THUẬT TOÁN ỨNG DỤNG

CẤU TRÚC DỮ LIỆU VÀ THƯ VIỆN

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

Nội dung

- Danh sách tuyến tính
- Tập hợp
- Ánh xạ
- Ngăn xếp
- Hàng đợi
- Sắp xếp

Danh sách tuyến tính

- Lưu trữ các đối tượng theo quan hệ tuyến tính (trước – sau)
- Thao tác: thêm, xóa, tìm kiếm

List

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    list<int> L;
    for(int i = 1; i<=5;i++){
        L.push_back(i);
    }
    list<int>::iterator it;
    it = find(L.begin(),L.end(),3);
    L.insert(it,10);
    for(it = L.begin(); it != L.end(); it++){
        cout << *it << endl;
    }
}
```

Vector

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    vector<int> V(3,100); // initialize 3 elements 100
    for(int v = 0; v <= 10; v++)
        V.push_back(v);

    cout << "vector: ";
    for(int i = 0; i < V.size(); i++){
        cout << V[i] << " ";
    }
}
```

Tập hợp

- Lưu các đối tượng, không trùng nhau
- Thao tác: thêm, xóa, tìm kiếm

Tập hợp

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    set<int> Y;
    for(int i = 1; i <= 10; i++){
        Y.insert(i);
    }
    for(set<int>::iterator it =  Y.begin(); it != Y.end(); it++){
        cout << *it << endl;
    }
    if(Y.find(7) != Y.end())
        cout << "Y contains 7" << endl;
    else
        cout << "Y does not contains 7" << endl;
}
```

Ánh xạ

- Cấu trúc dữ liệu cất trữ các cặp (khóa, giá trị)
- Phục vụ tìm kiếm nhanh với khóa đầu vào

Ánh xạ

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    map<int,int> m;
    for(int i = 1; i <= 5; i++)
    m.insert(pair<int,int>(i,10*i));
    m[6] = 100;
    for(int k = 1; k <= 6; k++) cout << m[k] << endl;

    map<string, string> m1;
    m1["abc"] = "abcabc";
    m1["xyz"] = "xyzxyz";
    string s = "abc";
    cout << m1[s] << endl;
}
```

Ánh xạ

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    map<pair<int,int>, pair<int,int> > m2;
    m2[pair<int,int>(2,5)] = pair<int,int>(20,50);
    m2[pair<int,int>(3,5)] = pair<int,int>(30,50);

    int i = 3;
    int j = 5;
    pair<int,int> p = m2[pair<int,int>(i,j)];
    cout << p.first << "," << p.second << endl;
}
```

Ngăn xếp

- Cấu trúc dữ liệu cất trữ các đối tượng một cách tuyến tính
- Thao tác
  - Thêm 1 phần tử
  - Lấy ra 1 phần tử
- Nguyên tắc: Vào trước – ra sau

Hàng đợi

- Cấu trúc dữ liệu cất trữ các đối tượng một cách tuyến tính
- Thao tác
  - Thêm 1 phần tử
  - Lấy ra 1 phần tử
- Nguyên tắc: vào trước – ra trước

Stack

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    stack<int> S;
    for(int i = 0; i < 5; i++){
        S.push(i);
    }
    while(!S.empty()){
        int v = S.top(); S.pop();
        cout << v << endl;
    }}
```

Queue

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    queue<int> Q;
    for(int i = 0; i < 5; i++){
        Q.push(i);
    }
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        cout << v << endl;
    }
}
```

Sắp xếp

```
#include <algorithm>
#include <iostream>
using namespace std;
int main(){
    int N = 6;
    double a[N] = {1.1, 5.5, 7.7, 2.2, 8.8, 3.3};
    sort(a+3,a+N,greater<double>());// decreasing order
    for(int i = 0; i < N; i++) cout << a[i] << " ";
    cout << endl;
    sort(a,a+N);
    for(int i = 0; i < N; i++) cout << a[i] << " ";
}
```

THUẬT TOÁN ỨNG DỤNG

ĐỆ QUY QUAY LUI

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

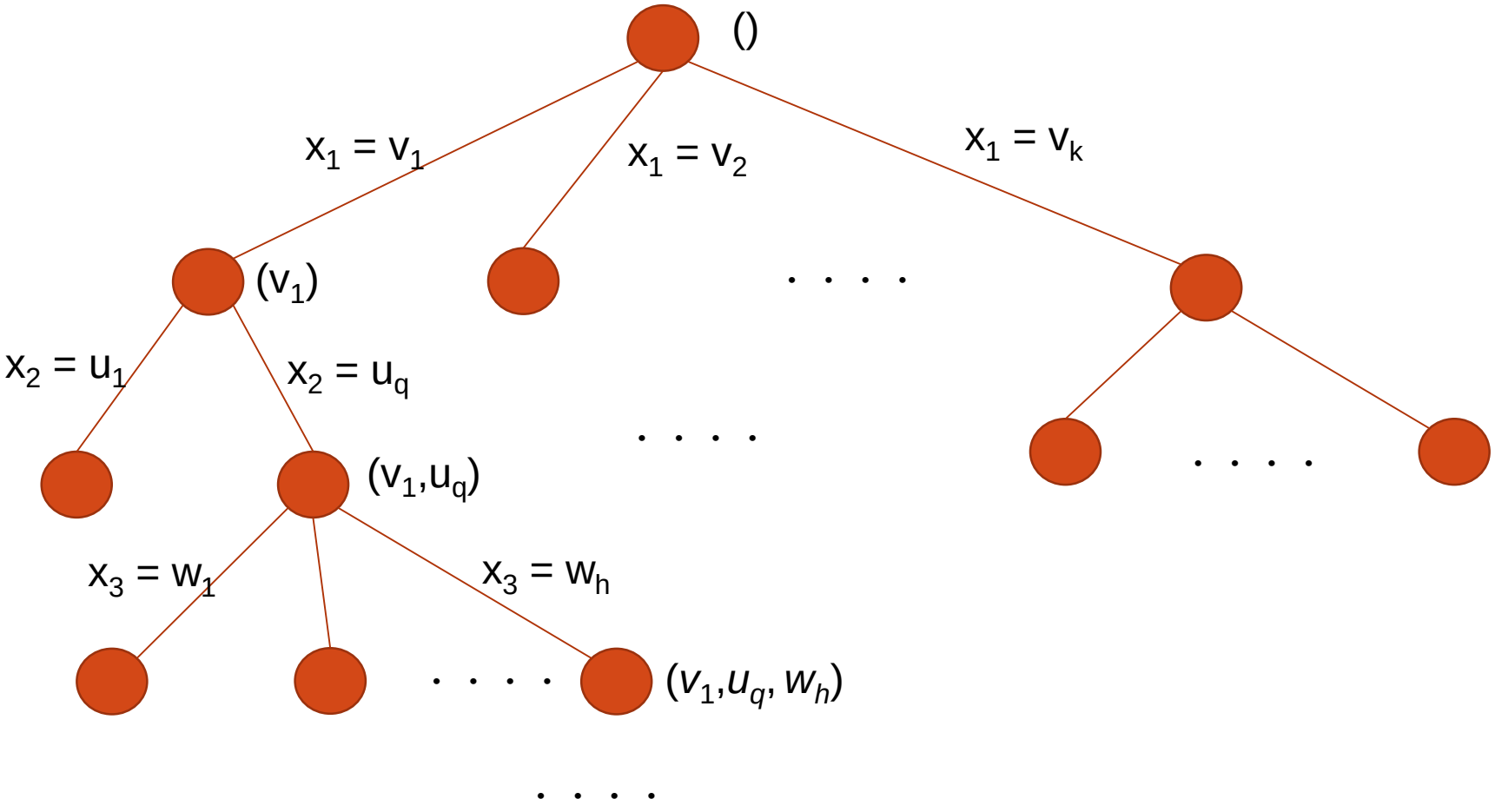
Nội dung

- Đệ quy quay lui
  - Tổng quan đệ quy quay lui
  - Bài toán liệt kê xâu nhị phân
  - Bài toán liệt kê nghiệm nguyên dương phương trình tuyến tính
  - Bài toán liệt kê TSP
  - Bài toán liệt kê hành trình taxi
  - Bài toán liệt kê CBUS
  - Bài toán liệt kê BCA
  - Bài toán liệt kê CVRP
- Thuật toán nhánh và cận
  - Tổng quan nhánh và cận
  - Bài toán tối ưu TSP
  - Bài toán tối ưu hành trình taxi
  - Bài toán tối ưu CBUS
  - Bài toán tối ưu BCA
  - Bài toán tối ưu CVRP

Đệ quy quay lui

- Phương pháp dùng để liệt kê các cấu hình tổ hợp cũng như giải bài toán tối ưu tổ hợp
  - Liệt kê: liệt kê tất cả các bộ $x = (x_1,x_2,\ldots, x_N)$ trong đó $x_i \in A_i$ - tập rời rạc, đồng thời $(x_1,x_2,..., x_N)$ thỏa mãn các ràng buộc C cho trước
  - Tối ưu tổ hợp: trong số các bộ (phương án) $x = (x_1,x_2,\ldots, x_N)$ trong đó $x_i \in A_i$ - tập rời rạc, đồng thời $(x_1,x_2,..., x_N)$ thỏa mãn các ràng buộc C cho trước, cần tìm phương án có $f(x) \rightarrow$ min(max)
- Thử lần lượt từng giá trị cho mỗi biến
  - Chiến lược chọn biến, ví dụ $x_1, x_2, x_3,\ldots, x_N$
  - Chiến lược chọn giá trị cho biến, ví dụ từ nhỏ đến lớn hoặc ngược lại

Đệ quy quay lui



Đệ quy quay lui

- Tại mỗi thời điểm, ta có phương án bộ phận ($x_1 = v_1$, $x_2 = v_2$, …, $x_{k-1} = v_{k-1}$) → cần thử duyệt tiếp các khả năng cho x_k ?

```
try(k){// thử giá trị cho x_k
 forall v ∈ A_k do{
  if(check(v,k)) then{// kiểm tra ràng buộc C của bài toán
    x_k = v;
    [cập nhật các cấu trúc dữ liệu liên quan]
    if(k = N) then solution();
    else try(k+1);
    [khôi phục trạng thái các cấu trúc dữ liệu khi quay lui]
  }
 }
}
```

Liệt kê xâu nhị phân độ dài N

```
#include <iostream>
using namespace std;
#define MAX 100
int N;
int x[MAX];// bieu dien loi phuong an cua bai toan

bool check(int v, int k){
    return true;
}
void solution(){
    for(int i = 1; i <= N; i++) cout << x[i] << " "; cout << endl;
}
```

Liệt kê xâu nhị phân độ dài N

```
void TRY(int k){
    for(int v = 0; v <= 1; v++){
        if(check(v,k)){
            x[k] = v;
            if(k == N) solution();
            else TRY(k+1);
        }
    }
}
int main(){
    N = 3;
    TRY(1);// bat dau thu gia tri cho x[1]
}
```

Liệt kê nghiệm nguyên dương của phương trình tuyến tính

$$X_1 + X_2 + \ldots + X_n = M$$

- Phương án bộ phận (X_1, …, X_{k-1}) có tổng bộ phận là T
- Khởi tạo
  - Phương án bộ phận rỗng: (), T = 0
- Thử giá trị cho X_k
  - $X_1 + X_2 + \ldots + X_{k-1} + X_k + X_{k+1} + \ldots + X_n = M$
  - $X_k = M - T - (X_{k+1} + X_{k+2} + \ldots + X_n)$
  - $1 \le X_k \le M - T - (n - k)$

Liệt kê nghiệm nguyên dương của phương trình tuyến tính

$$X_1 + X_2 + \ldots + X_n = M$$

```
#include <stdio.h>
#define MAX 100

int n,M;
int X[MAX];
int T;// accumulated sum

int check(int v, int k){
    if(k < n) return 1;
    return T + v == M;
}
void solution(){
    for(int i = 1; i <= n; i++) printf("%d ",X[i]); printf("\n");
}
```

Liệt kê nghiệm nguyên dương của phương trình tuyến tính

```
void TRY(int k){
    for(int  v = 1; v <= M-T - n+k; v++){
        if(check(v,k)){
            X[k] = v;
            T = T + X[k];// update incrementally
            if(k == n) solution();
            else TRY(k+1);
            T = T - X[k];// recover when backtracking
        }
    }
}
int main(){
    n = 6; M = 15;
    T = 0;
    TRY(1);
}
```

Liệt kê hành trình giao hàng

- Một shipper nhận hàng từ cửa hàng (điểm 0) và phải đi qua tất cả *N* khách hàng 1, 2, 3, . . ., *N* (mỗi khách hàng đi qua đúng 1 lần) để giao hàng. Hãy liệt kê tất cả các phương án cho shipper

Liệt kê hành trình giao hàng

```
#include <bits/stdc++.h>
#define MAX 100
using namespace std;

int N;
int X[MAX];// permutation 1,2,...,N
int appear[MAX];// appear[v] = 1 indicates that v has appeared

void solution(){
   for(int i = 1; i <= N; i++) cout << X[i] << " "; cout << endl;
}
bool check(int v, int k){
    return appear[v] == 0;
}
```

Liệt kê hành trình giao hàng

```
void TRY(int k){// thu gia tri cho X[k]
    for(int v = 1; v <= N; v++){
        if(check(v,k)){
            X[k] = v;
            appear[v] = 1;// update
            if(k == N) solution();
            else TRY(k+1);
            appear[v] = 0;// recover
        }
    }
}
int main(){
    N = 3;
    for(int v = 1; v <= N; v++) appear[v] = 0;
    TRY(1);
}
```

Liệt kê hành trình đón trả khách cho taxi

- Một taxi xuất phát từ điểm 0 và cần phục vụ đón trả *N* khách 1, 2, …, *N*. Khách thứ i có điểm đón là *i* và điểm trả là *N+i*. Hãy liệt kê tất cả các phương án đón trả cho taxi.

- Giải pháp
  - Đưa về bài toán liệt kê hoán vị
  - Do tính chất vận chuyển taxi nên điểm *i* sẽ nằm ngay trước điểm *i+N* trên mỗi hoán vị của 1, 2, …, 2*N*

  → mỗi hoán vị của 1, 2, …, *N*, chèn thêm *i+N* vào ngay sau giá trị *i (với mọi i = 1, 2, …, N)*

Liệt kê hành trình đón trả khách cho taxi

```
#include <bits/stdc++.h>
#define MAX 100
using namespace std;

int N;
int X[MAX];// permutation 1,2,...,N
int appear[MAX];// appear[v] = 1 indicates that v has appeared

void solution(){
  for(int i = 1; i <= N; i++)
    cout << X[i] << " " << X[i] + N << "  ";
  cout << endl;
}
bool check(int v, int k){
    return appear[v] == 0;
}
```

Liệt kê hành trình đón trả khách cho taxi

```
void TRY(int k){// thu gia tri cho X[k]
    for(int v = 1; v <= N; v++){
        if(check(v,k)){
            X[k] = v;
            appear[v] = 1;// update
            if(k == N) solution();
            else TRY(k+1);
            appear[v] = 0;// recover
        }
    }
}
int main(){
    N = 3;
    for(int v = 1; v <= N; v++) appear[v] = 0;
    TRY(1);
}
```

Liệt kê hành trình đón trả khách cho bus

- Một xe bus xuất phát từ điểm 0 và cần phục vụ đón trả *N* khách 1, 2, …, *N*. Khách thứ i có điểm đón là *i* và điểm trả là *N+i*. Xe bus chở được tối đa *Q* khách cùng một lúc. Hãy liệt kê tất cả các phương án đón trả cho xe bus.

Liệt kê hành trình đón trả khách cho bus

- Một xe bus xuất phát từ điểm 0 và cần phục vụ đón trả *N* khách 1, 2, …, *N*. Khách thứ i có điểm đón là *i* và điểm trả là *N+i*. Xe bus chở được tối đa *Q* khách cùng một lúc. Hãy liệt kê tất cả các phương án đón trả cho xe bus.

- Thiết kế giải pháp
  - Kiểm soát số hành khách trên xe bằng biến q: số khách đang có mặt trên xe
    - Mỗi khi hành trình đi qua 1 điểm đón thì tăng q lên 1
    - Mỗi khi hành trình đi qua 1 điểm trả thì giảm q đi 1 đơn vị

Liệt kê hành trình đón trả khách cho bus

```
#include <bits/stdc++.h>
using namespace std;
#define MAX_N 100
int N;// so khach
int Q;// so cho tren bus cho hanh khach
int X[2*MAX_N + 1];// bieu dien phuong an lo trinh X[1], X[2], . . . X[2N]
int q;// so khach thuc su dang co tren xe ung voi phuong an bo phan hien tai
bool appear[2*MAX_N+1];
bool check(int v, int k){
    if(appear[v]) return false;
    if(v <= N){// v is pickup
        if(q >= Q) return false;
    }else{// v > N means drop-off
        if(!appear[v-N]) return false;
    }
    return true;
}
```

Liệt kê hành trình đón trả khách cho bus

```
void solution(){
    for(int i = 1; i <= 2*N; i++) cout << X[i] << " "; cout << endl;
}
void TRY(int k){// thu gia tri cho X[k]
    for(int v = 1; v <= 2*N; v++){
        if(check(v,k)){
            X[k] = v;
            appear[v] = true;
            if(v <= N) q++; else q--;// update q incrementally
            if(k == 2*N) solution();
            else TRY(k+1);
            appear[v] = false;
            if(v <= N) q--; else q++;// recover status q
        }
    }
}
```

Liệt kê hành trình đón trả khách cho bus

```
int main(){
    N = 3; Q = 2;
    q = 0;
    for(int v = 1; v <= 2*N; v++) appear[v] = false;
    TRY(1);
}
```

DIGITS

- Write C program that reads an integer value ***N*** from stdin, prints to stdout the number Q ways to assign values 1, 2, …, 9 to characters I, C, T, H, U, S, K (characters are assigned with different values)

ICT – K62 + HUST = N

stdin	stdout
1234	24

DIGITS

```
#include <stdio.h>

int X[7];// X[0] = I, X[1] = C, X[2] = T, X[4] = H, X[5] = U, X[6] = S, X[3] = K
int appeared[10];
int ans,N;

void solution(){
    int T = X[0]*100 + X[1]*10 + X[2] - X[3]*100 - 62 +
    X[4]*1000 + X[5]*100 + X[6]*10 + X[2];
    if(T == N){
        ans++;
        //printf("%d%d%d - %d62 + %d%d%d%d\n",X[0],X[1],X[2],X[3],X[4],X[5],X[6],X[2]);
    }
}
void init(){
    for(int v = 1; v <= 9; v++) appeared[v] = 0;
}
```

DIGITS

```
void TRY(int k){
    for(int v = 1; v <= 9; v++){
        if(appeared[v] == 0){
            X[k] = v;
            appeared[v] = 1;
            if(k == 6){
                solution();
            }else{
                TRY(k+1);
            }
            appeared[v] = 0;
        }
    }
}
```

DIGITS

```
void solve(){
    scanf("%d",&N);
    init();
    ans = 0;
    TRY(0);
    printf("%d",ans);
}
int main(){
    solve();
}
```

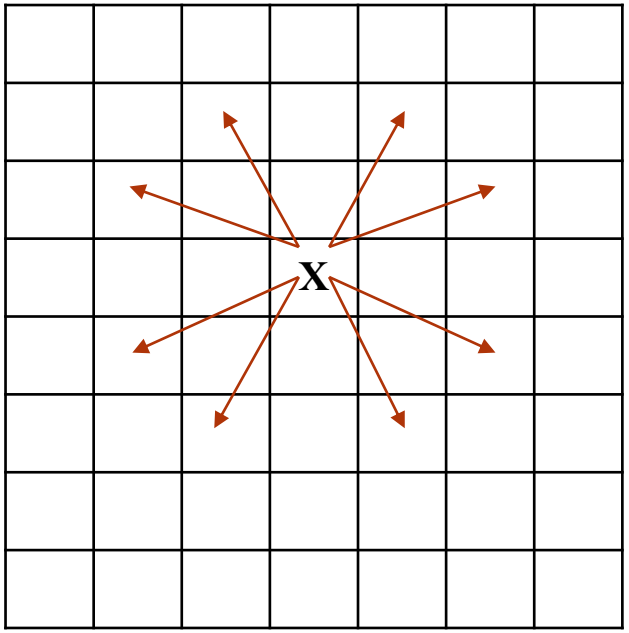
KNIGHT

- Cho một bàn cờ quốc tế N x N. Một quân mã xuất phát từ ô (i, j). Hãy liệt kê tất cả các hành trình cho quân mã di chuyển (theo luật cờ vua) đến tất cả các ô của bàn cờ, mỗi ô đúng 1 lần
- Input
  - Duy nhất 1 dòng chứa N, i, j ($1 \le i, j \le N \le 8$)
- Output
  - Ghi ra dãy tọa độ các ô trên hành trình của quân mã

KNIGHT

- Từ mỗi ô (*i*,*j*), quân mã có thể nhảy đến các ô (*i*+*di*, *j*+*dj*) với (*di*,*dj*) ∈{(1,2), (1,-2), (-1,2), (-1,-2), (2,1), (2,-1),(-2,1),(-2,-1)}
- Biểu diễn phương án
  - Xi[1..N*N], Xj[1..N*N]
  - (Xi[1],Xj[1]) → (Xi[2],Xj[2]) → ..
- Mảng đánh dấu mark[i][j] = true

có nghĩa ô (i,j) đã được đi đến



KNIGHT

```
#include <bits/stdc++.h>
using namespace std;
#define MAX 30
int di[8] = {1,  1, 2,  2, -1, -1, -2, -2};
int dj[8] = {2, -2, 1, -1, 2, -2,   1, -1};

bool mark[MAX][MAX];
int Xi[MAX*MAX];
int Xj[MAX*MAX];

int N;
int I,J; // start cell (I,J)
bool found;
int cnt;
```

KNIGHT

```
bool check(int r, int c){
    if(r < 1 || r > N) return false;
    if(c < 1 || c > N) return false;
    return !mark[r][c];
}
void solution(){
    found = true;
    cnt++;
    for(int k = 1; k <= N*N; k++) cout << "(" << Xi[k] << "," << Xj[k] << ") ";
    cout << endl;
}
```

KNIGHT

```
void TRY(int k){// current cell (Xi[k-1],Xj[k-1])
    for(int q = 0; q < 8; q++){
        if(check(Xi[k-1] + di[q], Xj[k-1] + dj[q])){
            Xi[k] = Xi[k-1] + di[q];
            Xj[k] = Xj[k-1] + dj[q];
            mark[Xi[k]][Xj[k]] = true;// update
            if(k == N*N) solution();
            else TRY(k+1);
            mark[Xi[k]][Xj[k]] = false;// recover
        }
    }
}
```

KNIGHT

```
int main(){
    cin >> N >> I >> J;
    for(int i = 1; i <= N; i++)
        for(int j = 1; j <= N; j++)
            mark[i][j] = false;
    Xi[1] = I; Xj[1] = J;// starting cell
    mark[I][J] = true;
    found = false;
    cnt = 0;
    TRY(2);
    cout << cnt;
}
```

Liệt kê phương án phân công giảng dạy

- Có *n* môn học 1, 2, …, *n* cần được phân cho *m* giáo viên 1, 2, …, *m*. Mỗi giáo viên có danh sách các môn mà người này có thể giảng dạy (tùy thuộc chuyên ngành hẹp của giáo viên).
- Vì đã được xếp thời khóa biểu từ trước nên giữa n môn học này sẽ có các cặp 2 môn trùng thời khóa biểu và do đó không thể được phân công cho cùng một giáo viên được và được thể hiện bởi 0-1 ma trận $A(i,j)$ trong đó $A(i,j) = 1$ có nghĩa môn *i* và *j* trùng thời khóa biểu.
- Hãy liệt kê tất cả các phương án phân công giảng dạy các môn cho giáo viên

Liệt kê phương án phân công giảng dạy

- Dữ liệu đầu vào
  - Dòng 1: n, m ($1 \le m < n \le 10$)
  - Dòng i+1 ($i = 1,\ldots,n$): k, t_1, t_2, …, t_k trong đó k là số giáo viên có thể dạy môn i và t_1, t_2, …, t_k là các giáo viên có thể dạy môn i
  - Dòng i+n+1 ($i = 1,\ldots,n$): chứa các phần tử dòng thứ i của ma trận A

Liệt kê phương án phân công giảng dạy

```
#include <bits/stdc++.h>
#define MAX_N 100
#define MAX_M 30
using namespace std;
// input data structures
int N;// number of course
int M;// number of teachers
int sz[MAX_N];// sz[c] is the number of possible teachers for course c
int t[MAX_N][MAX_M];// t[c][i]: the ith teacher that can teach course c
int h[MAX_N];// h[c] is the number of hours of course c each week
int A[MAX_N][MAX_N];// A[i][j] = 1 indicates that course i and j are
conflict
int f[MAX_M];
int cnt; // number of solutions;
// variables
int X[MAX_N];
```

Liệt kê phương án phân công giảng dạy

```
void input(){
    cin >> N >> M;
    for(int i = 1; i <= N; i++)
        cin >> h[i];
    for(int i = 1; i <= N; i++){
        cin >> sz[i];
        for(int j = 0; j < sz[i]; j++)
            cin >> t[i][j];
    }
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++)
            cin >> A[i][j];
    }
}
```

Liệt kê phương án phân công giảng dạy

```
int check(int v, int k){
    for(int i = 1; i <= k-1; i++){
        if(A[i][k] && v == X[i]) return 0;
    }
    return 1;
}
void solution(){
    cnt++;
    cout << "solution " << cnt << endl;
    for(int t = 1; t <= M; t++){
        cout << "course of teacher " << t << ": ";
        for(int i = 1; i <= N; i++) if(X[i] == t) cout << i << ", ";
        cout << " hour = " << f[t] << endl;
    }
    cout << "--------------------" << endl;
}
```

Liệt kê phương án phân công giảng dạy

```
void TRY(int k){
    for(int i = 0; i < sz[k]; i++){
        int v = t[k][i];
        if(check(v,k)){
                X[k] = v;
                f[v] += h[k];
                if(k == N){
                    solution();
                }else{
                    TRY(k+1);
                }
                f[v] -= h[k];
            }

