QUI HOẠCH ĐỘNG

Lập trình thi đấu

Đỗ Phan Thuận

Bộ môn Khoa Học Máy Tính
Viện Công Nghệ Thông Tin và Truyền Thông
Đại học Bách Khoa Hà Nội

Ngày 25 tháng 8 năm 2018

Richard Ernest Bellman (1920-1984): Multistage decision processes, 1949

Qui hoạch động là gì?

- Là một mô hình giải bài
- Nhiều điểm tương đồng với hai phương pháp Chia để trị và Quay lui
- Nhắc lại Chia để trị:
  - Chia bài toán 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 ban đầu
- Phương pháp qui hoạch động:
  - Chia bài toán 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 ban đầu
  - *Không tìm nhiều hơn một lần lời giải của cùng một bài toán*

Công thức Qui hoạch động

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

Công thức đệ qui

```
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[Q] = result;
    return result;
}
```

Dãy Fibonacci

Hai số đầu tiên của dãy Fibonacci là 1 và 1. Tất cả các số khác của dãy được tính bằng tổng của hai số ngay trước nó trong dãy

- Yêu cầu: Tìm số Fibonacci thứ n
- Thử giải bài toán bằng phương pháp Qui hoạch động

1. Tìm công thức truy hồi:

$$\text{fibonacci}(1) = 1$$
$$\text{fibonacci}(2) = 1$$
$$\text{fibonacci}(n) = \text{fibonacci}(n-2) + \text{fibonacci}(n-1)$$

Dãy Fibonacci

2. Cài đặt công thức qui hoạch động

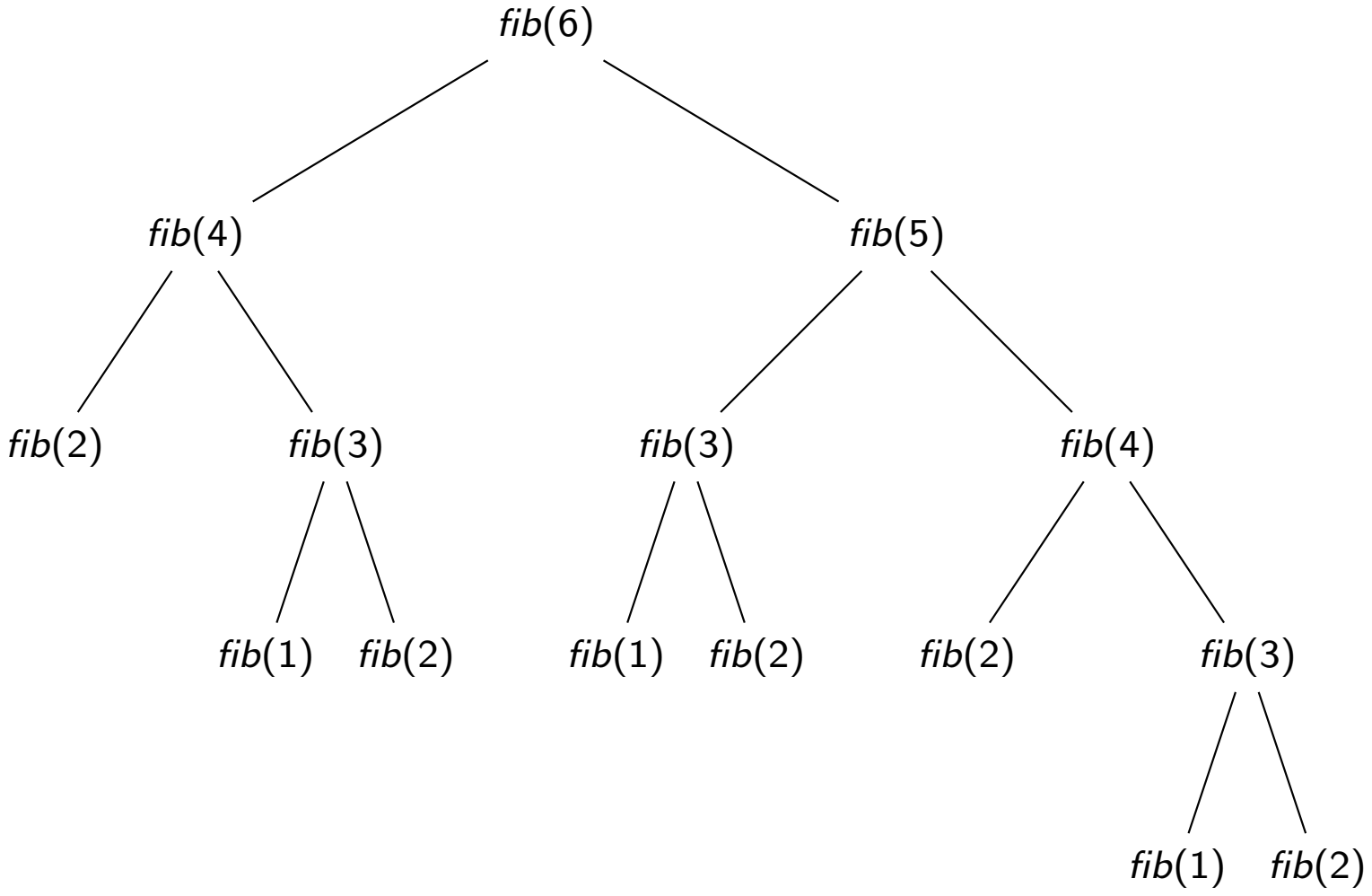
```
int fibonacci(int n) {
    if (n <= 2) {
        return 1;
    }

    int res = fibonacci(n - 2) + fibonacci(n - 1);

    return res;
}
```

