Contents

1 語法

1.1 c++

```
// c++ code
#include <bits/stdc++.h>
lower_bound(a, a + n, k);     //最左邊 ≥ k 的位置
upper_bound(a, a + n, k);     //最左邊 > k 的位置
upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置
[lower_bound, upper_bound) //等於 k 的範圍
equal_range(a, a+n, k);

// 從小到大
priority_queue<int, vector<int>, greater<int>>pq

insert(it,x)//向vector的任意迭代器it處插入一個元素x
erase(it)//刪除迭代器爲it處的元素，erase(first,last)
//刪除一個區間[first,last)內的所有元素，時間複雜度均爲O(N)

set
insert(x) //將x插入set中   O(log(n))
count(x)  //回傳x是否存在於set中()  O(log(n))
erase(x)  //刪除在set中的x  O(log(n))
clear() //刪除set中所有元素 O(n)
empty() //回傳是否為空  O(1)
size()  //回傳共有幾個元素  O(1)

map
insert(x) //將x這個pair插入map中  O(log(n))
count(x)  //回傳x這個key是否在map中 O(log(n))
erase(x)  //刪除在map中key為x的 O(log(n))


#include <bits/stdc++.h>
using namespace std;

int main(){
  set<int>s;
  for(int i = 0; i < 10; i++){
    s.insert(i);
  }
  cout << "lower bound: " << *s.lower_bound(5) <<
      '\n';// 5
  cout << "upper bound: " << *s.upper_bound(5) <<
      '\n';// 6

  if(s.lower_bound(20) == s.end()){
    cout << "all elements are less than 20\n";
  }
}
```

1.2 python

```
sorted((4,1,9,6),reverse=True)
fruits = ['apple', 'watermelon', 'pear', 'banana']
a = sorted(fruits, key = lambda x : len(x))
print(a)
# 輸出：['pear', 'apple', 'banana', 'watermelon']
divmod(a,b)
把除數和餘數運算結果結合起來，
返回一個包含商和餘數的元組(a // b, a % b)

pow(base, exp[, mod])
>>> pow(38, -1, mod=97)
23
>>> 23 * 38 % 97 == 1
True

eof 寫法
try:
  while True:
    s = input()
except EOFError:
  pass

eval(expression, globals=None, locals=None)


list(map(int, input().split()))
 L.append(r)
my_list = ['This' , 'is' , 'a' , 'string' , 'in' ,
    'Python']
my_string = " ".join(my_list)
#This is a string in Python
test = [[0 for j in range(m)] for i in range(n)]
```

2 Graph

2.1 Bellman-Ford

```
#include<iostream>
using namespace std;
const int INF = 1e9;
const int MAXN = 1000;
const int MAXM = 1000;
struct Edge {
    int u;
    int v;
    int w;
};

int n, m;
Edge edges[MAXM];
int dis[MAXN];

// s是起點
bool bellman(int s) {
    for (int i = 0; i < n; i++) {
        dis[i] = INF;
    }
    dis[s] = 0;
    bool relax;
    // 做 n 輪
    for (int i = 0; i < n; i++) {
        relax = false;
        for (int j = 0; j < m; j++) {
            int u = edges[j].u;
            int v = edges[j].v;
            int w = edges[j].w;
            if (dis[u] == INF) {
                continue;
            }
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                relax = true;
            }
        }
        if (!relax) {
            break;
        }
    }
    return relax;
}


int main(){

}
```