    }
}
```

Liệt kê phương án phân công giảng dạy

```
void solve(){
    for(int i = 1; i <= M; i++) f[i] = 0;
    cnt = 0;
    TRY(1);
 }
int main(){
    input();
    solve();
}
```

Bài toán CVRP

- Một đội gồm *K* xe tải giống nhau cần được phân công để vận chuyển hàng hóa pepsi từ kho trung tâm (điểm 0) đến các điểm giao hàng 1,2,…,N.
- Mỗi xe tải có tải trọng Q (mỗi chuyển chỉ vận chuyển tối đa Q thùng).
- Mỗi điểm giao hàng i có lượng hàng yêu cầu là d[i] thùng
- Cần xây dựng phương án vận chuyển sao cho
  - Mỗi xe đều phải được phân công vận chuyển
  - Mỗi điểm giao chỉ được giao bởi đúng 1 xe
  - Tổng lượng hàng trên xe không vượt quá tải trọng của xe đó
- Cần liệt kê tất cả các phương án vận chuyển

Bài toán CVRP

- Ví dụ N = 3, K = 2

→ có 6 phương án vận chuyển sau

Route[1] = 0 – 1 – 0 Route[2] = 0 – 2 – 3 – 0	Route[1] = 0 – 1 – 2 – 0 Route[2] = 0 – 3 – 0
Route[1] = 0 – 1 – 3 – 0 Route[2] = 0 – 2 – 0	Route[1] = 0 – 2 – 0 Route[2] = 0 – 3 – 1 – 0
Route[1] = 0 – 1 – 0 Route[2] = 0 – 3 – 2 – 0	Route[1] = 0 – 2 – 1 – 0 Route[2] = 0 – 3 – 0

Bài toán CVRP

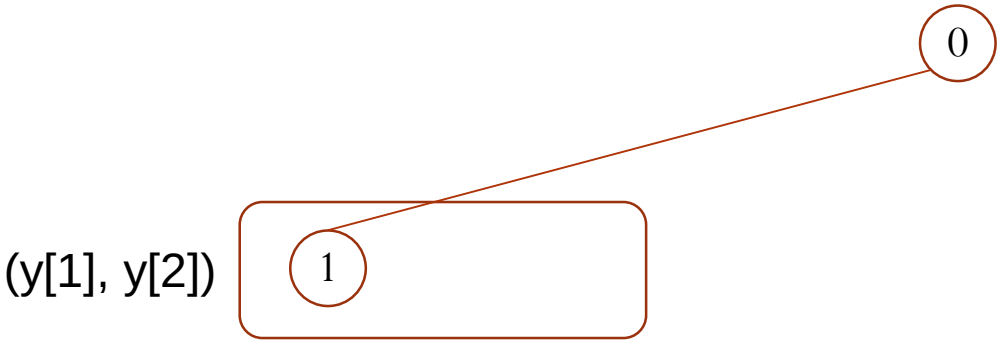
- Thiết kế dữ liệu
  - y[k] điểm giao đầu tiên của xe thứ k (y[k] $\in${1,2,...,N}, với k=1,2,…,K)
  - x[i] là điểm tiếp theo của điểm giao i trên lộ trình (x[i] $\in${0,1,2,…,N}, với i = 1,2,...,N)
  - Do các xe giống hệt nhau nên giả định y[1] < y[2] < … < y[K]
  - visited[v] = true nếu v đã được thăm bởi 1 xe nào đó

Bài toán CVRP

- Chiến lược duyệt
  - Bắt đầu bằng việc duyệt bộ giá trị cho (y[1],. . ., y[K])
  - Với mỗi bộ giá trị đầy đủ của (y[1],. . ., y[K]), bắt đầu duyệt bộ giá trị cho x[1,...,N] xuất phát từ x[y[1]]
  - Mỗi khi thử giá trị x[v] = u cho xe thứ k thì
    - Nếu u > 0 (chưa phải điểm xuất phát) thử duyệt tiếp giá trị cho x[u] vẫn trên chuyến xe thứ k
    - Nếu u = 0 (điểm xuất phát) thì
      - Nếu k = K (đã đủ hết các chuyến cho K) xe và điểm giao nào cũng được thăm thì ghi nhận 1 phương án
      - Ngược lại, thử duyệt tiếp giá trị cho chuyến của xe k+1 bắt đầu bởi cho x[y[k+1]]
  - Biến segments
    - Ghi nhận số chặng (đoạn nối giữa 2 điểm liên tiếp trên đường đi)
    - Khi segments = N+K thì thu được phương án đầy đủ

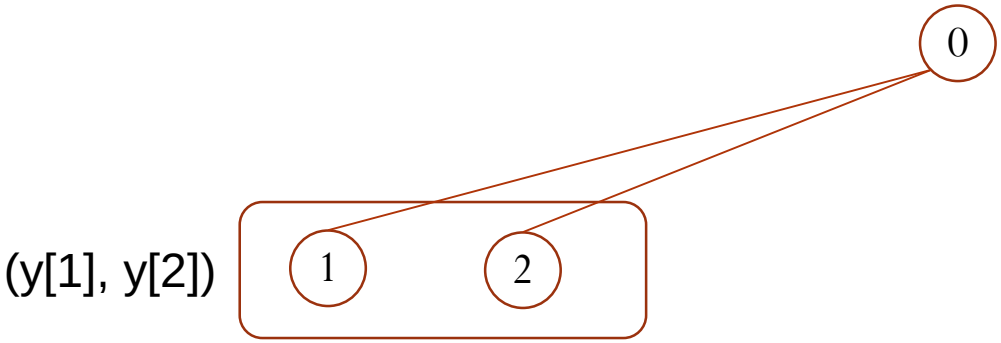
Bài toán CVRP

- K = 2



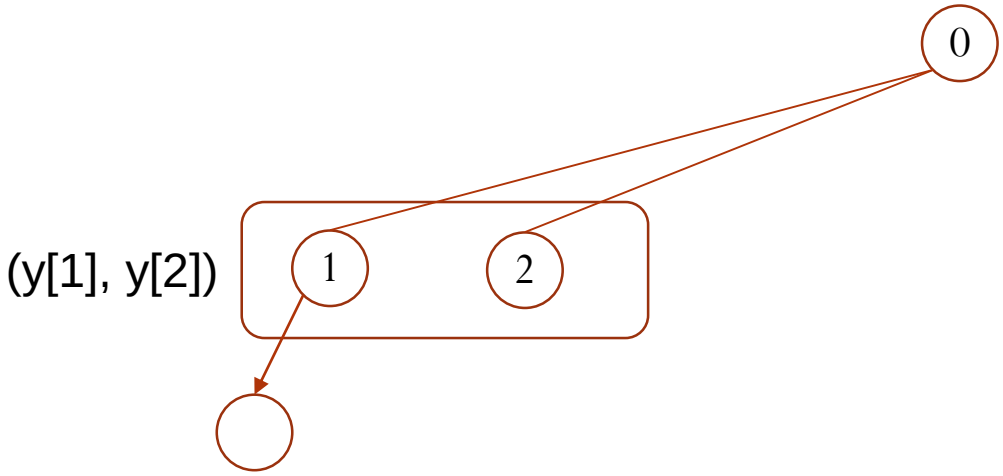
Bài toán CVRP

- K = 2



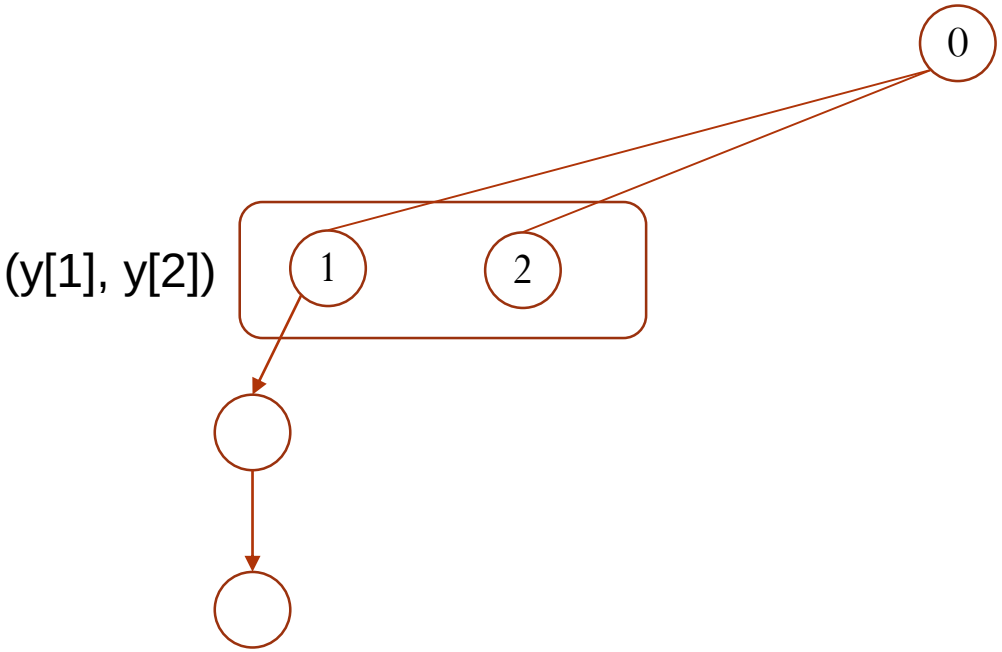
Bài toán CVRP

- K = 2



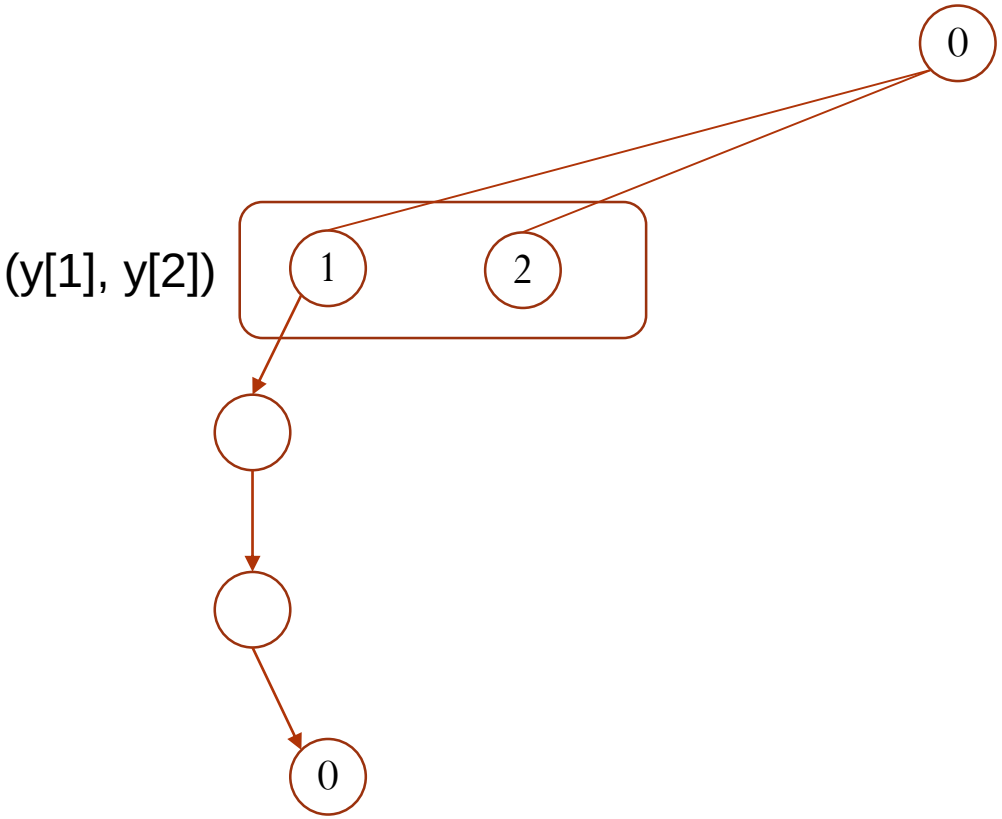
Bài toán CVRP

- K = 2



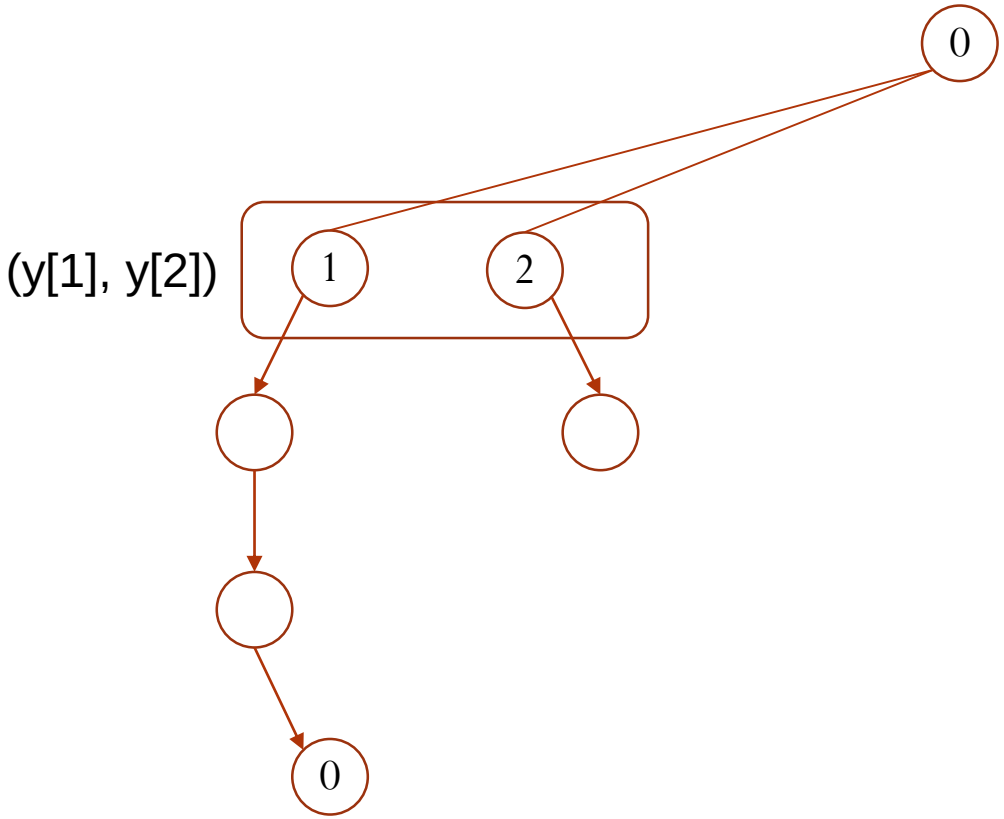
Bài toán CVRP

- K = 2



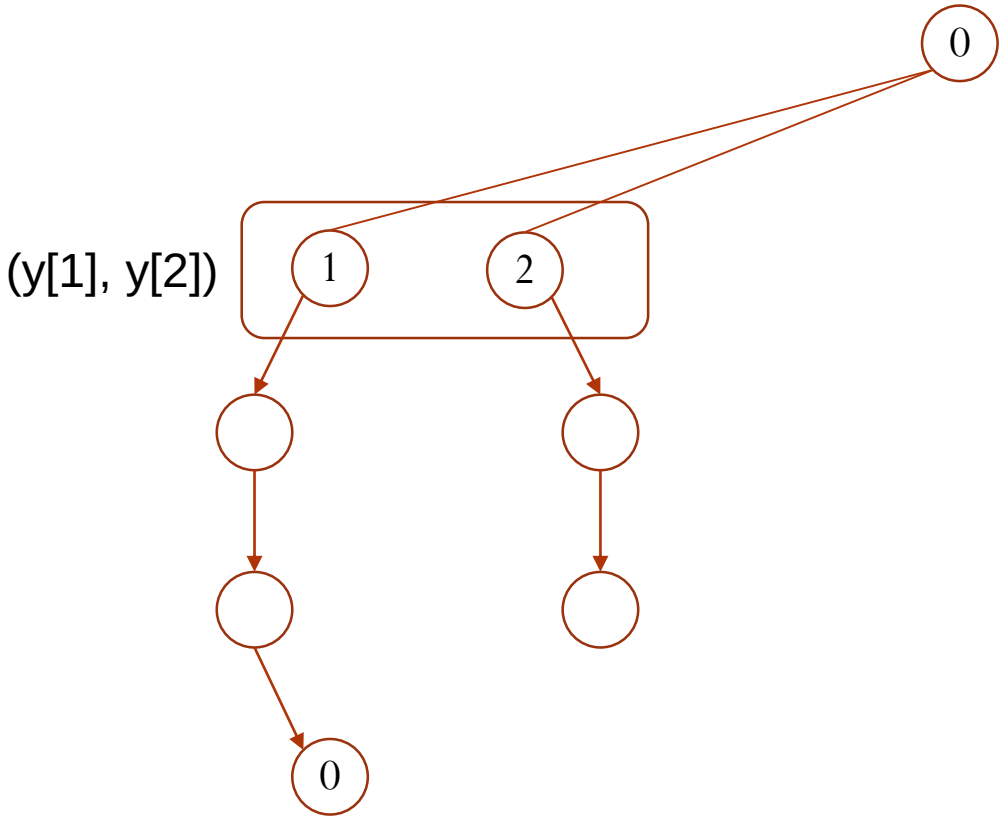
Bài toán CVRP

- K = 2



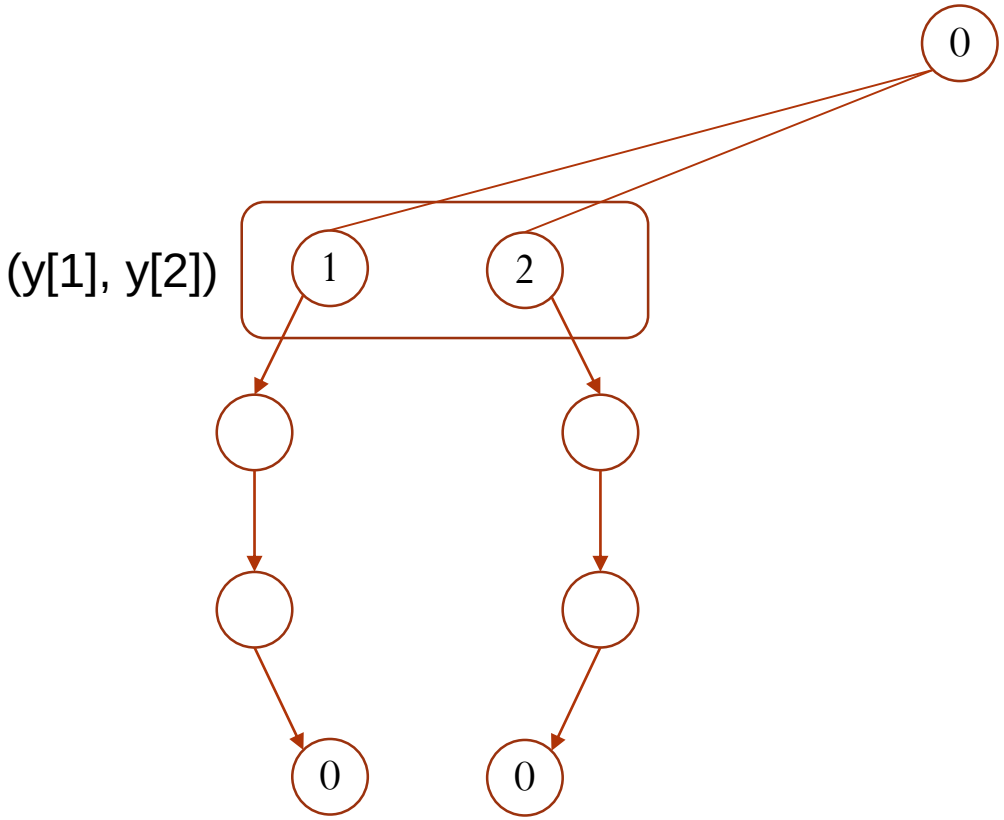
Bài toán CVRP

- K = 2



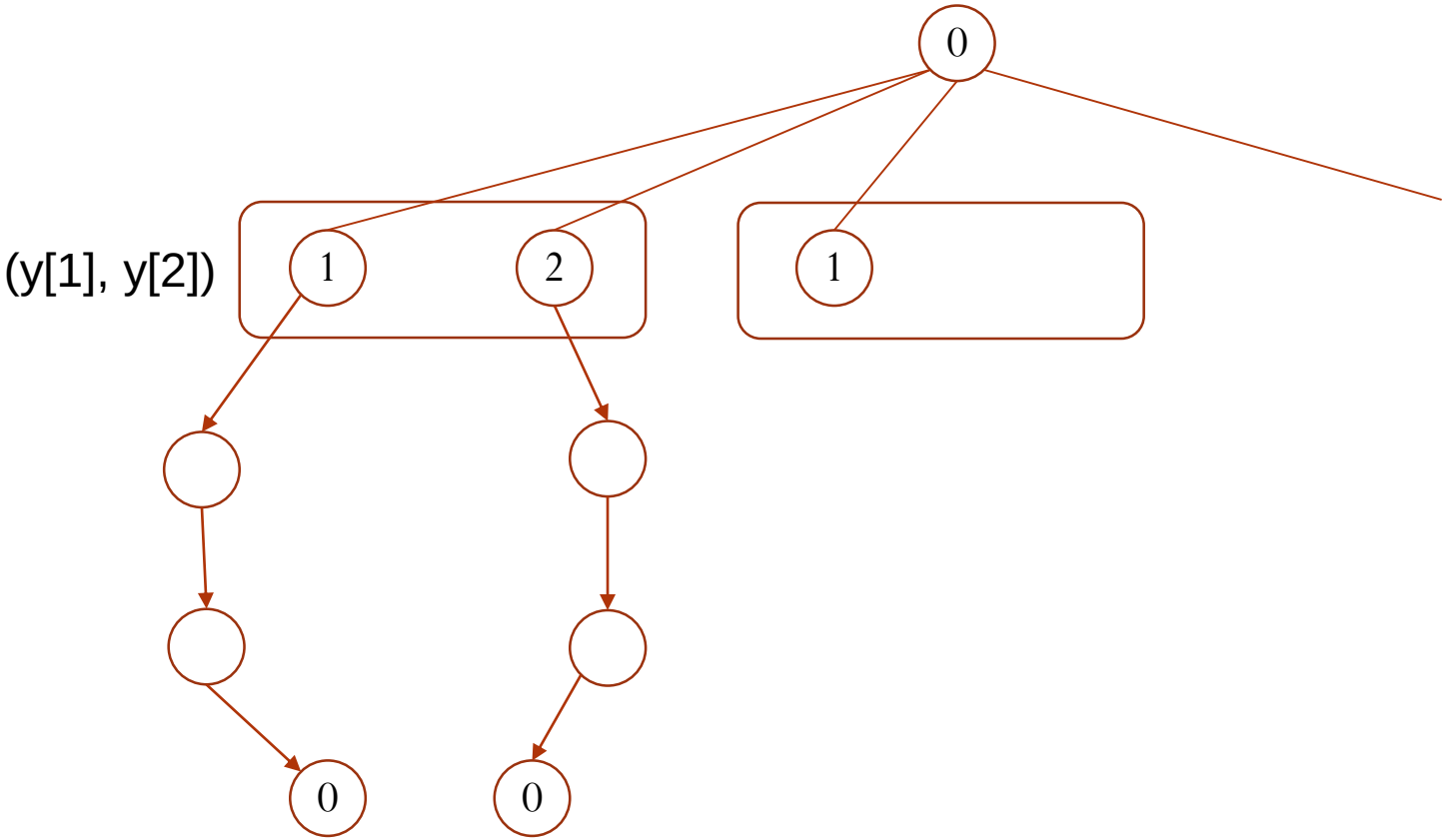
Bài toán CVRP

- K = 2



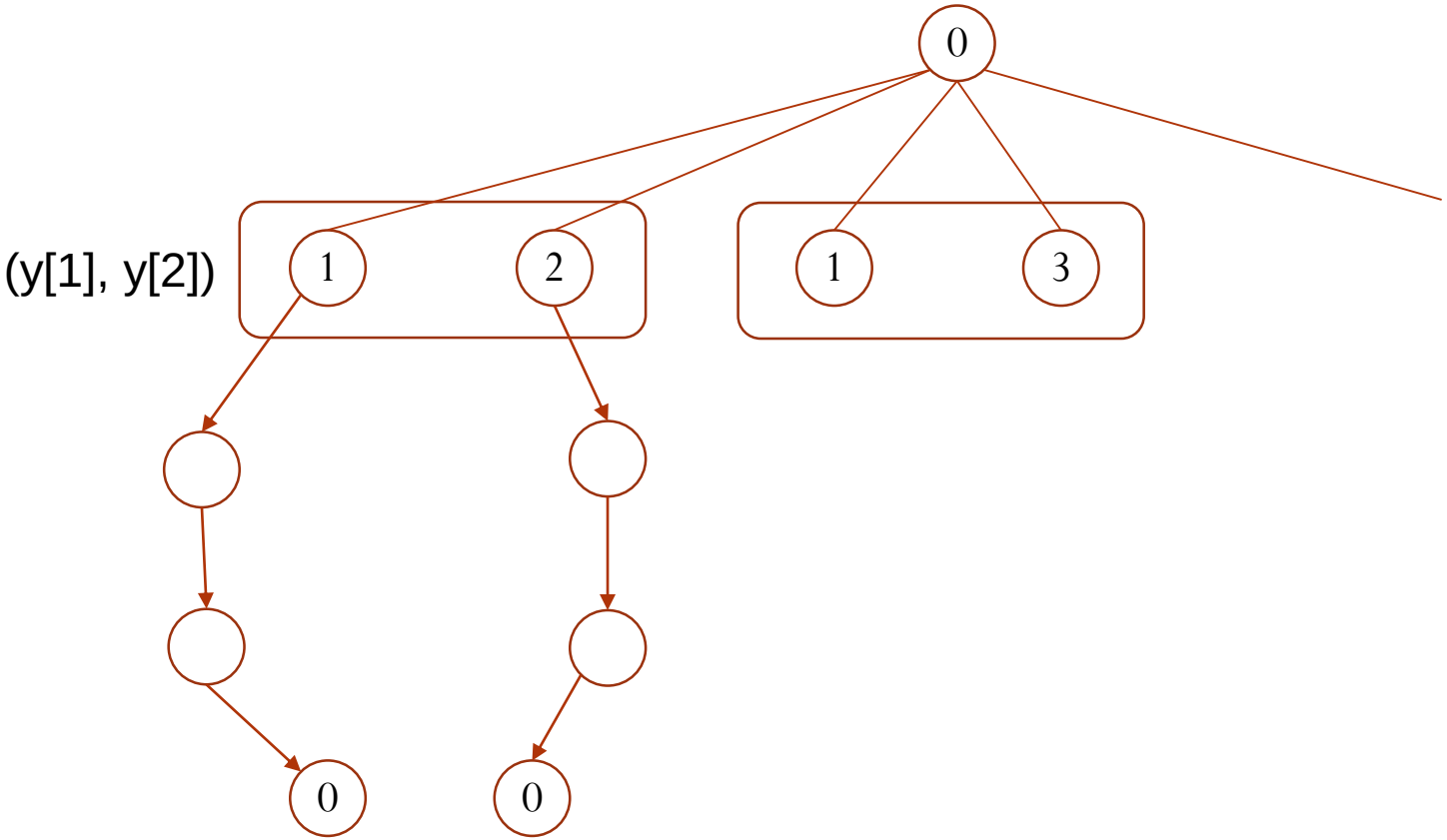
Bài toán CVRP

- K = 2



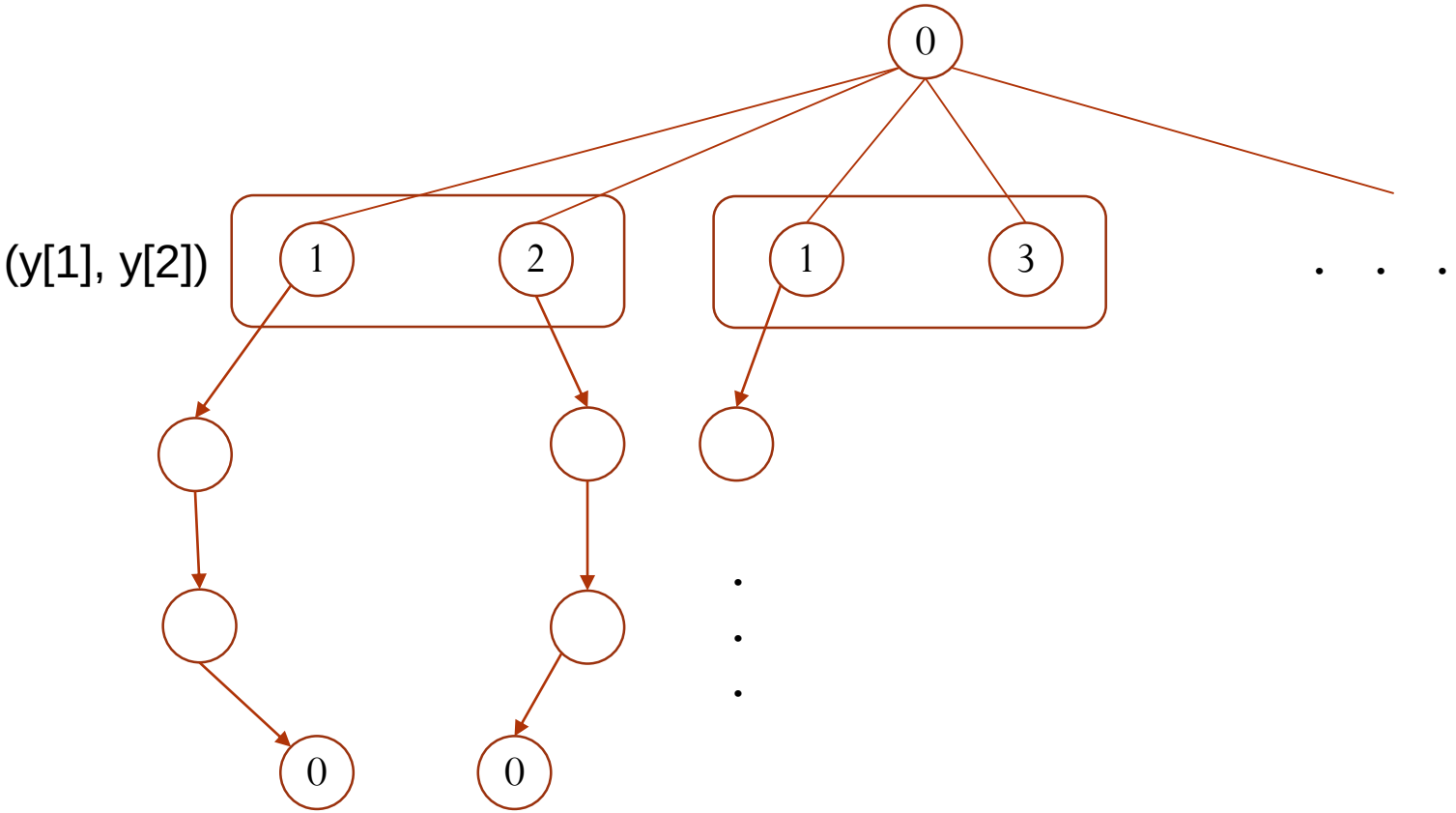
Bài toán CVRP

- K = 2



Bài toán CVRP

- K = 2



Bài toán CVRP

```
#include <stdio.h>
#define MAX 50

int n,K,Q;
int d[MAX];
int x[MAX]; // x[i] is the next point of i (i = 1,...,n), x[i] \in
            // {0,1,...,n}
int y[MAX];// y[k] is the start point of route k
int load[MAX];
int visited[MAX];// visited[i] = 1 means that client point i has been
visited
int segments;// number of segments accumulated
int nbRoutes;
int ans; // records number of solutions
```

Bài toán CVRP

```
void solution(){
    ans++;
    for(int k = 1; k <= K; k++){
        int s = y[k];
        printf("route[%d]:  0 ",k);
        for(int v = s; v != 0; v = x[v]){
            printf("%d ",v);
        }
        printf("0\n");
    }
    printf("--------------------------\n");
}
```

Bài toán CVRP

```
int checkX(int v,int k){
    if(v > 0 && visited[v]) return 0;
    if(load[k] + d[v] > Q) return 0;
    return 1;
}
void input(){
    scanf("%d%d%d",&n,&K,&Q);
    for(int i = 1; i <= n; i++){
        scanf("%d",&d[i]);
    }
    d[0] = 0;
}
```

Bài toán CVRP

```
void TRY_X(int s, int k){
    for(int v = 0; v <= n; v++){
        if(checkX(v,k)){
            x[s] = v;
            visited[v] = 1;  load[k] += d[v]; segments++;
            if(v > 0)   TRY_X(v,k);
            else{
                if(k == K){
                     if(segments == n+nbRoutes) solution();
                }else  TRY_X(y[k+1],k+1);
            }
            segments--;  load[k] -= d[v];   visited[v] = 0;
        }
    }
}
```

Bài toán CVRP

```
int checkY(int v, int k){
    if(v == 0) return 1;
    if(load[k] + d[v] > Q) return 0;
    return !visited[v];
}
```

Bài toán CVRP

```
void TRY_Y(int k){
    for(int v = y[k-1] + 1; v <= n; v++){// 0 < y[1] < y[2] < . . . . < y[K]
        if(checkY(v,k)){
            y[k] = v;
            segments += 1;
            visited[v] = 1;    load[k] += d[v];
            if(k < K){
                TRY_Y(k+1);
            }else{
                nbRoutes = segments;
                TRY_X(y[1],1);// du bo y[1],...,y[K], bat dau duyet cho diem tiep
                              // theo cua y[1]
            }
            load[k] -= d[v];  visited[v] = 0;
            segments -= 1;
        }
    }
}
```

Bài toán CVRP

```
void solve(){
    for(int v = 1; v <= n; v++) visited[v] = 0;
    y[0] = 0;
    ans = 0;
    TRY_Y(1);
    printf(“Number of solutions = %d",ans);
}
int main(){
    input();
    solve();
}
```

Thuật toán nhánh và cận

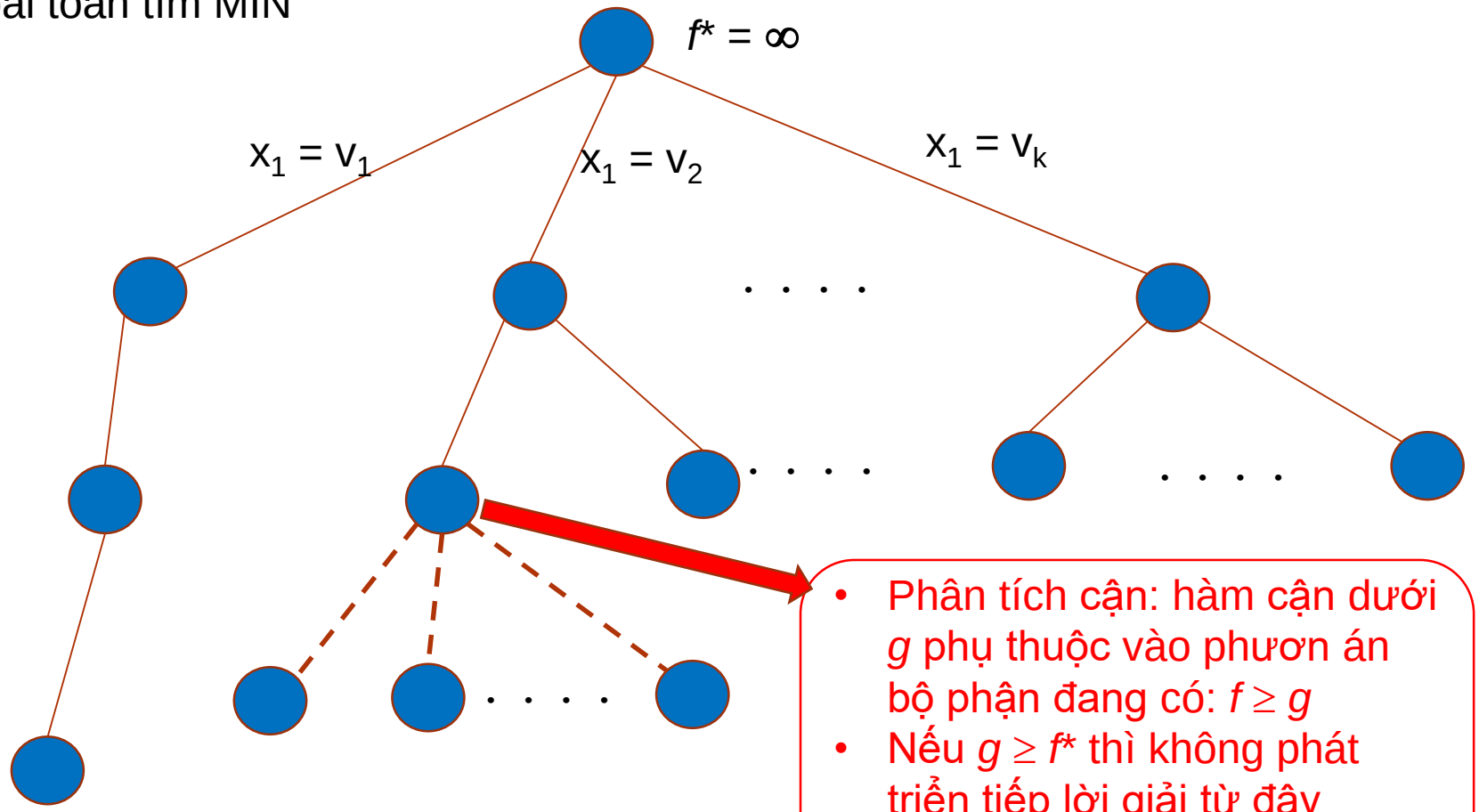
- Một trong số các phương pháp giải bài toán tối ưu tổ hợp
  - Dùng đệ quy quay lui để duyệt toàn bộ không gian lời giải
  - Dùng kỹ thuật phân tích đánh giá cận để cắt bớt nhánh tìm kiếm không có ích

Thuật toán nhánh và cận

- Bài toán tối ưu tổ hợp
  - Phương án $x = (x_1, x_2, \ldots, x_n)$ trong đó $x_i \in A_i$ cho trước
  - Phương án thoả mãn ràng buộc C
  - Hàm mục tiêu $f(x) \rightarrow$ min (max)

Thuật toán nhánh và cận

Xét bài toán tìm MIN



- Phân tích cận: hàm cận dưới *g* phụ thuộc vào phương án bộ phận đang có: $f \geq g$
- Nếu $g \geq f^*$ thì không phát triển tiếp lời giải từ đây

Thuật toán nhánh và cận

- Duyệt nhánh và cận:
  - Phương án bộ phận (a_1,…, a_k) trong đó a_1 gán cho x_1,… a_k gán cho x_k
  - Phương án (a_1,…, a_k, b_{k+1},…,b_n) là một phương án đầy đủ được phát triển từ (a_1,…,a_k) trong đó b_{k+1} gán cho x_{k+1},…, b_n được gán cho x_n
  - Với mỗi phương án bộ phận (x_1,…, x_k), hàm cận dưới $g(x_1,…, x_k)$ có giá trị không lớn hơn giá trị hàm mục tiêu của phương án đầy đủ phát triển từ (x_1,…,x_k)
  - Nếu $g(x_1,…,x_k) \geq f^*$ thì không phát triển lời giải từ (x_1,…,x_k)

```
TRY(k) {
  Foreach v thuộc A_k
      if check(v,k) {
        x_k = v;
        if(k = n) {
            ghi_nhan_cau_hinh;
            cập nhật kỷ lục f*;
         } {
            if g(x1,…,x_k) < f*
              TRY(k+1);
        }
       }
}
Main()
{ f* = ∞;
  TRY(1);
}
```

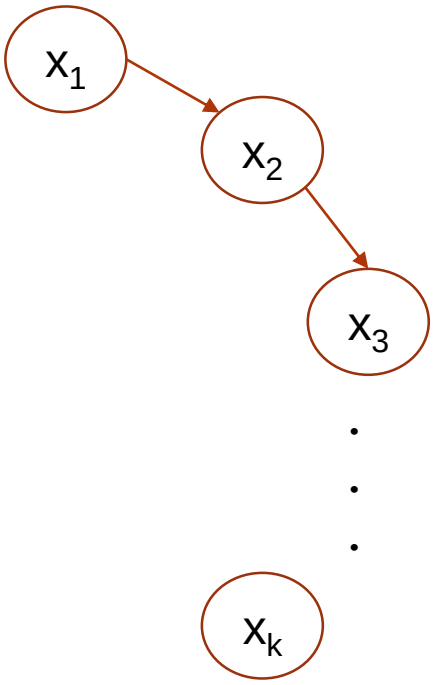
Thuật toán nhánh và cận

- Bài toán TSP
  - c_m là chi phí nhỏ nhất trong số các chi phí đi giữa 2 thành phố khác nhau
  - Phương án bộ phận $(x_1,\ldots, x_k)$
    - Chi phí bộ phận $\bar{f} = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{k-1}, x_k)$
    - Hàm cận dưới

      $$g(x_1,\ldots, x_k) = \bar{f} + c_m \times (n-k+1)$$

    - Hàm mục tiêu f của các lời giải pháp triển từ lời giải hiện tại

      $$f \geq g(x_1,\ldots, x_k)$$



Thuật toán nhánh và cận

```
void TRY(int k){
  for(int v = 1; v <= n; v++){
    if(marked[v] == false){
      x[k] = v;
      f = f + c[x[k-1]][x[k]];
      marked[v] = true;
      if(k == n){
        solution();
      }else{
        int g = f + cmin*(n-k+1);
        if(g < f_min)
          TRY(k+1);
      }
      marked[v] = false;
      f = f - c[x[k-1]][x[k]];
    }
  }
}
```

```
void solution() {
  if(f + c[x[n]][x[1]] < f_min){
    f_min = f + c[x[n]][x[1]];
  }
}
void main() {
  f_min = 9999999999;
  for(int v = 1; v <= n; v++)
    marked[v] = false;
  x[1] = 1; marked[1] = true;
  f = 0;
  TRY(2);
}
```

THUẬT TOÁN ỨNG DỤNG

CHIA ĐỂ TRỊ

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

Nội dung

- Tổng quan chia để trị
- Ví dụ minh họa
- Độ phức tạp chia để trị
- Giảm để trị

Tổng quan chia để trị

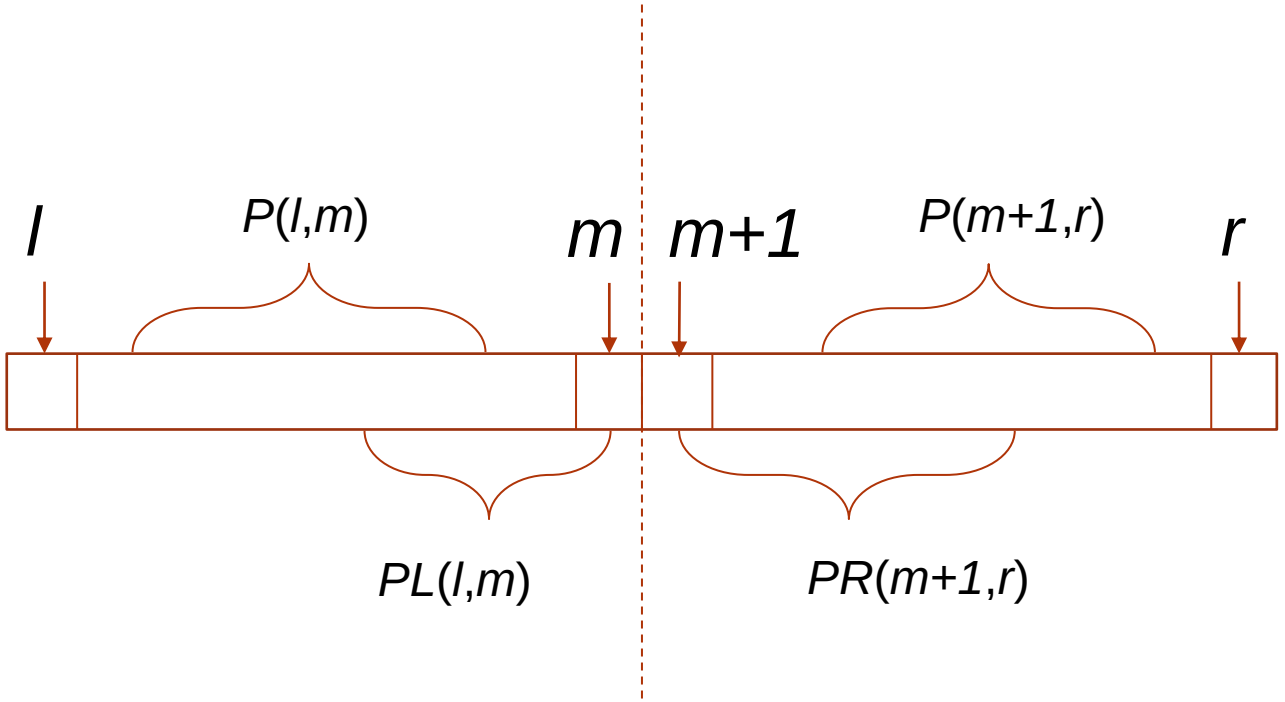
- Chia bài toán cần giải ban đầu thành các bài toán con độc lập nhau
- Giải (trị) các bài toán con
- Tổng hợp lời giải của các bài toán con để dẫn ra lời giải của bài toán xuất phát

Ví dụ minh họa

- Bài toán dãy con dài nhất: cho dãy số nguyên $a = a_1, a_2, \ldots, a_n$. Tìm dãy con gồm một số liên tiếp các phần tử có tổng lớn nhất
- Phân chia: ký hiệu $P(i, j)$ là lời giải của bài toán tìm dãy con liên tiếp của dãy $a_i, a_{i+1}, \ldots, a_j$ có tổng cực đại
- Tổng hợp lời giải
  - Ký hiệu $PL(i, j)$ là lời giải của bài toán tìm dãy con liên tiếp của dãy $a_i, a_{i+1}, \ldots, a_j$ sao cho phần tử cuối cùng là a_j có tổng cực đại
  - Ký hiệu $PR(i, j)$ là lời giải của bài toán tìm dãy con liên tiếp của dãy $a_i, a_{i+1}, \ldots, a_j$ sao cho phần tử đầu tiên là a_i có tổng cực đại

Ví dụ minh họa

- Xét đoạn [*l*,*l*+1,...,*r*]. Ký hiệu *m* = (*l*+*r*)/2
- *P*(*l*,*r*) = MAX{*P*(*l*, *m*), *P*(*m*+1,*r*), *PL*(*l*,*m*) + *PR*(*m*+1,*r*)}



Ví dụ minh họa

```
#include <bits/stdc++.h>
using namespace std;
#define INF 1e9
#define MAX 1000000

int a[MAX];
int n;
void input(){
    cin >> n;
    for(int i = 0; i < n; i++) cin >> a[i];
}
```

Ví dụ minh họa