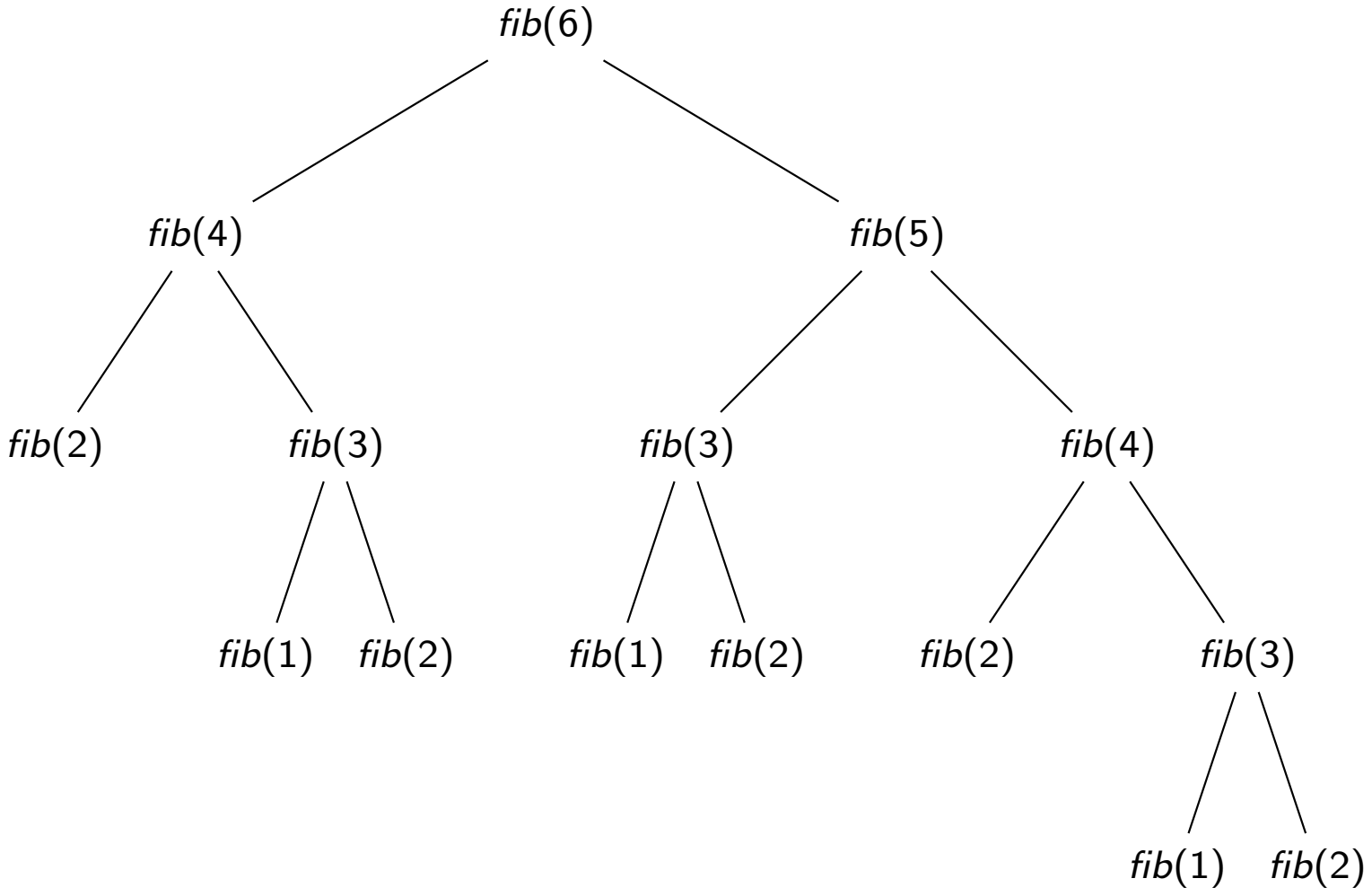
Dãy Fibonacci

- Độ phức tạp là bao nhiêu?



Dãy Fibonacci

- Độ phức tạp là bao nhiêu? Hàm mũ, gần như $O(2^n)$



Dãy Fibonacci

3. Lưu trữ kết quả các hàm đã tính

```
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;
}
```

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;
}
```

Dãy Fibonacci

- Bây giờ độ phức tạp là bao nhiêu?
- Ta có n khả năng input cho hàm đệ qui: 1, 2, . . . , n.
- Với mỗi input:
  - hoặc là kết quả được tính và lưu trữ lại
  - hoặc là lấy luôn ra từ bộ nhớ nếu như trước đây đã được tính
- Mỗi input sẽ được tính tốt đa một lần
- Thời gian tính là $O(n \times f)$, với f là thời gian tính toán của hàm với một input, với giả thiết là kết quả đã tính trước đây sẽ được lấy trực tiếp từ bộ nhớ, chỉ trong $O(1)$
- Do ta chỉ tốn một lượng hằng số phép tính đối với một input của hàm, nên $f = O(1)$
- Thời gian tính tổng cộng là $O(n)$

Tổng lớn nhất trong mảng

- Cho một mảng số nguyên $\mathrm{arr}[0]$, $\mathrm{arr}[1]$, ..., $\mathrm{arr}[n-1]$, hãy tìm một đoạn trong mảng có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-15	8	-2	1	0	6	-3

Tổng lớn nhất trong mảng

- Cho một mảng số nguyên $\mathrm{arr}[0]$, $\mathrm{arr}[1]$, ..., $\mathrm{arr}[n-1]$, hãy tìm một đoạn trong mảng có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-15	8	-2	1	0	6	-3

- Tổng của đoạn có trọng số lớn nhất trong mảng là 13

Tổng lớn nhất trong mảng

