THUẬT TOÁN ỨNG DỤNG

Đệ quy – Quay lui – Nhánh cận

Nội dung

1. Đệ quy
   - Đệ quy
   - Đệ quy có nhớ
2. Quay lui
   - Nhị phân
   - Tập con
   - Hoán vị
   - Phân tích
   - Đặt hậu
3. Nhánh cận
   - Bài toán người bán hàng (TSP – Traveling Salesman Problem)
4. Bài tập

Phần 1

Đệ quy

Đệ quy: khái niệm

- Hàm đệ quy = Hàm có lời gọi lại chính nó trong quá trình thực hiện
  - Đệ quy trực tiếp: gọi lại chính nó ngay trong thân hàm
  - Đệ quy gián tiếp: gọi lại chính nó thực hiện trong các hàm con

```
// in các số nguyên từ 1 đến n viết đệ quy
void print(int n) {
    if (n > 1) print(n-1);
    cout << " " << n;
}

// tính tổ hợp chập k của n dựa trên công thức
// C(k, n) = C(k-1, n-1) + C(k, n-1)
int C(int k, int n) {
    if (k == n || k == 0) return 1;
    return C(k-1, n-1) + C(k, n-1);
}
```

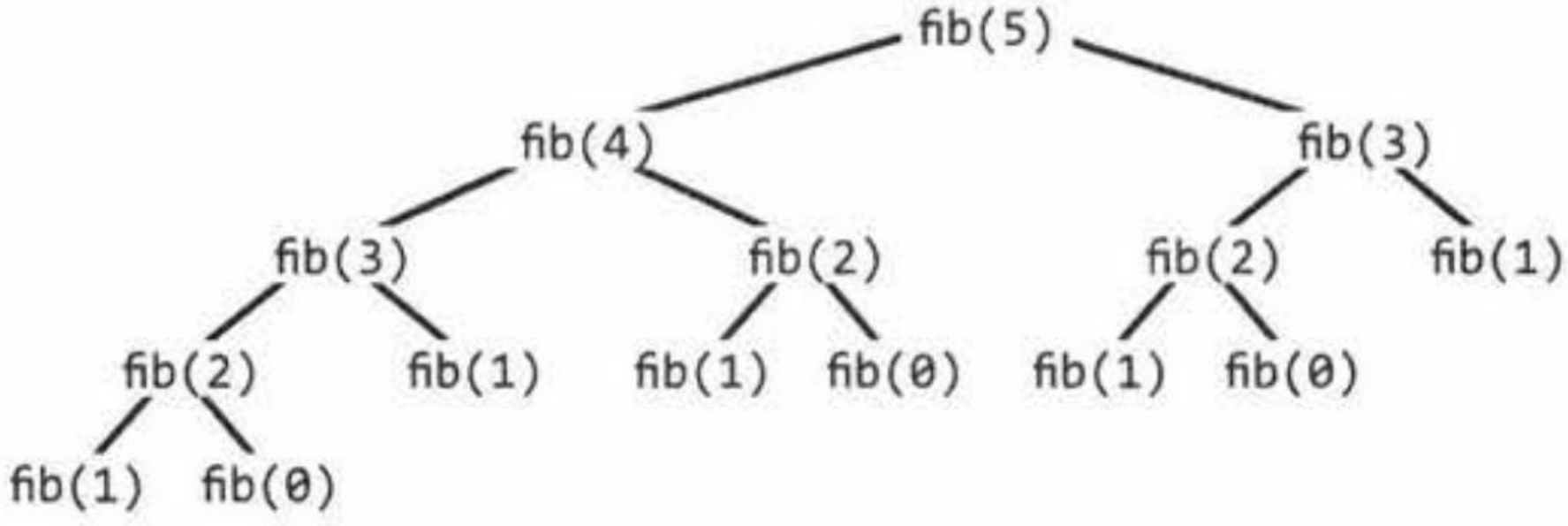
Đệ quy: đặc điểm

- Đơn giản:
  - Đệ quy phù hợp với tiếp cận từ trên xuống (chia bài toán lớn thành các bài toán nhỏ)
  - Mã ngắn gọn, dễ hiểu, thể hiện chính xác tiếp cận top-down
- Chậm:
  - Chi phí thời gian cho việc gọi hàm đệ quy
  - Một hàm có thể bị gọi lại nhiều lần
- Chuyển về vòng lặp (khử đệ quy): hầu hết các hàm đệ quy đơn (single recursion – hàm đệ quy chỉ gọi chính nó một lần) đều có thể chuyển về vòng lặp khá đơn giản
  - Mọi hàm đệ quy đều có thể chuyển về vòng lặp, vấn đề là việc chuyển như vậy đơn giản hay phức tạp mà thôi