```
int PL(int l, int r){
    int rs = -INF;
    int s = 0;
    for(int i = r; i >= l; i--){
        s += a[i];
        rs = max(rs,s);
    }
    return rs;
}

int PR(int l, int r){
    int rs = -INF;
    int s = 0;
    for(int i = l; i <= r; i++){
        s += a[i];
        rs = max(rs,s);
    }
    return rs;
}
```

Ví dụ minh họa

```
int P(int l, int r){
    if(l == r) return a[r];
    int m = (l+r)/2;
    return max(max(P(l,m),P(m+1,r)), PL(l,m)+PR(m+1,r));
}
void solve(){
    cout << P(0,n-1);
}
int main(){
    input();
    solve();
}
```

Độ phức tạp tính toán

- Chia bài toán xuất phát thành *a* bài toán con, mỗi bài toán con kích thước *n*/*b*
- *T*(*n*): thời gian của bài toán kích thước *n*
- Thời gian phân chia (dòng 4): *D*(*n*)
- Thời gian tổng hợp lời giải (dòng 6): *C*(*n*)
- Công thức truy hồi:

$$T(n) = \begin{cases} \Theta(1), & n \leq n0 \\ aT\left(\frac{n}{b}\right) + C(n) + D(n), & n > n0 \end{cases}$$

```
procedure D-and-C(n) {
1.   if(n ≤ n0)
2.     xử lý trực tiếp
3.   else{
4.     chia bài toán xuất phát
thành a bài toán con kích thước
n/b
5.     gọi đệ quy a bài toán con
6.     tổng hợp lời giải
7.   }
}
```

Độ phức tạp tính toán

- Độ phức tạp của thuật toán chia để trị (định lí thợ)
- Công thức truy hồi:

  $T(n) = aT(n/b) + cn^k$, với các hằng số $a \geq 1$, $b > 1$, $c > 0$
- Nếu $a > b^k$ thì $T(n) = \Theta(n^{log_b a})$
- Nếu $a = b^k$ thì $T(n) = \Theta(n^k logn)$ với $logn = log_2 n$
- Nếu $a < b^k$ thì $T(n) = \Theta(n^k)$

Độ phức tạp tính toán

- Độ phức tạp của thuật toán chia để trị (định lí thợ)
- Công thức truy hồi:

  $T(n) = aT(n/b) + cn^k$, với các hằng số $a \geq 1$, $b > 1$, $c > 0$
- Nếu $a > b^k$ thì $T(n) = \Theta(n^{log_b a})$
- Nếu $a = b^k$ thì $T(n) = \Theta(n^k logn)$ với $logn = log_2 n$
- Nếu $a < b^k$ thì $T(n) = \Theta(n^k)$

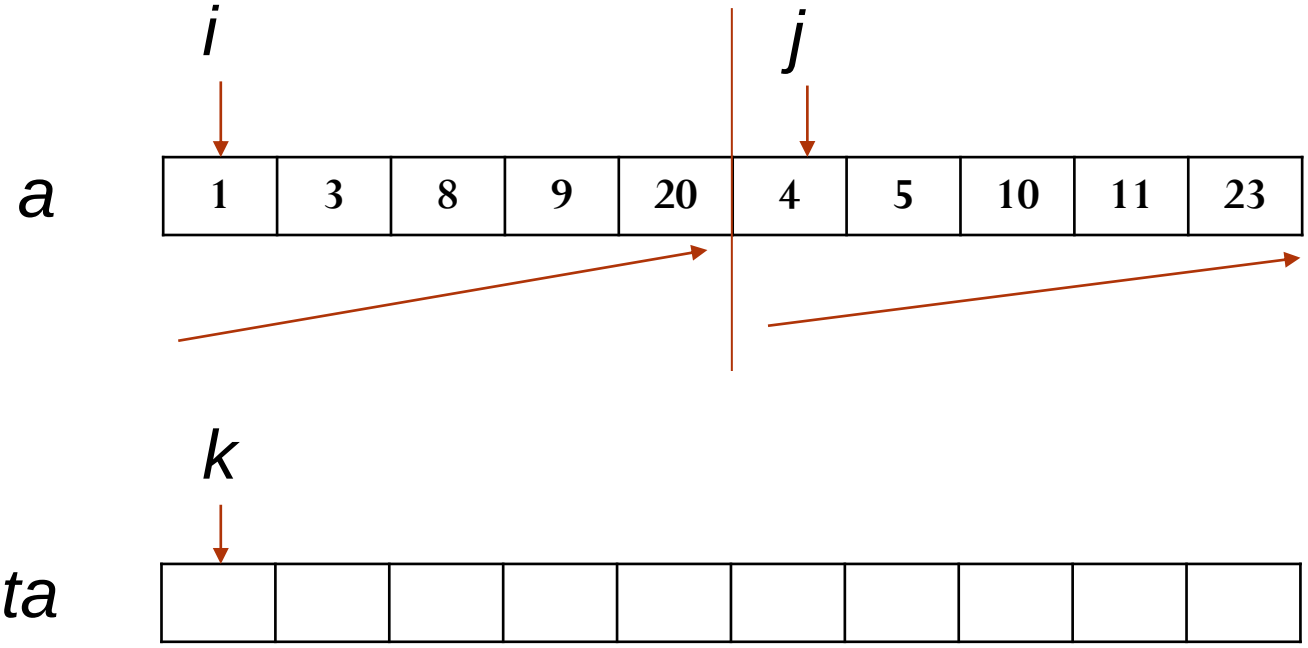
→ Thuật toán chia để trị giải bài toán tổng con cực đại có độ phức tạp là O(*nlogn*)

Sắp xếp trộn

- Phân chia: Chia dãy $a_1, \ldots, a_n$ thành 2 dãy con có độ dài bằng nhau
- Trị đệ quy: Sắp xếp 2 dãy con bằng thuật toán sắp xếp trộn
- Tổng hợp: Trộn 2 dãy con đã được sắp với nhau để thu được dãy ban đầu được sắp thứ tự

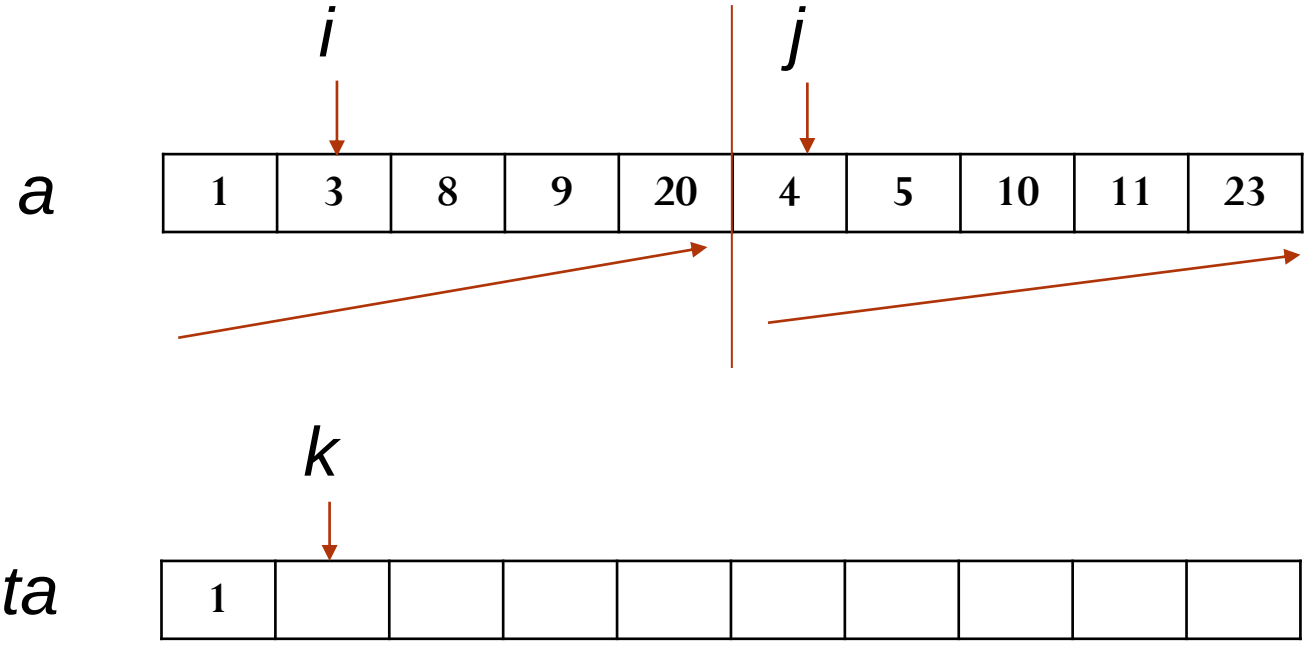
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



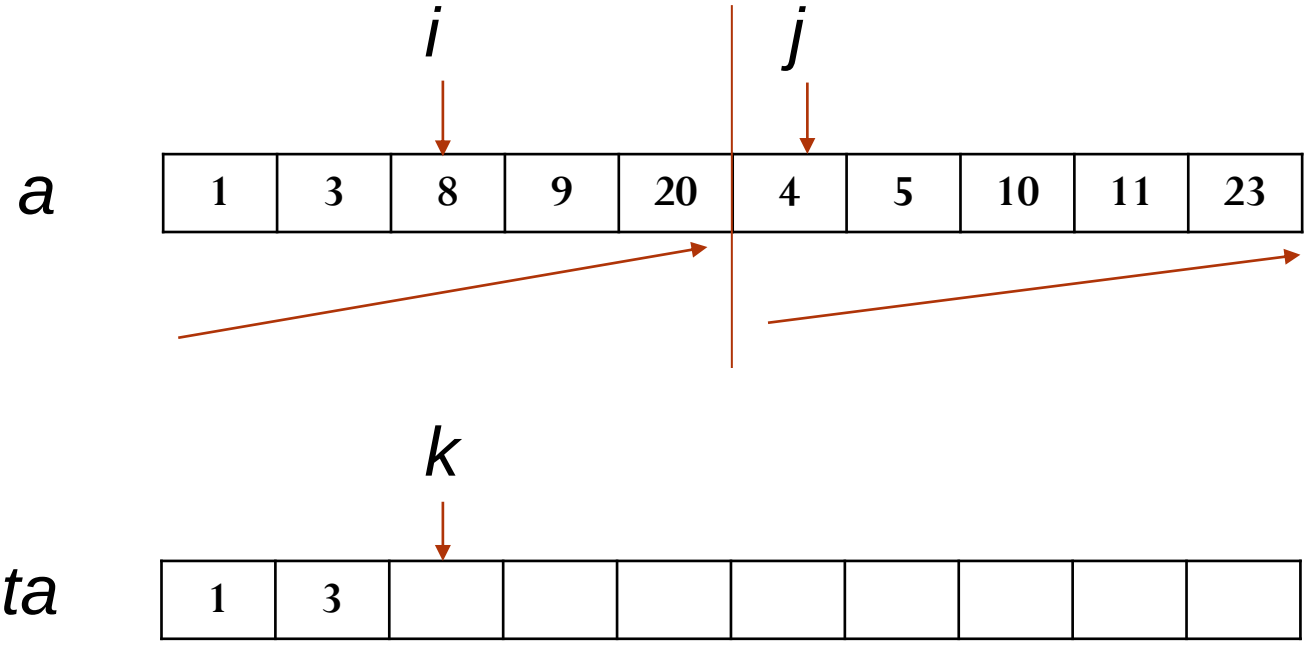
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



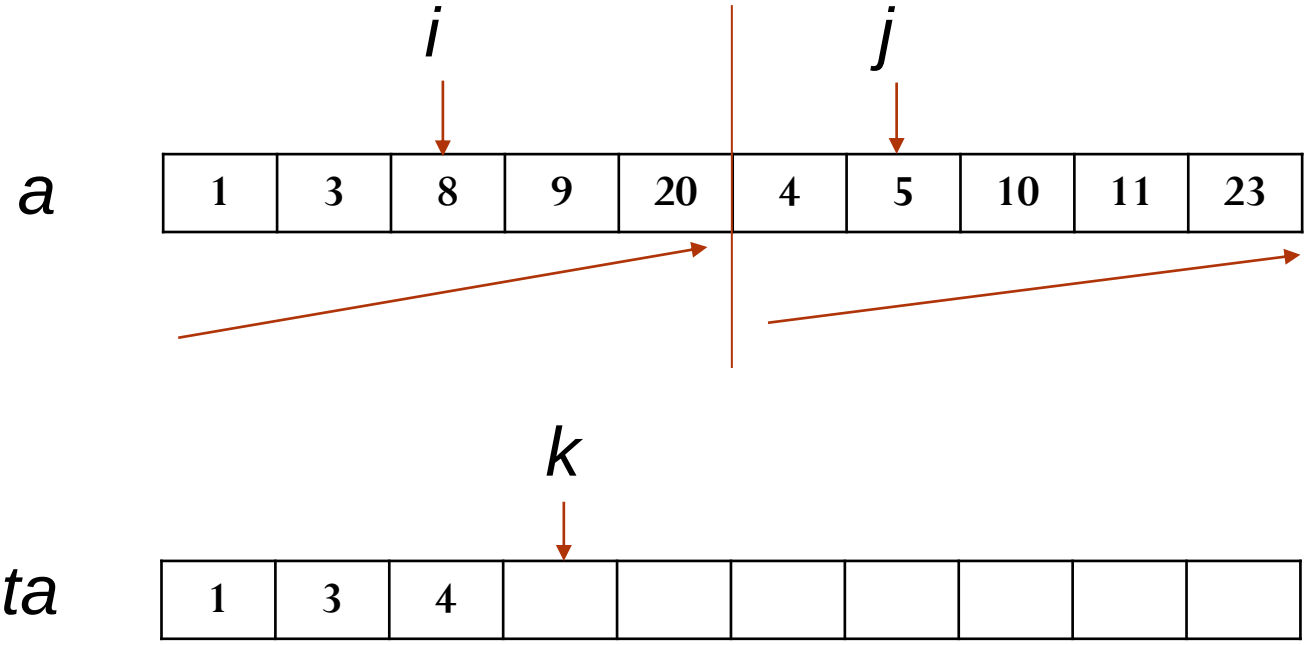
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



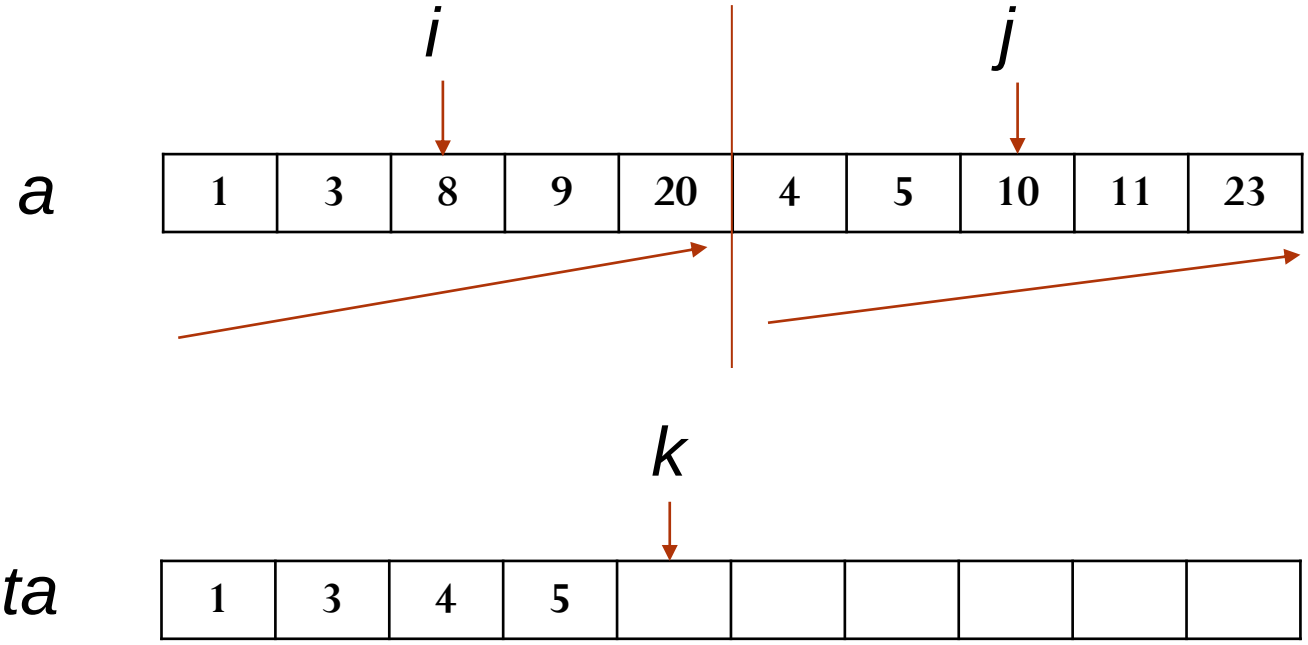
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



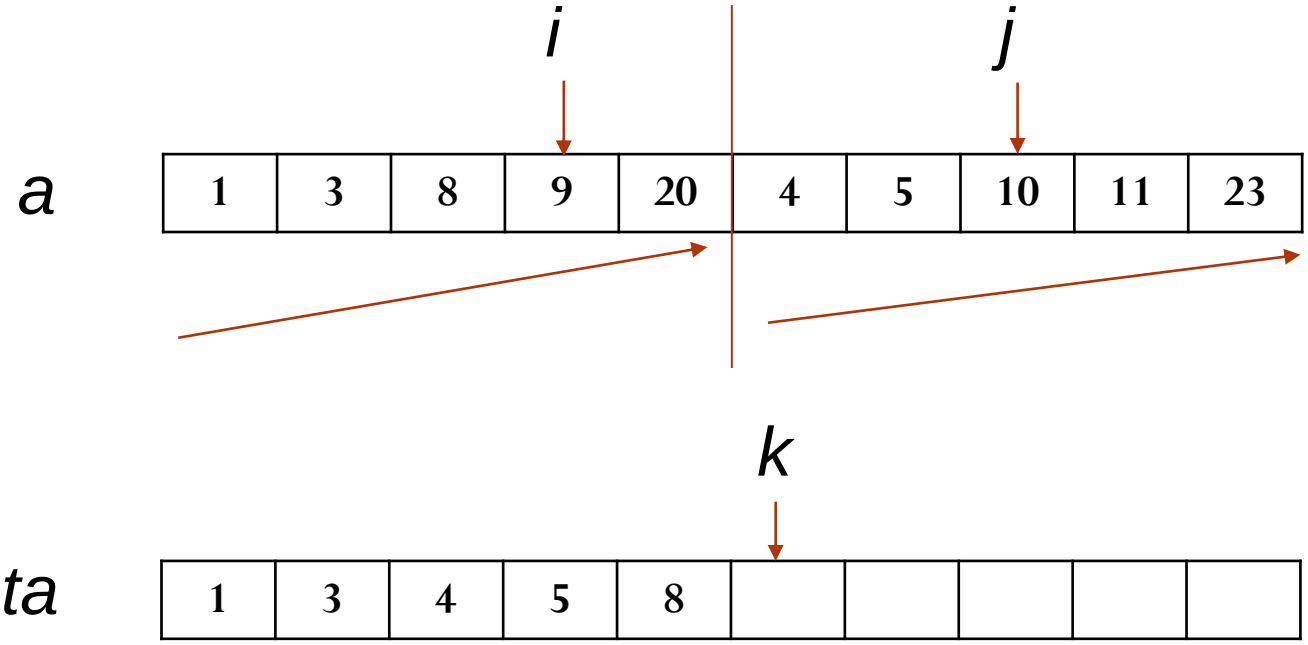
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



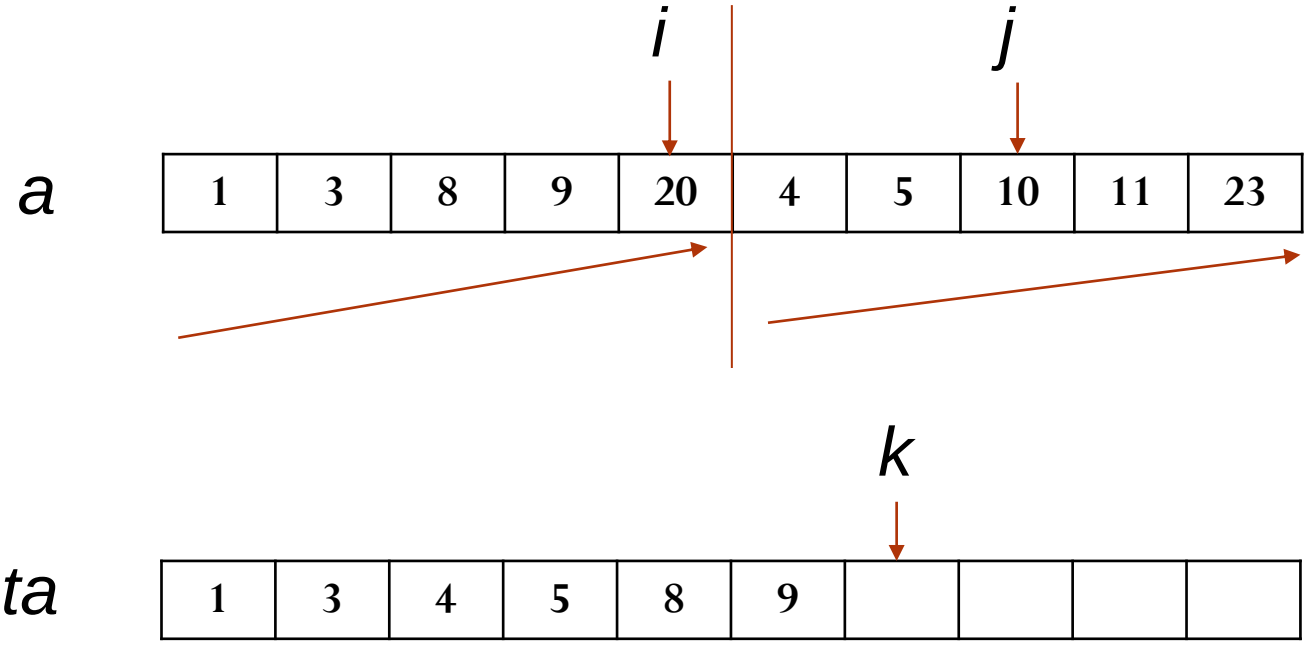
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



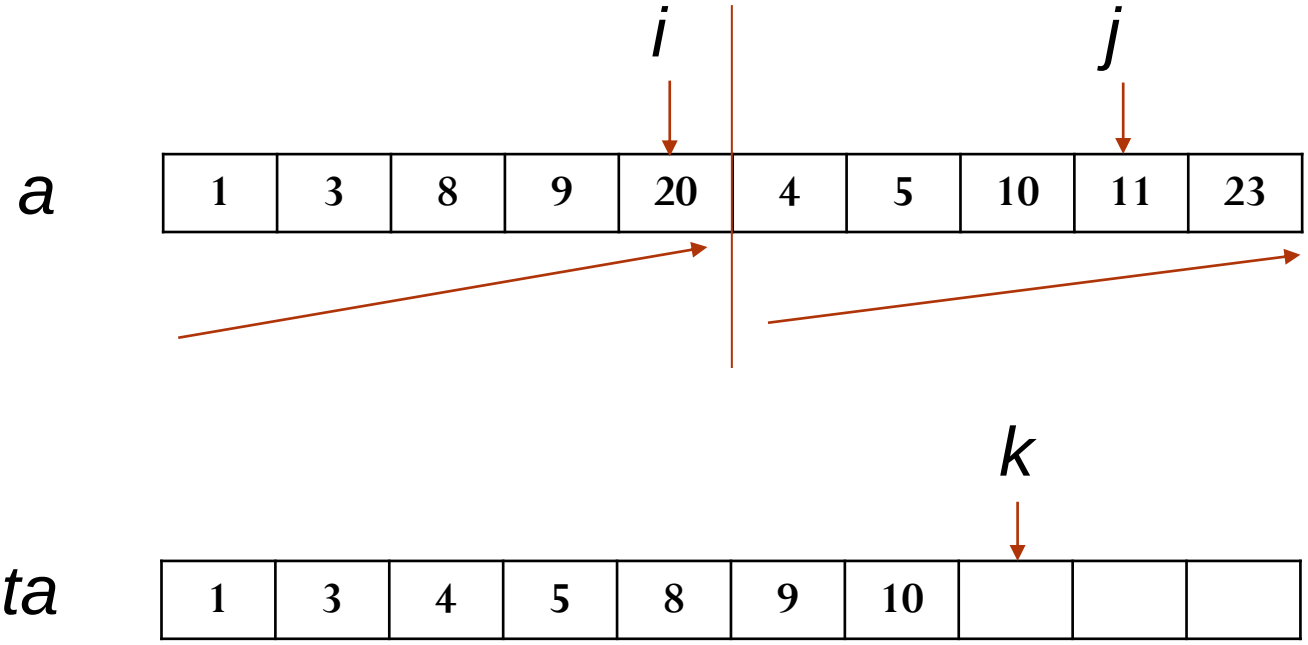
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



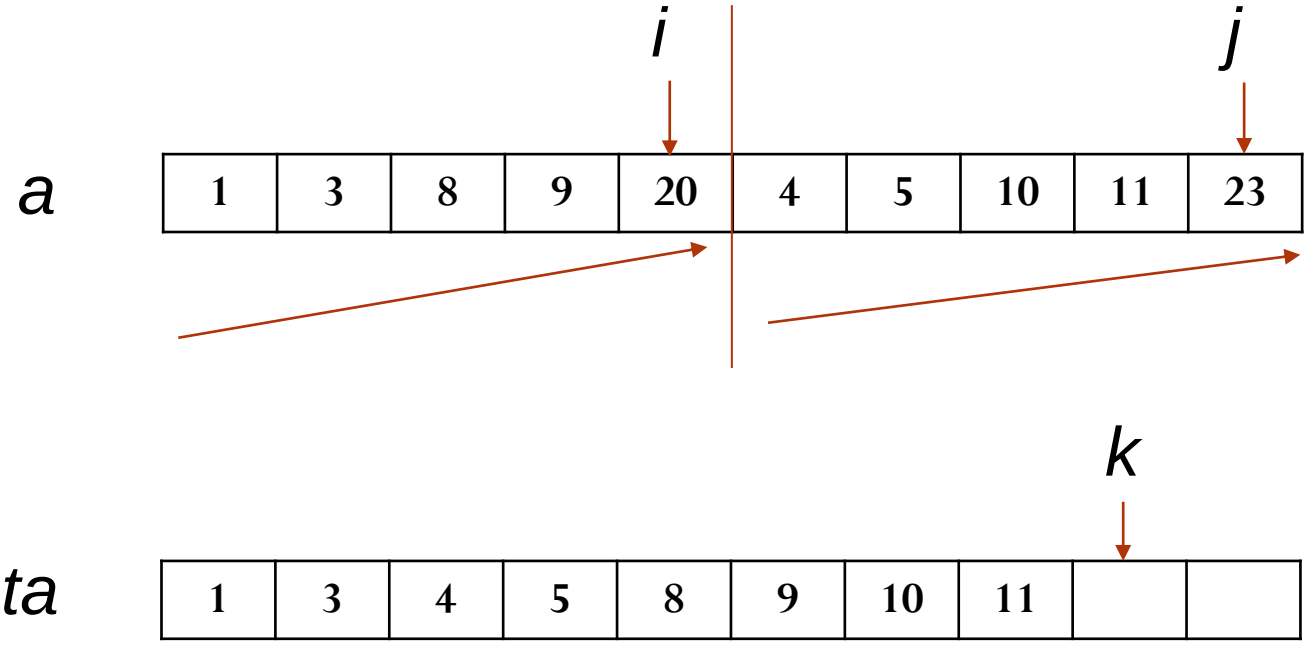
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



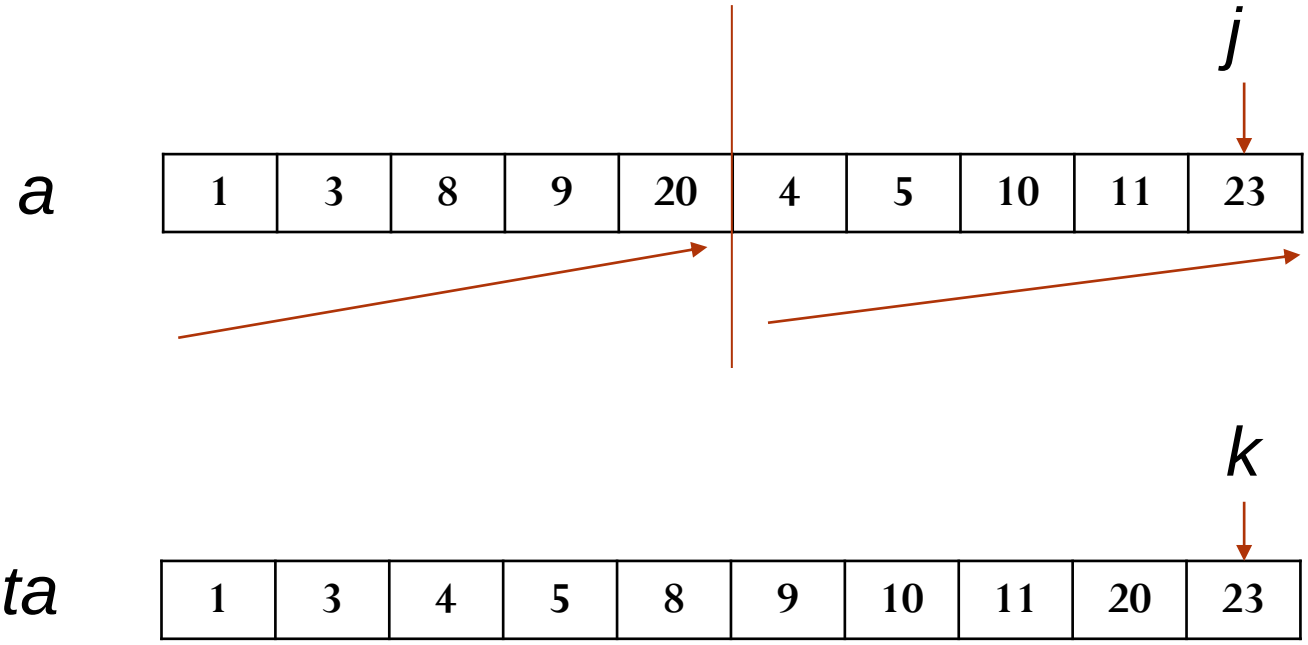
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



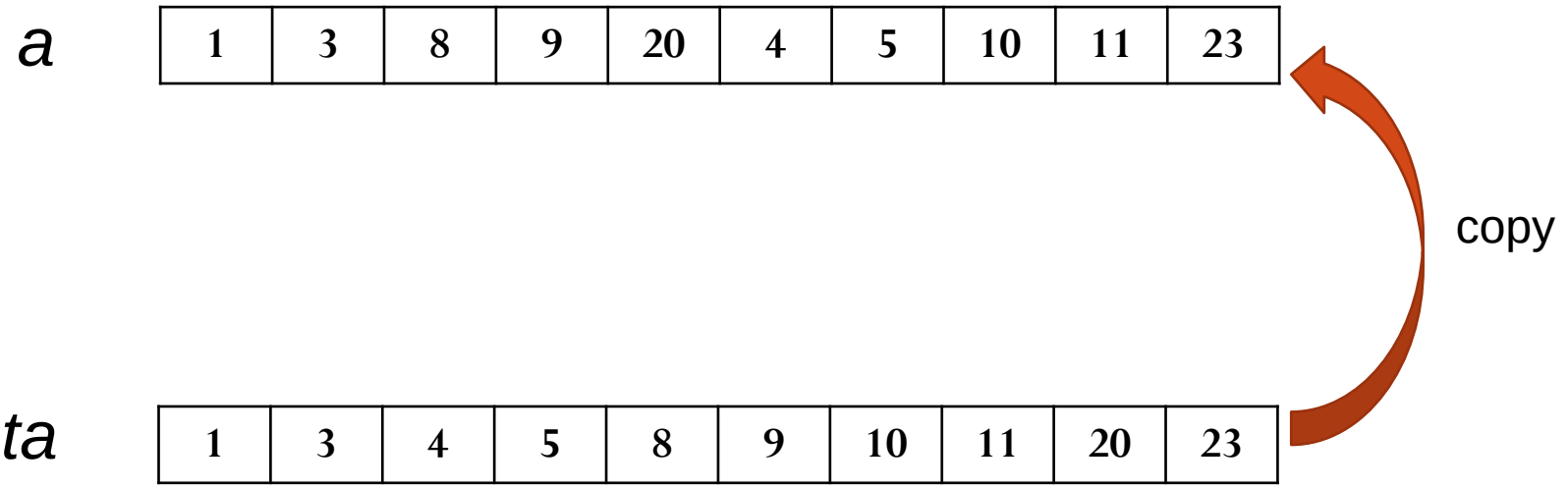
Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp



Sắp xếp trộn

- Trộn hai dãy đã được sắp xếp thành dãy mới được sắp xếp

a

1	3	4	5	8	9	10	11	20	23

ta

1	3	4	5	8	9	10	11	20	23

Sắp xếp trộn

```
#include <bits/stdc++.h>
using namespace std;
#define MAX 1000000
int a[MAX];
int n;
int ta[MAX];

void input(){
    cin >> n;
    for(int i = 0; i < n; i++) cin>> a[i];
}
void print(){
    for(int i = 0; i < n; i++) cout << a[i] << " ";
}
```

Sắp xếp trộn

```
void merge(int b, int m, int e){
    int i = b;
    int j = m+1;
    for(int k = b; k <= e; k++){
        if(i > m){  ta[k] = a[j]; j++; }
        else if(j > e){ta[k] = a[i]; i++;}
        else{
            if(a[i] > a[j]){ta[k] = a[j]; j++;}
            else{ta[k] = a[i]; i++;}
        }
    }
    for(int k = b; k <= e; k++) a[k] = ta[k];
}
```

Sắp xếp trộn

```
void mergeSort(int b, int e){
    if(b == e) return;
    int m = (b+e)/2;
    mergeSort(b,m);
    mergeSort(m+1,e);
    merge(b,m,e);
}
int main(){
    input();
    mergeSort(0,n-1);
    print();
    return 0;
}
```

Giảm để trị

- Chia bài toán (chia theo dữ liệu) xuất phát thành các bài toán con
- Giải 1 bài toán con và dẫn ra lời giải của bài toán xuất phát (các bài toán con khác không cần giải → tránh dư thừa)
  - Tìm kiếm nhị phân
  - Tính lũy thừa

Tìm kiếm nhị phân

Mã giả

```
bSearch(a,left, right, x){
  if left = right then{
    if a[left] = x return left; else return NOT_FOUND;
  }
  mid = (left + right)/2;
  if a[mid] = x then return mid;
  if a[mid] < x return bSearch(a,mid+1, right, x);
  else return bSearch(a,left, mid-1, x);
}
```

Độ phức tạp O(logn), với n là độ dài dãy từ chỉ số `left` đến chỉ số `right`

Tính x^n

Mã giả

```
exp(x,n){
  if n = 1 then return x;
  n2 = n/2;
  a = exp(x,n2);
  if n mod 2 = 0 then
    return a*a;
  else
    return a*a*x;
}
```

Độ phức tạp O(logn)

Bài tập ví dụ

- EXPMOD
  - Cho số nguyên dương x và N, hãy tính x^N mod 10^9+7

- TRIPLE
  - Cho dãy N số nguyên dương $a_1, a_2, \ldots, a_N$ và số nguyên dương K. Hãy đếm xem có bao nhiêu bộ chỉ số (i,j,k) sao cho $1 \le i < j < k \le N$ và $a_i + a_j + a_k = K$

THUẬT TOÁN ỨNG DỤNG

QUY HOẠCH ĐỘNG

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

Nội dung

- Tổng quan chia để trị
- Dãy con cực đại
- Dãy con tăng dần dài nhất

Quy hoạch động

- Sơ đồ chung
  - Chia bài toán xuất phát thành các bài toán con không nhất thiết độc lập với nhau
  - Giải các bài toán con từ nhỏ đến lớn, lời giải được lưu trữ lại vào 1 bảng
  - Bài toán con nhỏ nhất phải được giải 1 cách trực tiếp
  - Xây dựng lời giải của bài toán lớn hơn từ lời giải đã có của các bài toán con nhỏ hơn (truy hồi)
    - Số lượng bài toán con cần được bị chặn bởi đa thức của kích thước dữ liệu đầu vào
  - Phù hợp để giải hiệu quả một số bài toán tối ưu tổ hợp

Bài toán dãy con cực đại

- Cho dãy số nguyên $a = a_1, a_2, \ldots, a_N$. Hãy tìm dãy con bao gồm các phần tử liên tiếp của dãy a có tổng lớn nhất

Bài toán dãy con cực đại

- Phân chia
  - Ký hiệu P_i là bài toán tìm dãy con bao gồm các phần tử liên tiếp có tổng cực đại mà phần tử cuối cùng là a_i, với mọi $i = 1,\ldots, n$
  - Ký hiệu S_i là tổng các phần tử của lời giải của P_i, $\forall\ i = 1,\ldots, n$
  - $S_1 = a_1$
  - $S_i = \begin{cases} S_{i-1} + a_i, \text{ nếu } S_{i-1} > 0 \\ a_i, \text{ nếu } S_{i-1} \leq 0 \end{cases}$
  - Tổng các phần tử của dãy con cực đại của bài toán xuất phát là

$$\max\{S_1, S_2, \ldots, S_n\}$$

Bài toán dãy con cực đại

