- **Duyệt toàn bộ:** duyệt toàn bộ các hoán vị của tập n phần tử $\{0, 1, \dots, n-1\} \rightarrow$ Độ phức tạp $O(n!)$ chỉ chạy được với $n \leq 11$
- Liệu có thể làm tốt hơn với **quy hoạch động???**



TSP: Quy hoạch động

Không mất tính tổng quát, giả sử thành phố xuất phát là 0.

Bước 1. Tìm công thức quy hoạch động

- Gọi $TSP(k, S)$ là chi phí của đường đi ngắn nhất bắt đầu từ đỉnh k đi qua mỗi đỉnh trong S đúng một lần, rồi quay trở về đỉnh kết thúc 0

Ví dụ: $S = \{1, 2, 3\}$ và $k = 2$, ta sẽ phải tìm đường đi ngắn nhất trong số 2 đường đi : $0 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 0$

và

$$0 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 0$$

- Neo đê quy: $TSP(k, \emptyset) = c_{k0}$
- Công thức đê quy cho $TSP(k, S) = ???$

$$TSP(k, S) = \min_{i \in S \setminus \{k\}} \{c_{ki} + TSP(i, S \setminus \{i\})\}$$

Ta có thể tính $TSP(k, S)$ nhờ lặp luân sau đây: Đường đi với chi phí nhỏ nhất cần tìm sẽ phải xuất phát từ k , từ k đến một thành phố $i \in S \setminus \{k\}$, sau đó đi qua mỗi thành phố còn lại trong S đúng 1 lần, rồi quay trở về 0 (nghĩa là: k là thành phố đi ngay trước i trong đường đi đó).

NGUYỄN KHÁNH PHƯƠNG 58
KHMT – SOICT - DHBKHN

TSP: Quy hoạch động

Không mất tính tổng quát, giả sử thành phố xuất phát là 0.

Bước 1. Tìm công thức quy hoạch động

- Gọi $TSP(k, S)$ là chi phí của đường đi ngắn nhất bắt đầu từ đỉnh k đi qua mỗi đỉnh trong S đúng một lần, rồi quay trở về đỉnh kết thúc 0

– Gọi $\text{visited} = \{0, 1, \dots, n-1\} \setminus S$

- Neo đê quy: $TSP(k, \emptyset) = c_{k0}$

• $TSP(k, \{0, 1, \dots, n-1\} \setminus \{k\}) = c_{k0}$

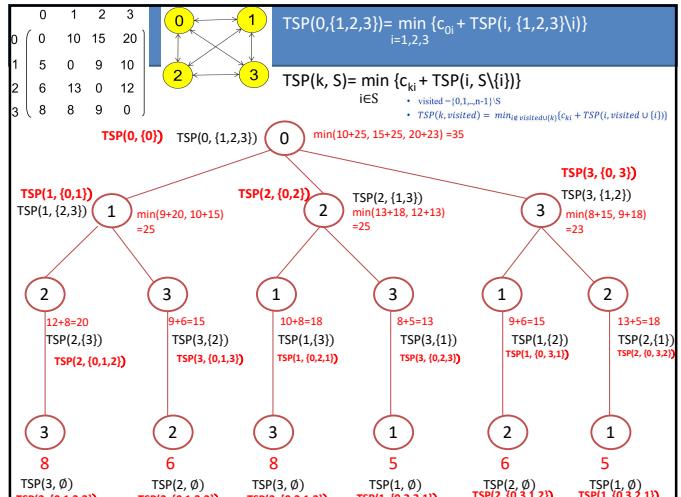
- Công thức đê quy cho $TSP(k, S) = ???$

– $TSP(k, S) = \min_{i \in S \setminus \{k\}} \{c_{ki} + TSP(i, S \setminus \{i\})\}$

• $\text{visited} = \{0, 1, \dots, n-1\} \setminus S$

• $TSP(k, \text{visited}) = \min_{i \in \text{visited} \cup \{k\}} \{c_{ki} + TSP(i, \text{visited} \cup \{i\})\}$

NGUYỄN KHÁNH PHƯƠNG 59
KHMT – SOICT - DHBKHN



TSP: Quy hoạch động

```

const int N = 20;
int c[N][N];
int mem[N][1<<N];
int n;

int tsp(int k, int visited)
{
    if(visited == ((1 << n) - 1)) //visited da gom tat ca cac thanh pho 0,...,n-1
        return c[k][0]; // quay tro ve thanh pho xuat phat 0

    if(mem[k][visited] != INT_MAX) //bai toan con da giat
        return mem[k][visited];

    int minTSPk = INT_MAX;
    for(int i = 0; i < n; ++i)
    {
        if(i == k || (visited & (1 << i))) //i= k hoac i da co trong tap visited
            continue;

        int distance = c[k][i] + tsp(i, visited | (1 << i)); //them i vao tap visited
        if (distance < minTSPk) minTSPk = distance;
    }
    mem[k][visited]=minTSPk;
    return minTSPk;
}