Đệ quy có nhớ

- Các tiếp cận đệ quy đôi khi làm cho việc gọi hàm con bùng nổ tổ hợp

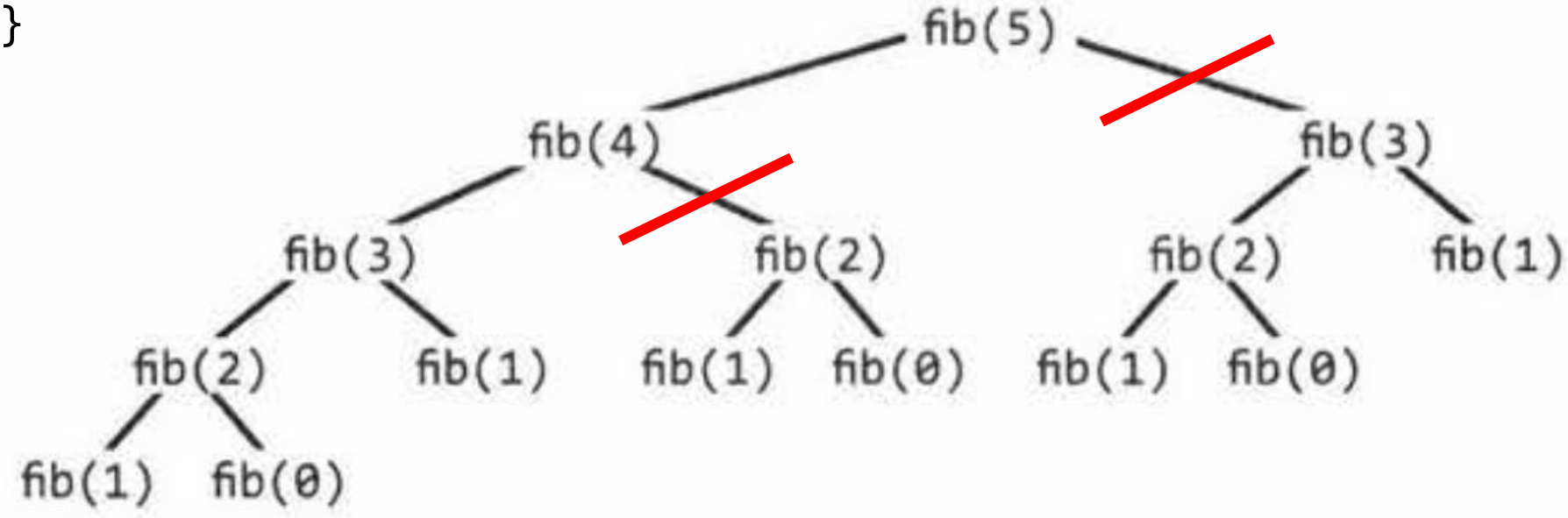
```
int fibo(int n) {
    if (n < 2) return n;
    return fibo(n-1) + fibo(n-2);
}
```



Đệ quy có nhớ

- Giải quyết: dùng bộ nhớ lưu lại kết quả để dùng lại

```
int fibo(int n) {
    if (n < 2) return n;
    // nếu chưa tính hàm fibo(n) thì tính và lưu vào f[n]
    if (f[n] = -1)
        f[n] = fibo(n-1) + fibo(n-2);
    return f[n];
}
```