```
maxsubseq(a1,a2,. . ., aN){
 s[1] = a[1];
 res = s[1];
 for i = 2 → N do{
  if s[i-1] > 0 then {
    s[i] = s[i-1] + a[i];
  }else{
    s[i] = a[i];
  }
  if s[i] > res then res = s[i];
 }
 return res;
}
```

Bài toán dãy con tăng dần dài nhất

- Cho dãy số nguyên $a = a_1, a_2,\ldots, a_N$. Hãy tìm dãy con tăng dần bao gồm các phần tử không nhất thiết liên tiếp nhau của dãy a có số phần tử lớn nhất

Bài toán dãy con tăng dần dài nhất

- Ký hiệu P_i là bài toán tìm dãy con cực đại mà phần tử cuối cùng là a_i, với mọi $i = 1,\ldots, n$
- Ký hiệu S_i là số phần tử của lời giải của P_i, $\forall\ i = 1,\ldots, n$
- $S_1 = 1$
- $S_i = \max\{1, \max\{S_j + 1 \mid j < i \wedge a_j < a_i\}\}$
- Số phần tử của dãy con cực đại của bài toán xuất phát là

$$\max\{S_1, S_2, \ldots, S_n\}$$

Bài toán dãy con tăng dần dài nhất

```
incsubseq(a1,a2,. . ., aN){
 s[1] = 1;
 res = s[1];
 for i = 2 → N do{
  s[i] = 1;
  for j = 1 → i-1 do{
   if a[j] < a[i] then{
    if s[i] < s[j] + 1 then{
     s[i] = s[j] + 1;
    }
   }
  }
  if s[i] > res then res = s[i];
 }
 return res;
}
```

Dãy con chung dài nhất

- Ký hiệu $X = \langle X_1, X_2, \ldots, X_n \rangle$, một dãy con của X là dãy được tạo ra bằng việc loại bỏ 1 số phần tử nào đó của X đi
- Đầu vào
  - Cho 2 dãy $X = \langle X_1, X_2, \ldots, X_n \rangle$ và $Y = \langle Y_1, Y_2, \ldots, Y_m \rangle$
- Đầu ra
  - Tìm dãy con chung của X và Y có độ dài lớn nhất

Dãy con chung dài nhất

- Phân rã
  - Ký hiệu $S(i, j)$ là độ dài dãy con chung dài nhất của dãy $\langle X_1, \ldots, X_i \rangle$ và $\langle Y_1, \ldots, Y_j \rangle$, với $\forall\ i = 1, \ldots, n$ và $j = 1, \ldots, m$
  - Bài toán con nhỏ nhất
    - $\forall\ j = 0,1,\ldots, m$: $S(0, j) = 0$
    - $\forall\ i = 0,1,\ldots, n$: $S(i, 0) = 0$
- Tổng hợp lời giải, với $i = 1,\ldots,n$ và $j = 1,\ldots,m$

$$S(i, j) = \begin{cases} S(i\text{-}1, j\text{-}1) + 1, \text{ nếu } X_i = Y_j \\ \max\{S(i\text{-}1, j), S(i, j\text{-}1)\} \end{cases}$$

Dãy con chung dài nhất

```
lcs([X1,…,Xn], [Y1,…,Ym]){
 for i = 0 → n do S[i][0] = 0;
 for j = 0 → m do S[0][j] = 0;
 for i = 1 → n do{
  for j = 1 → m do{
   if Xi = Yj then{
     S[i][j] = S[i-1][j-1] + 1;
   }else{
    if S[i-1][j] > S[i][j-1] then{
     S[i][j] = S[i-1][j];
    }else{
      S[i][j] = S[i][j-1];
    }
   }
  }
 }
 return S[n][m];
}
```

Truy vết

- Mỗi bước xây dựng lời giải của một bài toán con từ lời giải của các bài toán con nhỏ hơn ta thường quyết định lựa chọn giữa các phương án

→ ghi nhớ quyết định lựa chọn đó vào cấu trúc dữ liệu để truy vết và dẫn ra lời giải đầy đủ cho bài toán ban đầu

Truy vết

- Bài toán dãy con cực đại
  - start[i]: chỉ số của phần tử đầu tiên của lời giải bài toán con Pi
  - select: chỉ số của bài toán con mà lời giải của bài toán con đó là lời giải của bài toán xuất phát

Truy vết

```
maxsubseq(a1,a2,. . ., aN){
 s[1] = a[1]; start[1] = 1;
 res = s[1]; select = 1;
 for i = 2 → N do{
  if s[i-1] > 0 then {
    s[i] = s[i-1] + a[i]; start[i] = start[i-1];
  }else{
    s[i] = a[i]; start[i] = i;
  }
  if s[i] > res then{
   res = s[i]; select = i;
  }
 }
 output('day con tu ', start[select], ' den ', select);
 return res;
}
```

Truy vết

- Bài toán dãy con tăng dần dài nhất
  - prev[i]: chỉ số của phần tử trước phần tử a[i] trong lời giải của bài toán con P_i.
  - select: chỉ số của bài toán con mà lời giải của bài toán con đó là lời giải của bài toán xuất phát

Truy vết

```
incsubseq(a1,a2,. . ., aN){
 s[1] = 1; prev[1] = -1;
 res = s[1]; select = 1;
 for i = 2 → N do{
  s[i] = 1; prev[i] = -1;
  for j = 1 → i-1 do{
   if a[j] < a[i] and s[i] < s[j] + 1 then{
     s[i] = s[j] + 1; prev[i] = j;
   }
  }
  if s[i] > res then {res = s[i]; select = i;}
 }
 i = select;
 while(i > -1){ output(i); i = prev[i]; }
 return res;
}
```

Truy vết

- Bài toán dãy con chung dài nhất
  - nếu S[i][j] = S[i-1][j-1] + 1 thì a[i][j] = ‘c’ (đi chéo)
  - Ngược lại
    - Nếu S[i-1][j] > S[i][j-1] thì a[i][j] = ‘u’; (đi lên)
    - Ngược lại a[i][j] = ‘l’ (đi sang trái)

Truy vết

```
lcs([X1,…,Xn], [Y1,…,Ym]){
 for i = 0 → n do S[i][0] = 0;
 for j = 0 → m do S[0][j] = 0;
 for i = 1 → n do{
  for j = 1 → m do{
   if Xi = Yj then{
     S[i][j] = S[i-1][j-1] + 1; end_i = i; end_j = j; a[i][j] = ‘c’;
   }else{
    if S[i-1][j] > S[i][j-1] then{
     S[i][j] = S[i-1][j]; a[i][j] = ‘u’;
    }else{
      S[i][j] = S[i][j-1]; a[i][j] = ‘l’;
    }
   }
  }
 }
 return S[n][m];
}
```

Truy vết

```
trace(end_i, end_j, a){
  i = end_i; j = end_j;
  while(true){
   if(a[i][j] == 'c’){
     output(i,j);
     i = i-1; j = j-1;
   }else if(a[i][j] == 'u'){
     i = i-1;
   }else{
     j = j-1;
   }
   if(i == 0 || j == 0) break;
   }
}
```

THUẬT TOÁN ỨNG DỤNG

CẤU TRÚC DEQUE

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

DEQUE

- Cấu trúc dữ liệu tuyến tính có tính chất của cả ngăn xếp và hàng đợi:
  - Thêm 1 phần tử vào cuối deque
  - Lấy 1 phần tử ở đầu deque ra
  - Lấy 1 phần tử ở cuối deque ra
- Trong C++
  - Khai báo: deque<int>
  - Phương thức: push_back(), push_front(), pop_front(), pop_back(), back(), front(), empty()

Quy hoạch động sử dụng deque

- Cho dãy $a_1, a_2, \ldots, a_n$ và 2 số nguyên dương $L_1 < L_2$. Hãy tìm dãy con $1 \le j_1 < j_2 < \ldots < j_k \le n$ sao cho $L_1 \le j_{q+1} - j_q \le i_2$ và $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$ có tổng cực đại

Quy hoạch động sử dụng deque

- Định nghĩa bài toán con
  - $S(i)$: tổng cực đại của dãy con của dãy $a_1, \ldots, a_i$ thỏa mãn đề bài mà phần tử cuối cùng là a_i
- Công thức quy hoạch động
  - $S(i) = \max(a_i + S(j) \mid L_1 \le i - j \le L_2\}$

Quy hoạch động sử dụng deque

- Định nghĩa bài toán con
  - $S(i)$: tổng cực đại của dãy con của dãy $a_1, \ldots, a_i$ thỏa mãn đề bài mà phần tử cuối cùng là a_i
- Công thức quy hoạch động
  - $S(i) = \max(a_i + S(j) \mid L_1 \le i - j \le L_2\}$
- Khởi tạo deque, lưu trữ các chỉ số j sao cho $S(j)$ không tăng và là ứng cử viên để tính toán các bài toán con $S(i)$
- Mỗi khi xét đến chỉ số i ($i = 1,\ldots, n$) thì
  - Đưa hết các chỉ số j ở đầu deque tại đó $j < i - L_2$ ra ngoài (vì nó ko là ứng cử viên để xác định S(i), S(i+1),…)
  - Đưa hết các chỉ số j ở cuối deque tại đó $S(j) < S(i\text{-}L_1)$ (do những chỉ số j như vậy không có ý nghĩa nữa trong việc xác định $S(i)$, $S(i+1)$,…

Quy hoạch động sử dụng deque

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6+1;
int a[N], S[N];
int n,L1,L2,ans;
int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> L1 >> L2;
    for(int i = 1; i <= n; i++)
       cin >> a[i];
    deque<int> q;
    ans = 0;
```

```
 for(int i = 1; i <= n; i++){
  while(!q.empty() && (q.front() <
                                  i - L2))
      q.pop_front();
  if(i - L1 >= 1){
     while(!q.empty() && S[q.back()] <
             S[i- L1])
       q.pop_back();
     q.push_back(i-L1);
   }
   S[i] = a[i] + (q.empty() ? 0 :
                           S[q.front()]);
   ans = max(ans,S[i]);
 }
 cout << ans;
}
```

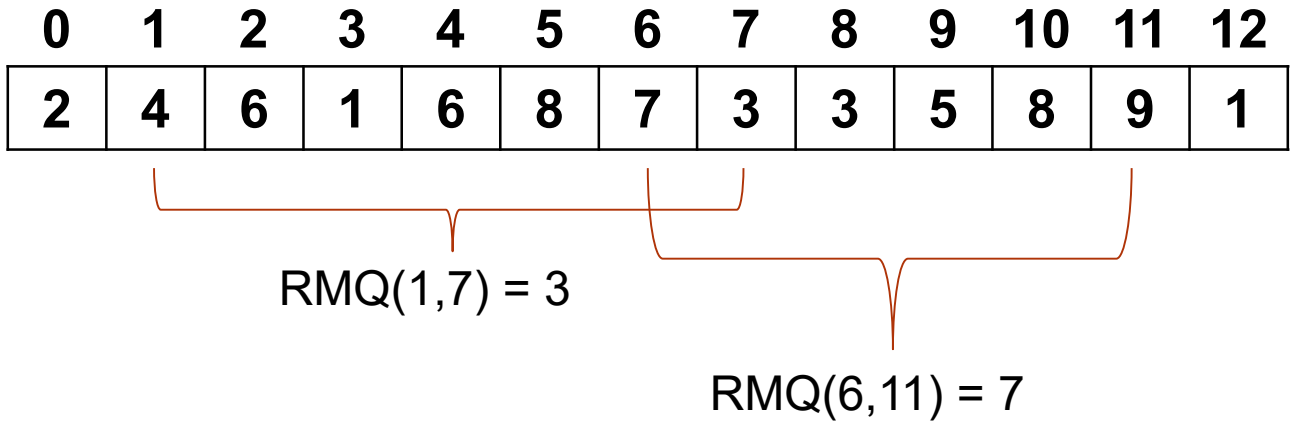
THUẬT TOÁN ỨNG DỤNG

QUY HOẠCH ĐỘNG
Range Minimum Query

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

Bài toán Range Minimum Query RMQ

- Cho dãy $a[0], a[1], \ldots, a[N-1]$. Với mỗi bộ chỉ số $0 \le i < j \le N-1$, hãy thực hiện truy vấn RMQ(i, j) tìm và trả về chỉ số của phần tử nhỏ nhất trong dãy con $a[i], a[i+1], \ldots, a[j]$.



Bài toán Range Minimum Query RMQ

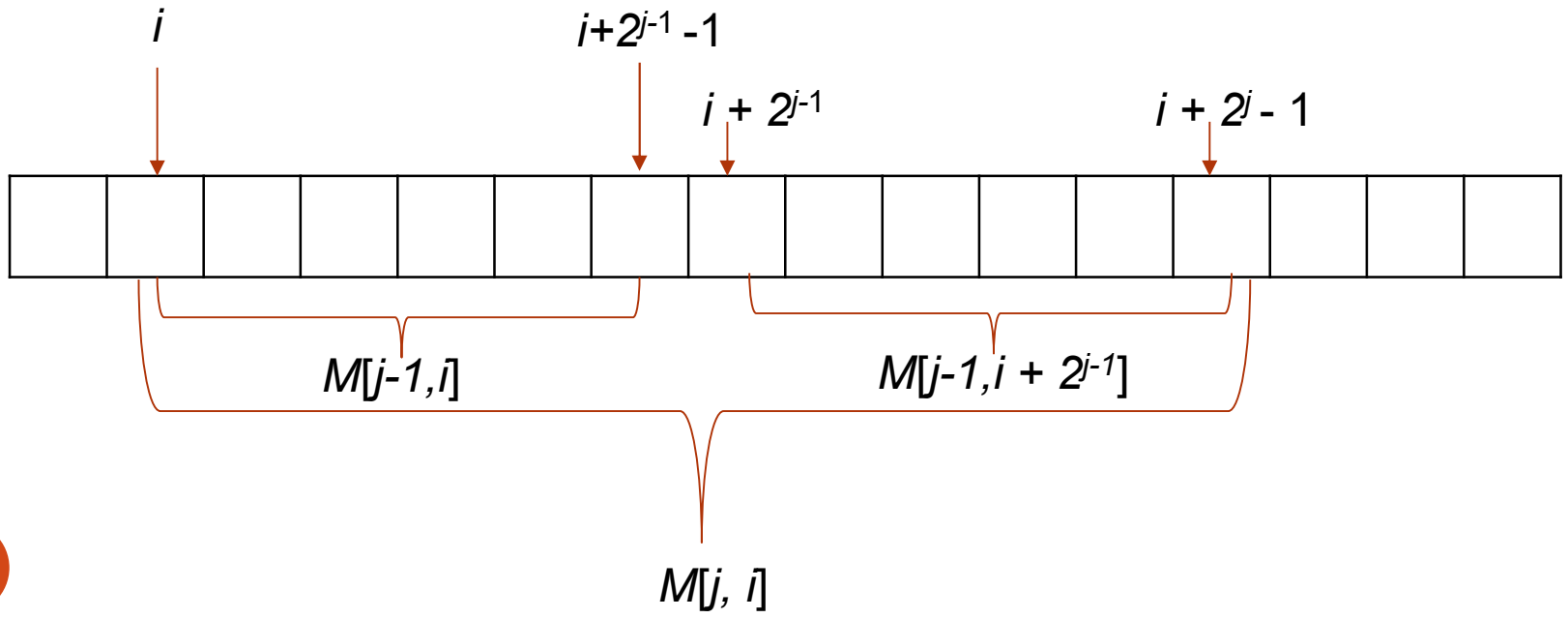
- Ký hiệu $M[j, i]$ là chỉ số phần tử nhỏ nhất của dãy $a[i]$, $a[i+2]$,…, $a[i+2^j-1]$ (dãy bắt đầu từ chỉ số i và có độ dài là 2^j).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	**4**	**6**	**1**	**6**	**8**	**7**	**3**	**3**	**5**	**8**	**9**	**1**	**2**	**6**	**4**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	1	3	3	4	6	7	8	8	9	10	12	12	13	15	-
2	3	3	3	3	7	8	8	8	8	12	12	12	12	-	-	-
3	3	3	3	3	8	12	12	12	12	-	-	-	-	-	-	-
4	12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Bài toán Range Minimum Query RMQ

- Bài toán con nhỏ nhất M[0,i] = i, i = 0,…, N-1
- Công thức truy hồi
- $M[j,i] = \begin{cases} M[j-1,i] \text{ nếu } a[M[j-1,i]] < a[M[j-1,i+2^{j-1}] \\ M[j-1,i+2^{j-1}], \text{ ngược lại} \end{cases}$



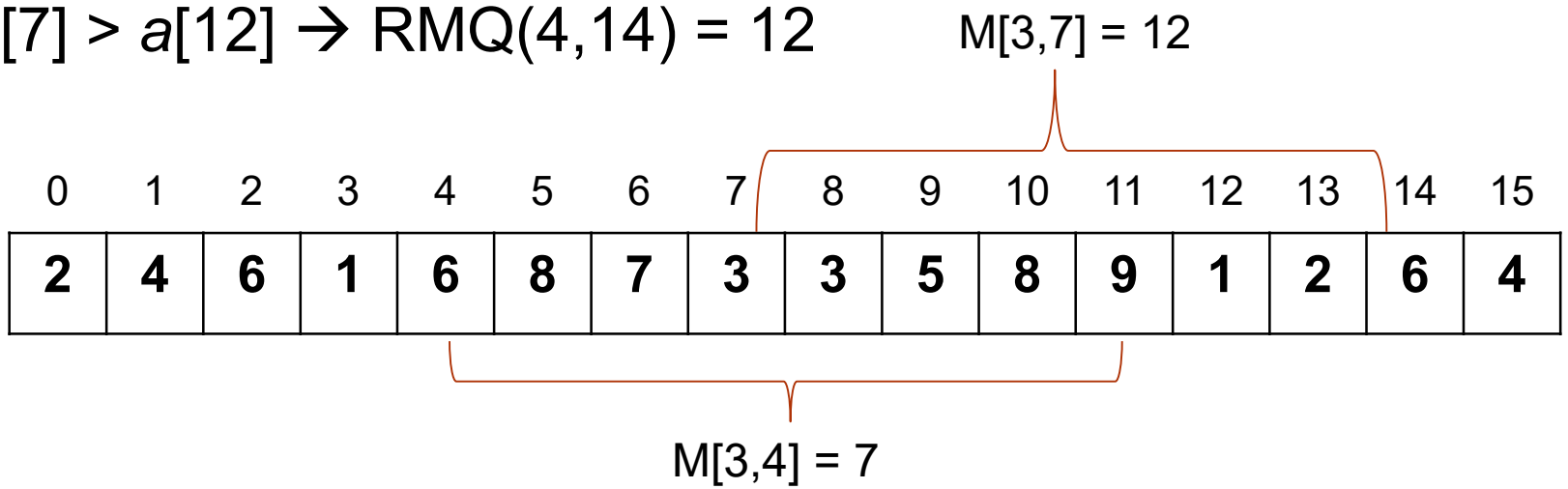
Bài toán Range Minimum Query RMQ

```
preprocessing(){
  for (i = 0; i < N; i++) M[0,i] = i;

  for (j = 0; 2^j ≤ N; j++){
    for(i = 0; i + 2^j -1 < N; i++){
      if a[M[j-1,i]] < a[M[j-1,i+2^(j-1)]] then{
        M[j,i] = M[j-1,i];
      }else{
        M[j,i] = M[j-1,i+2^(j-1)];
      }
    }
  }
}
```

Bài toán Range Minimum Query RMQ

- Truy vấn RMQ(i,j)
- k = [log(j-i+1)]
- RMQ(i,j) = $M[k,i]$ nếu $a[M[k,i]] \le a[M[k, j-2^k+1]]$
  $M[k, j-2^k+1]]$, ngược lại
- RMQ(4,14) = ?
  - k = [log(14-4+1)]=3
  - a[7] > a[12] → RMQ(4,14) = 12



THUẬT TOÁN ỨNG DỤNG

PHAM QUANG DUNG

Graphs

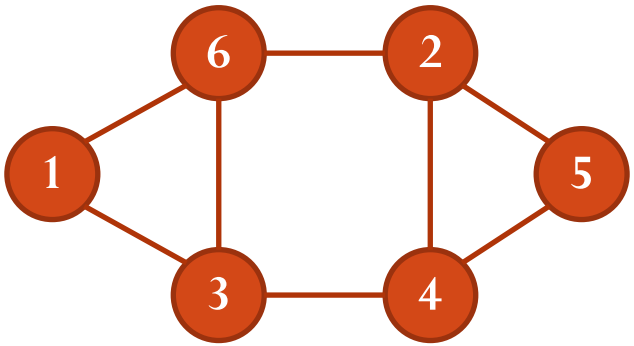
Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

Nội dung

- Đồ thị và các thuật ngữ liên quan
- Tìm kiếm theo chiều sâu
- Tìm kiếm theo chiều rộng
- Chu trình Euler
- Thuật toán Dijkstra sử dụng hàng đợi ưu tiên
- Thuật toán Kruskal sử dụng disjoint-set structure
- Exercises

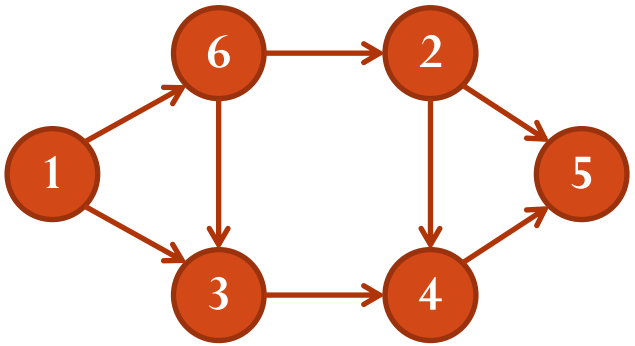
Đồ thị

- Đối tượng toán học bao gồm các đỉnh (node) và các liên kết giữa các đỉnh (cạnh, cung)
- Đồ thị $G = (V,E)$, trong đó V là tập đỉnh, E là tập cạnh (cung)
  - $(u,v) \in E$, chúng ta nói u kề với v



Undirected graph

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 3), (1,6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 6), (4, 5)\}$



Directed graph

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 3), (1,6), (2, 4), (2, 5), (6, 2), (3, 4), (6, 3), (4, 5)\}$

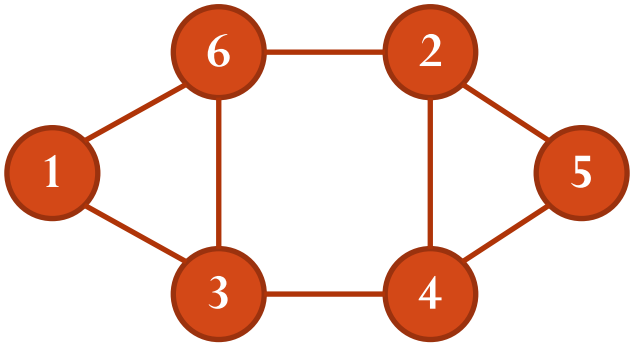
Đồ thị

- Bậc của một đỉnh là số đỉnh kề với nó

$$deg(v) = \#\{u \mid (u, v) \in E\}$$

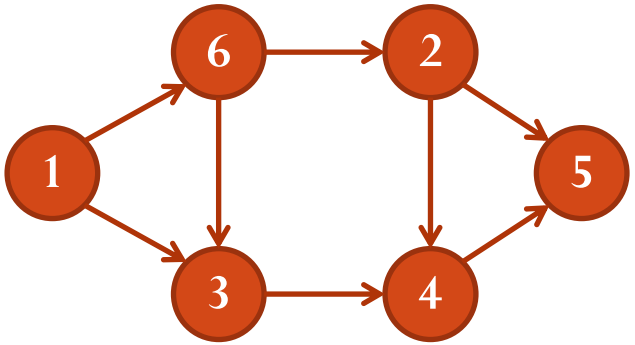
- Bán bậc vào (bán bậc ra) của một đỉnh là số cung đi vào (đi ra) khỏi đỉnh đó trên đồ thị có hướng:

$$deg^-(v) = \#\{u \mid (u, v) \in E\},\ deg^+(v) = \#\{u \mid (v, u) \in E\}$$



Undirected graph

- $deg(1) = 2$, $deg(6) = 3$

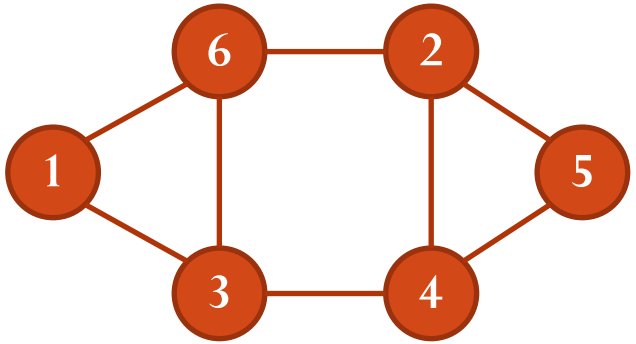


Directed graph

- $deg^-(1) = 0$, $deg^+(1) = 2$

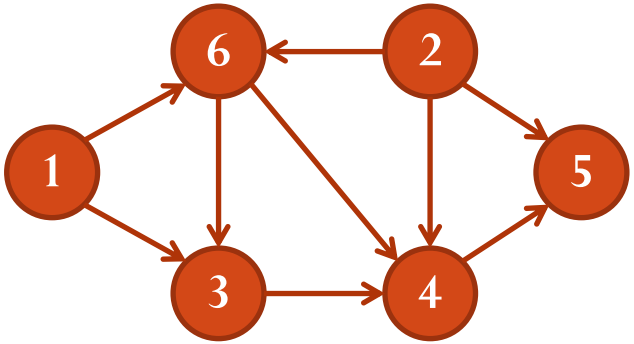
Đồ thị

- Cho đồ thị $G=(V, E)$ và 2 đỉnh $s, t \in V$, một đường đi từ s đến t trên G là chuỗi $s = x_0, x_1, \ldots, x_k = t$ trong đó $(x_i, x_{i+1}) \in E, \forall\ i = 0, 1, \ldots, k\text{-}1$



Path from 1 to 5:

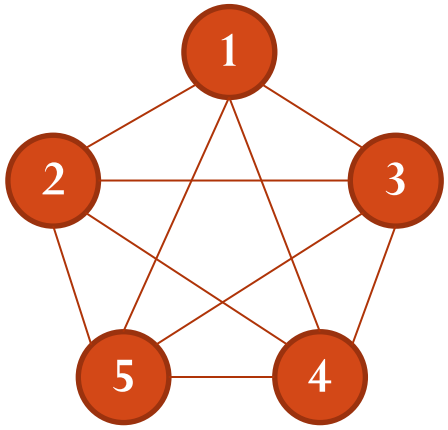
- 1, 3, 4, 5
- 1, 6, 2, 5



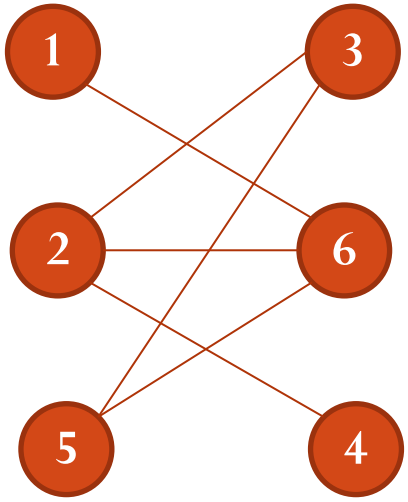
Path from 1 to 5:

- 1, 3, 4, 5
- 1, 6, 4, 5

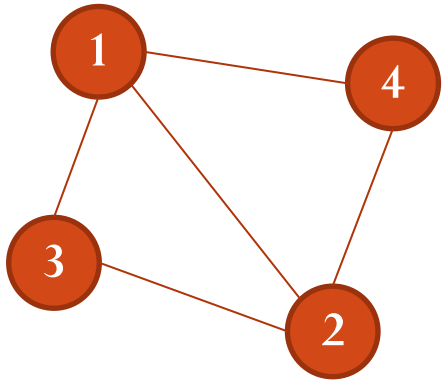
Đồ thị đặc biệt



Đồ thị đầy đủ



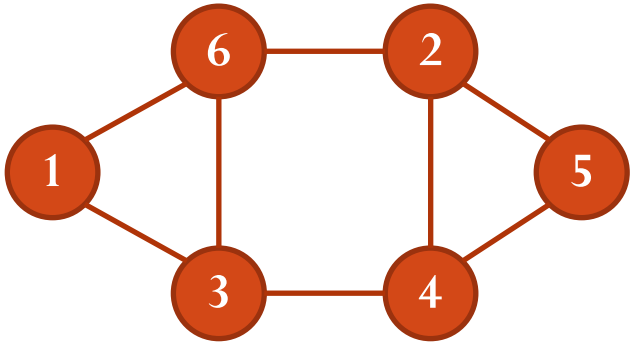
Đồ thị hai phía



Đồ thị phẳng

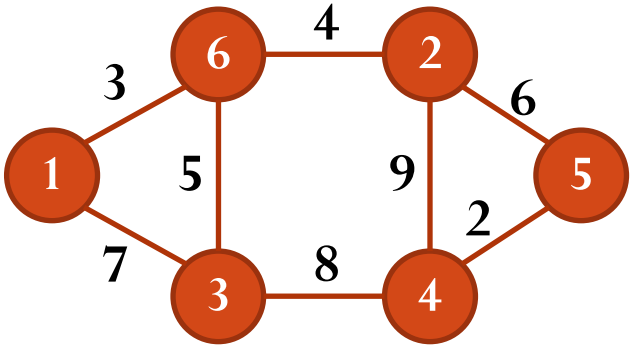
Đồ thị

- Ma trận kề



	1	2	3	4	5	6
1	0	0	1	0	0	1
2	0	0	0	1	1	1
3	1	0	0	1	0	1
4	0	1	1	0	1	0
5	0	1	0	1	0	0
6	1	1	1	0	0	0

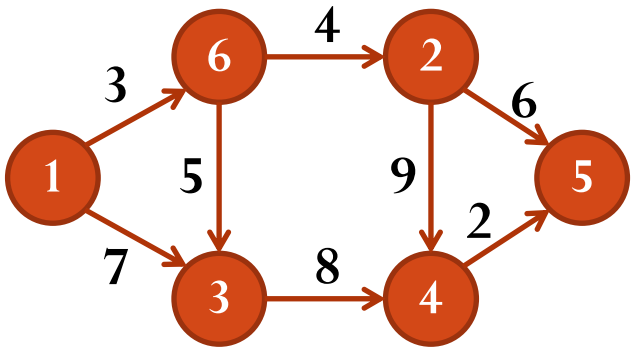
- Ma trận trọng số



	1	2	3	4	5	6
1	0	0	7	0	0	3
2	0	0	0	9	6	4
3	7	0	0	8	0	5
4	0	9	8	0	2	0
5	0	6	0	4	0	0
6	3	4	5	0	0	0

Biểu diễn đồ thị

- Danh sách kề
  - Với mỗi $v \in V$, $A(v)$ là tập các bộ (v, u, w) trong đó w là trọng số cung (v, u)
  - $A(1) = \{(1, 6, 3), (1, 3, 7)\}$
  - $A(2) = \{(2, 4, 9), (2, 5, 6)\}$
  - $A(3) = \{(3, 4, 8)\}$
  - $A(4) = \{(4, 5, 2)\}$
  - $A(5) = \{\}$
  - $A(6) = \{(6, 3, 5), (6, 2, 4)\}$



Duyệt đồ thị

- Thăm các đỉnh của đồ thị theo một thứ tự nào đó
- Mỗi đỉnh thăm đúng 1 lần
- Có 2 phương pháp chính
  - Duyệt theo chiều sâu: Depth-First Search (DFS)
  - Duyệt theo chiều rộng: Breadth-First Search (BFS)

Depth-First Search (DFS)

- DFS(*u*): DFS bắt đầu thăm từ đỉnh *u*
  - Nếu tồn tại một đỉnh *v* trong danh sách kề của *u* mà chưa được thăm → tiền hành thăm *v* và gọi DFS(*v*)
  - Nếu tất cả các đỉnh kề với *u* đã được thăm → DFS quay lui trở lại đỉnh *x* từ đó thuật toán thăm *u* và tiến hành thăm các đỉnh khác kề với *x* (gọi DFS(*x*)) mà chưa được thăm. Lúc này, đỉnh *u* được gọi là *đã duyệt xong*

Depth-First Search (DFS)

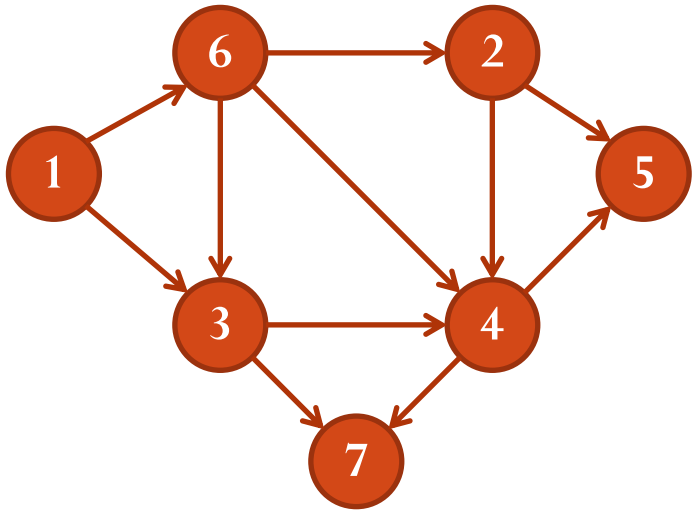
- Các thông tin liên quan đến mỗi đỉnh *v*
  - *p*(*v*): là đỉnh mà từ đó, DFS thăm đỉnh *v*
  - *d*(*v*): thời điểm đỉnh v *được thăm* nhưng *chưa duyệt xong*
  - *f*(*v*): thời điểm đỉnh v *đã duyệt xong*
  - color(*v*)
    - WHITE: *chưa thăm*
    - GRAY: *đã được thăm* nhưng *chưa duyệt xong*
    - BLACK: *đã duyệt xong*

Depth First Search - DFS

```
DFS(u) {
  t = t + 1;
  d(u) = t;
  color(u) = GRAY;
  foreach(adjacent node v to u)
  {
    if(color(v) = WHITE) {
      p(v) = u;
      DFS(v);
    }
  }
  t = t + 1;
  f(u) = t;
  color(u) = BLACK;
}
```

```
DFS() {
  foreach (node u of V) {
    color(u) = WHITE;
    p(u) = NIL;
  }
  t = 0;
  foreach(node u of V) {
    if(color(u) = WHITE) {
      DFS(u);
    }
  }
}
```

Depth First Search - DFS

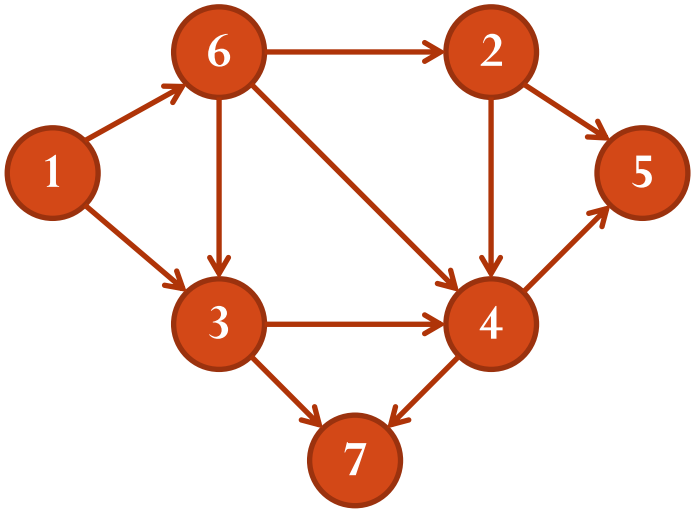


Node	1	2	3	4	5	6	7
d							
f							
p	-	-	-	-	-	-	-
color	W	W	W	W	W	W	W

Depth First Search - DFS

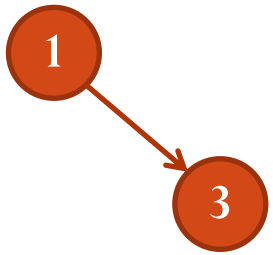
DFS(1)