2.2 Dijkstra

```
#include<bits/stdc++.h>
using namespace std;
#define M 100005
#define INF 1e9
struct Edge{
    int v, w;
    Edge(int a, int b):v(a), w(b){};
};
struct node{
    int u, dis;
    node(){};
    node(int a, int b):u(a), dis(b){};
    bool operator<(const node &r)const{
        return dis > r.dis;
    }
};
int dis[M]; //距離
vector<Edge> G[M];
void init(){
    fill(dis, dis+M, INF);
    for(int i = 0; i < M; i++){
        G[i].clear();
    }
}
void dijkstra(int start){
    dis[start] = 0;
    priority_queue<node> pq;
    pq.push(node(start, 0));
    while(!pq.empty()){
        node now = pq.top();
        pq.pop();
        if(now.dis > dis[now.u]) continue;
        for(Edge i : G[now.u]){
            if(dis[i.v] > now.dis + i.w){
                dis[i.v] = now.dis + i.w;
                pq.push(node(i.v, dis[i.v]));
                // printf("push(%d, %d)\n", i.v, dis[i.v]);
            }
        }
    }
}

int main(){
  int point, side;
    cin >> point >> side;
    init();
    for(int i = 0; i < side; i++){
        int s, t, w;
        cin >> s >> t >> w;
        G[s].push_back(Edge(t, w));
        G[t].push_back(Edge(s, w));
    }
    dijkstra(1);
    for(int i = 2; i <= point; i++){
        cout << dis[i] << '\n';
    }

}
```

2.3 Floyd-Warshall

```
#include<bits/stdc++.h>
using namespace std;
#define M 1005
#define INF 1e9

int dis[M][M];
// int G[M][M];
void init(int n){
    for(int i = 0; i <= n; i++){
        for(int j = 0; j <= n; j++){
            dis[i][j] = INF;
            if(i == j) dis[i][j] = 0;
        }
    }
}
void Floyd(int n){
    for(int k = 1; k <= n; k++){
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= i; j++){
                dis[i][j]= dis[j][i] = min(dis[i][k]+dis[k][j], dis[i][j]);
            }
        }
    }
}
void printarr(int r, int c){
    for(int i = 1; i <= r; i++){
        for(int j = 1; j <= c; j++){
            if(dis[i][j] == INF) cout << "INF ";
            else cout << dis[i][j] << ' ';
        }
        cout << '\n';
    }
}
int main(){
  int point, side;
    cin >> point >> side;
    init(point);
    for(int i = 0; i < side; i++){
        int s, t, w;
        cin >> s >> t >> w;
        dis[s][t] = w;
        dis[t][s] = w;
    }
    Floyd(point);
    int Cas;
    cin >> Cas;
    while(Cas--){
        int i, j;
        cin >> i >> j;
        cout << dis[i][j] << '\n';
    }
    // printarr(point, point);

}
```

2.4 SPFA

```
const int INF = 1e9;
const int MAXN = 1000;
struct Edge {
    int v;
    int w;
};
int n, m;
vector<Edge> G[MAXN];   //向量記圖
int dis[MAXN];
void SPFA(int s) {
    // 記錄目前的點是否在 queue 中
    bool inq[n];
    for (int i = 0; i < n; i++) {
        dis[i] = INF;
        inq[i] = false;
    }
    dis[s] = 0;
    inq[s] = true;
    queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (Edge e : G[u]) {
            if (dis[e.v] > dis[u] + e.w) {
                dis[e.v] = dis[u] + e.w;
                if (!inq[e.v]) {
                    inq[e.v] = true;
                    q.push(e.v);
                }
            }
        }
    }
}
```

2.5 SPFA

```
const int INF = 1e9;
const int MAXN = 1000;
struct Edge {
    int v;
    int w;
};
int n, m;
vector<Edge> G[MAXN];   //向量記圖
int dis[MAXN];
void SPFA(int s) {
    // 記錄目前的點是否在 queue 中
    bool inq[n];
    for (int i = 0; i < n; i++) {
        dis[i] = INF;
        inq[i] = false;
    }
    dis[s] = 0;
    inq[s] = true;
    queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (Edge e : G[u]) {
            if (dis[e.v] > dis[u] + e.w) {
                dis[e.v] = dis[u] + e.w;
                if (!inq[e.v]) {
                    inq[e.v] = true;
                    q.push(e.v);
                }
            }
        }
    }
}
```

3 Other

3.1 thm