Đệ quy có nhớ: nguyên tắc triển khai

- Sử dụng bộ nhớ để lưu kết quả:
  - Tính toán những trường hợp nhỏ, ghi vào bộ nhớ
  - Những phần chưa được tính toán thì đánh dấu lại (chẳng hạn như ghi tạm giá trị là -1)
- Khi thực hiện đệ quy:
  - Tìm trong bộ nhớ xem đã có kết quả chưa, nếu có rồi thì trả ngay về kết quả đã có
  - Nếu chưa có thì thực hiện đệ quy như bình thường, lưu lại kết quả tính được vào bộ nhớ
  - Trả về kết quả vừa tính được
- Chú ý: không phải lúc nào cũng có thể dùng bộ nhớ để lưu lại kết quả tính toán

Đệ quy có nhớ: ví dụ triển khai


```
#include <iostream>

const int MAX = 100;

int ckn[MAX][MAX];

int C(int k, int n) {
    if (k == n || k == 0) return 1;
    if (ckn[k][n] == -1)
        ckn[k][n] = C(k-1, n-1) + C(k, n-1);
    return ckn[k][n];
}

int main() {
    for (int i = 0; i < MAX; i++)
        for (int j = 0; j < MAX; j++) ckn[i][j] = -1;
    std::cout << C(15, 30);
}
```

Phần 2

Quay lui

Vấn đề duyệt toàn bộ