Node	1	2	3	4	5	6	7
d	1						
f							
p	-	-	-	-	-	-	-
color	G	W	W	W	W	W	W

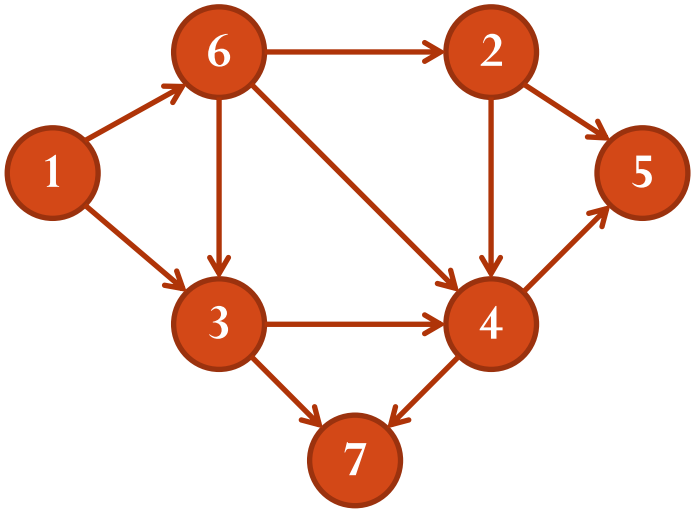


Depth First Search - DFS

DFS(1)

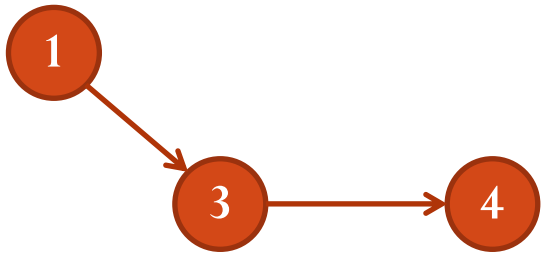


Node	1	2	3	4	5	6	7
d	1		2				
f							
p	-	-	1	-	-	-	-
color	G	W	G	W	W	W	W

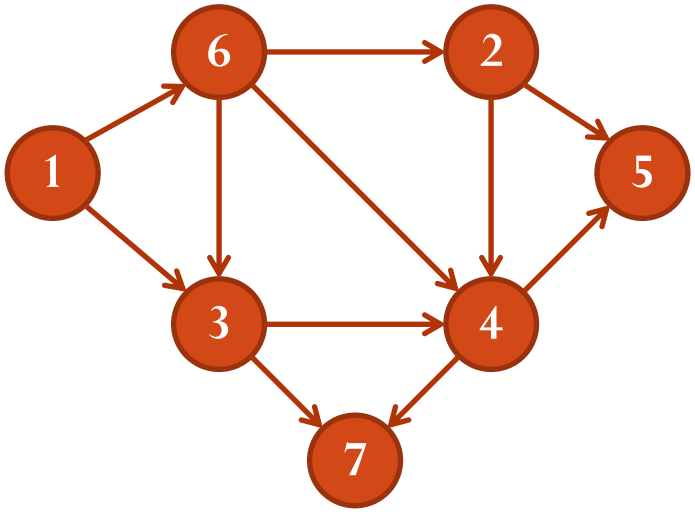


Depth First Search - DFS

DFS(1)

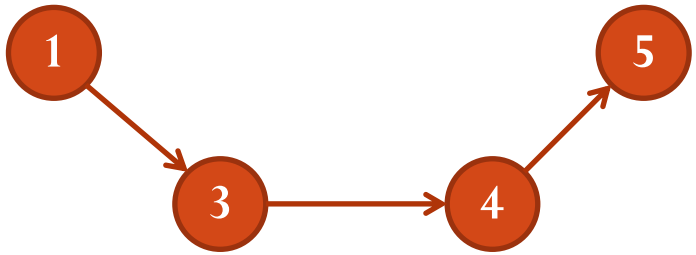


Node	1	2	3	4	5	6	7
d	1		2	3			
f							
p	-	-	1	3	-	-	-
color	G	W	G	G	W	W	W

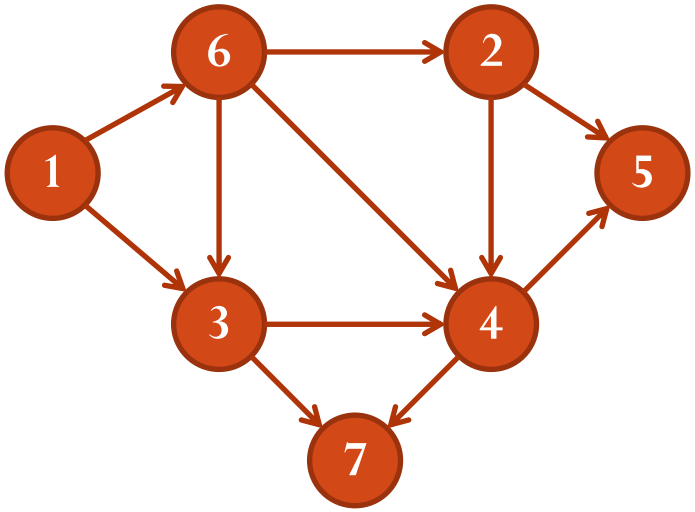


Depth First Search - DFS

DFS(1)

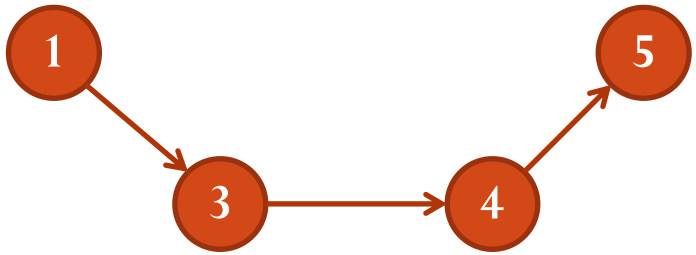


Node	1	2	3	4	5	6	7
d	1		2	3	4		
f							
p	-	-	1	3	4	-	-
color	G	W	G	G	G	W	W

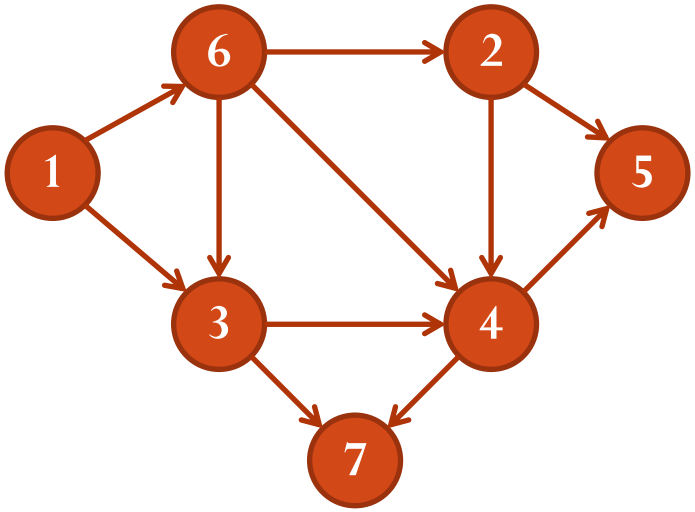


Depth First Search - DFS

DFS(1)

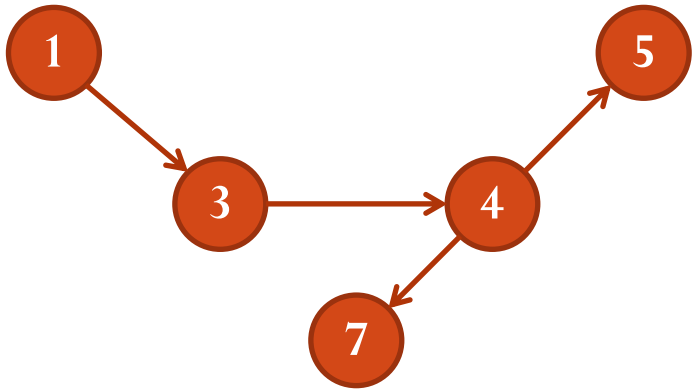


Node	1	2	3	4	5	6	7
d	1		2	3	4		
f					5		
p	-	-	1	3	4	-	-
color	G	W	G	G	B	W	W

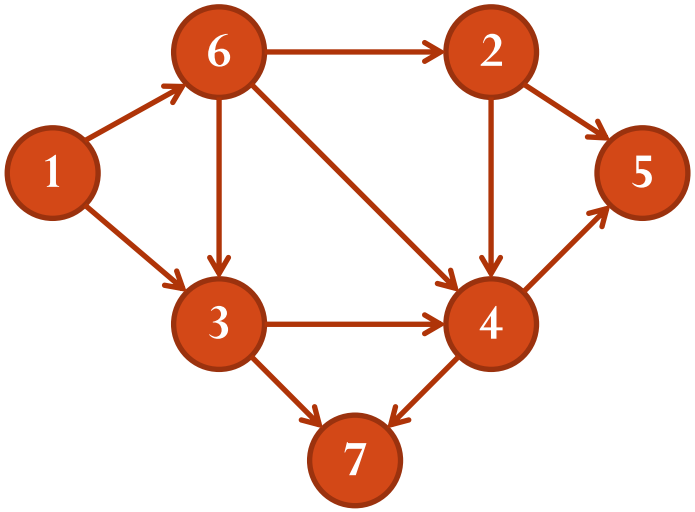


Depth First Search - DFS

DFS(1)

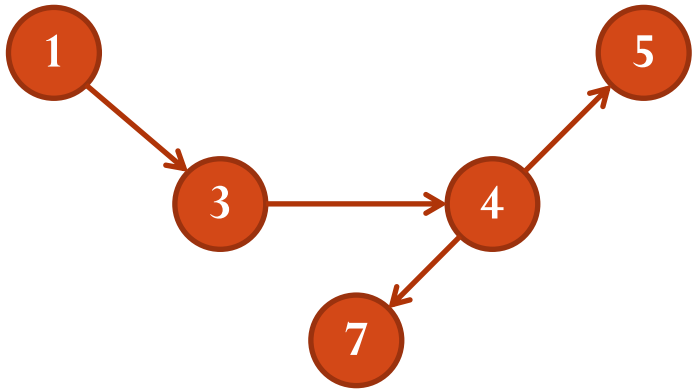


Node	1	2	3	4	5	6	7
d	1		2	3	4		6
f					5		
p	-	-	1	3	4	-	4
color	G	W	G	G	B	W	G

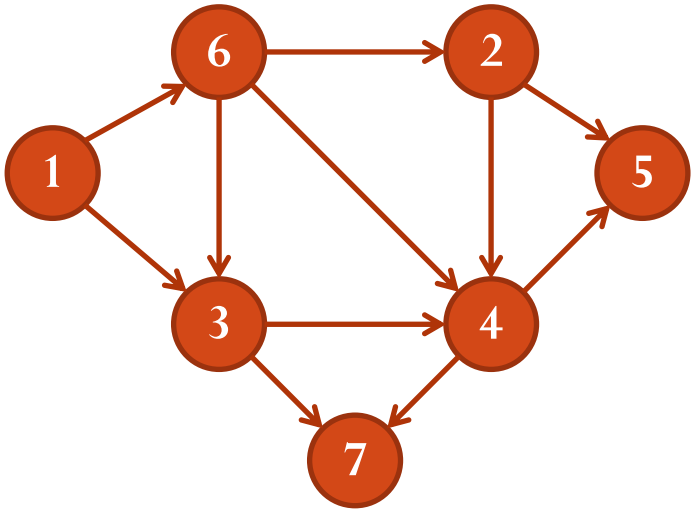


Depth First Search - DFS

DFS(1)

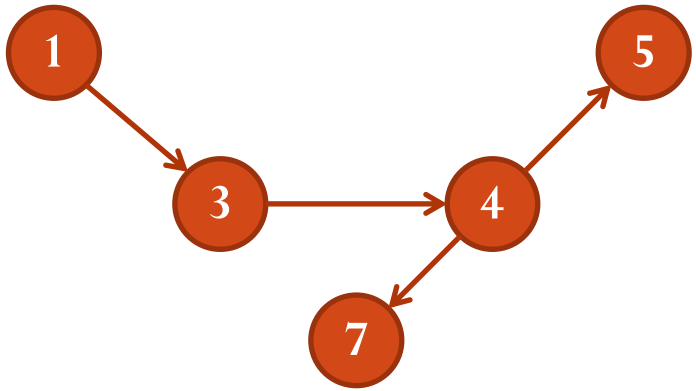


Node	1	2	3	4	5	6	7
d	1		2	3	4		6
f					5		7
p	-	-	1	3	4	-	4
color	G	W	G	G	B	W	B

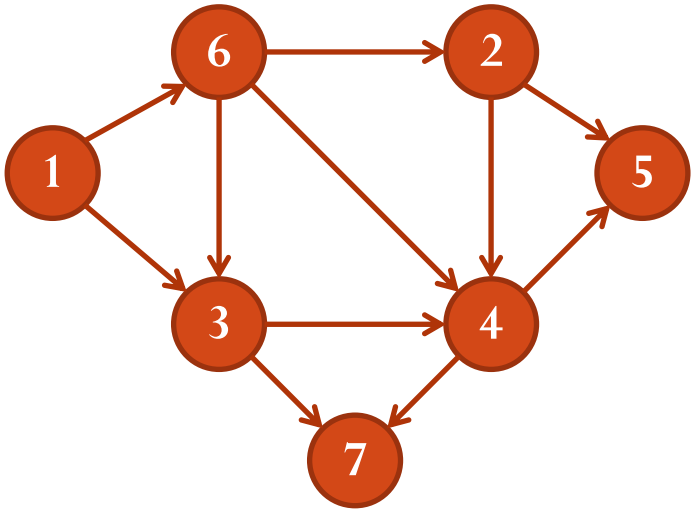


Depth First Search - DFS

DFS(1)

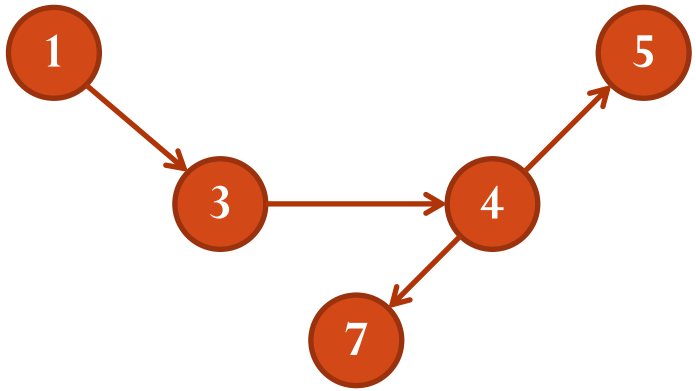


Node	1	2	3	4	5	6	7
d	1		2	3	4		6
f				8	5		7
p	-	-	1	3	4	-	4
color	G	W	G	B	B	W	B

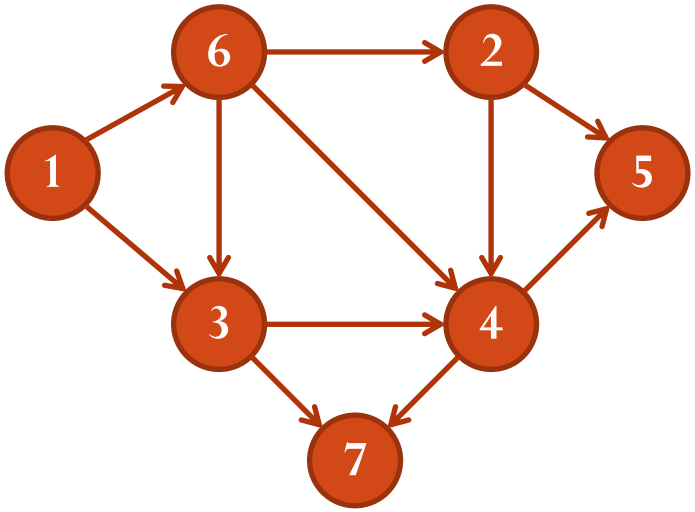


Depth First Search - DFS

DFS(1)

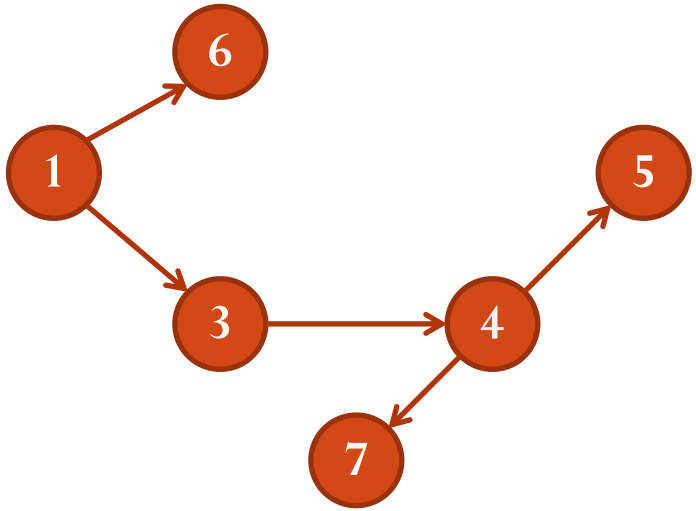


Node	1	2	3	4	5	6	7
d	1		2	3	4		6
f			9	8	5		7
p	-	-	1	3	4	-	4
color	G	W	B	B	B	W	B

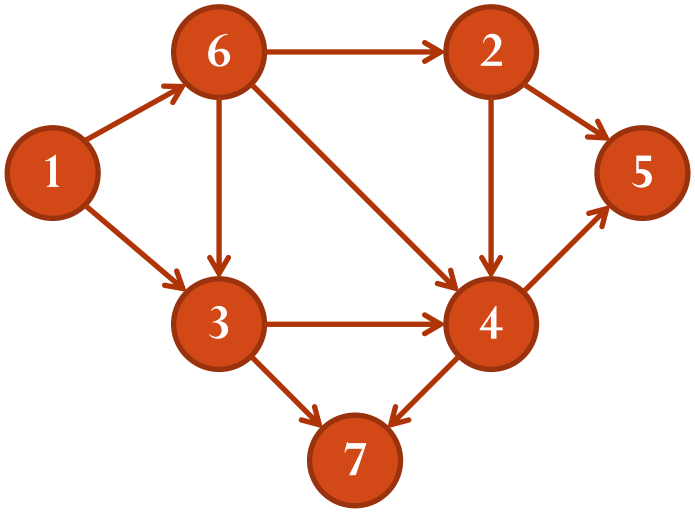


Depth First Search - DFS

DFS(1)

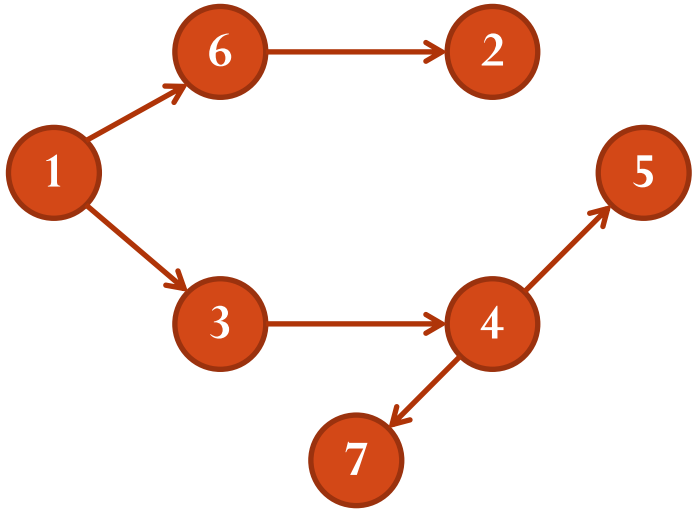


Node	1	2	3	4	5	6	7
d	1		2	3	4	10	6
f			9	8	5		7
p	-	-	1	3	4	1	4
color	G	W	B	B	B	G	B

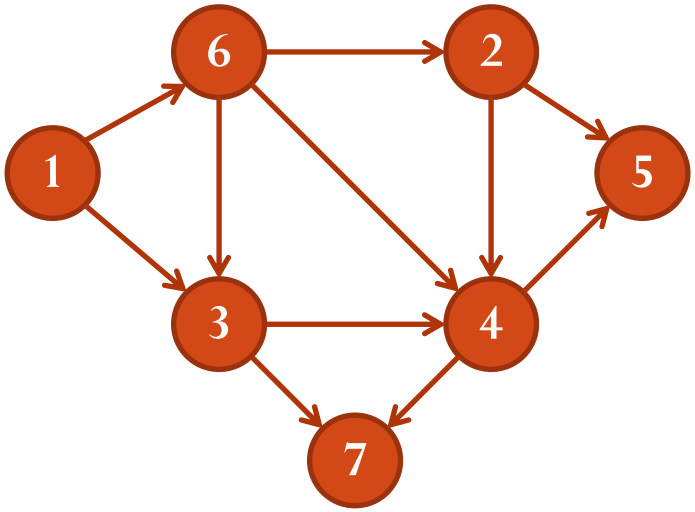


Depth First Search - DFS

DFS(1)

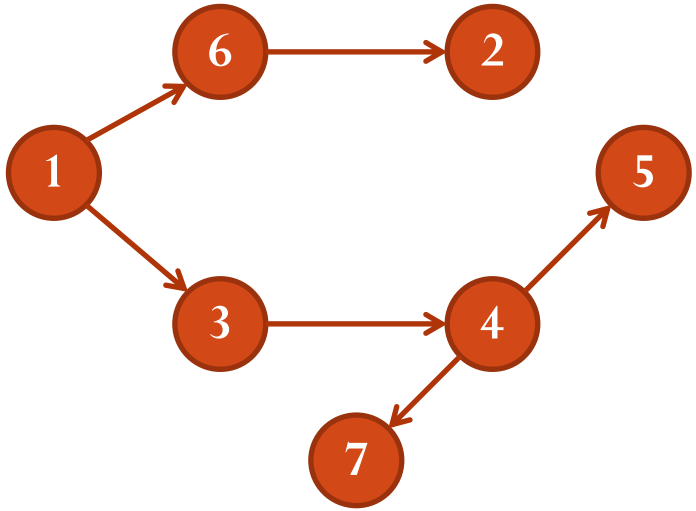


Node	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f			9	8	5		7
p	-	6	1	3	4	1	4
color	G	G	B	B	B	G	B

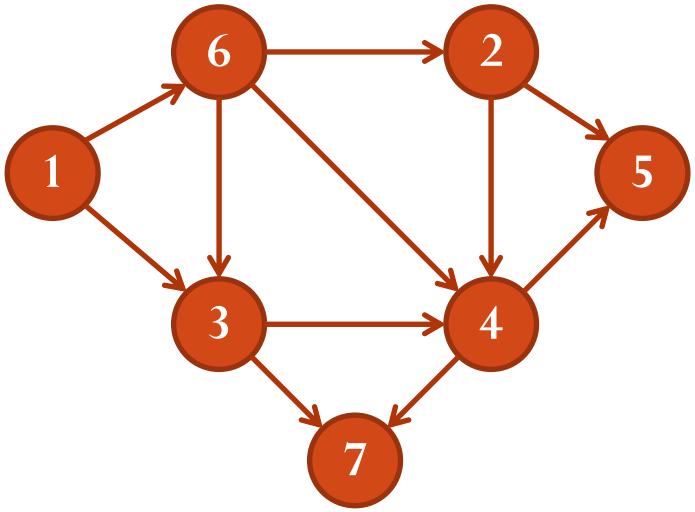


Depth First Search - DFS

DFS(1)

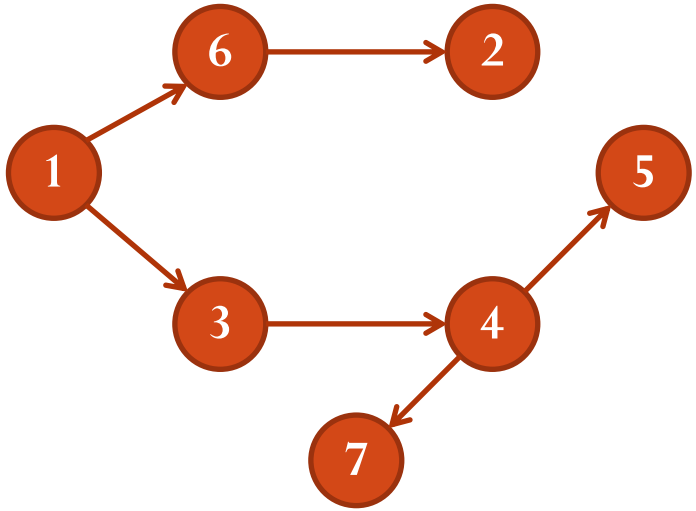


Node	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f		12	9	8	5		7
p	-	6	1	3	4	1	4
color	G	B	B	B	B	G	B

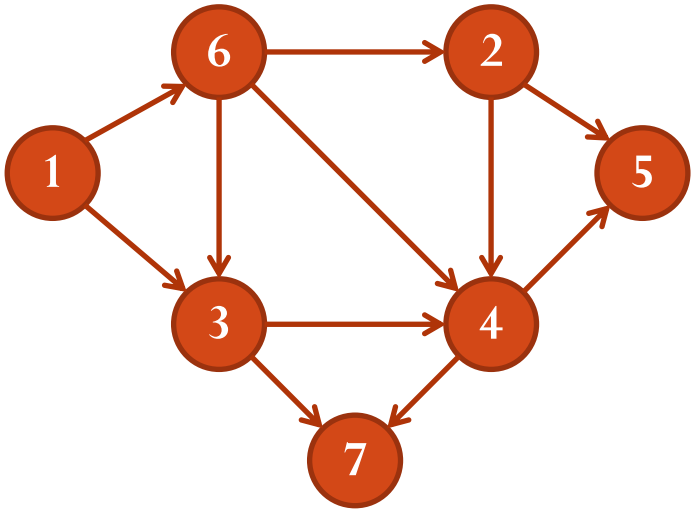


Depth First Search - DFS

DFS(1)

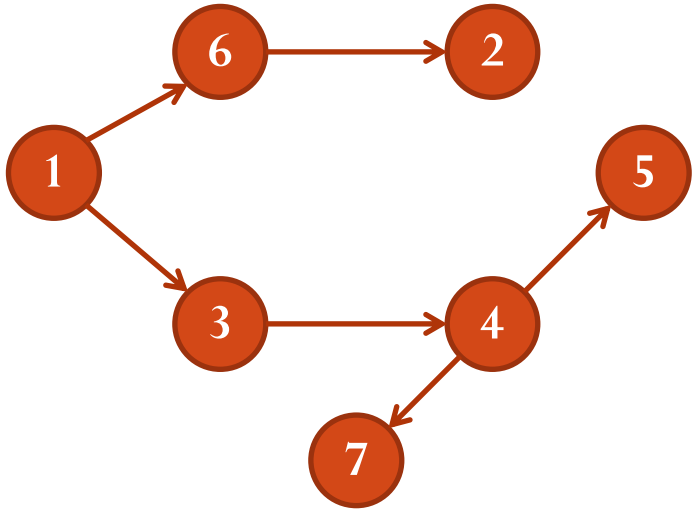


Node	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f		12	9	8	5	13	7
p	-	6	1	3	4	1	4
color	G	B	B	B	B	B	B

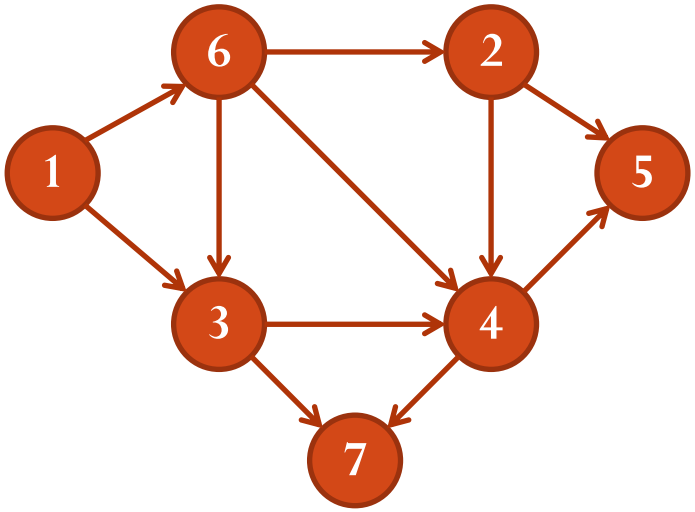


Depth First Search - DFS

DFS(1)



Node	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f	14	12	9	8	5	13	7
p	-	6	1	3	4	1	4
color	B	B	B	B	B	B	B

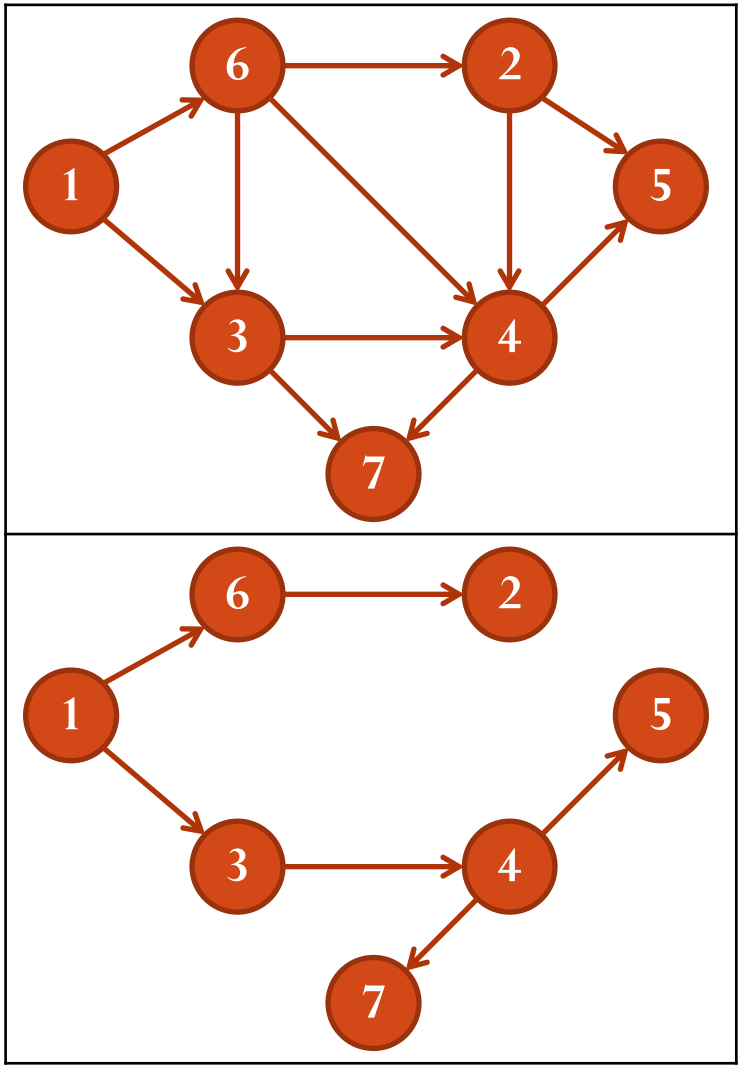


Depth First Search - DFS

- Kết quả của DFS là tập các cây DFS
- Phân loại cạnh
  - tree edge: (u,v) là cạnh cây nếu v được thăm từ u
  - back edge: (u,v) là cạnh ngược nếu v là tổ tiên của u
  - forward edge: (u,v) là cạnh xuôi nếu u là tổ tiên của v
  - crossing edge: cạnh ngang (các cạnh còn lại)

Depth First Search - DFS

- Phân loại cạnh
  - Cạnh cây: (1, 6), (1, 3), (6, 2), (3, 4), (4, 5), (4, 7)
  - Cạnh ngược:
  - Cạnh xuôi: (3, 7)
  - Cạnh ngang: (6, 3), (6, 4), (2, 4), (2,5)



Breadth First Search - BFS

- BFS(*u*): BFS xuất phát từ đỉnh *u*
  - Thăm *u*
  - Thăm các đỉnh kề với u và chưa được thăm (gọi là đỉnh mức 1)
  - Thăm các đỉnh kề với đỉnh mức 1 mà chưa được thăm (gọi là đỉnh mức 2)
  - Thăm các đỉnh kề với đỉnh ở mức 2 mà chưa được thăm (gọi là đỉnh mức 3)
  - . . .
- Cài đặt sử dụng cấu trúc hàng đợi

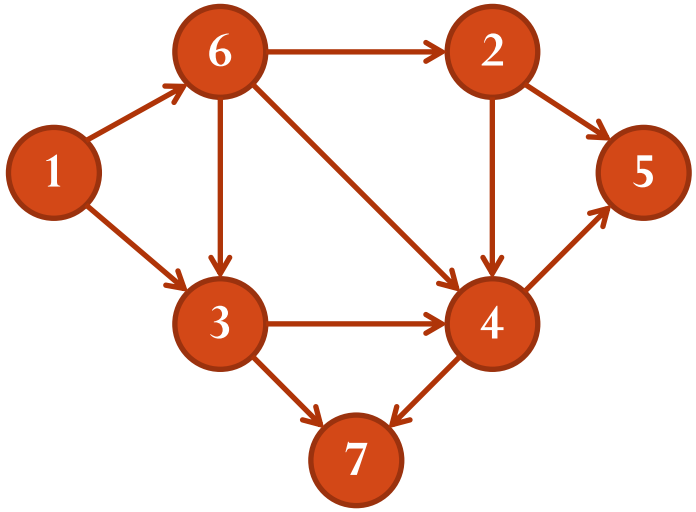
Breadth First Search - BFS

```
BFS(u) {
  d(u) = 0;
  Init a queue Q;
  enqueue(Q,u);
  color(u) = GRAY;
  while(Q not empty) {
   v = dequeue(Q);
   foreach(x adjacent node to
v) {
      if(color(x) = WHITE){
        d(x) = d(v) + 1;
        color(x) = GRAY;
        enqueue(Q,x);
      }
   }
  }
}
```

```
BFS() {
  foreach (node u of V) {
    color(u) = WHITE;
    p(u) = NIL;
  }
  foreach(node u of V) {
    if(color(u) = WHITE) {
       BFS(u);
    }
  }
}
```

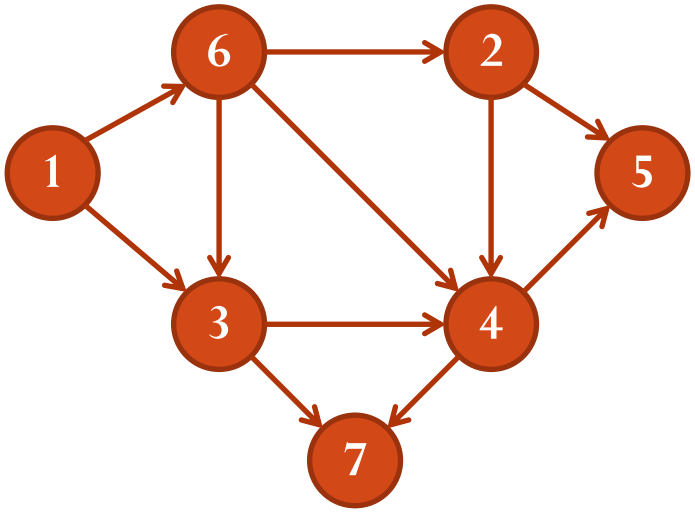
Breadth First Search - BFS

BFS(1)