- Cho một mảng số nguyên $\mathrm{arr}[0], \mathrm{arr}[1], \ldots, \mathrm{arr}[n-1]$, hãy tìm một đoạn trong mảng có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-15	8	-2	1	0	6	-3

- Tổng của đoạn có trọng số lớn nhất trong mảng là 13
- Cách giải thế nào?
  - Phương pháp trực tiếp thử tất cả gần $\approx n^2$ khoảng, và tính trọng số mỗi đoạn, cho độ phức tạp$O(n^3)$
  - Ta có thể xử lý kỹ thuật bởi một "mẹo" lưu trữ cố định trong vòng lặp để giảm độ phức tạp về $O(n^2)$
  - Liệu có thể làm tốt hơn với phương pháp Qui hoạch động?

Tổng lớn nhất trong mảng

- Bước đầu tiên là đi tìm công thức qui hoạch động
- Gọi $\max\_\text{sum}(i)$ là trọng số của đoạn có trọng số lớn nhất giới hạn trong đoạn $0, \ldots, i$
- Neo đệ qui: $\max\_\text{sum}(0) = \max(0, arr[0])$
- $\max\_\text{sum}(i)$?
- Liên hệ gì với $\max\_\text{sum}(i-1)$?
- Liệu có thể kết hợp lời giải của các bài toán con có kích thước bé hơn i thành lời giải bài toán có kích thước bằng i?
- Không hoàn toàn hiển nhiên phải không ?...

Tổng lớn nhất trong mảng

- Hãy thay đổi hàm mục tiêu:
- Gọi $\max\_\text{sum}(i)$ là trọng số đoạn có trọng số lớn nhất giới hạn bởi $0, \ldots, i$, *mà phải kết thúc tại* i
- Neo đệ qui: $\max\_\text{sum}(0) = arr[0]$
- $\max\_\text{sum}(i) = \max(arr[i], arr[i] + \max\_\text{sum}(i-1))$
- Vậy công thức cuối cùng chỉ là $\max_{0 \le i < n} \{ \max\_\text{sum}(i) \}$

Tổng lớn nhất trong mảng

- Bước tiếp theo là cài đặt công thức qui hoạch động