Cho một dãy số nguyên A = (a_1, a_2,..., a_N). Hãy chia các phần tử trong A thành hai nhóm tách rời P và Q (một phần tử thuộc và chỉ thuộc một nhóm) sao cho chênh lệch tổng các phần tử của P và Q là nhỏ nhất.

Ví dụ:

- N = 11
- A = {1, 2, 3, 4, 5, 8, 7, 7, 10, 20, 1}
- Phương án chia tối ưu thành 2 nhóm như sau:
  - P = {1, 2, 3, 4, 5, 8, 10, 1}
  - Q = {7, 7, 20}
  - Tổng hai nhóm chênh lệch nhau nhỏ nhất là 0

Quay lui

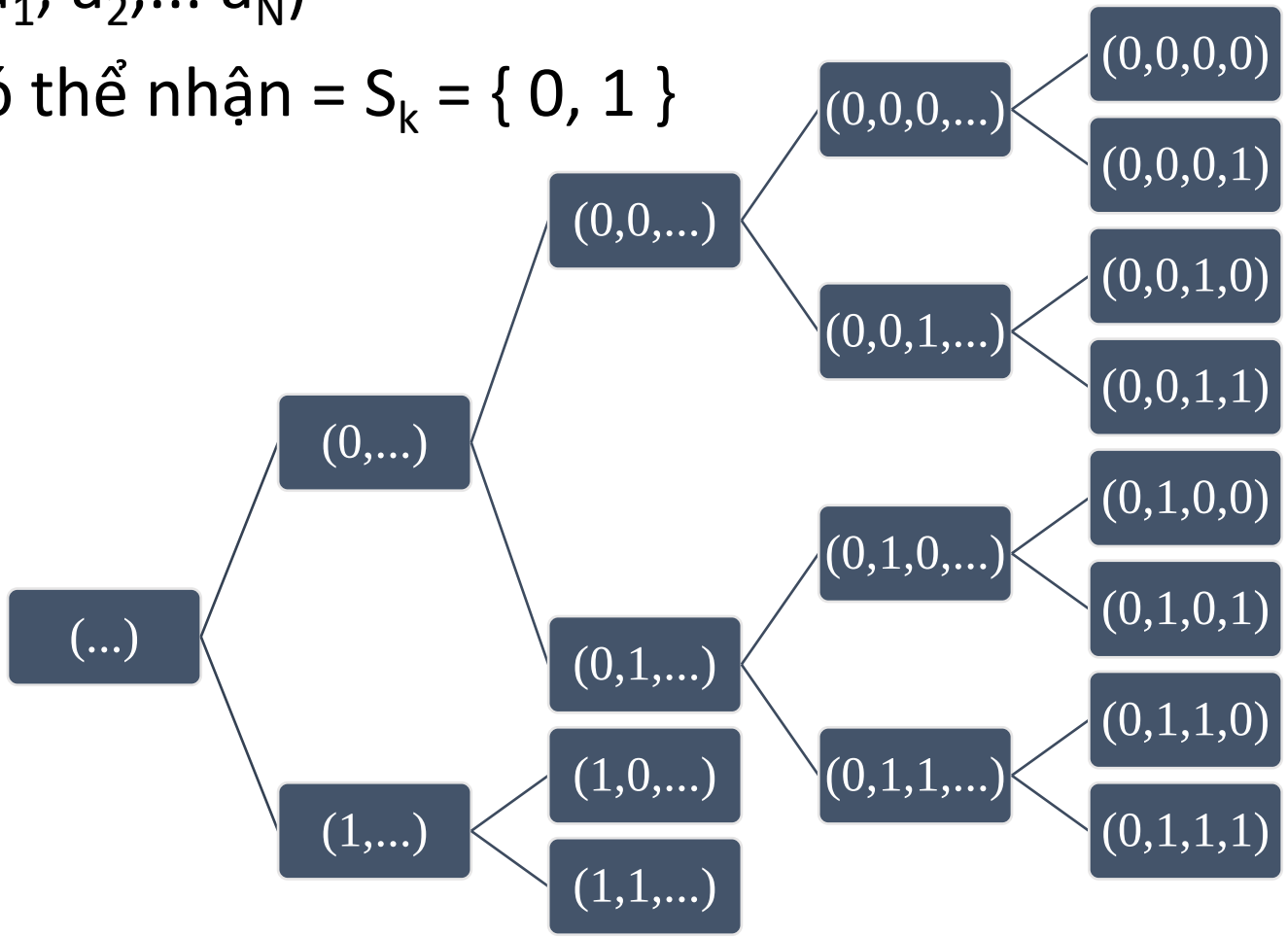
- Tên tiếng Anh: backtracking (Lehmer, 1950)
- Chiến lược tìm kiếm lời giải cho các bài toán thỏa mãn ràng buộc bằng cách xét mọi tổ hợp
- Bài toán tổng quát: Liệt kê mọi cấu hình A = (a_1, a_2,... a_N) thỏa mãn một số ràng buộc nào đó
  - Nhị phân: liệt kê mọi chuỗi nhị phân độ dài N
  - Tập con: liệt kê mọi cách chọn N phần tử trong số M phần tử
  - Hoán vị: liệt kê mọi hoán vị của (1,2,...,N)
  - Phân tích: liệt kê mọi cách phân tích số M thành tổng N số nguyên dương
  - Đặt hậu: liệt kê mọi cách đặt N quân hậu lên bàn cờ N x N để hai quân bất kỳ không ăn nhau