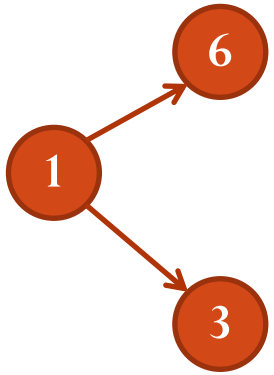
Breadth First Search - BFS

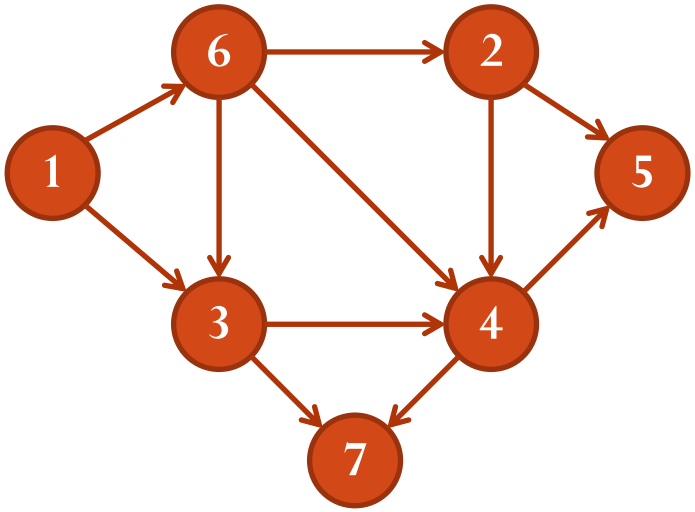
BFS(1)



Breadth First Search - BFS

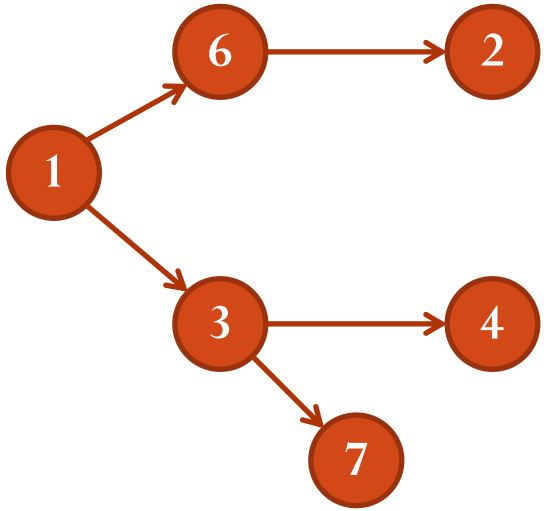
BFS(1)

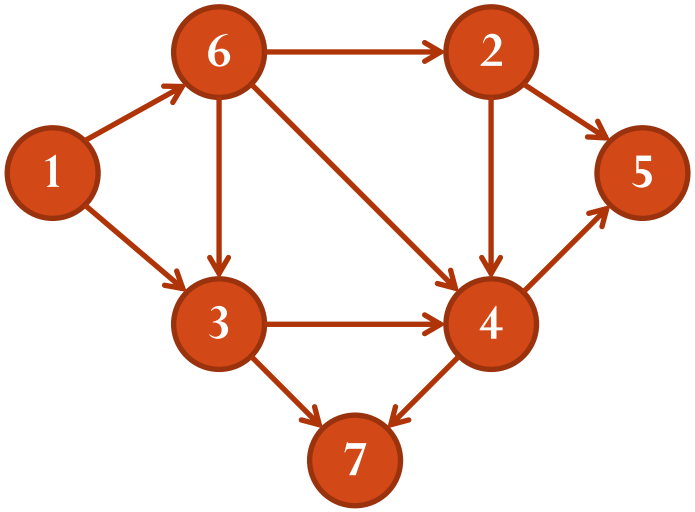




Breadth First Search - BFS

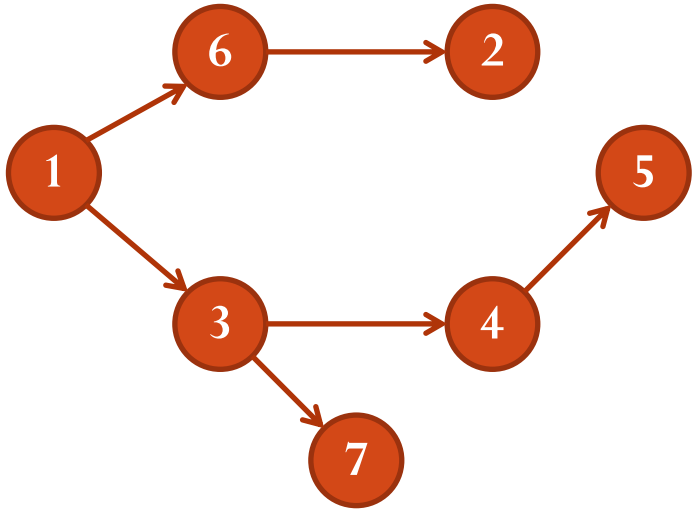
BFS(1)

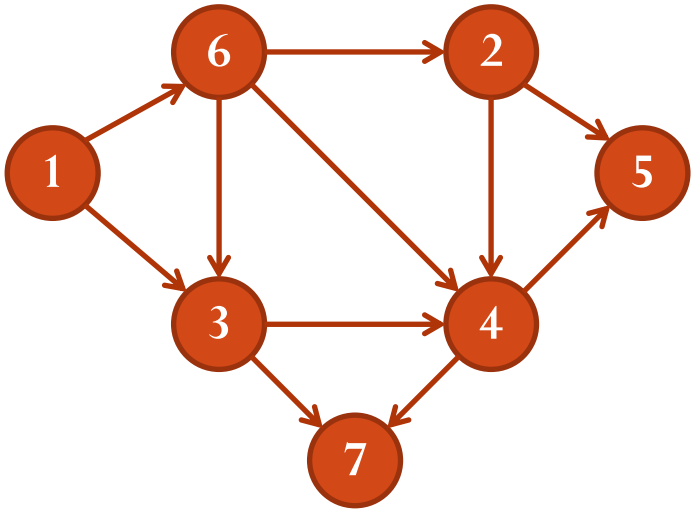




Breadth First Search - BFS

BFS(1)



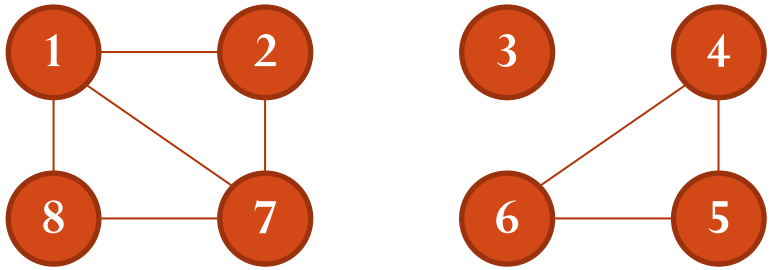


Ứng dụng DFS, BFS

- Tính toán thành phần liên thông của đồ thị vô hướng
- Tính thành phần liên thông mạnh của đồ thị có hướng
- Kiểm tra đồ thị hai phía
- Phát hiện chu trình
- Sắp xếp topo
- Tìm đường đi dài nhất trên cây

EXERCISE
CONNECTED COMPONENT

- Given a undirected graph G=(V, E) in which set of nodes V = {1,2,…,N}. Compute number of connected components.
- Example: following graph has 3 connected components
  - {1, 2, 7, 8}
  - {3}
  - {4, 5, 6}



EXERCISE
CONNECTED COMPONENT

- Input
  - Line 1: N and M ($1 \le N \le 10^6$, $1 \le M \le 10^7$)
  - Line i+1 (i = 1,…,M): u_i and v_i which are endpoints of the i^{th} edge
- Ouput
  - Number of connected components

Stdin	stdout
8 8 1 2 1 7 1 8 2 7 4 5 4 6 5 6 7 8	3

EXERCISE
CONNECTED COMPONENT

```
#include <bits/stdc++.h>
#include <vector>
#include <iostream>
#define MAX_N 100001
using namespace std;
int N,M;
vector<int> A[MAX_N];// A[v] la vector<int> ds cac dinh ke voi v
int visited[MAX_N];
int ans;
void input(){
    cin >> N >> M;
    for(int i = 1; i <= M; i++){
        int u,v;
        cin >> u >> v;
        A[u].push_back(v);  A[v].push_back(u);
    }
}
```

EXERCISE
CONNECTED COMPONENT

```
void init(){
    for(int i = 1; i<=N; i++) visited[i] = 0;
}
void DFS(int u){
    for(int j = 0; j < A[u].size(); j++){
        int v = A[u][j];
        if(!visited[v]){
            visited[v] = 1;
            DFS(v);
        }
    }
}
```

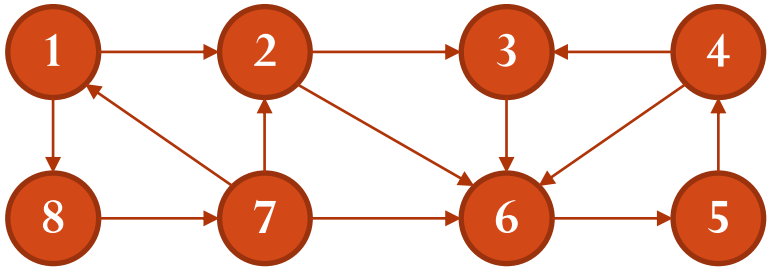
EXERCISE
CONNECTED COMPONENT

```
void solve(){
    init();
    ans = 0;
    for(int  v = 1; v <= N; v++)if(!visited[v]){
        ans++;// chuan bi dem mot thanh phan lien thong tiep theo
        DFS(v);
    }
    cout << ans;
}

int main(){
    input();
    solve();
}
```

EXERCISE
STRONGLY CONNECTED COMPONENT

- Given a directed graph G = (V,E) where V = {1,. . ., N} is the set of nodes. Compute the number of strongly connected components of G.
- Example, the graph below has 3 strongly connected components
  - {1,7,8}
  - {2}
  - {3,4,5,6}



EXERCISE
STRONGLY CONNECTED COMPONENT

- Input
  - Line 1: N and M ($1 \le N \le 10^6$, $1 \le M \le 10^7$)
  - Line i+1 ($i = 1,\ldots,M$): u_i and v_i which are endpoints of the i^{th} arc
- Ouput
  - Number of strongly connected components

stdin	stdout
8 13 1 2 1 8 2 3 2 6 3 6 4 3 4 6 5 4 6 5 7 1 7 2 7 6 8 7	3

EXERCISE
STRONGLY CONNECTED COMPONENT

```
#include <bits/stdc++.h>
#include <vector>
#include <iostream>
using namespace std;
#define MAX_N 100001

int n;
vector<int> A[MAX_N];
vector<int> A1[MAX_N];// residual graph

// data structure for DFS
int f[MAX_N];// finishing time
char color[MAX_N];
int t;
int icc[MAX_N];// icc[v] index of the strongly connected component containing v
int ncc;// number of connected components in the second DFS
int x[MAX_N];// sorted-list (inc finishing time) of nodes visited by DFS
int idx;
```

EXERCISE
STRONGLY CONNECTED COMPONENT

```
void buildResidualGraph(){// xay dung do thi bu
    for(int u = 1; u <= n; u++){
        for(int j = 0; j < A[u].size(); j++){
            int v = A[u][j];
            A1[v].push_back(u);
        }
    }
}
void init(){
    for(int v = 1; v <= n; v++){
        color[v] = 'W';
    }
    t = 0;
}
```

EXERCISE
STRONGLY CONNECTED COMPONENT

```
// DFS on the original graph
void dfsA(int s){
    t++;
    color[s] = 'G';
    for(int j = 0; j < A[s].size(); j++){
        int v = A[s][j];
        if(color[v] == 'W'){
            dfsA(v);
        }
    }
    t++;
    f[s] = t;
    color[s] = 'B';
    idx++;
    x[idx] = s;
}
```

EXERCISE
STRONGLY CONNECTED COMPONENT

```
void dfsA(){
    init();
    idx = 0;
    for(int v = 1; v <= n; v++){
        if(color[v] == 'W'){
            dfsA(v);
        }
    }
}
```

EXERCISE
STRONGLY CONNECTED COMPONENT

```
// DFS on the residual graph
void dfsA1(int s){
    t++;
    color[s] = 'G';
    icc[s] = ncc;
    for(int j = 0; j < A1[s].size(); j++){
        int v= A1[s][j];
        if(color[v] == 'W'){
            dfsA1(v);
        }
    }
    color[s] = 'B';
}
```

EXERCISE
STRONGLY CONNECTED COMPONENT

```
void dfsA1(){
    init();
    ncc = 0;
    for(int i = n; i >= 1; i--){
        int v = x[i];
        if(color[v] == 'W'){
            ncc++;
            dfsA1(v);
        }
    }
}
```

EXERCISE
STRONGLY CONNECTED COMPONENT

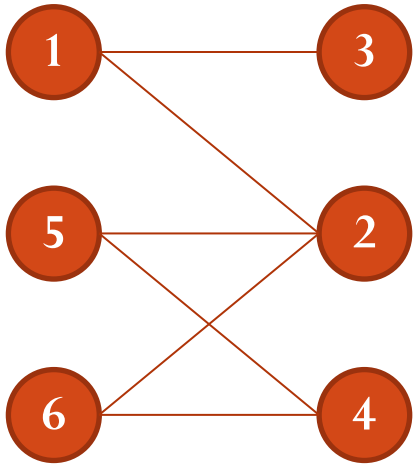
```
void solve(){
    dfsA();
    buildResidualGraph();
    dfsA1();
    cout << ncc;
}
void input(){
    int m;
    cin >> n >> m;
    for(int k = 1; k <= m; k++){
        int u,v;
        cin >> u >> v;
        A[u].push_back(v);
    }
}
int main(){
    input(); solve();
}
```

EXERCISE
STRONGLY CONNECTED COMPONENT

```
void input(){
    int m;
    cin >> n >> m;
    for(int k = 1; k <= m; k++){
        int u,v;
        cin >> u >> v;
        A[u].push_back(v);
    }
}
int main(){
    input();
    solve();
}
```

EXERCISE
BIPARTIE GRAPH

- Given undirected graph G = (V, E). Check whether or not G is a bipartie graph.



EXERCISE
BIPARTIE GRAPH

- Input
  - Line 1: N and M ($1 \le N \le 10^5$, $1 \le M \le 10^5$)
  - Line i+1 (i = 1,…,M): u_i and v_i which are endpoints of the i^{th} edge
- Ouput
  - Write 1 if G is bipartie graph, and write 0, otherwise

stdin	stdout
6 6 1 2 1 3 2 5 2 6 4 5 4 6	1

EXERCISE
BIPARTIE GRAPH

```
#include <bits/stdc++.h>
#include <vector>
#include <queue>
#include <iostream>
using namespace std;
#define MAX_N 100001
int N, M;
vector<int> A[MAX_N];
int d[MAX_N];// d[v] is the level of d
void input(){
    cin >> N >> M;
    for(int i = 1; i <= M; i++){
        int u,v;
        cin >> u >> v;
        A[u].push_back(v);
        A[v].push_back(u);
    }
}
```

EXERCISE
BIPARTIE GRAPH

```
int BFS(int u){
    queue<int> Q;
    Q.push(u);
    d[u] = 0;
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        for(int i = 0; i < A[v].size(); i++){
            int x = A[v][i];
            if(d[x] > -1){
                if(d[v] % 2 == d[x] % 2) return 0;
            }else{
                d[x] = d[v] + 1;
                Q.push(x);
            }
        }
    }
}
```

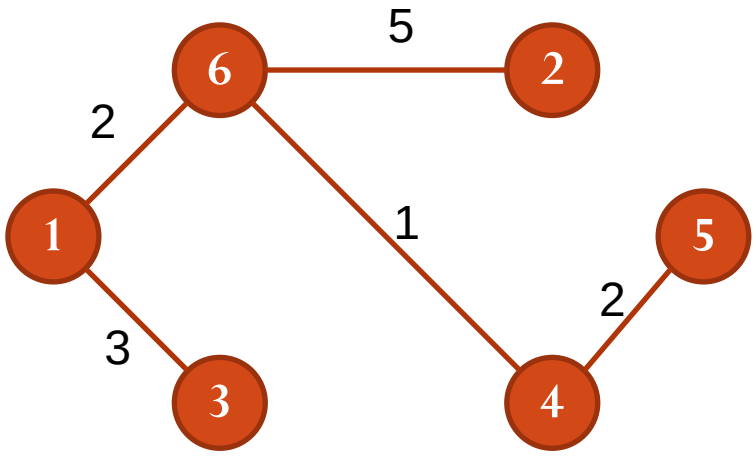
EXERCISE
BIPARTIE GRAPH

```
void solve(){
    init();
    int ans = 1;
    for(int v= 1; v <= N; v++) if(d[v]== -1){
        if(!BFS(v)){
            ans = 0; break;
        }
    }
    cout << ans ;
}

int main(){
    input();
    solve();
}
```

EXERCISE
LONGEST PATH ON A TREE

- Given a undirected tree G = (V, E) in which V = {1,…,N} is the set of nodes. Each edge (u,v) ∈ E has weight w(u,v). The length of a path is defined to be the sum of weights of edges of this path. Find the longest elementary path on G.



EXERCISE
LONGEST PATH ON A TREE

- Input
  - Line 1: N ($1 \le N \le 10^5$)
  - Line i + 1 (i = 1,…,N-1): u, v, w in which w is the weight of edge (u,v) ($1 \le w \le 100$)
- Output
  - The weight of the longest path on the given tree

stdin	stdout
6 1 3 3 1 6 2 2 6 5 4 5 2 4 6 1	10

EXERCISE
LONGEST PATH ON A TREE

```
#include <bits/stdc++.h>
#include <vector>
#include <iostream>
using namespace std;
#define MAX_N 100001

int N;
vector<int> A[MAX_N];// A[v][i] is the i^th adjacent node to v
vector<int> c[MAX_N];// c[v][i] is the weight of the i^th adjacent edge to v
int d[MAX_N];// d[v] is the distance from the source node to v in BFS
int p[MAX_N];// p[v] is the parent of v in BFS
```

EXERCISE
LONGEST PATH ON A TREE

```
void input(){
    std::ios::sync_with_stdio(false);
    cin >> N;
    for(int i = 1; i <= N-1; i++){
        int u,v,w;
        cin >> u >> v >> w;
        A[v].push_back(u);
        A[u].push_back(v);
        c[v].push_back(w);
        c[u].push_back(w);
    }
}
```

EXERCISE
LONGEST PATH ON A TREE

```
void BFS(int u){
    queue<int> Q;
    d[u] = 0;
    Q.push(u);
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        for(int i = 0; i < A[v].size(); i++){
            int x = A[v][i];
            if(d[x] > -1){ if(p[v] != x) cout << "FALSE" << endl;continue;}
            int w = c[v][i];
            Q.push(x);
            d[x] = d[v] + w;
            p[x] = v;
        }
    }
}
```

EXERCISE
LONGEST PATH ON A TREE

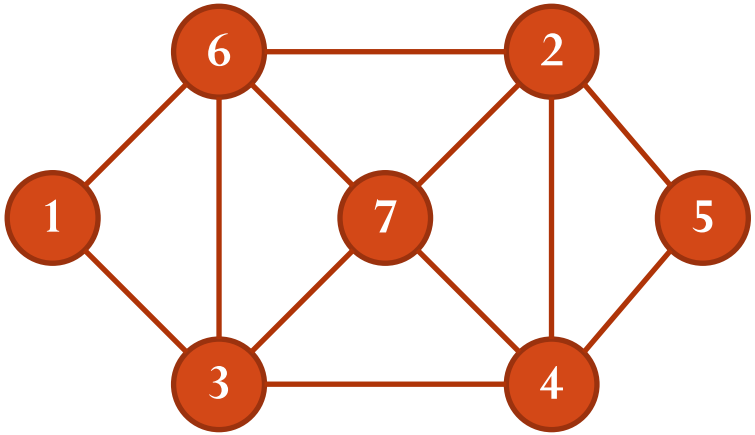
```
int findMax(){
    int max_d = -1;
    int u = -1;
    for(int v = 1; v <= N; v++){
        if(max_d < d[v]){
            max_d = d[v];
            u = v;
        }
    }
    return u;
}
void init(){
    for(int v = 1; v <= N; v++){ d[v] = -1; p[v] = -1;}
}
```

EXERCISE
LONGEST PATH ON A TREE

```
void solve(){
    init();
    BFS(1);
    int u = findMax();
    init();
    BFS(u);
    u = findMax();
    cout << d[u];
}
int main(){
    input();
    solve();
}
```

Eulerian and Hamiltonian cycles

- Given undirected graph G = (V, E)
  - Eulerian cycle of G is a cycle that passes each edge of G exactly once
  - Hamiltonian cycle of G is a cycle that visits each node of G exactly once
- A graph containing an Eulerian cycle is called Eulerian graph
- A graph containing a Hamiltonian cycle is called Hamiltonian graph



- Eulerian cycle: 1, 6, 3, 7, 6, 2, 5, 4, 2, 7, 4, 3, 1
- Hamiltonian cycle: 1, 6, 2, 5, 4, 7, 3, 1

Algorithm for finding an Eulerian cycle

```
euler(G = (V, A)) {
  Init stacks S, CE;
  select v of V;
  push(S,v);
  while(S is not empty) {
    x  = top(S);
    if(A(x) is not empty) {
      select y ∈ A(x);
      push(S,y);
      remove (x,y) from G;
    }else{
      x = pop(S); push(CE,x);
    }
  }
  sequence of nodes in CE forms an euler;
}
```

Algorithm for finding a Hamiltonian cycle

- Use backtracking
- Input $G = (V, E)$ in which
  - $V = \{1, 2, \ldots, n\}$
  - $A(v)$ set of adjacent nodes to v
- Solution representation: $x[1..n]$, the cycle will be x[1] → x[2] → . . . → x[n] → x[1]

```
TRY(k) {// try values for x[k] being aware of
        // x[1],…,x[k-1]
  for(v ∈ A(x[k-1]) {
    if(not mark[v]) {
      x[k] = v;
      mark[v] = true;
      if(k == n) {
        if(v ∈ A(x[1]) {
          retrieve a Hamiltonian cycle x;
        }
      }else{
        TRY(k+1);
      }
      mark[v] = false;
    }
  }
}
```

Minimum Spanning Tree

- Given a undirected graph $G = (V, E, w)$.
  - Each edge $(u, v) \in E$ has weight $w(u, v)$
  - If $(u, v) \notin E$ then $w(u, v) = \infty$
- A Spanning tree of G is a undirected connected graph with no cycle, and contains all node of G.
  - $T = (V, F)$ in which $F \subseteq E$
  - Weight of T: $w(T) = \sum_{e \in F} w(e)$
- Find a spanning tree such that the weight is minimal

Minimum Spanning Tree - KRUSKAL

- Main idea (greedy)
  - Each step, select the minimum-cost edge and insert it into the spanning tree (under construction) if no cycle is created.

```
KRUSKAL(G = (V,E)){
   ET = {}; C = E;
   while(|ET| < |V|-1 and |C| > 0){
      e = select minimum-cost edge of C;
      C = C \ {e};
      if(ET ∪ {e} create no cycle){
         ET = ET ∪ {e};
      }
   }
   if(|ET| = |V|-1) return ET;
   else return null;
}
```

Minimum Spanning Tree - KRUSKAL

```
#include <iostream>
#define MAX 100001

using namespace std;

// data structure for input graph
int N, M;
int u[MAX];
int v[MAX];
int c[MAX];
int ET[MAX];
int nET;
// data structure for disjoint-set
int r[MAX];// r[v] is the rank of the set v
int p[MAX];// p[v] is the parent of v
long long rs;
```

Minimum Spanning Tree - KRUSKAL

```
void link(int x, int y){
    if(r[x] > r[y]) p[y] = x;
    else{
        p[x] = y;
        if(r[x] == r[y]) r[y] = r[y] + 1;
    }
}
void makeSet(int x){
    p[x] = x;
    r[x] = 0;
}
int findSet(int x){
    if(x != p[x])
        p[x] = findSet(p[x]);
    return p[x];
}
```

Minimum Spanning Tree - KRUSKAL

```
void swap(int& a, int& b){
    int tmp = a; a = b; b = tmp;
}
void swapEdge(int i, int j){
    swap(c[i],c[j]);    swap(u[i],u[j]);    swap(v[i],v[j]);
}
int partition(int L, int R, int index){
    int pivot = c[index];
    swapEdge(index,R);
    int storeIndex = L;
    for(int i = L; i <= R-1; i++){
        if(c[i] < pivot){
            swapEdge(storeIndex,i);
            storeIndex++;
        }
    }
    swapEdge(storeIndex,R);
    return storeIndex;
}
```

Minimum Spanning Tree - KRUSKAL

```
void quickSort(int L, int R){
    if(L < R){
        int index = (L+R)/2;
        index = partition(L,R,index);
        if(L < index) quickSort(L,index-1);
        if(index < R) quickSort(index+1,R);
    }
}
void quickSort(){
    quickSort(0,M-1);
}
```

Minimum Spanning Tree - KRUSKAL

```
void solve(){
    for(int x = 1; x <= N; x++)  makeSet(x);
    quickSort();
    rs = 0;
    int count = 0;
    nET = 0;
    for(int i = 0;i < M; i++){
        int ru = findSet(u[i]);
        int rv = findSet(v[i]);
        if(ru != rv){
            link(ru,rv);
            nET++;  ET[nET] = i;
            rs += c[i];
            count++;
            if(count == N-1) break;
        }
    }
    cout << rs;
}
```

Minimum Spanning Tree - KRUSKAL

```
void input(){
    cin >> N >> M;
    for(int i = 0; i < M; i++){
        cin >> u[i] >> v[i] >> c[i];
    }
}
int main(){
    input();
    solve();
 }
```

Minimum Spanning Tree - PRIM

- Main idea (greedy)
  - Each step, select a node having minimum distance to the tree under construction for insertion
- Data structures
  - Each $v \notin V_T$
    - $d(v)$ is the distance from v to V_T:

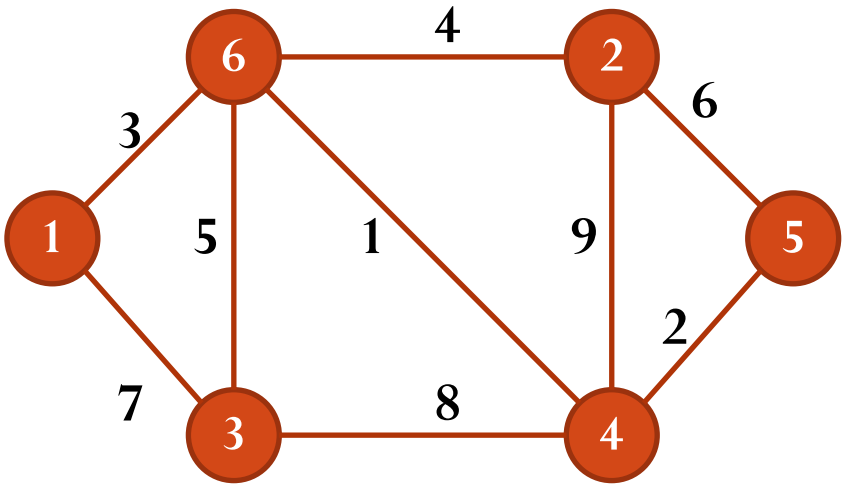
$$d(v) = \min\{w(v, u) \mid u \in V_T, (u, v) \in E\}$$

    - *near(v)*: the node $\in V_T$ having $w(v, \text{near}(v)) = d(v)$;

Minimum Spanning Tree - PRIM

```
PRIM(G = (V,E,w)) {
  select a random node s of V;
  for(v ∈ V) {
     d(v) = w(s,v); near(v) = s;
  }
  E_T = {}; V_T = {s};
  while(|V_T| ≠ |V|) {
    v = select a node ∈ V \ V_T having minimum d(v);
    V_T = V_T∪{v}; E_T = E_T ∪ {(v, near(v))};
    for(x ∈ V \ V_T) {
      if(d(x) > w(x,v) {
        d(x) = w(x,v);
        near(x) = v;
      }
    }
  }
  return (V_T, E_T);
}
```

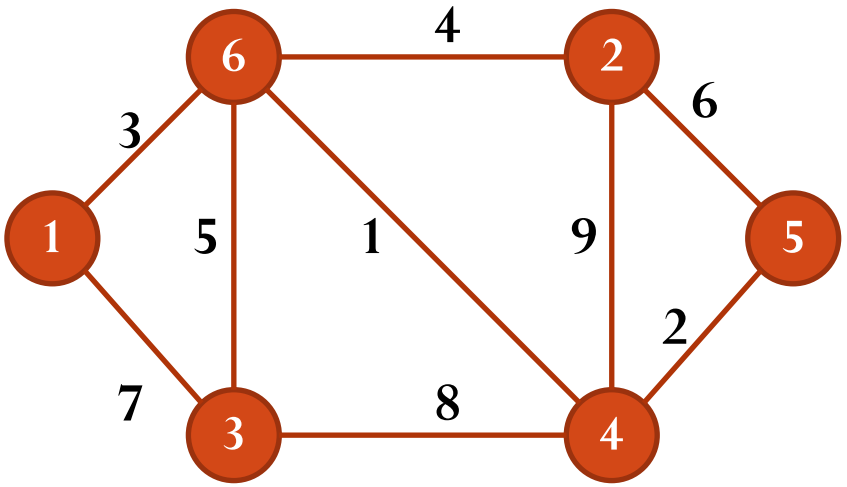
Minimum Spanning Tree - PRIM



- Each cell associated with a node v has label $(d(v), \text{near}(v))$
- Starting node $s = 1$

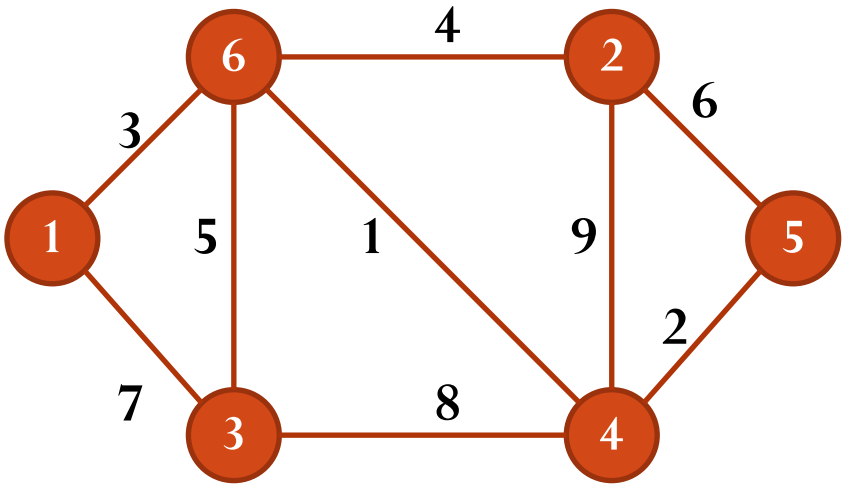
	1	2	3	4	5	6	E_T
Init	**(0,1)**	**(∞,1)**	**(7, 1)**	**(∞, 1)**	**(∞, 1)**	**(3,1)**	
Step 1	-						
Step 2	-						
Step 3	-						
Step 4	-						
Step 5	-						

Minimum Spanning Tree - PRIM



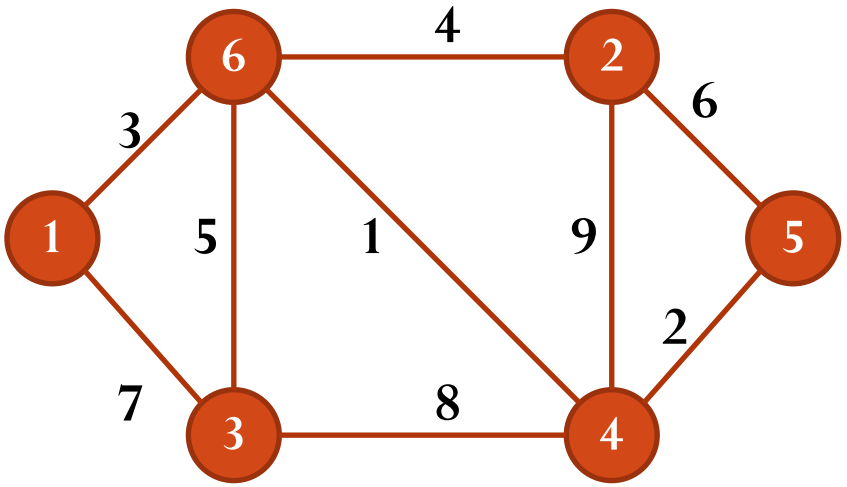
	1	2	3	4	5	6	E_T
Init	**(0,1)**	**(∞,1)**	**(7, 1)**	**(∞, 1)**	**(∞, 1)**	**(3,1) ***	**(1,6)**
Step 1	**-**	**(4,6)**	**(5,6)**	**(1,6)**	**(∞,1)**	**-**	
Step 2	**-**					**-**	
Step 3	**-**					**-**	
Step 4	**-**					**-**	
Step 5	**-**					**-**	

Minimum Spanning Tree - PRIM



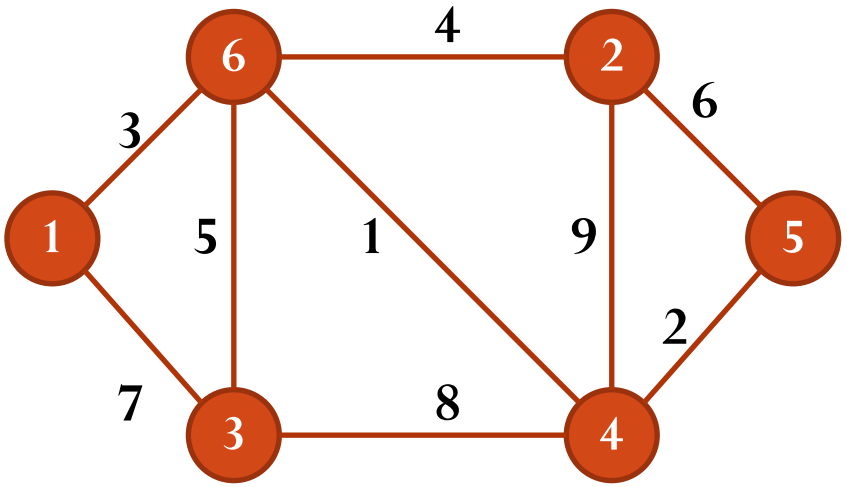
	1	2	3	4	5	6	E_T
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *	(1,6)
Step 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6), (4,6)
Step 2	-	(4,6)	(5,6)	-	(2,4)	-	
Step 3	-			-		-	
Step 4	-			-		-	
Step 5	-			-		-	

Minimum Spanning Tree - PRIM



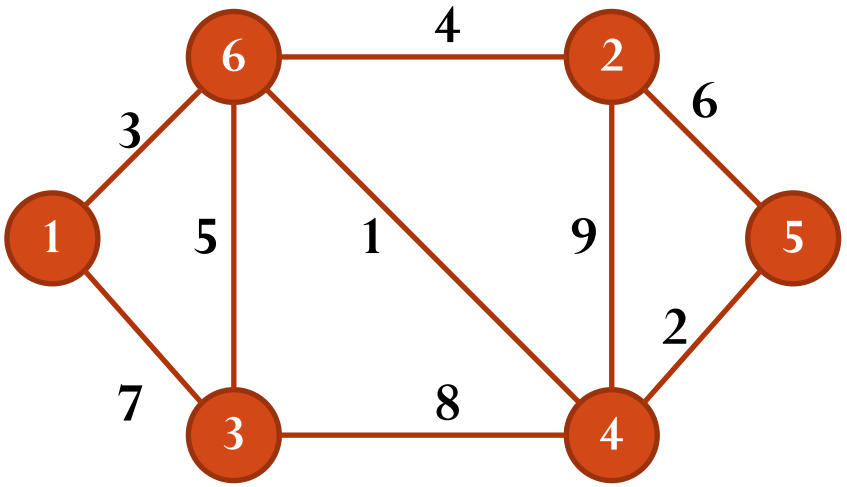
	1	2	3	4	5	6	E_T
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *	(1,6)
Step 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6),(4,6)
Step 2	-	(4,6)	(5,6)	-	(2,4) *	-	(1,6),(4,6),(4,5)
Step 3	-	(4,6)	(5,6)	-	-	-	
Step 4	-			-	-	-	
Step 5	-			-	-	-	

Minimum Spanning Tree - PRIM



	1	2	3	4	5	6	E_T
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *	(1,6)
Step 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6),(4,6)
Step 2	-	(4,6)	(5,6)	-	(2,4) *	-	(1,6),(4,6),(4,5)
Step 3	-	(4,6) *	(5,6)	-	-	-	(1,6),(4,6),(4,5),(2,6)
Step 4	-	-	(5,6)	-	-	-	
Step 5	-	-		-	-	-	