```
int arr[1000];

int max_sum(int i) {
    if (i == 0) {
        return arr[i];
    }

    int res = max(arr[i], arr[i] + max_sum(i - 1));
    return res;
}
```

Tổng lớn nhất trong mảng

- Bước cuối cùng là lưu trữ các hàm đã tính

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

int max_sum(int i) {
    if (i == 0) {
        return arr[i];
    }
    if (comp[i]) {
        return mem[i];
    }

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

Tổng lớn nhất trong mảng

- Bây giờ lời giải đơn giản là giá trị lớn nhất của trong các giá trị $\max\_\text{sum}(i)$

```
int maximum = 0;
for (int i = 0; i < n; i++) {
    maximum = max(maximum, best_sum(i));
}

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

Tổng lớn nhất trong mảng

- Lưu ý nếu bài toán yêu cầu tìm đoạn có trọng số lớn nhất trong nhiều mảng khác nhau, thì hãy nhớ xóa bộ nhớ khi kết thúc tính toán mỗi mảng

Tổng lớn nhất trong mảng

- Độ phức tạp tính toán ?
- Có n khả năng input cho hàm đệ qui
- Mỗi input được tính trong $O(1)$, giả thiết là mỗi phép gọi đệ qui là $O(1)$
- Thời gian tính toán tổng cộng là $O(n)$

Đổi tiền

- Cho trước một mảng các đồng tiền mệnh giá $d_0, d_2, \ldots, d_{n-1}$, và một mệnh giá x. Hãy tìm số lượng ít nhất các đồng tiền để đổi cho mệnh giá x?
- Có ai nhớ thuật toán tham lam cho bài Đổi tiền này?
- Lời giải thuật toán tham lam không hề chắc chắn đưa ra lời giải tối ưu, thậm chí nhiều trường hợp còn không đưa ra được lời giải...

- Có thể sử dụng phương pháp Qui hoạch động?

Đổi tiền

- Bước đầu tiên: xây dựng công thức Qui hoạch động
- Gọi $\mathrm{opt}(i, x)$ là số lượng tiền ít nhất cần để đổi mệnh giá x nếu chỉ được phép sử dụng các đồng tiền mệnh giá $d_0, \ldots, d_i$
- Neo đệ qui: $\mathrm{opt}(i, x) = \infty$ if $x < 0$
- Neo đệ qui: $\mathrm{opt}(i, 0) = 0$
- Neo đệ qui: $\mathrm{opt}(-1, x) = \infty$
- $\mathrm{opt}(i, x) = \min \begin{cases} 1 + \mathrm{opt}(i, x - d_i) \\ \mathrm{opt}(i - 1, x) \end{cases}$

Đổi tiền

```
int INF = 100000;
int d[10];

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

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

    return res;
}
```