Quay lui

- Quy tắc: xây dựng từng thành phần cho đến khi đạt được cấu hình theo yêu cầu
- Cấu hình đầu tiên là rỗng: A = ()
- Tìm cách xây dựng dần dần các phần tử a_1, a_2,... a_N
- Quy tắc xây dựng phần tử a_k:
  - Nếu k > N: cấu hình A đã hoàn chỉnh, in ra và quay lui
  - Xây dựng tập S_k chứa mọi giá trị có thể của a_k
  - Nếu $S_k = \emptyset$, quay lui trở về hàm gọi
  - Nếu $S_k \neq \emptyset$:
    - Cho a_k lần lượt nhận các giá trị trong S_k
    - Gọi đệ quy xây dựng phần tử a_{k+1}

Ví dụ: "Nhị phân"

- Liệt kê mọi chuỗi nhị phân độ dài N
- Cấu hình A = (a_1, a_2,... a_N)
- Các giá trị a_k có thể nhận = S_k = { 0, 1 }

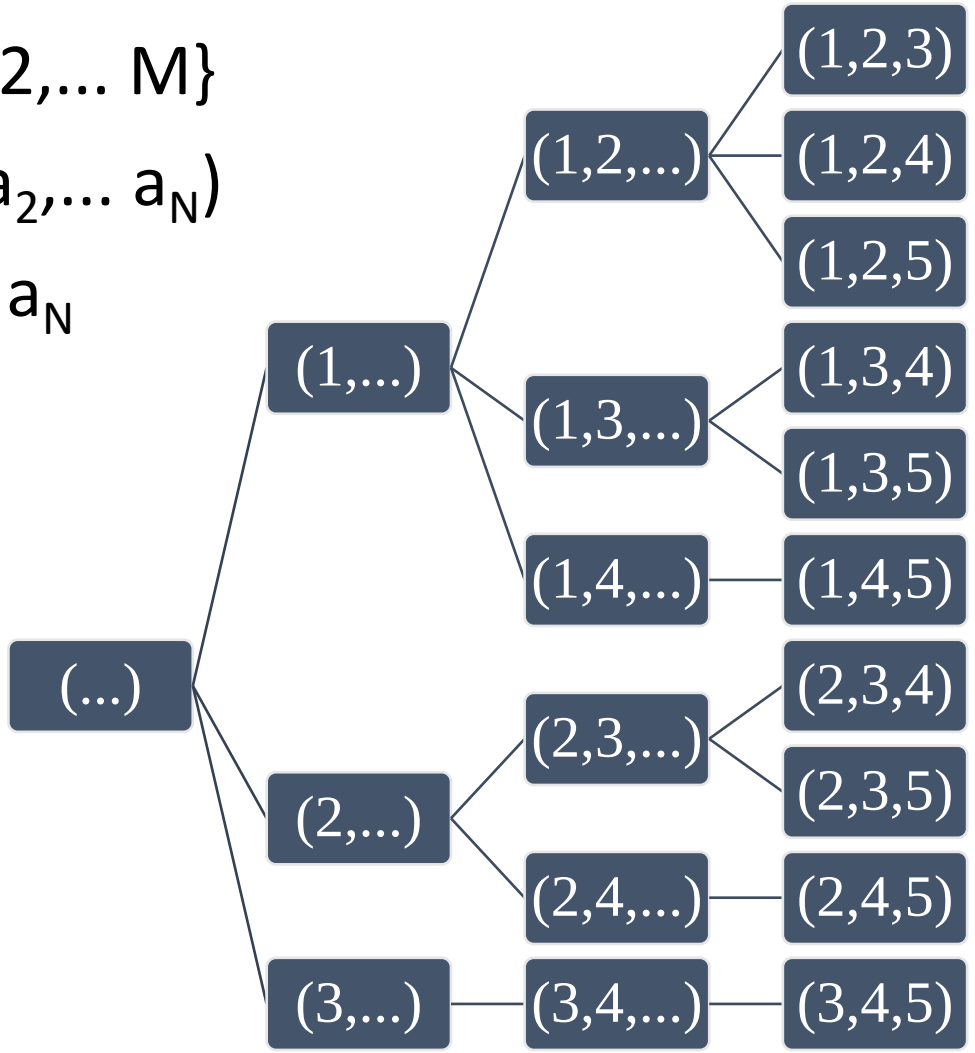


Ví dụ: "Nhị phân"

```
#include <iostream>
using namespace std;

const int MAX = 100;

int a[MAX], n;

void print(int n) {
    for (int i = 1; i <= n; i++) cout << a[i];
    cout << endl;
}

void gen(int k) {
    if (k > n) { print(n); return; }
    a[k] = 0; gen(k+1);
    a[k] = 1; gen(k+1);
}

int main() {
    n = 5;
    gen(1);
}
```

Ví dụ: "Tập con"

- Liệt kê mọi cách chọn N phần tử trong tập M phần tử
- Đơn giản hóa: đặt M = {1, 2,... M}
- Cấu hình tập hợp A = (a_1, a_2,... a_N)
- Đơn giản hóa: $a_1 < a_2 <...< a_N$
- S_k = { a_{k-1}+1,... M-N+k }



Ví dụ: "Tập con"

```
#include <iostream>
using namespace std;

const int MAX = 100;

int a[MAX], n, m;

void print(int n) {
    for (int i = 1; i <= n; i++) cout << a[i]; cout << endl;
}

void gen(int k) {
    if (k > n) { print(n); return; }
    for (int i = a[k-1] + 1; i <= m+n-k; i++) {
        a[k] = i; gen(k+1);
    }
}

int main() {
    n = 3; m = 5; a[0] = 0;
    gen(1);
}
```