Minimum Spanning Tree - PRIM



	1	2	3	4	5	6	E_T
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1)	
Step 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6)
Step 2	-	(4,6)	(5,6)	-	(2,4) *	-	(1,6), (4,6)
Step 3	-	(4,6) *	(5,6)	-	-	-	(1,6), (4,6), (4,5)
Step 4	-	-	(5,6) *	-	-	-	(1,6), (4,6), (4,5), (2,6)
Step 5	-	-	-	-	-	-	(1,6), (4,6), (4,5), (2,6), (3,6)

Shortest Path Problem - Dijkstra

- Given a weighted graph $G = (V, E, w)$.
  - Each edge $(u, v) \in E$ has weight $w(u, v)$ which is non-negative
  - If $(u, v) \notin E$ then $w(u, v) = \infty$
- Given a node s of V, find the shortest path from s to other nodes of G

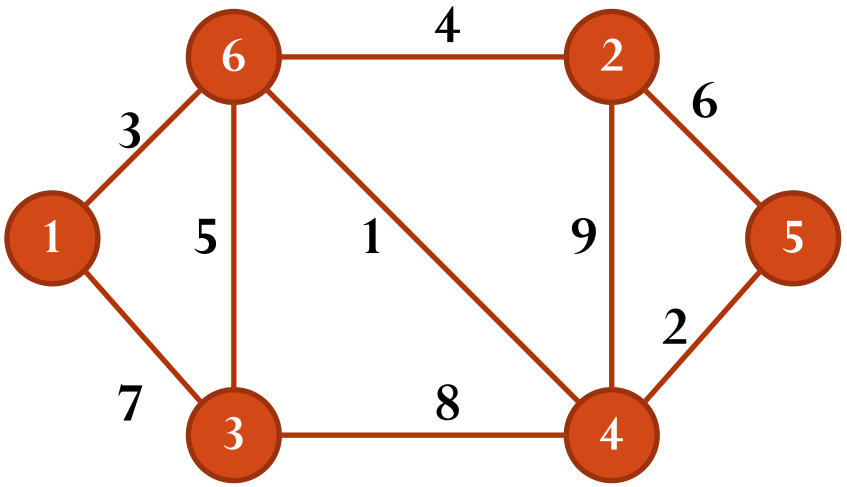
Shortest Path Problem - Dijkstra

- Main idea Dijkstra:
  - Each $v \in V$:
    - $\mathscr{P}(v)$ upper bound of the shortest path from s to v
    - $d(v)$: weight of $\mathscr{P}(v)$
    - $p(v)$: predecessor of v on $\mathscr{P}(v)$
  - Initialization
    - $\mathscr{P}(v) = \langle s, v \rangle$, $d(v) = w(s,v)$, $p(v) = s$
  - Upper bound improvement
    - If there exists a node u such that $d(v) > d(u) + w(u, v)$ then update:
      - $d(v) = d(u) + w(u, v)$
      - $p(v) = u$

Shortest Path Problem - Dijkstra

```
Dijkstra(G = (V,E,w)) {
  for(v ∈ V) {
     d(v) = w(s,v); p(v) = s;
  }
  S = V \ {s};
  while(S ≠ {}) {
    u = select a node ∈ S having minimum d(u);
    S = S \ {u};
    for(v ∈ S) {
      if(d(v) > d(u) + w(u,v) {
        d(v) = d(u) + w(u,v);
        p(v) = u;
      }
    }
  }
}
```

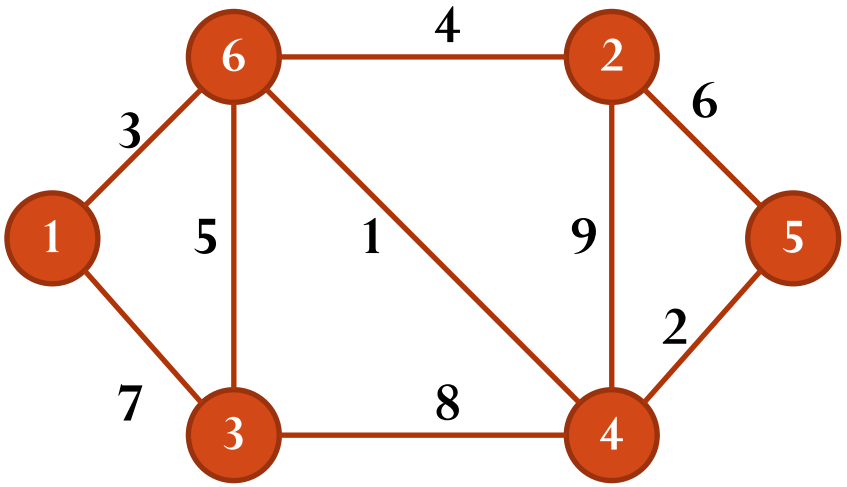
Shortest Path Problem - Dijkstra



- Each cell associated with v of the table has label $(d(v), p(v))$
- Starting node $s = 1$

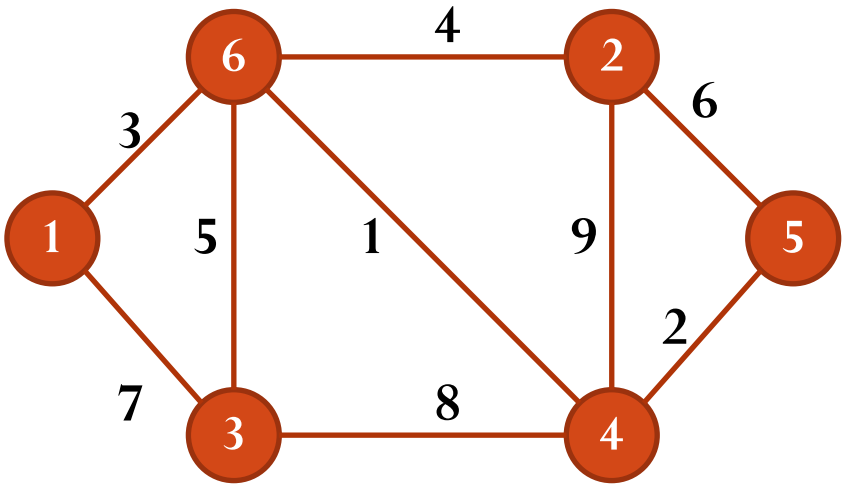
	1	2	3	4	5	6
Init	**(0,1)**	**(∞,1)**	**(7, 1)**	**(∞, 1)**	**(∞, 1)**	**(3,1)**
Step 1	**-**					
Step 2	**-**					
Step 3	**-**					
Step 4	**-**					
Step 5	**-**					

Shortest Path Problem - Dijkstra



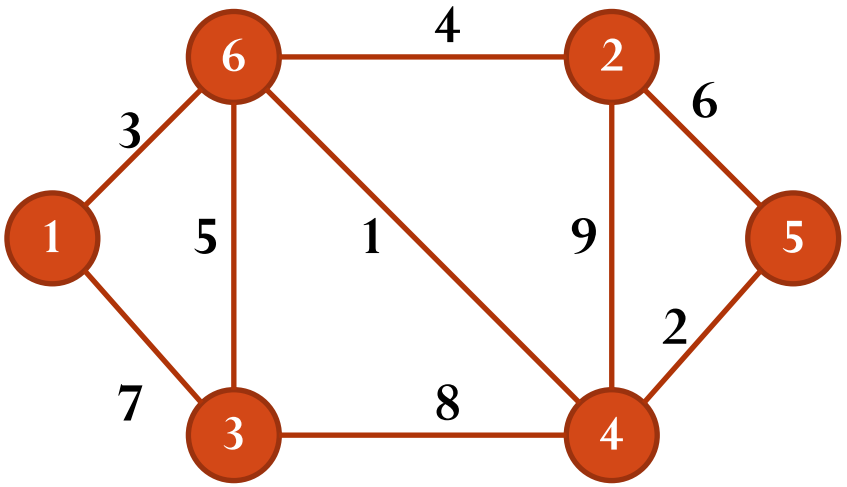
	1	2	3	4	5	6
Init	**(0,1)**	**(∞,1)**	**(7, 1)**	**(∞, 1)**	**(∞, 1)**	**(3,1) ***
Step 1	**-**	**(7,6)**	**(7,1)**	**(4,6)**	**(∞,1)**	**-**
Step 2	**-**					**-**
Step 3	**-**					**-**
Step 4	**-**					**-**
Step 5	**-**					**-**

Shortest Path Problem - Dijkstra



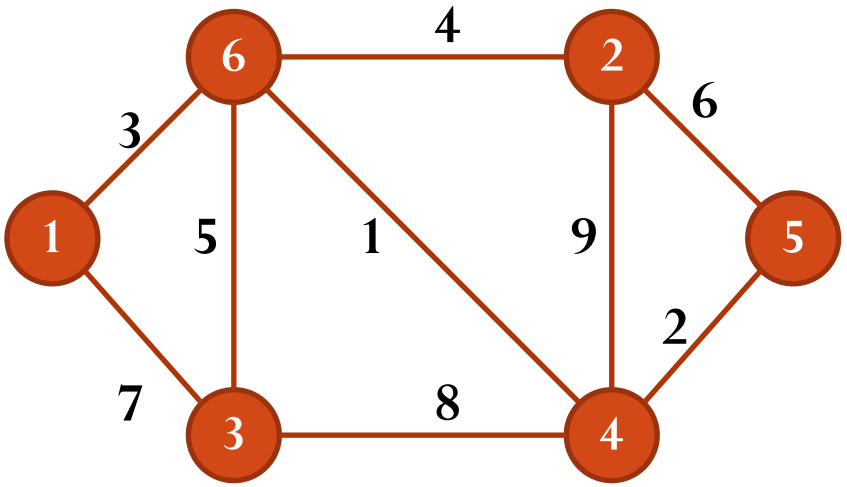
	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Step 2	-	(7,6)	(7,1)	-	(6, 4)	-
Step 3	-			-		-
Step 4	-			-		-
Step 5	-			-		-

Shortest Path Problem - Dijkstra



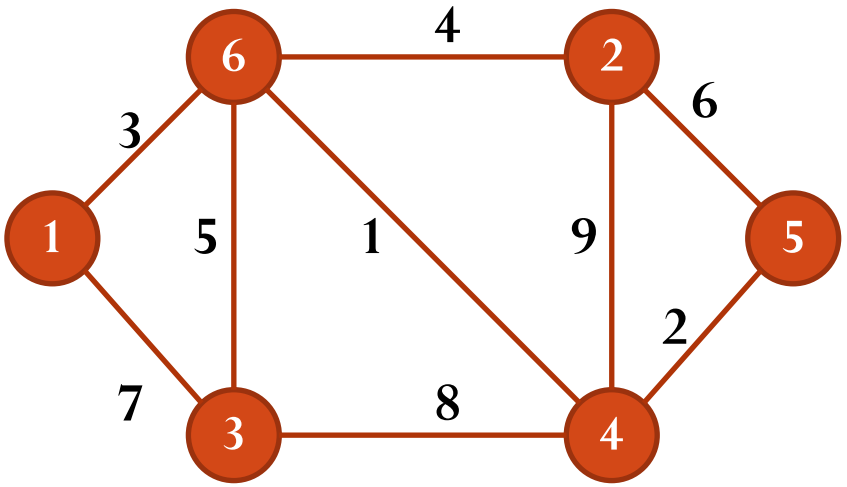
	1	2	3	4	5	6
Init	**(0,1)**	**(∞,1)**	**(7, 1)**	**(∞, 1)**	**(∞, 1)**	**(3,1) ***
Step 1	**-**	**(7,6)**	**(7,1)**	**(4,6) ***	**(∞,1)**	**-**
Step 2	**-**	**(7,6)**	**(7,1)**	**-**	**(6, 4) ***	**-**
Step 3	**-**	**(7,6)**	**(7,1)**	**-**	**-**	**-**
Step 4	**-**			**-**	**-**	**-**
Step 5	**-**			**-**	**-**	**-**

Shortest Path Problem - Dijkstra



	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Step 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Step 3	-	(7,6)	(7,1) *	-	-	-
Step 4	-	(7,6)	-	-	-	-
Step 5	-		-	-	-	-

Shortest Path Problem - Dijkstra

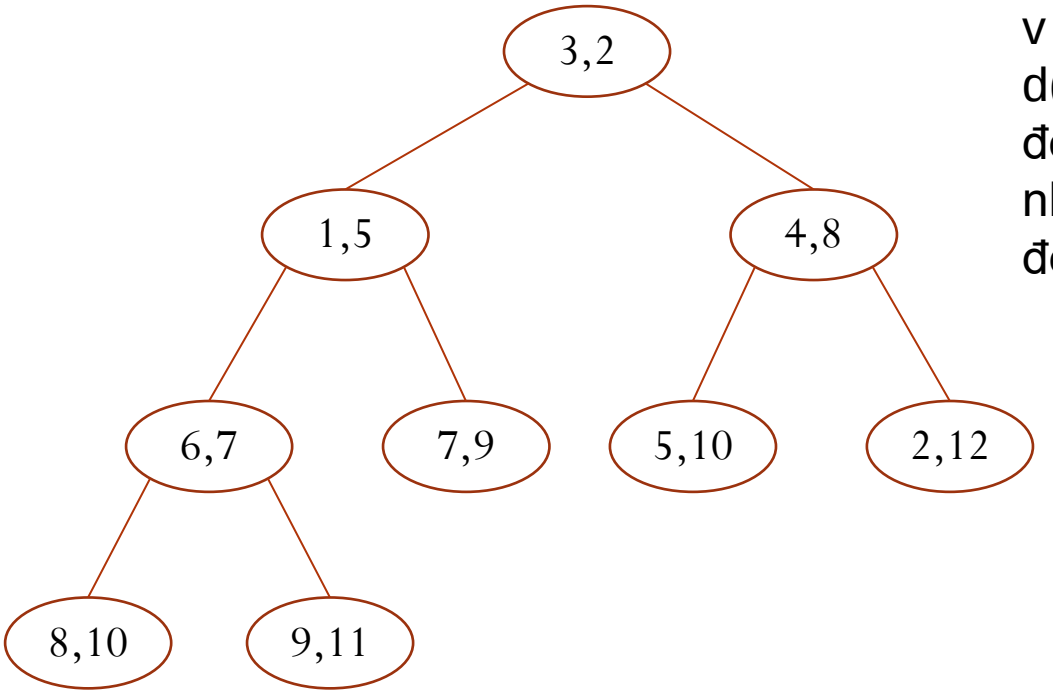


	1	2	3	4	5	6
Init	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Step 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Step 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Step 3	-	(7,6)	(7,1) *	-	-	-
Step 4	-	(7,6) *	-	-	-	-
Step 5	-	-	-	-	-	-

Shortest Path Problem - Dijkstra

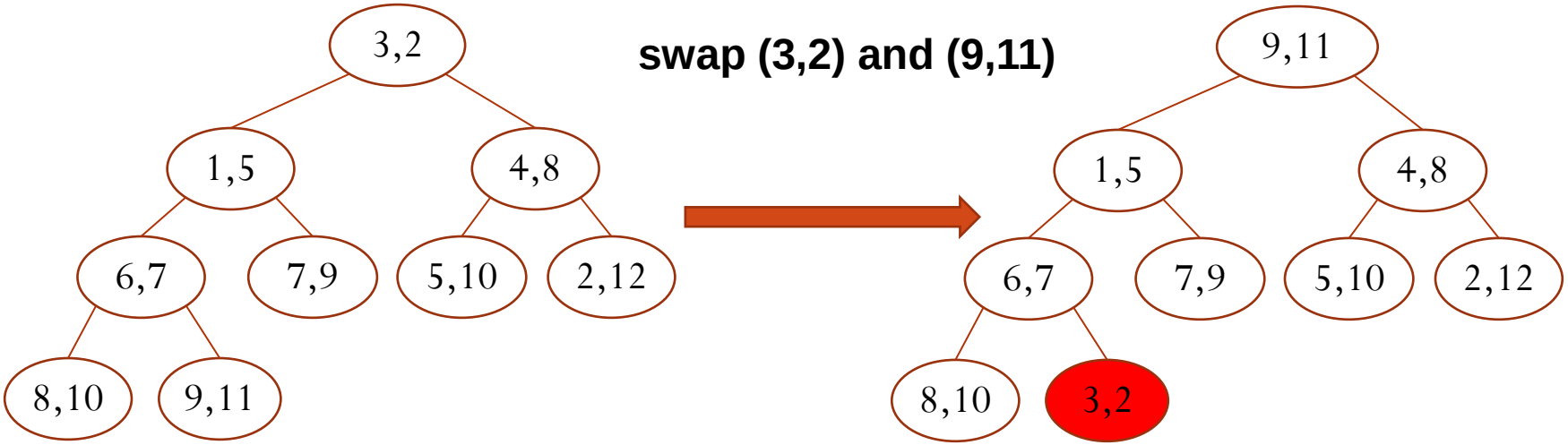
- Priority queues
  - Data structure storing elements and their keys
  - Efficient operations
    - (e,k) = deleteMin(): extract element e having minimum key k
    - insert(v,k): insert element e and its key k into the queue
    - updateKey(v,k): update the element with new key k
  - Implementation as binary min-heap
    - Elements are organized in a complete binary tree
    - Key of each element is less than or equal to the keys of its children

Shortest Path Problem - Dijkstra



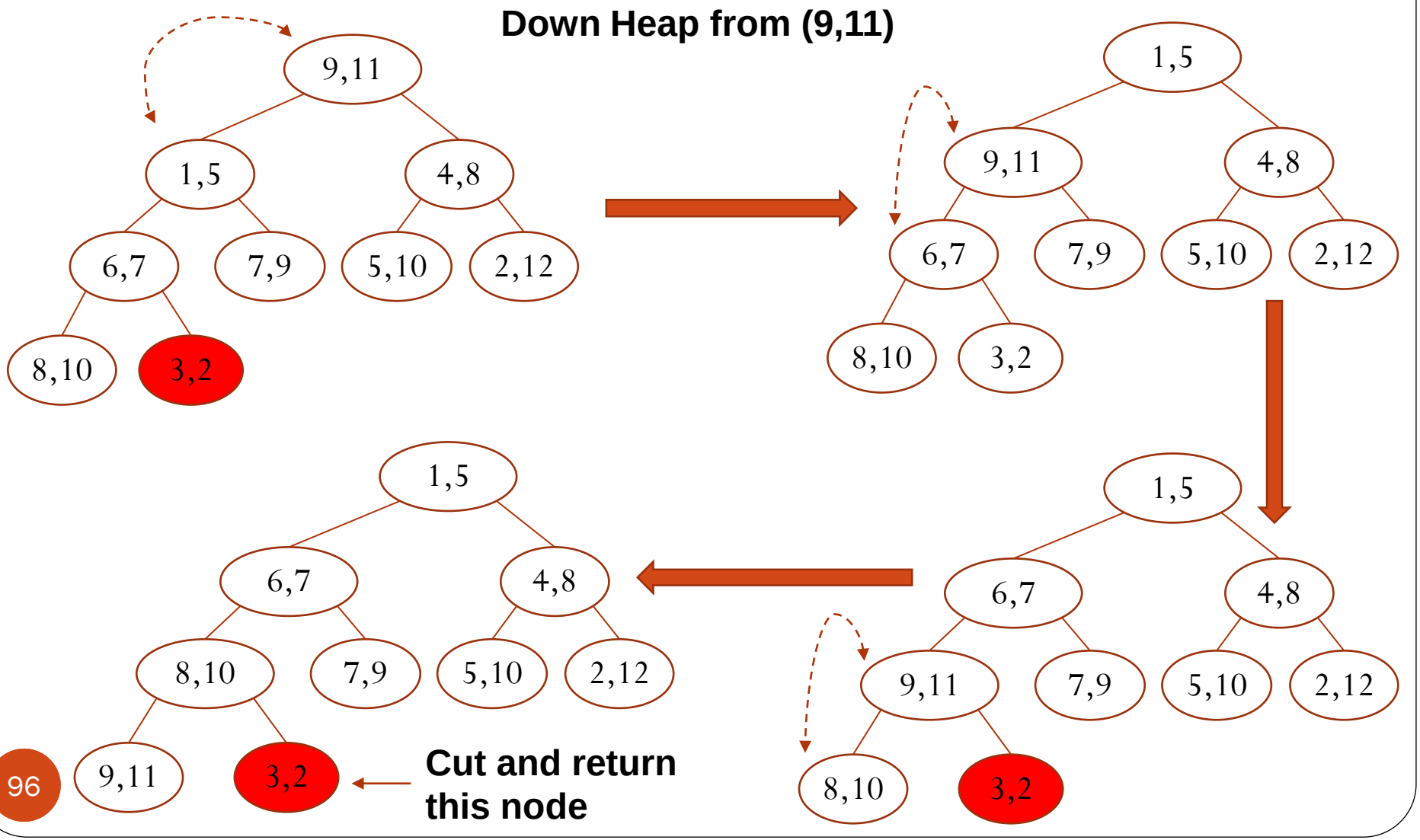
Mỗi nút của min-heap chứa (v,d(v)) trong đó v là đỉnh của đồ thị và d(v) là cận trên của độ dài đường đi ngắn nhất từ đỉnh xuất phát đến v

Shortest Path Problem - Dijkstra



Thao tác deleteMin

Shortest Path Problem - Dijkstra



Shortest Path Problem - Dijkstra

```
#include <stdio.h>
#include <vector>
#define MAX 100001
#define INF 1000000
using namespace std;

vector<int> A[MAX]; // A[v][i] is the i^th adjacent node to v
vector<int> c[MAX]; // c[v][i] is the weight of the i^th adjacent arc
                    // (v,A[v][i]) to v
int n,m; // number of nodes and arcs of the given graph
int s,t; // source and destination nodes

// priority queue data structure (BINARY HEAP)
int d[MAX];// d[v] is the upper bound of the length of the shortest path
from s to v (key)
int node[MAX];// node[i] the i^th element in the HEAP
int idx[MAX];// idx[v] is the index of v in the HEAP (idx[node[i]] = i)
int sH;// size of the HEAP
bool fixed[MAX];
```

Shortest Path Problem - Dijkstra

```
void swap(int i, int j){
    int tmp = node[i]; node[i] = node[j]; node[j] = tmp;
    idx[node[i]] = i; idx[node[j]] = j;
}
void upHeap(int i){
    if(i == 0) return;
    while(i > 0){
        int pi = (i-1)/2;
        if(d[node[i]] < d[node[pi]]){
            swap(i,pi);
        }else{
            break;
        }
        i = pi;
    }
}
```

Shortest Path Problem - Dijkstra

```
void downHeap(int i){
    int L = 2*i+1;
    int R = 2*i+2;
    int maxIdx = i;
    if(L < sH && d[node[L]] < d[node[maxIdx]]) maxIdx = L;
    if(R < sH && d[node[R]] < d[node[maxIdx]]) maxIdx = R;
    if(maxIdx != i){
        swap(i,maxIdx); downHeap(maxIdx);
    }
}
void insert(int v, int k){
    // add element key = k, value = v into HEAP
    d[v] = k;
    node[sH] = v;
    idx[node[sH]] = sH;
    upHeap(sH);
    sH++;
}
```

Shortest Path Problem - Dijkstra

```
int inHeap(int v){
    return idx[v] >= 0;
}
void updateKey(int v, int k){
    if(d[v] > k){
        d[v] = k;
        upHeap(idx[v]);
    }else{
        d[v] = k;
        downHeap(idx[v]);
    }
}
```

Shortest Path Problem - Dijkstra

```
int deleteMin(){
    int sel_node = node[0];
    swap(0,sH-1);
    sH--;
    downHeap(0);
    return sel_node;
}
```

Shortest Path Problem - Dijkstra

```
void input(){
    scanf("%d%d",&n,&m);
    for(int k = 1; k <= m; k++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        A[u].push_back(v);
        c[u].push_back(w);

    }
    scanf("%d%d",&s,&t);
}
```

Shortest Path Problem - Dijkstra

```
void init(int s){
    sH = 0;
    for(int v = 1; v <= n; v++){
        fixed[v] = false;  idx[v] = -1;
    }
    d[s] = 0;    fixed[s] = true;
    for(int i = 0; i < A[s].size(); i++){
        int v = A[s][i];
        insert(v,c[s][i]);
    }
}
```

Shortest Path Problem - Dijkstra

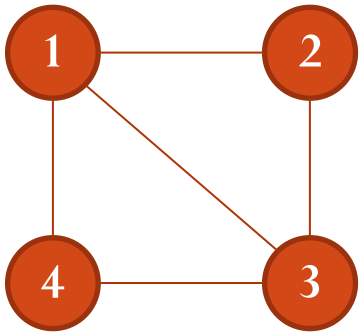
```
void solve(){
    init(s);
    while(sH > 0){
        int u = deleteMin();
        fixed[u] = true;
        for(int i = 0; i < A[u].size(); i++){
            int v = A[u][i];
            if(fixed[v]) continue;
            if(!inHeap(v)){
                    int w = d[u] + c[u][i];
                    insert(v,w);
            }else{
                if(d[v] > d[u] + c[u][i]){
                    updateKey(v,d[u]+c[u][i]);
                }
            }
        }
    }
    int rs = d[t]; if(!fixed[t]) rs = -1;
    printf("%d",rs);
}
```

Shortest Path Problem - Dijkstra

```
int main(){
    input();
    solve();
}
```

Exercises
COUNT SPANNING TREE

- Given a undirected connected graph G = (V,E) in which V = {1,…,N} is the set of nodes. Count the number of spanning trees of G.
- There are 8 spanning trees represented by list of edges as follows
  - (1,2) (1,3) (1,4)
  - (1,2) (1,3) (3,4)
  - (1,2) (1,4) (2,3)
  - (1,2) (1,4) (3,4)
  - (1,2) (2,3) (3,4)
  - (1,3) (1,4) (2,3)
  - (1,3) (2,3) (3,4)
  - (1,4) (2,3) (3,4)



Exercises
COUNT SPANNING TREE

- Input
  - Line 1: contains positive integers N and M ($1 \le N \le 20$, $1 \le M \le 25$)
  - Line i+1 (i = 1,…,M): contains u and v which are endpoints of the i^{th} edge of G
- Output
  - Write the number of spanning trees of G

stdin	stdout
4 5 1 2 1 3 1 4 2 3 3 4	8

Exercises
COUNT SPANNING TREE

```
#include <bits/stdc++.h>
#define MAX_N 101
#define MAX_M 1000
using namespace std;
int N,M;
int b[MAX_M];
int e[MAX_M];// (b[i],e[i]) la canh thu i cua do thi

int X[MAX_N];// model solution, set of indices of edges of spanning
trees
long long ans;

// data structure for disjoint-set
int r[MAX_N];// r[v] is rank of set v
int p[MAX_N];// p[v] is parent of v
long long rs;
```

Exercises
COUNT SPANNING TREE

```
void link(int x, int y){
    if(r[x] > r[y]) p[y] = x;
    else{
        p[x] = y;
        if(r[x] == r[y]) r[y] = r[y] + 1;
    }
}
void makeSet(int x){
    p[x] = x;
    r[x] = 0;
}
int findSet(int x){
    if(x != p[x])
        p[x] = findSet(p[x]);
    return p[x];
}
```

Exercises
COUNT SPANNING TREE

```
void input(){
    cin >> N >> M;
    for(int i = 1; i<= M; i++){
        cin >> b[i] >> e[i];
    }
}
void solution(){
    ans++;
}
```

Exercises
COUNT SPANNING TREE

```
int checkNoCycle(int val, int k){
    // check if set edges (b[X[1]],e[X[1]]), (b[X[2]],e[X[2]]),
    // ..., (b[X[val]],e[X[val]]) induces a cycle
    for(int i =1; i <= N; i++) makeSet(i);
    for(int j = 1; j < k; j++){
        int u = b[X[j]]; int v = e[X[j]];
        int ru = findSet(u); int rv = findSet(v);
        if(ru == rv) return 0;// node u and v  belong to the
                              // same set --> creating a cycle
        link(ru,rv);// otherwise, link two sets together
    }
    if(findSet(b[val]) == findSet(e[val])) return 0;
    return 1;
}
```

Exercises
COUNT SPANNING TREE

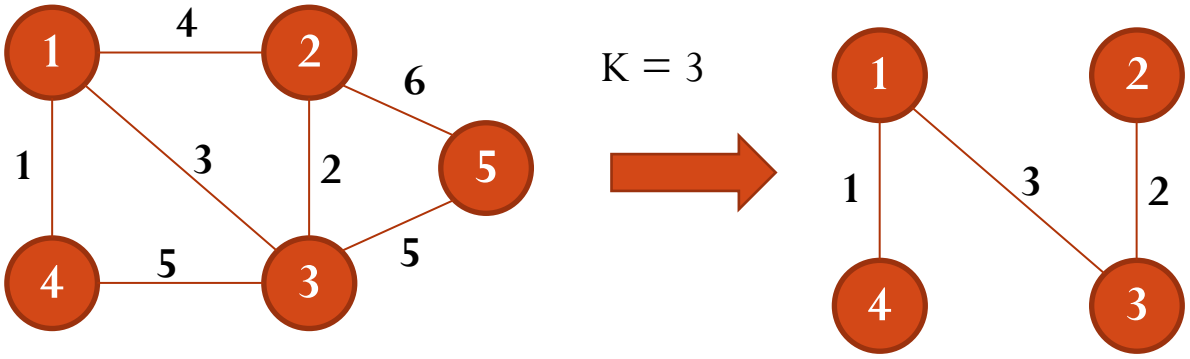
```
void TRY(int k){
    for(int v = X[k-1] + 1; v <= M; v++){
        if(checkNoCycle(v,k)){
            X[k] = v;
            if(k == N-1){
                solution();
            }else{
                TRY(k+1);
            }
        }
    }
}
```

Exercises
COUNT SPANNING TREE

```
void solve(){
    ans = 0;
    X[0] = 0;
    TRY(1);
    cout << ans;
}
int main(){
    input();
    solve();
}
```

Exercises
K-MST

- Given a undirected graph G=(V,E), w(e) is the weight of the edge e (e $\in$ E). Given a positive integer K, find the subgraph of G which is a tree containing exactly K edges having minimal weight.



Exercises
K-MST

```
#include <bits/stdc++.h>
#define MAX_N 101
#define MAX_M 1000
#define INF 10000000
using namespace std;
int N,M,K;
vector<int> A[MAX_N];
int b[MAX_M];
int e[MAX_M];
int w[MAX_M];
int X[MAX_N];// model solution, set of indices of edges of spanning trees
set<int> Ax[MAX_N];// Ax[v] is the set of adjacent nodes to v in the solution
int ans;
int W;
// data structures for DFS
int visited[MAX_N];
// data structure for disjoint-set
int r[MAX_N];// r[v] is rank of set v
int p[MAX_N];// p[v] is parent of v
long long rs;
```

Exercises
K-MST

```
void DFS(int u){
    visited[u] = 1;
    for(set<int>::iterator it = Ax[u].begin(); it !=
                Ax[u].end(); it++){
        int v = *it;
        if(!visited[v]){
            DFS(v);
        }
    }
}
```

Exercises
K-MST

```
int checkConnected(){
    set<int> Vx;
    for(int i = 1; i <= K; i++){
        Vx.insert(b[X[i]]); Vx.insert(e[X[i]]);
    }
    for(set<int>::iterator it = Vx.begin(); it != Vx.end(); it++){
        int u = *it; visited[u] = 0;
    }
    int cnt = 0;
    for(set<int>::iterator it = Vx.begin(); it != Vx.end(); it++){
        int u = *it;
        if(visited[u] == 0){
            cnt++;
            DFS(u);
        }
    }
    return cnt == 1;
}
```

Exercises
K-MST

```
void swap(int&a, int&b){
    int tmp = a; a = b; b = tmp;
}
void swapEdge(int i, int j){
    swap(w[i],w[j]);
    swap(b[i],b[j]);
    swap(e[i],e[j]);
}
int partition(int L, int R, int index){
    int pivot = w[index];
    swapEdge(index,R);
    int storeIndex = L;
    for(int i = L; i <= R-1; i++){
        if(w[i] < pivot){
            swapEdge(storeIndex,i);
            storeIndex++;
        }
    }
    swapEdge(storeIndex,R);
    return storeIndex;
}
```

Exercises
K-MST

```
void quickSort(int L, int R){
    if(L < R){
        int index = (L+R)/2;
        index = partition(L,R,index);
        if(L < index) quickSort(L,index-1);
        if(index < R) quickSort(index+1,R);
    }
}
void quickSort(){
    quickSort(1,M);
}
```

Exercises
K-MST

```
void input(){
    cin >> N >> M >> K;
    for(int i = 1; i<= M; i++){
        int u,v,wuv;
        cin >> u >> v >> wuv;
        A[u].push_back(v);
        A[v].push_back(u);
        b[i] = u;
        e[i] = v;
        w[i] = wuv;
    }
}
```

Exercises
K-MST

```
int check(int val, int k){
    // check if set edges (b[X[1]],e[X[1]]), (b[X[2]],e[X[2]]),
    // ..., (b[X[val]],e[X[val]])
    // induces a cycle
    for(int i =1; i <= N; i++) makeSet(i);
    for(int j = 1; j < k; j++){
        int u = b[X[j]]; int v = e[X[j]];
        int ru = findSet(u); int rv = findSet(v);
        if(ru == rv) return 0;
        link(ru,rv);
    }
    if(findSet(b[val]) == findSet(e[val])) return 0;
    return 1;
}
```

Exercises
K-MST

```
void solution(){
  if(checkConnected())
    if(W < ans) ans = W;
}
```

Exercises
K-MST

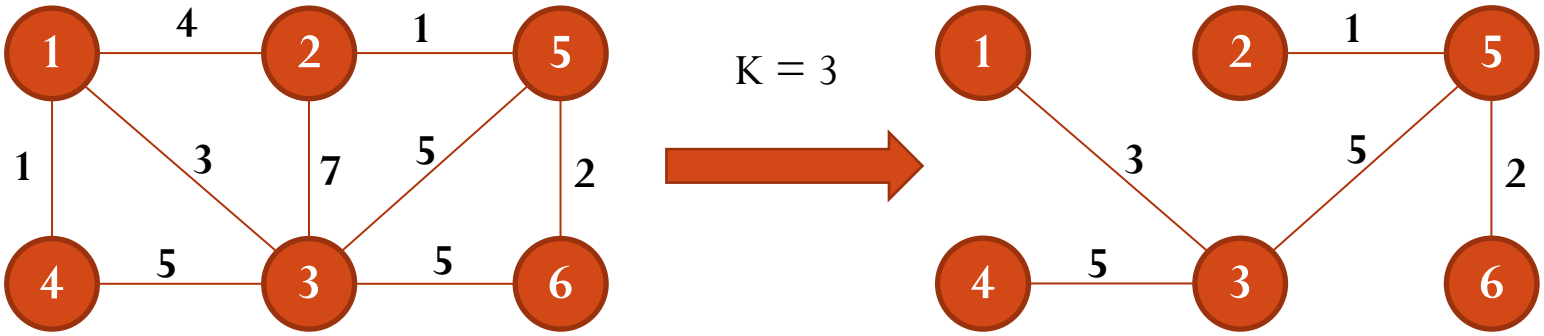
```
void TRY(int k){
    for(int v = X[k-1] + 1; v <= M - K + k; v++){
        if(check(v,k)){
            X[k] = v;
            W += w[v];
            Ax[b[v]].insert(e[v]);   Ax[e[v]].insert(b[v]);
            if(k == K){
                solution();
            }else{
                int g = W;
                for(int j = 1; j <= K-k; j++) g += w[X[k] + j];
                if(g < ans)
                    TRY(k+1);
            }
            Ax[b[v]].erase(e[v]);    Ax[e[v]].erase(b[v]);
            W -= w[v];
        }
    }
}
```

Exercises
K-MST

```
void solve(){
    quickSort();
    ans = INF;
    X[0] = 0;
    W = 0;
    TRY(1);
    cout << ans;
}
int main(){
    input();
    solve();
}
```

Exercises
BOUNDED-MST

- The diameter of a tree is defined to be the length of the longest path (in term of number of edges of the path) on that tree. Given a undirected graph G=(V,E), w(e) is the weight of the edge e (e $\in$ E). Given a positive integer K, find the minimum spanning tree T of G such that the diameter of T is less than or equal to K.



