Contents

1 Math

1.1 FindPrime

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

//查找[0,2^15]中的所有質數 共有3515

const int MAXN = 32768;  //2^15=32768
bool primes[MAXN];
vector<int> p;  //3515

//質數篩法Sieve of Eratosthenes
inline void findPrimes() {
    for (int i = 0; i < MAXN; i++) {
        primes[i] = true;
    }
    primes[0] = false;
    primes[1] = false;
    for (int i = 4; i < MAXN; i += 2) {
        //將2的倍數全部刪掉(偶數不會是質數)
        primes[i] = false;
    }
    //開始逐個檢查--->小心i*i會有overflow問題--->使用long long
    for (long long i = 3; i < MAXN; i += 2) {
        if (primes[i]) {

            //如果之前還未被刪掉 才做篩法
            for (long long j = i * i; j < MAXN; j += i) {
                //從i*i開始(因為i*2,i*3...都被前面處理完)
                primes[j] = false;
            }
        }
    }
    //搜集所有質數
    for (int i = 0; i < MAXN; i++) {
        if (primes[i]) {
            p.emplace_back(i);
        }
    }
}
```

2 Graph

2.1 Kruskal

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

//節點從0號開始
struct Edge {
    int v, w, wt;
    Edge(int a, int b, int c) {
        v = a;
        w = b;
        wt = c;
    }
    bool operator<(const Edge &e) const {
        return wt < e.wt;
    }
};

const int maxN = 100000 + 5;  // 最多maxN個節點
int V, E;                     //有V個節點E條鞭
int parent[maxN];
vector<Edge> edges;

int do_find(int p) {
    while (parent[p] >= 0) {
        p = parent[p];
    }
    return p;
}

void do_union(int p, int q) {
    if (parent[p] > parent[q]) {
        parent[q] += parent[p];
        parent[p] = q;
    } else {
        parent[p] += parent[q];
        parent[q] = p;
    }
}

void init() {
    edges.clear();
    for (int i = 0; i < V; i++) {
        parent[i] = -1;
    }
}

int kruskal() {
    sort(edges.begin(), edges.end());
    int mstWeight = 0;
    int pRoot, qRoot;
    for (auto e : edges) {
        pRoot = do_find(e.v);
        qRoot = do_find(e.w);
        if (pRoot != qRoot) {
            mstWeight += e.wt;
            do_union(pRoot, qRoot);
        }
    }
    return mstWeight;
}

int main() {
    int ta, tb, tc;
    while (~scanf("%d %d", &V, &E)) {
        init();
        for (int i = 0; i < E; i++) {
            scanf("%d %d %d", &ta, &tb, &tc);
            edges.push_back({ta, tb, tc});
        }
        printf("%d\n", kruskal());
    }
    return 0;
}
```

2.2 Dijkstra

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

//節點從1號開始
struct Edge {
    int v, wt;
    Edge(int a, int c) {
        v = a;
        wt = c;
```

```
    }
    Edge() {}
};
struct Info {
    int v;
    int wt;
    Info(int a, int b) : v(a), wt(b) {}
    Info() {}

    bool operator<(const Info &i) const {
        return wt > i.wt;
    }
};

const int maxN = 100000 + 5;  // 最多maxN個節點
int V, E;                     //有V個節點E條邊
vector<Edge> g[maxN];
vector<bool> visied(maxN);
vector<int> dis(maxN);
priority_queue<Info> pq;

void init() {
    for (int i = 0; i < V; i++) {
        g[i].clear();
        visied[i] = false;
        dis[i] = 0x3f3f3f;
    }
    while (!pq.empty()) {
        pq.pop();
    }
}

void dijkstra(int s) {
    Info info;
    dis[s] = 0;
    visied[s] = true;
    pq.push({s, 0});

    while (!pq.empty()) {
        info = pq.top();
        pq.pop();
        visied[info.v] = true;
        if (dis[info.v] > info.wt) {
            dis[info.v] = info.wt;
        }
        for (auto e : g[info.v]) {
            if (!visied[e.v]) {
                pq.push({e.v, dis[info.v] + e.wt});
            }
        }
    }
}

int main() {
    int ta, tb, tc;
    while (~scanf("%d %d", &V, &E)) {
        init();
        while (E--) {
            scanf("%d %d %d", &ta, &tb, &tc);
            g[ta].push_back({tb, tc});
            g[tb].push_back({ta, tc});
        }
        dijkstra(1);  //從1號節點開始
    }
    return 0;
}
```

2.3 BellmanFord

```
#include <bits/stdc++.h>
using namespace std;
//節點從0開始(適用於無向圖)
//備註:如果圖為無向圖且包含負權邊 則必定有負權環
struct Edge {
    int f, t, wt;
    Edge() {}
    Edge(int a, int b, int c) {
        f = a;
        t = b;
        wt = c;
    }
};

int V, E;                     //節點與邊數
const int maxN = 100000;  //最多maxN個節點
vector<vector<Edge>> G(maxN);
vector<int> distTo(maxN);  //到節點i的權重
bool hasNegtiveCycle;
Edge e;

void init() {
    for (int i = 0; i < V; i++) {
        G[i].clear();
        distTo[i] = 0x3f3f3f;
    }
}

bool detectHasCycle() {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < G[i].size(); j++) {
            e = G[i][j];
            if (distTo[e.f] + e.wt < distTo[e.t]) {
                return true;
            }
        }
    }
    return false;
}

void bellmanFord(int s) {  //從s點開始
    distTo[s] = 0;
    //執行節點-1次鬆弛
    for (int pass = 1; pass < V; pass++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < G[i].size(); j++) {
                e = G[i][j];
                if (distTo[e.f] + e.wt < distTo[e.t])
                    {
                    distTo[e.t] = distTo[e.f] + e.wt;
                }
            }
        }
    }
    //檢測負權環
    hasNegtiveCycle = detectHasCycle();
}

int main() {
    scanf("%d %d", &V, &E);
    init();
    for (int i = 0; i < E; i++) {
        scanf("%d %d %d", &e.f, &e.t, &e.wt);
        G[e.f].push_back(e);
    }
    bellmanFord(0);  //從節點0開始
    if (!hasNegtiveCycle) {
        for (int i = 0; i < V; i++) {
            printf("%d ", distTo[i]);
        }
        printf("\n");
    } else {
        printf("Has Negtive Cycle.");
    }
    return 0;
}
```

3 DataStructure

3.1 BitIndexTree

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

// bit陣列索引從1開始
const int maxN = 100000 + 5;  // bit容量
const int dataSize = 5;       //資料大小
    arr[0,5)--->bit[1,5]
int bit[maxN];

int query(int x) {
    // query prefix sum in BIT
    int ret = 0;
    while (x) {
        ret += bit[x];
        x -= x & (-x);
    }
    return ret;
}

//更新bit[x]的值
void update(int x, int d) {
    while (x <= dataSize) {
        bit[x] += d;
        x += x & (-x);
    }
}

// 區間和 [l,r]
int rSum(int l, int r) {
    return query(r) - query(l - 1);
}

int main() {
    memset(bit, 0, sizeof(bit));
    int arr[dataSize] = {1, 2, 3, 4, 5};
    for (int i = 0; i < dataSize; i++) {
        update(i + 1, arr[i]);  //
            arr[i]放bit[i+1]的位置
    }
    printf("%d\n", rSum(2, 4));  // arr[2,4]=2+3+4
    return 0;
}
```