Ví dụ: "Tập con" – Bài tập ứng dụng

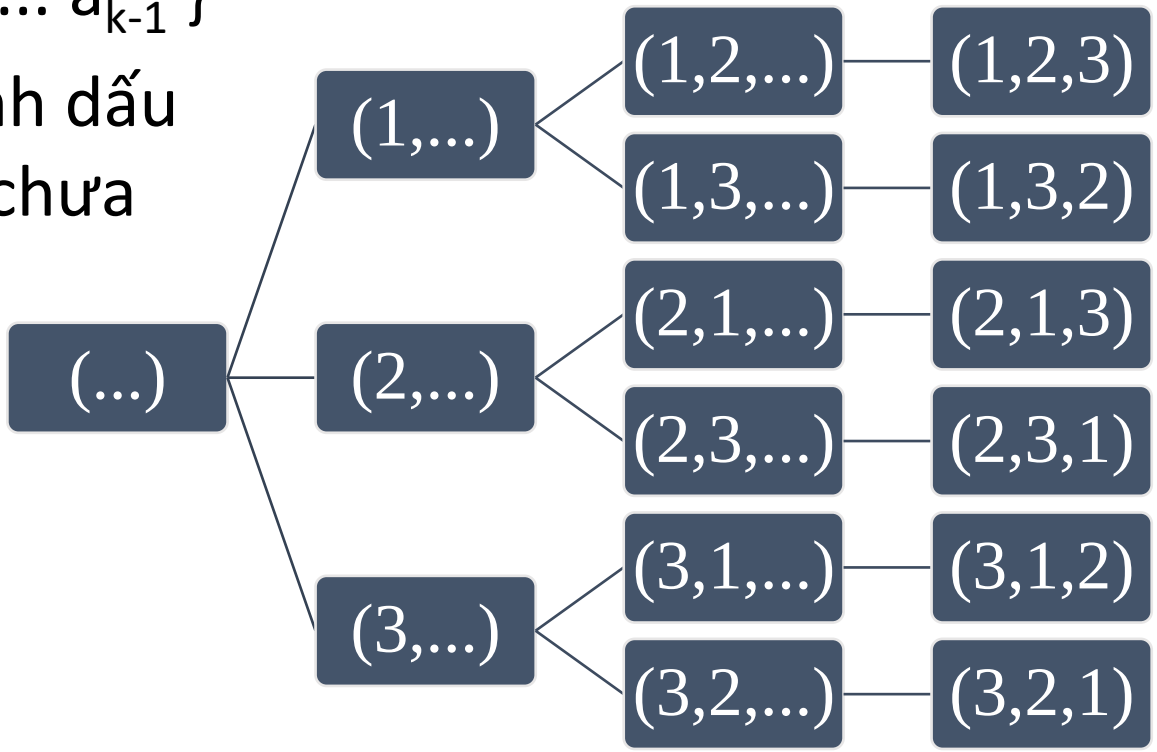
Xét các số nguyên dương nhỏ hơn 20: 1, 2,..., 20.

Có bao nhiêu cách chọn bộ 5 số trong 20 số trên thỏa mãn đồng thời các điều kiện:

- Số số chẵn trong bộ nhiều hơn số số lẻ
- Có ít nhất một số chia hết cho 5
- Có ít nhất một cặp số mà tích chia hết cho 8
- Không có quá 3 số lớn hơn 13

Ví dụ: "Hoán vị"

- Liệt kê mọi hoán vị của (1,2,...,N)
- Cấu hình A = (a_1, a_2,... a_N)
- Giá trị a_k khác với những số nằm trước
- S_k = { 1..N } - { a_1, a_2,... a_{k-1} }
- Dùng một mảng đánh dấu xem giá trị đã dùng chưa



Ví dụ: "Hoán vị"

```
#include <iostream>
using namespace std;

const int MAX = 100;

int a[MAX], n;

bool b[MAX];    // mảng đánh dấu, true nghĩa là chưa dùng

// in cấu hình A
void print(int n) {
    for (int i = 1; i <= n; i++) cout << a[i];
    cout << endl;
}
```

Ví dụ: "Hoán vị"

```
// sinh phần tử thứ k
void gen(int k) {
    // nếu đã sinh được n phần tử thì in ra và thoát
    if (k > n) {
        print(n); return;
    }
    // chọn giá trị cho a[k]
    for (int i = 1; i <= n; i++)
        if (b[i]) {              // nếu chưa đánh dấu
            b[i] = false;        // đánh dấu
            a[k] = i; gen(k+1); // chọn giá trị và sinh tiếp
            b[i] = true;         // bỏ đánh dấu
        }
}
int main() {
    n = 4;
    for (int i = 1; i <= n; i++) b[i] = true;
    gen(1);
}
```