```

NGUYỄN KHÁNH PHƯƠNG 61
KHMT – SOICT – DHBKHN

TSP: Quy hoạch động

Như vậy, lời giải được đưa ra bằng cách gọi hàm:

```
cout<<"Gia tri toi uu: "<<tsp(0, 1<<0)<<endl;
```

NGUYỄN KHÁNH PHƯƠNG 62
KHMT – SOICT – DHBKHN

TSP: Quy hoạch động

```

int tsp(int k, int visited)
{
    if(visited == ((1 << n) - 1)) //visited da gom tat ca cac thanh pho 0,...,n-1
        return c[k][0]; // quay tro ve thanh pho xuat phat 0

    if(mem[k][visited] != INT_MAX) //bai toan con da giat
        return mem[k][visited];

    int minTSPk = INT_MAX;
    for(int i = 0; i < n; ++i)
    {
        if(i == k || (visited & (1 << i))) //i= k hoac i da co trong tap visited
            continue;

        int distance = c[k][i] + tsp(i, visited | (1 << i)); //them i vao tap visited
        if (distance < minTSPk) minTSPk = distance;
    }
    mem[k][visited]=minTSPk;
    return minTSPk;
}

```

Độ phức tạp?

Khi gọi `tsp(0, 1<<0)`: có $n * 2^n$ khả năng input cho hàm

Với mỗi input:

- hoặc là kết quả được lấy luôn từ bộ nhớ nếu như trước đây đã từng được tính: $O(1)$ nếu giả thiết lấy trực tiếp từ bộ nhớ chỉ mất $O(1)$
- hoặc là kết quả được tính và lưu lại: thời gian tính $O(n)$, giả thiết mỗi lời gọi đệ quy mất $O(1)$

Mỗi input sẽ được tính tối đa 1 lần

Thời gian tính tổng cộng là $O(n^2 2^n)$

➔ Có thể chạy nhanh được đến $n \leq 20$

NGUYỄN KHÁNH PHƯƠNG 63
KHMT – SOICT – DHBKHN

TSP: Quy hoạch động

```

int minTSPk = INT_MAX;int mini;
for(int i = 0; i < n; ++i)
{
    if(i == k || (visited & (1 << i))) //i= k hoac i da co trong tap visited
        continue;

    int distance = c[k][i] + tsp(i, visited | (1 << i)); //them i vao tap visited
    if(distance < minTSPk)
    {
        minTSPk = distance;
        mini = i;
    }
}
mem[k][visited]=minTSPk;
succ[k][visited] = mini;

```

Làm thế nào để đưa ra hành trình tối ưu?

- Công thức để quy $TSP(k, visited) = \min_{i \notin \text{visited} \cup \{k\}} \{c_{ki} + TSP(i, visited \cup \{i\})\}$
- Để tìm hành trình tối ưu cần ghi nhận $\text{Succ}(k, visited)$ là chỉ số i đạt min trong biểu thức trên.

64

TSP: Đưa ra hành trình tối ưu

```

int tsp(int k, int visited)
{
    if(visited == ((1 << n) - 1)) //visited da gom tat ca cac thanh pho 0...n-1
        return succ[k][0]; // quay tro ve thanh pho xuat phat
    if(mem[k][visited] != INT_MAX) //doi ton con da giat
        return mem[k][visited];

    int minTSPk = INT_MAX; int mini;
    for(int i = 0; i < n; ++i)
    {
        if(i == k || (visited & (1 << i))) //i = k hoac i da co trong tap visited
            continue;
        int distance = succ[k][i] + tsp(i, visited | (1 << i)); //them i vao tap visited
        if(distance < minTSPk)
        {
            minTSPk = distance;
            mini = i;
        }
    }
    mem[k][visited] = minTSPk;
    succ[k][visited] = mini;
    return minTSPk;
}

int main()
{
    int i, j;
    memset(mem, -1, sizeof(mem));
    cin >> n;
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            scanf("%d", &c[i][j]);
    cout<<"Gia tri toi uu: "<<tsp(0, 1<<0)<<endl;
    cout<<"Hanh trinh toi uu: ";
    trace_tsp(0, 1<<0);
}

```

65

```

void trace_tsp(int k, int visited)
{
    cout<<k<<" --> ";
    if((visited == (1<<n)-1))
    {
        cout<<"0 ";
        return;
    }
    int i = succ[k][visited];
    trace_tsp(i, visited|(i<<i));
}

```

```

9 14 18 15
3 0 4 22 20
17 9 0 16 4
10 20 13 18
15 11 5 8

```

```

Gia tri toi uu: 25
Hanh trinh toi uu: 0 --> 1 --> 2 --> 4 --> 3 --> 0

```

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Dãy con lớn nhất

2.2. Dãy con chung dài nhất

2.3. Dãy con tăng dài nhất

2.4. Bài toán đói tiền

2.5. Bài toán người du lịch

2.6. Bài toán cái túi

NGUYỄN KHÁNH PHƯƠNG 66
KHMT – SOICT- ĐHQKHN

Bài toán cái túi (Knapsack problem – KP)

- Cho cái túi với dung lượng L , và tập S gồm n đồ vật.
- Đồ vật i có trọng lượng w_i và giá trị sử dụng v_i (các số w_i , v_i và L là nguyên không âm)
- Bài toán:** Cần cho vào túi những đồ vật nào để đạt được tổng giá trị của các đồ vật chất trong túi là lớn nhất?