Exercises
BOUNDED-MST

```
#include <bits/stdc++.h>
#define MAX_N 101
#define MAX_M 1000
#define INF 10000000
using namespace std;
int N,M,K;
//vector<int> A[MAX_N];
int b[MAX_M];
int e[MAX_M];
int c[MAX_M];// c[i] is the weight of edge (b[i],e[i])
int X[MAX_N];// model solution, set of indices of edges of spanning trees
int W;
int ans;

// data structure for disjoint-set
int r[MAX_N];// r[v] is rank of set v
int p[MAX_N];// p[v] is parent of v
long long rs;
```

Exercises
BOUNDED-MST

```
void link(int x, int y){
    if(r[x] > r[y]) p[y] = x;
    else{
        p[x] = y;
        if(r[x] == r[y]) r[y] = r[y] + 1;
    }
}
void makeSet(int x){
    p[x] = x;
    r[x] = 0;
}
int findSet(int x){
    if(x != p[x])
        p[x] = findSet(p[x]);
    return p[x];
}
```

Exercises
BOUNDED-MST

```
void input(){
    cin >> N >> M >> K;
    for(int i = 1; i<= M; i++){
        int u,v,w;
        cin >> u >> v >> w;
        b[i] = u;
        e[i] = v;
        c[i] = w;
    }
}
```

Exercises
BOUNDED-MST

```
int check(int val, int k){
    // check if set edges (b[X[1]],e[X[1]]), (b[X[2]],e[X[2]]), ...,
    //  (b[X[val]],e[X[val]])
    // induces a cycle
    for(int i =1; i <= N; i++) makeSet(i);
    for(int j = 1; j < k; j++){
        int u = b[X[j]]; int v = e[X[j]];
        int ru = findSet(u); int rv = findSet(v);
        if(ru == rv) return 0;
        link(ru,rv);
    }
    if(findSet(b[val]) == findSet(e[val])) return 0;
    return 1;
}
```

Exercises
BOUNDED-MST

```
int selectMax(int d[],int N){
    int max_d = -1;
    int u = -1;
    for(int v = 1; v <= N; v++){
        if(max_d < d[v]){ max_d = d[v]; u = v;}
    }
    return u;
}
```

Exercises
BOUNDED-MST

```
int diameter(){
    vector<int> A[MAX_N];
    for(int i= 1; i <= N-1; i++){
        int u = b[X[i]];        int v = e[X[i]];
        A[u].push_back(v);      A[v].push_back(u);
    }
    queue<int> Q;
    int d[MAX_N] = {-1};
    for(int v = 1; v <= N; v++) d[v] = -1;
    int s = 1;
    d[s] = 0;
    Q.push(s);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = 0; i < A[u].size(); i++){
            int v = A[u][i];
            if(d[v] < 0){
                d[v] = d[u] + 1;
                Q.push(v);
            }
        }
    }
```

Exercises
BOUNDED-MST

```
    int x = selectMax(d,N);
    for(int v = 1; v <= N; v++) d[v] = -1;
    while(!Q.empty()) Q.pop();
    d[x] = 0;
    Q.push(x);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = 0; i < A[u].size(); i++){
            int v = A[u][i];
            if(d[v] < 0){
                d[v] = d[u] + 1; Q.push(v);
            }
        }
    }
    int y = selectMax(d,N);
    return d[y];
}
```

Exercises
BOUNDED-MST

```
void solution(){
   int dia = diameter();
   if(dia <= K){
        if(W < ans){
            ans = W;
        }
    }
}
```

Exercises
BOUNDED-MST

```
void TRY(int k){
    for(int v = X[k-1] + 1; v <= M; v++){
        int ok = check(v,k);
        if(ok){
            X[k] = v;
            W += c[v];
            if(k == N-1){
                solution();
            }else{
                TRY(k+1);
            }
            W -= c[v];
        }
    }
}
```

Exercises
BOUNDED-MST

```
void solve(){
    ans = INF;
    X[0] = 0;
    TRY(1);
    cout << ans;
}
int main(){
    input();
    solve();

}
```

Exercises
MaxClique

- Cho đồ thị vô hướng G=(V,E). Một đồ thị G'=(V', E') được gọi là đồ thị con của G nếu V' là tập con của V và E' là tập con của E. Hãy tìm đồ thị con của G là đồ thị đầy đủ và có số đỉnh lớn nhất

Exercises
MaxClique

```
#include <bits/stdc++.h>
#define MAX 1000

using namespace std;

int N,M;
set<int> A[MAX]; // Adj[v] la list cac dinh ke voi v

int X[MAX];
int best;// kich thuoc be lon nhat
int X_best[MAX];// luu tap dinh cua be cuc dai
```

Exercises
MaxClique

```
void input(){
    cin >> N >> M;
    for(int i = 0; i < M; i++){
        int u,v;
        cin >> u >> v;
        A[u].insert(v);
        A[v].insert(u);
    }
}
```

Exercises
MaxClique

```
bool check(int v, int k){
    for(int i = 1; i <= k-2; i++){
        if(A[X[i]].find(v) == A[X[i]].end()){
            return false;
        }
    }
    return true;
}
```

Exercises
MaxClique

```
void TRY(int k){// thu gia tri cho X[k]
    // da biet X[1,. . ., k-1]
    for(set<int>::iterator it = A[X[k-1]].begin(); it != A[X[k-1]].end(); it++){
        int v = *it;
        if(check(v,k)){
            X[k] = v;
            if(k > best){
                best = k;
                for(int i = 1; i <= k; i++) X_best[i] = X[i];
                //printf("Best = %d\n",best);
            }
            if(k < N){
                TRY(k+1);
            }
        }
    }
}
```

Exercises
MaxClique

```
void solve(){
    best = 1;
    for(int v = 1; v <= N; v++){
        X[1] = v;
        TRY(2);
    }
    cout << best;
    //printf("maxclique: ");
    //for(int i = 1; i <= best; i++) printf("%d ",X_best[i]); printf("\n");
}
int main(){
    input();
    solve();
}
```

THUẬT TOÁN ỨNG DỤNG

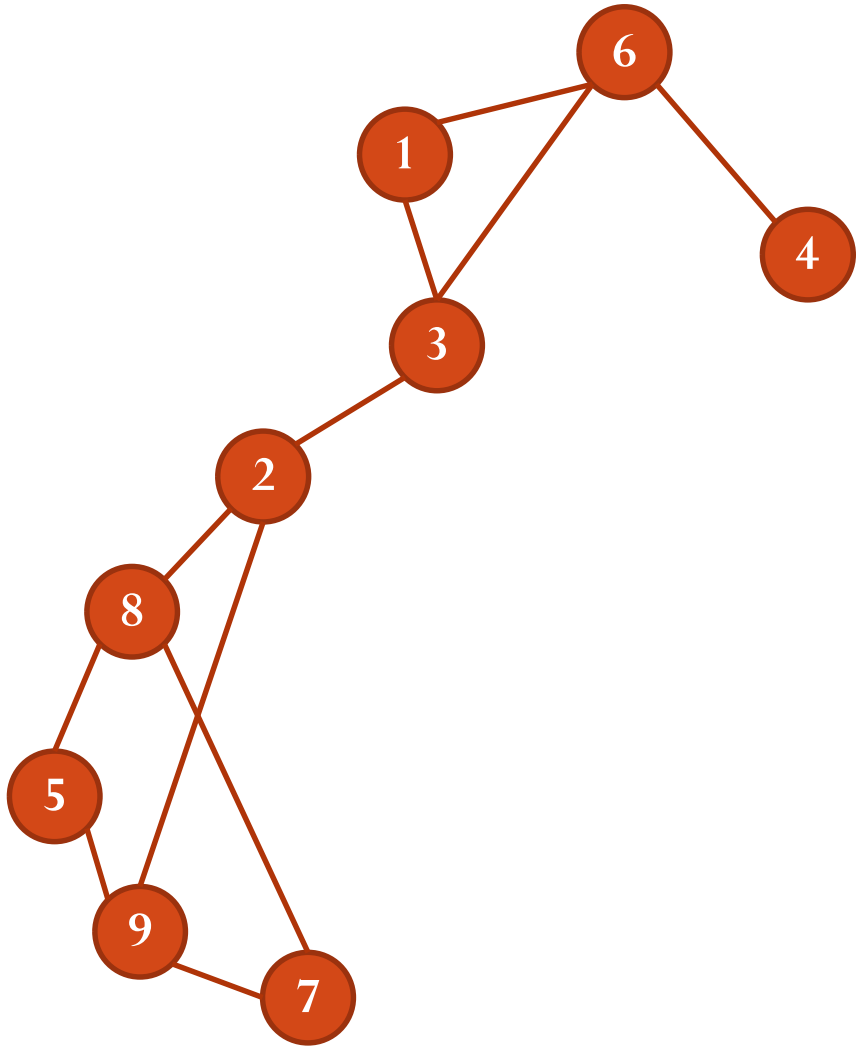
Tarjan DFS algorithm for finding Bridges and Articulation Points

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

Duyệt theo chiều sâu

- Cây DFS
  - DFS xuất phát từ một đỉnh cho phép thăm các đỉnh con cháu của nó trên cây DFS
- Cấu trúc dữ liệu duy trì
  - num[v]: thời điểm đỉnh v được thăm
  - low[v]: giá trị num nhỏ nhất của các đỉnh x sao cho có cạnh ngược (u,x) với u là 1 đỉnh con cháu nào đó của v

DFS(6)

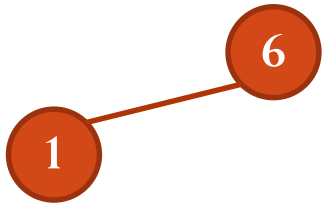


3

DFS(6)

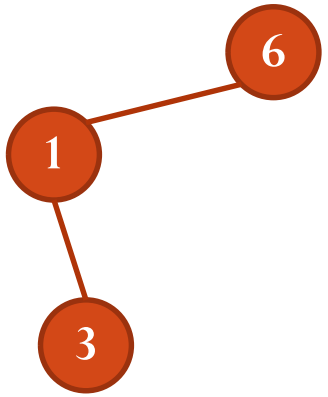
num[6] = 1, low[6] = 1

DFS(6)



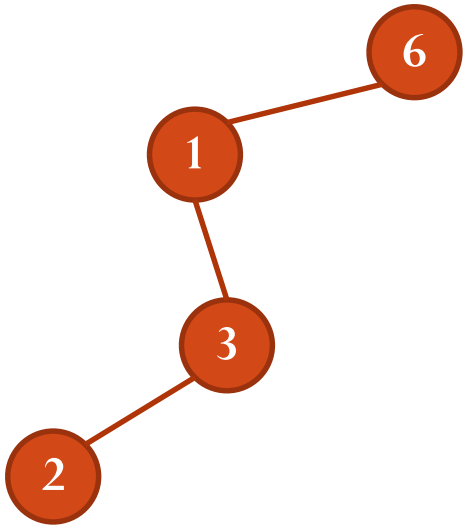
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2

DFS(6)



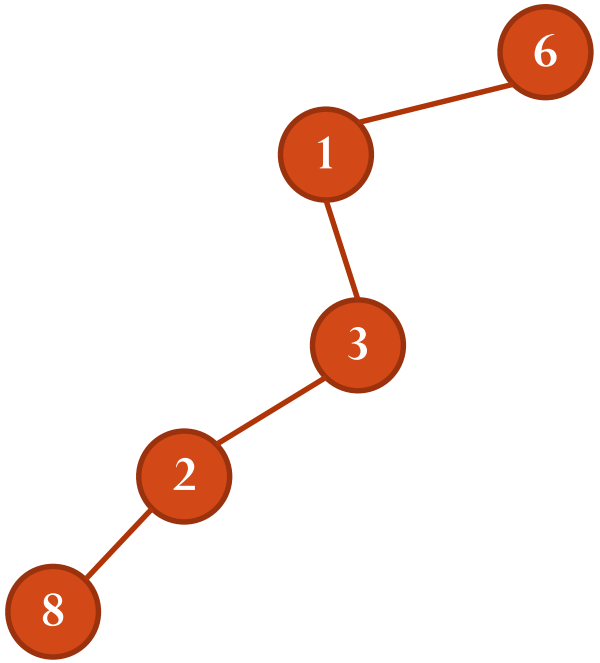
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3

DFS(6)



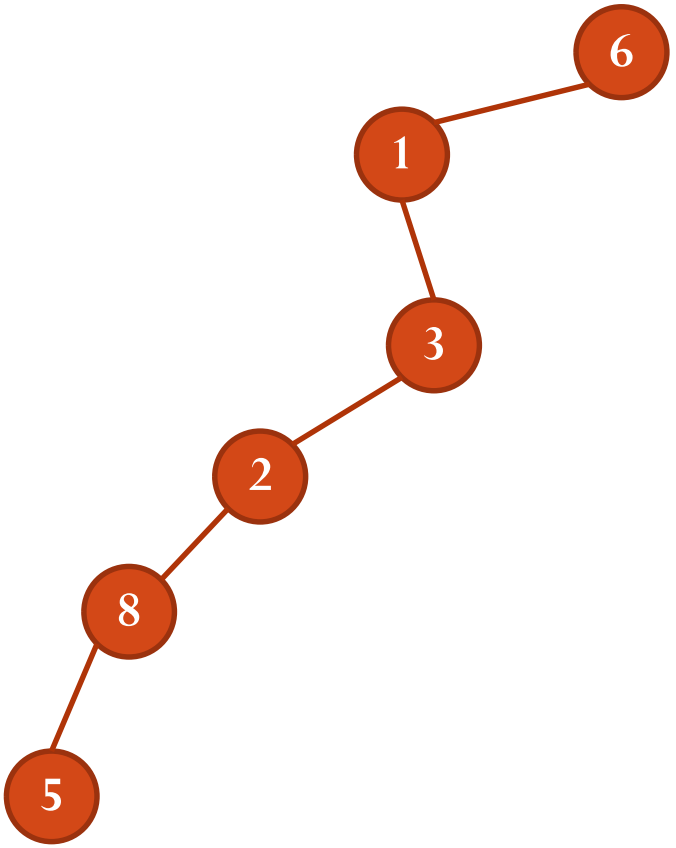
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4

DFS(6)



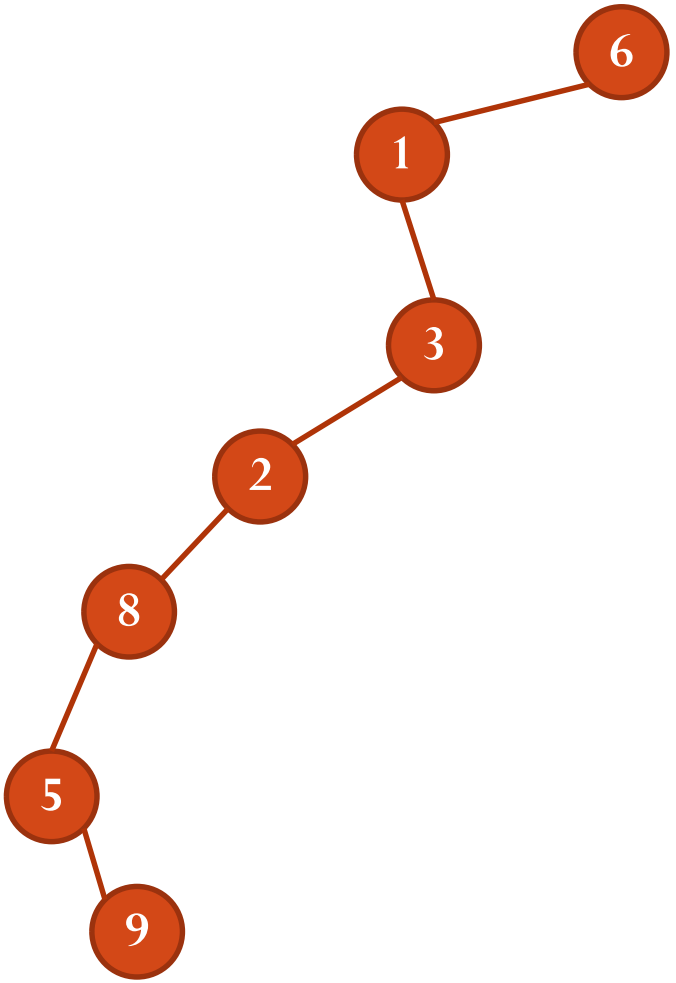
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5

DFS(6)



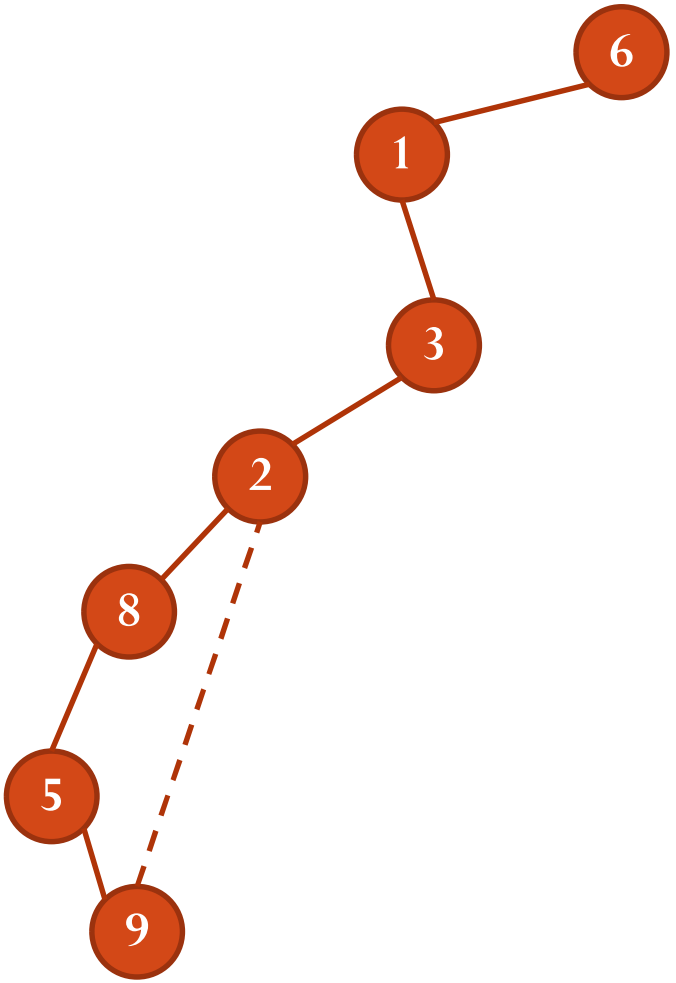
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6

DFS(6)



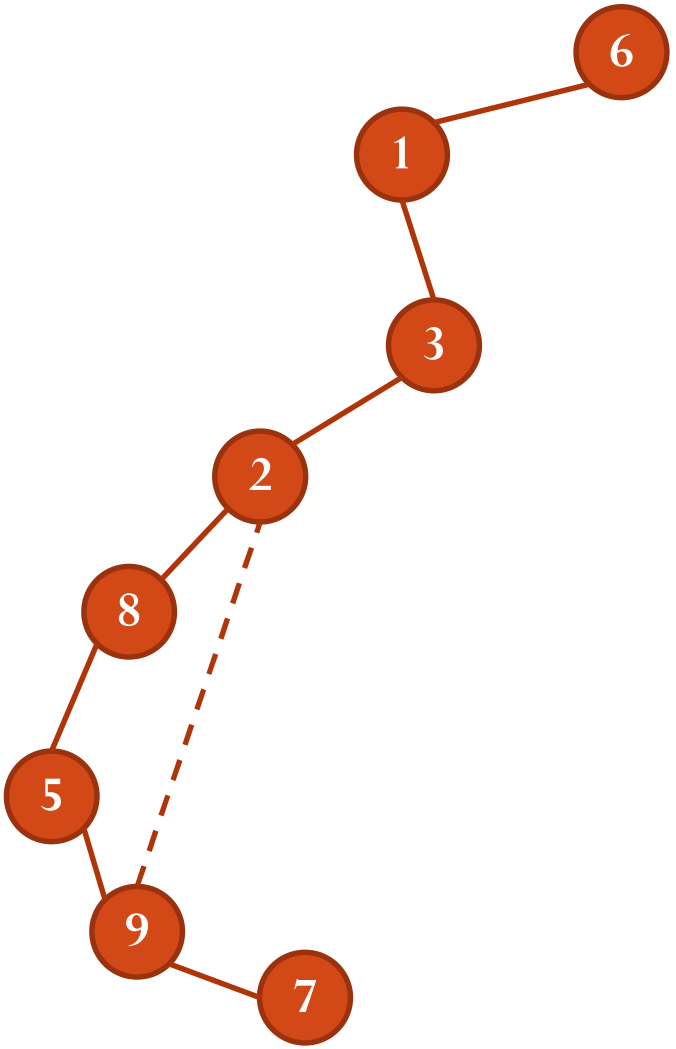
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = 7

DFS(6)



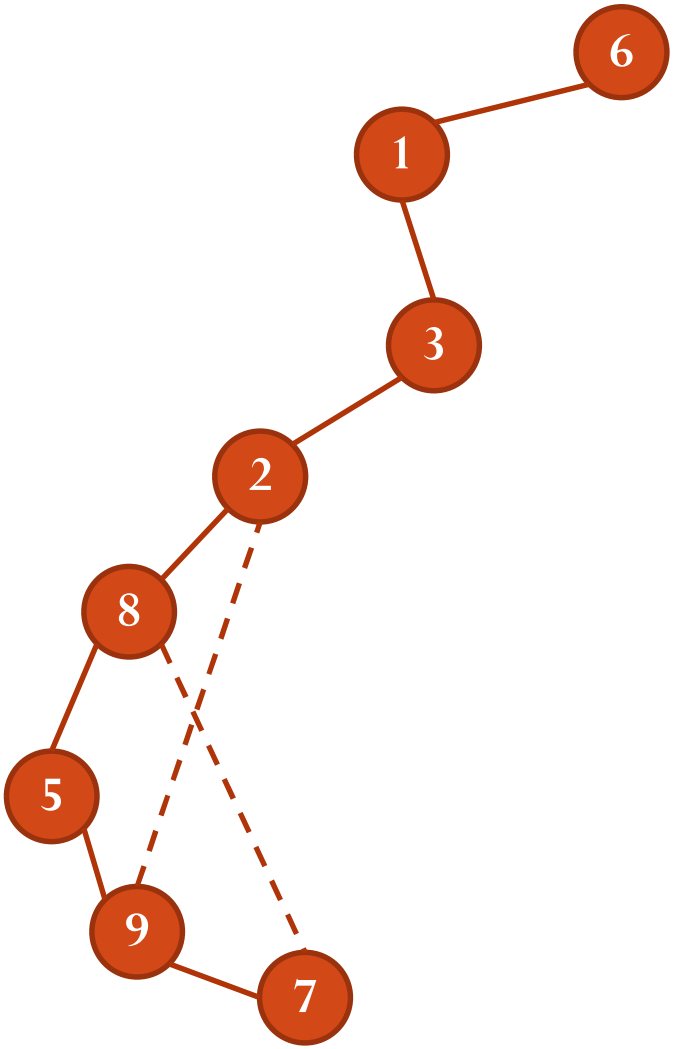
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = num[2] = 4

DFS(6)



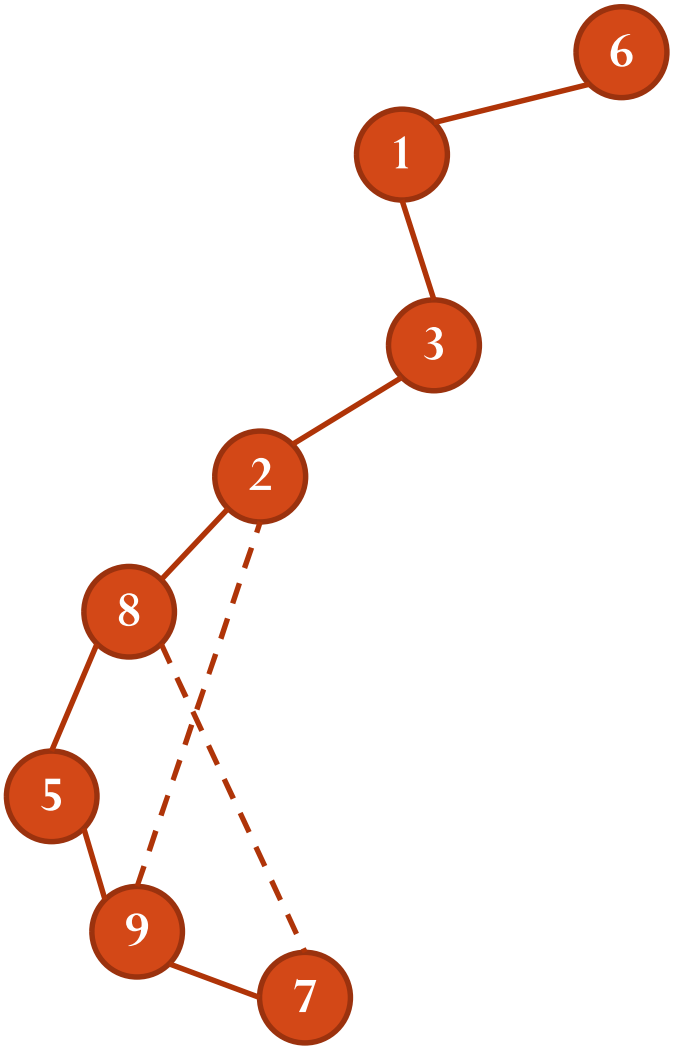
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = 8

DFS(6)



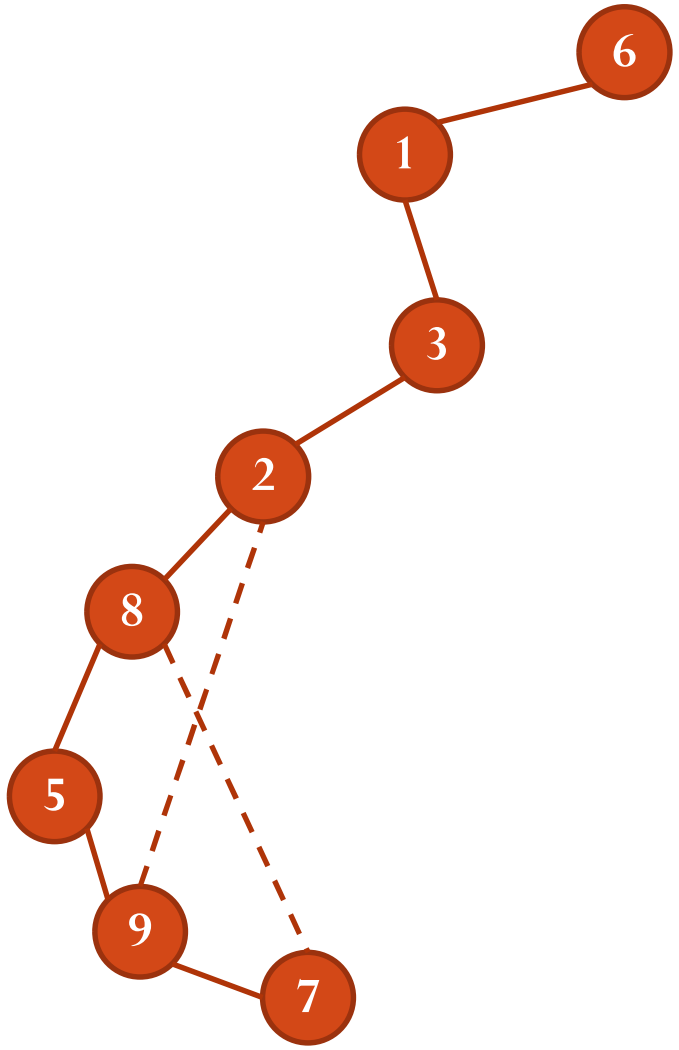
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

DFS(6)



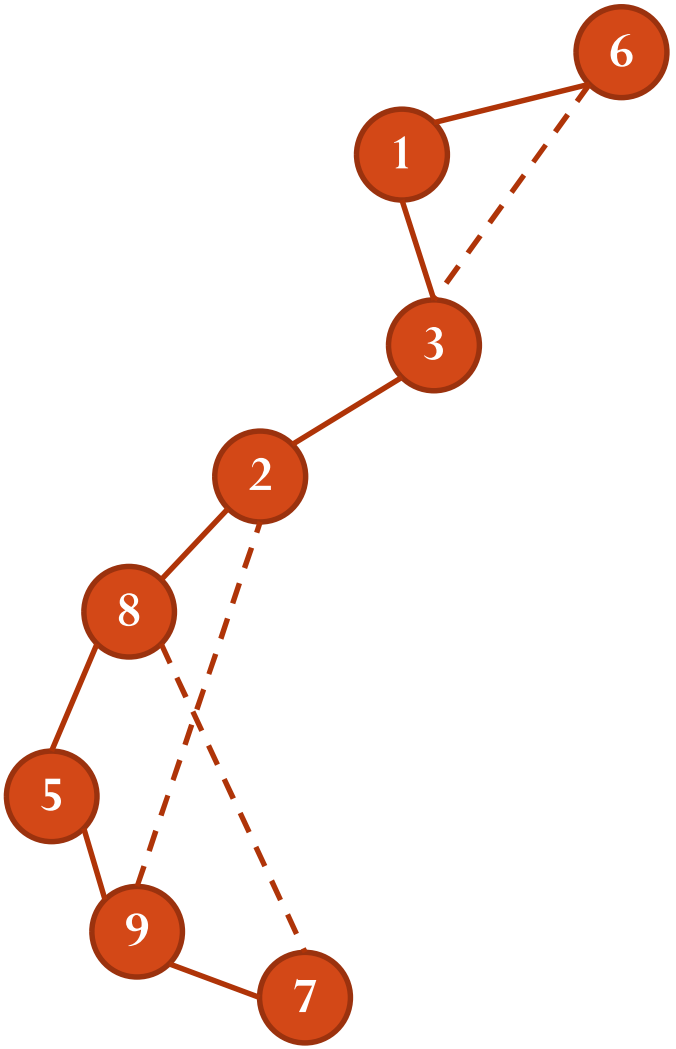
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

DFS(6)



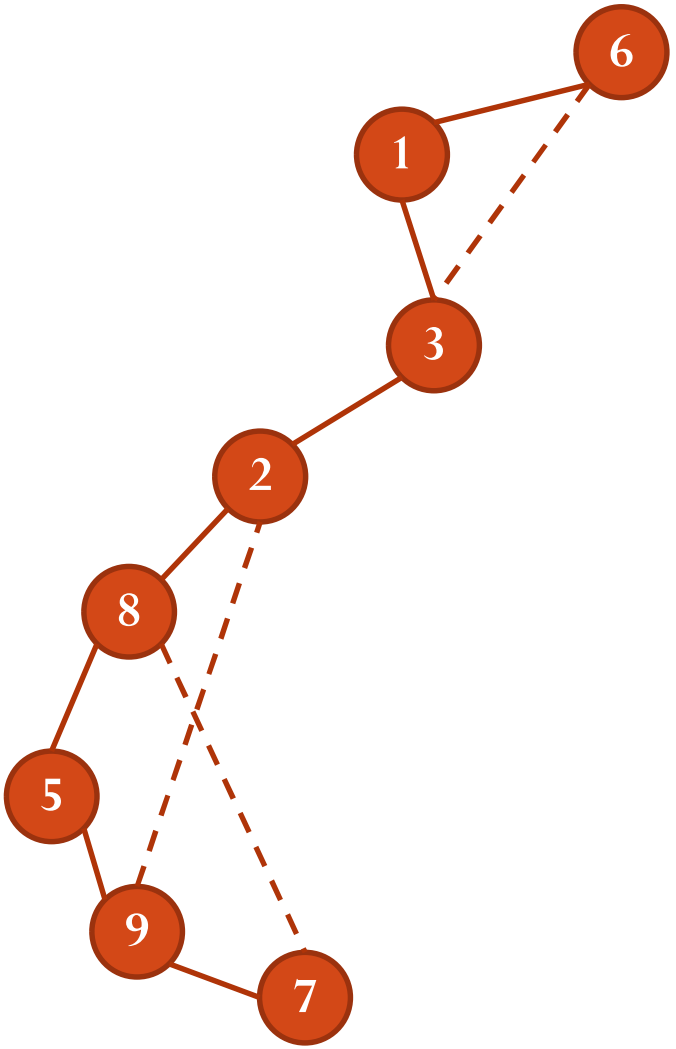
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

DFS(6)



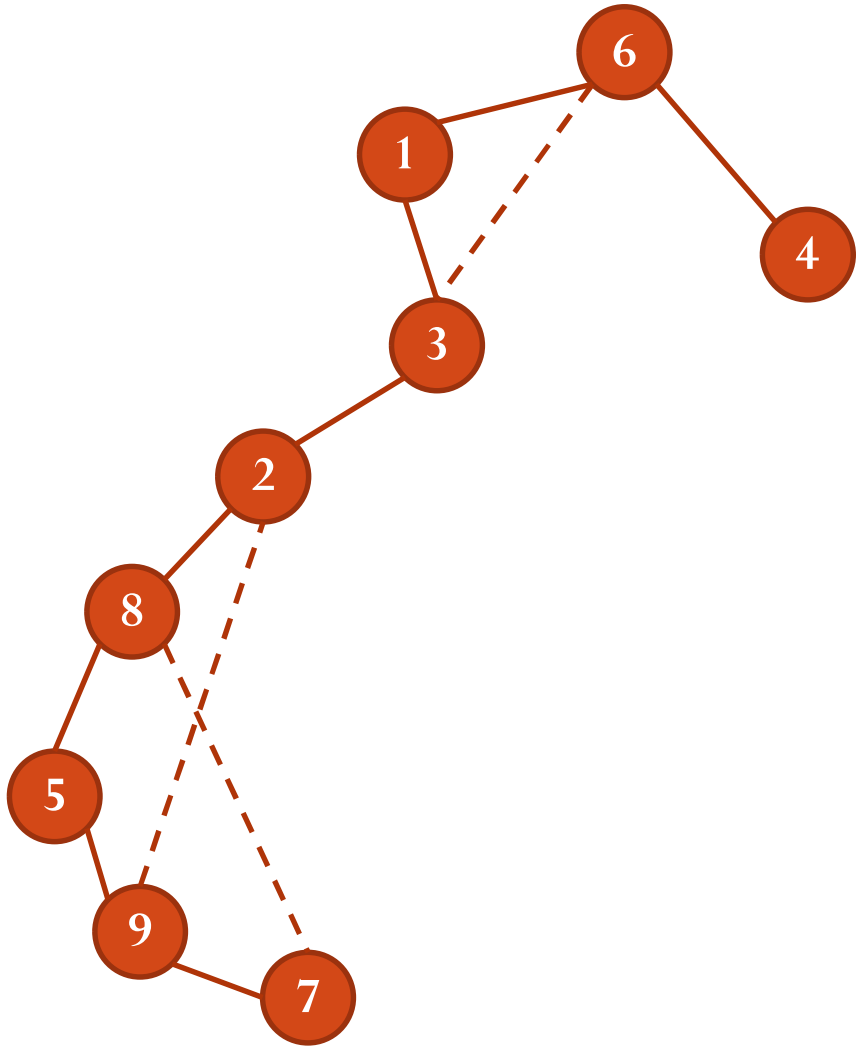
num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = num[6] = 1
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

DFS(6)



num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = low[3] = 1
num[3] = 3, low[3] = num[6] = 1
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

DFS(6)



num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = low[3] = 1
num[3] = 3, low[3] = num[6] = 1
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5
num[4] = 9, low[4] = 9

Sample code

```
#include <bits/stdc++.h>
using namespace std;
const int N = 10000;
int n,m;
vector<int> A[N];
bool visited[N];
int num[N];
int low[N];
int t;
vector<pair<int,int> > bridges;
void input(){
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n >> m;
    for(int i = 1; i <= m; i++){
        int u,v;
        cin >> u >> v;
        A[u].push_back(v);
        A[v].push_back(u);
    }
}
```

Sample code

```
void dfs(int s, int ps){
    // DFS from s with ps is the parent of s in the DFS tree
    t++;
    num[s] = t;
    low[s] = num[s];
    visited[s] = true;
    for(int i = 0;i < A[s].size(); i++){
        int v = A[s][i];
        if(v == ps) continue;
        if(visited[v]){
            low[s] = min(low[s],num[v]);
        }else{
            dfs(v,s);
            low[s] = min(low[s],low[v]);
            if(low[v] > num[s]){
                // discover a bridge (s,v)
                bridges.push_back(make_pair(s,v));
            }
        }
    }
}
```

Sample code

```
void init(){
    for(int v = 1; v <= n; v++) visited[v] = false;
}
void solve(){
    init();
    t = 0;
    for(int s = 1; s <= n; s++){
        if(!visited[s]){
            dfs(s,-1);
        }
    }
    cout << "bridges = ";
    for(int i = 0; i < bridges.size(); i++){
        cout << "(" << bridges[i].first << "," << bridges[i].second << ") ";
    }
}
int main(){
    input();
    solve();
}
```