Ví dụ: "Hoán vị" – Bài tập ứng dụng

Hãy điền các số từ 1 đến 9 vào chỗ trống dưới đây, mỗi ô một chữ số và mỗi chữ số chỉ xuất hiện một lần, để được công thức đúng:

_ + _ - _ + _ - _ + _ - _ + _ - _ = 23

Ví dụ: "Phân tích"

- Liệt kê mọi cách phân tích số M thành tổng N số nguyên dương
- Cấu hình A = (a_1, a_2,... a_N)
- Điều kiện: $a_1 + a_2 + ... + a_N = M$
- S_k = { 1 ... X }
- Tính X như thế nào?
  - $(a_1 + a_2 + ... + a_{k-1}) + a_k + (a_{k+1} + ... + a_N) = M$
  - $P = a_1 + a_2 + ... + a_{k-1}$ (giá trị này đã biết)
  - $Q = a_{k+1} + ... + a_N \geq N-K$ (vì mỗi số a_j đều nguyên dương)
  - Suy ra: $1 \leq X \leq M-P-N+K$

Ví dụ: "Đặt hậu"

- Liệt kê mọi cách đặt N quân hậu lên bàn cờ N x N để hai quân bất kỳ không ăn nhau
- Cấu hình A = (a_1, a_2,... a_N)
  - Mỗi dòng tất nhiên chỉ có một quân hậu
  - Ta quan tâm đến vị trí cột của quân hậu
  - Trong đó a_k là vị trí cột của quân hậu đặt trên dòng thứ k
- S_k = ?
- Các ràng buộc:
  - Không cùng cột: $a_k \neq a_i$
  - Không cùng đường chéo chính: $(a_k - k) \neq (a_i - i)$
  - Không cùng đường chéo phụ: $(a_k + k) \neq (a_i + i)$

Phần 3

Nhánh cận

Nhánh cận

- Tiếng Anh: Branch and Bound / Branch and Cut (cắt nhánh)
- Chiến lược tìm kiếm tối ưu tổ hợp:

$$z = min\{f(x)|x \in X\}$$

hoặc:

$$\bar{x} = argmin\{f(x)|x \in X\}$$

- "Tìm phương án tối ưu trong mọi tổ hợp nghiệm"
- Ứng dụng:
  - Định tuyến
  - Lập lịch
  - Cấp phát tài nguyên
  - ...

Nhánh cận

- Quay lui: (bản chất là) quá trình tìm kiếm theo chiều sâu
  - Đi theo chiều sâu (xác định dần các giá trị của x_k)
  - Quay lui (khi không còn giá trị x_k phù hợp)
- Nhánh cận: đưa ra quyết định quay lui sớm nếu nhánh hiện tại không "tốt"
  - Thế nào là "tốt"? Nhánh hiện tại không có khả năng ra nghiệm tối ưu hơn phương án đã biết
  - Quá trình quay lui:
    - Cấu hình đề cử A = (a_1, a_2,... a_N) sẽ được xây dựng dần dần
    - Cần xây dựng thành phần a_K, A' = (a_1, a_2,... a_{K-1})
    - Xây dựng hàm đánh giá p(A') xem có nên đi tiếp không
    - Nếu kì vọng p(A') thấp quá, ta sẽ không đi tiếp (cắt nhánh sớm)