NGUYỄN KHÁNH PHƯƠNG 67
KHMT – SOICT- ĐHQKHN

KP – Phát biểu bài toán

- Dưới dạng bài toán tối ưu: Tìm $\max\{f(T) = \sum_{i \in T} v_i \mid T \subseteq N: \sum_{i \in T} w_i \leq L\}$

$N = \{1, 2, \dots, n\}$ – tập chỉ số của các đồ vật

(Tên chính xác của bài toán: “Bài toán cái túi biến 0-1”)

- Thuật toán trực tiếp (Duyệt toàn bộ):

– Duyệt tất cả 2^n khả năng chất đồ vào túi: tìm cách chất có tổng giá trị lớn nhất trong số các cách chất có tổng trọng lượng của các đồ vật không quá dung lượng của túi.

– Thời gian tính: $O(n2^n)$

- Thuật toán quy hoạch động:

NGUYỄN KHÁNH PHƯƠNG 68
KHMT – SOICT- ĐHQKHN

KP: Quy hoạch động

Bước 1: Tìm công thức quy hoạch động

- Tính $Value[k, S]$ là tổng giá trị lớn nhất của các đồ vật được chọn trong số k đồ vật đầu tiên và có tổng trọng lượng không quá S với mỗi $1 \leq k \leq n$ và $0 \leq S \leq L$
- Có tất cả $n(L+1)$ bài toán.
- Giá trị tối ưu cần tìm là: $Value[n, L]$

NGUYỄN KHÁNH PHƯƠNG 69
KHMT – SOICT - ĐHBKHN

KP: Quy hoạch động

- Neo đê quy:** Với $k=0$ hoặc $S=0$, ta có $Value[0, S] = 0, Value[k, 0] = 0$.
- Tìm công thức đê quy cho $Value[k, S]$: Giả sử đã tính được $Value[i, j]$ với mọi $1 \leq i < k$ và $0 \leq j \leq L$. Khi đó, để tính $Value[k, S]$ ta có thể lập luận như sau:
 - Nếu $S < w_k$ thì đồ vật thứ k không thể chắt vào túi, do đó cách chắt tối ưu chỉ có thể sử dụng $k-1$ đồ vật trước đó và có giá trị là $Value[k-1, S]$.
 - Nếu $S \geq w_k$, thì cách chắt tối ưu cần lựa chọn trong 2 cách chắt sau:
 - Không chắt đồ vật k vào túi, khi đó giá trị của cách chắt tối ưu sẽ là $Value[k-1, S]$,
 - Chắt đồ vật thứ k vào túi, khi đó trọng lượng còn lại $S-w_k$ sẽ được chắt tối ưu từ $k-1$ đồ vật đầu tiên với giá trị tối ưu là $Value[k-1, S-w_k]$.

Từ đó ta có công thức đê qui: Với $S=0, 1, \dots, L$

$$Value[k, S] = \begin{cases} Value[k-1, S], & \text{nếu } S < w_k \\ \max\{ Value[k-1, S], v_k + Value[k-1, S-w_k] \}, & \text{nếu trái lại,} \end{cases}$$

KP: Cài đặt

$$Value[k, S] = \begin{cases} Value[k-1, S], & \text{nếu } S < w_k \\ \max\{ Value[k-1, S], v_k + Value[k-1, S-w_k] \}, & \text{nếu trái lại.} \end{cases}$$

```
for k=0 to n do
  for S=0 to L do
  {
    if (k==0 || S==0) Value[k, S] = 0;
    else if (S < w[k])
      Value[k, S] = Value[k-1, S];
    else
      Value[k, S] = max(Value[k-1, S], Value[k-1, S-w[k]]);
  }
  Thời gian tính: O(n*L)
```

NGUYỄN KHÁNH PHƯƠNG 71
KHMT – SOICT - ĐHBKHN

Nhận xét

- Thuật toán giải bài toán cái túi có đánh giá thời gian tính là $O(nL)$. Thời gian này phụ thuộc tuyến tính vào dữ liệu vào L , nhưng nếu xét nó như là hàm của độ dài dữ liệu vào $l = \lceil \log L \rceil$, thì sự phụ thuộc này lại là hàm mũ. Trên thực tế, thuật toán này chỉ làm việc hiệu quả khi L là không quá lớn.
- Người ta gọi thuật toán có thời gian tính bị chặn trên bởi đa thức của dữ liệu vào là **thuật toán giả đa thức** (pseudopolynomial algorithm).

NGUYỄN KHÁNH PHƯƠNG 72
KHMT – SOICT - ĐHBKHN