Đổi tiền

```
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));

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

Đổi tiền

- Độ phức tạp?
- Số lượng khả năng input là $n \times x$
- Mỗi input được xử lý trong $O(1)$, giả thiết mỗi lời gọi đệ qui thực hiện trong thời gian hằng số
- Thời gian tính toán tổng cộng là $O(n \times x)$

Đổi tiền

- Làm thế nào để xác định được những đồng tiền nào cho phương án tối ưu ?
- Hãy ghi nhận lại thêm các thông tin để truy vết ngược lại quá trình đệ qui

Dãy con tăng dài nhất

- Cho một mảng n số nguyên $a[0], a[1], \ldots, a[n-1]$, hãy tìm độ dài của dãy con tăng dài nhất?

- Định nghĩa dãy con?
- Nếu xoá đi 0 phần tử hoặc một số phần tử của mảng a thì sẽ thu được một dãy con của a
- Ví dụ: $a = [5, 1, 8, 1, 9, 2]$
- $[5, 8, 9]$ là một dãy con
- $[1, 1]$ là một dãy con
- $[5, 1, 8, 1, 9, 2]$ là một dãy con
- [] là một dãy con
- $[8, 5]$ is **not** là một dãy con
- $[10]$ is **not** là một dãy con

Dãy con tăng dài nhất

- Cho một mảng n số nguyên $a[0], a[1], \ldots, a[n-1]$, hãy tìm độ dài của dãy con tăng dài nhất?
- Một dãy con tăng của a là một dãy con của a sao cho các phần tử là tăng chặt từ trái sang phải
- $[5, 8, 9]$ và $[1, 8, 9]$ là hai daỹ con tăng của $a = [5, 1, 8, 1, 9, 2]$
- Làm thế nào để tính độ dài dãy con tăng dài nhất?
- Có 2^n dãy con, phương pháp đơn giản nhất là duyệt qua toàn bộ các dãy này
- Thuật toán cho độ phức tạp $O(n2^n)$, chỉ có thể chạy được với $n \leq 23$
- Phương pháp Qui hoạch động thì sao?

Dãy con tăng dài nhất

- Gọi $\mathrm{lis}(i)$ là độ dài dãy con tăng dài nhất của mảng $a[0], \ldots, a[i]$
- Nei đệ qui: $\mathrm{lis}(0) = 1$
- Công thức đệ qui cho $\mathrm{lis}(i)$?
- Nếu đặt hàm mục tiêu như vậy sẽ gặp phải vấn đề giống như bài toán tổng lớn nhất trong mảng ở trên, hãy thay đổi một chút hàm mục tiêu

Dãy con tăng dài nhất

- Gọi $\mathrm{lis}(i)$ là độ dài dãy con tăng dài nhất của mảng $a[0], \ldots, a[i]$, *mà kết thúc tại i*
- Neo đệ qui: không cần thiết
- $\mathrm{lis}(i) = \max(1, \max_{j \text{ s.t. } a[j]<a[i]}\{1 + \mathrm{lis}(j)\})$

Dãy con tăng dài nhất

```
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;
}
```

Dãy con tăng dài nhất

- Bây giờ độ dài dãy con tăng dài nhất chính là giá trị lớn nhất trong các giá trị $\mathrm{lis}(i)$:

```
int mx = 0;
for (int i = 0; i < n; i++) {
    mx = max(mx, lis(i));
}

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

Dãy con tăng dài nhất

- Độ phức tạp tính toán?

- Có n khả năng input
- Mỗi input được tính trong thời gian $O(n)$, giả thiết mỗi lời gọi đệ qui chỉ mất $O(1)$
- Thời gian tính tổng cộng là $O(n^2)$

- Có thể chạy được đến $n \leq 10\ 000$, tốt hơn rất nhiều sơ với phương pháp duyệt toàn bộ!

Dãy con chung dài nhất

- Cho hai xâu (hoặc hai mảng số nguyên) n phần tử $a[0], \ldots, a[n-1]$ và $b[0], \ldots, b[m-1]$, hãy tìm độ dài của dãy con chung dài nhất của hai xâu.