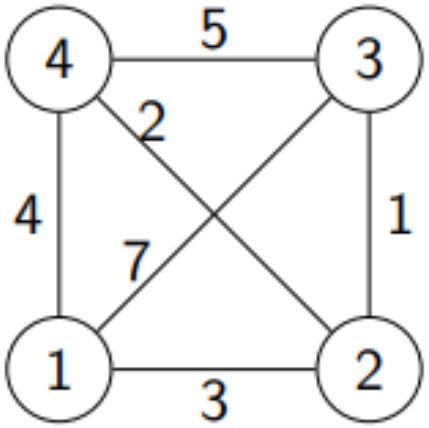
Nhánh cận

- Cấu hình đầu tiên là rỗng: A = (), $z = +\infty$
- Tìm cách xây dựng dần dần các phần tử a_1, a_2,... a_N
- Quy tắc xây dựng phần tử a_k:
  - Nếu k > N: cấu hình A đã hoàn chỉnh
    - Ghi nhận z mới nếu f(A) < z
    - Quay lui
  - Xây dựng tập S_k chứa mọi giá trị có thể của a_k
  - Nếu $S_k = \emptyset$, quay lui trở về hàm gọi
  - Nếu $S_k \neq \emptyset$:
    - Cho a_k lần lượt nhận các giá trị trong S_k
    - Nếu p(A) < z thì gọi đệ quy xây dựng phần tử a_{k+1}

Dễ thấy: nhánh cận dựa trên quay lui

Traveling Salesman Problem

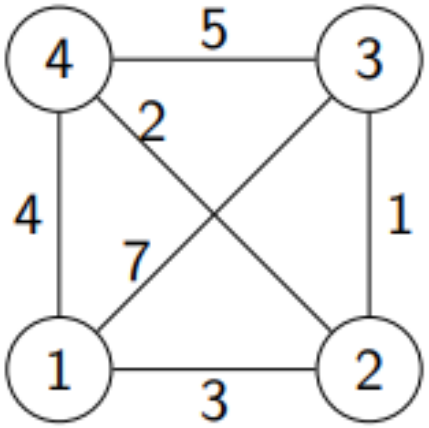
- Có N địa điểm và khoảng cách giữa từng cặp địa điểm
- Người bán hàng xuất phát từ một địa điểm và đi thăm tất cả các địa điểm còn lại mỗi địa điểm đúng một lần và trở về địa điểm ban đầu
- Xác định lộ trình tốt nhất (tổng quãng đường nhỏ nhất)
- $A = (a_1, a_2, \ldots a_N)$, lộ trình: $a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_N \rightarrow a_1$
- $F(A) = c(a_1, a_2) + c(a_2, a_3) + \ldots + c(a_N, a_1)$



$$\Rightarrow c = \begin{pmatrix} 0 & 3 & 7 & 4 \\ 3 & 0 & 1 & 2 \\ 7 & 1 & 0 & 5 \\ 4 & 2 & 5 & 0 \end{pmatrix}$$

Traveling Salesman Problem

- Giả sử đã đi được đến điểm K:
  - A' = (a_1, a_2,... a_K), lộ trình: $a_1 \rightarrow a_2 \rightarrow ... \rightarrow a_{K-1} \rightarrow a_K$
  - F(A') = $c(a_1, a_2) + c(a_2, a_3) + ... + c(a_{K-1}, a_K)$
- Xây dựng kì vọng p(A') như thế nào?
  - Còn phải đi (N-K+1) quãng đường
  - Giả sử c_{min} là khoảng cách ngắn nhất giữa hai địa điểm
  - $p(A) = c(a_1, a_2) + c(a_2, a_3) + ... + c(a_{K-1}, a_K) + (N-K+1) \times c_{min}$



$$\Rightarrow c = \begin{pmatrix} 0 & 3 & 7 & 4 \\ 3 & 0 & 1 & 2 \\ 7 & 1 & 0 & 5 \\ 4 & 2 & 5 & 0 \end{pmatrix}$$

Phần 4

Bài tập

Bài tập

1. Hàm f(N) với số N nguyên được định nghĩa như sau:
   f(0)=0 f(1)=1 f(2)=2
   f(3k)=f(2k), với k>0
   f(3k+1)=f(2k)+f(2k+1), với k>0
   f(3k+2)=f(2k)+f(2k+1)+f(2k+2), với k>0
   Nhiệm vụ: Nhập số N và in ra giá trị của f(N).
2. Chuỗi tam phân là chuỗi chỉ gồm những kí tự 0, 1 hoặc 2. Chuỗi tam phân không lặp là chuỗi tam phân mà không có hai chuỗi con liên tiếp giống nhau. Hãy nhập số nguyên dương N và liệt kê mọi chuỗi tam phân có độ dài N.
3. Nhập số nguyên dương N. Hãy chỉ ra một dãy tam phân không lặp độ dài N sử dụng ít kí tự 2 nhất.