- $a =$"b<u>an</u>an<u>inn</u>"
- $b =$"k<u>anin</u>a<u>n</u>"
- Dãy con chung dài nhất của a và b, "aninn", có độ dài 5

Dãy con chung dài nhất

- Gọi $\mathrm{lcs}(i, j)$ là độ dài dãy con chung dài nhất của $a[0], \ldots, a[i]$ và $b[0], \ldots, b[j]$
- Neo đệ qui: $\mathrm{lcs}(-1, j) = 0$
- Neo đệ qui: $\mathrm{lcs}(i, -1) = 0$
- $$\mathrm{lcs}(i, j) = \max \begin{cases} \mathrm{lcs}(i, j-1) \\ \mathrm{lcs}(i-1, j) \\ 1 + \mathrm{lcs}(i-1, j-1) \quad \text{if } a[i] = b[j] \end{cases}$$

Dãy con chung dài nhất

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

Dãy con chung dài nhất

- Độ phức tạp tính toán?

- Có n khả năng input
- Mỗi input được tính trong thời gian $O(1)$, giả thiết mỗi lời gọi đệ qui chỉ mất $O(1)$
- Thời gian tính tổng cộng là $O(n \times m)$

Qui hoạch động trên bitmask

- Có còn nhớ biểu diễn bitmask cho các tập con?
- Mỗi tập con của tập n phần tử được biểu diễn bởi một số nguyên trong khoảng $0, \ldots, 2^n - 1$
- Điều này có thể giúp thực hiện phương pháp qui hoạch động dễ dàng trên các tập con

Bài toán người du lịch

- Cho một đồ thi n đỉnh và giá trị trọng số $c_{i,j}$ trên mỗi cặp đỉnh i, j. Hãy tìm một chu trình đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần sao cho tổng các trọng số trên chu trình đó là nhỏ nhất
- Đây là bài toán NP-khó, vì vậy không tồn tại thuật toán tất định thời gian đa thức nào hiện biết để giải bài toán này
- Thuật toán duyệt toàn bộ đơn giản duyệt qua toàn bộ các hoán vị các đỉnh cho độ phức tạp là $O(n!)$, nhưng chỉ có thể chạy được đến $n \leq 11$
- Liệu có thể làm tốt hơn với phương pháp qui hoạch động?

Bài toán người du lịch

- Không mất tính tổng quát giả sử chu trình bắt đầu và kết thúc tại đỉnh 0
- Gọi $\mathrm{tsp}(i, S)$ cách sử dụng ít chi phí nhất để đi qua toàn bộ các đỉnh và quay trở lại đỉnh 0, nếu như hiện tại hành trình đang ở tại đỉnh i và người du lịch đã thăm tất cả các đỉnh trong tập S
- Neo đệ qui: $\mathrm{tsp}(i, \text{all vertices}) = c_{i,0}$
- Công thức đệ qui $\mathrm{tsp}(i, S) = \min_{j \notin S} \{ c_{i,j} + \mathrm{tsp}(j, S \cup \{j\}) \}$

Bài toán người du lịch

```
const int N = 20;
const int INF = 100000000;
int c[N][N];
int mem[N][1<<N];
memset(mem, -1, sizeof(mem));

int tsp(int i, int S) {
    if (S == ((1 << N) - 1)) {
        return c[i][0];
    }
    if (mem[i][S] != -1) {
        return mem[i][S];
    }

    int res = INF;
    for (int j = 0; j < N; j++) {
        if (S & (1 << j))
            continue;

        res = min(res, c[i][j] + tsp(j, S | (1 << j)));
    }
```

Bài toán người du lịch

- Như vậy lời giải tối ưu có thể được đưa ra như sau:

- `printf("%d\n", tsp(0, 1<<0));`

Bài toán người du lịch

- Độ phức tạp tính toán?

- Có $n \times 2^n$ khả năng input
- Mỗi input được tính trong thời gian $O(n)$, giả thiết mỗi lời gọi đệ qui chỉ mất $O(1)$
- Thời gian tính tổng cộng là $O(n^2 2^n)$

- Như vậy có thể tính được với n lên đến 20