

Contents

1	Math	1
1.1	FindPrime	1
2	Graph	1
2.1	Kruskal	1
2.2	Dijkstra	1
2.3	BellmanFord	2
3	DataSeture	2
3.1	BitIndexTree	2

1 Math

1.1 FindPrime

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 //查找[0,2^15]中的所有質數 共有3515
5
6 const int MAXN = 32768; //2^15=32768
7 bool primes[MAXN];
8 vector<int> p; //3515
9
10 //質數篩法 Sieve of Eratosthenes
11 inline void findPrimes() {
12     for (int i = 0; i < MAXN; i++) {
13         primes[i] = true;
14     }
15     primes[0] = false;
16     primes[1] = false;
17     for (int i = 4; i < MAXN; i += 2) {
18         //將2的倍數全部刪掉(偶數不會是質數)
19         primes[i] = false;
20     }
21     //開始逐個檢查--->小心i*i會有overflow問題--->使用long
22     for (long long i = 3; i < MAXN; i += 2) {
23         if (primes[i]) {
24             //如果之前還未被刪掉 才做篩法
25             for (long long j = i * i; j < MAXN; j += i) {
26                 //從i*i開始(因為i*2,i*3...都被前面處理完)
27                 primes[j] = false;
28             }
29         }
30     }
31     //搜集所有質數
32     for (int i = 0; i < MAXN; i++) {
33         if (primes[i]) {
34             p.emplace_back(i);
35         }
36     }
37 }

```

2 Graph

2.1 Kruskal

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //Kruskal (MST) 節點從0號開始
4 struct Edge {
5     int v, w, wt;
6     Edge(int a, int b, int c) {
7         v = a;
8         w = b;
9         wt = c;

```

```

10     }
11     bool operator<(const Edge &e) const {
12         return wt < e.wt;
13     }
14 };
15
16 const int maxN = 100000 + 5; //maxN個節點
17 int parent[maxN];
18 vector<Edge> edges;
19
20 int do_find(int p) {
21     while (parent[p] >= 0) {
22         p = parent[p];
23     }
24     return p;
25 }
26
27 void do_union(int p, int q) {
28     if (parent[p] > parent[q]) {
29         parent[q] += parent[p];
30         parent[p] = q;
31     } else {
32         parent[p] += parent[q];
33         parent[q] = p;
34     }
35 }
36
37 int m, n, ta, tb, tc, weight;
38
39 int main() {
40     while (~scanf("%d %d", &m, &n)) {
41         for (int i = 0; i < n; i++) {
42             scanf("%d %d %d", &ta, &tb, &tc);
43             edges.push_back({ta, tb, tc});
44         }
45         sort(edges.begin(), edges.end());
46         for (int i = 0; i <= m; i++) {
47             parent[i] = -1;
48         }
49         weight = 0;
50         for (auto e : edges) {
51             ta = do_find(e.v);
52             tb = do_find(e.w);
53             if (ta != tb) {
54                 weight += e.wt;
55                 do_union(ta, tb);
56             }
57         }
58         printf("%d\n", weight);
59     }
60     return 0;
61 }

```

2.2 Dijkstra

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 //節點從1號開始
5 const int maxN = 100000 + 5; //maxN個節點
6 struct Edge {
7     int v, wt;
8     Edge(int a, int c) {
9         v = a;
10        wt = c;
11    }
12    Edge() {}
13 };
14
15 vector<Edge> g[maxN];
16 vector<bool> visied(maxN);
17 vector<int> dis(maxN);
18
19 struct Info {
20     int v;

```

```

21     int wt;
22     Info(int a, int b) : v(a), wt(b) {}
23     Info() {}
24
25     bool operator<(const Info &i) const {
26         return wt > i.wt;
27     }
28 };
29
30 priority_queue<Info> pq;
31
32 void init() {
33     for (int i = 0; i < maxN; i++) {
34         g[i].clear();
35         visied[i] = false;
36         dis[i] = 0x3f3f3f;
37     }
38     while (!pq.empty()) {
39         pq.pop();
40     }
41 }
42
43 void dijkstra(int s) {
44     Info info;
45     dis[s] = 0;
46     visied[s] = true;
47     pq.push({s, 0});
48
49     while (!pq.empty()) {
50         info = pq.top();
51         pq.pop();
52         visied[info.v] = true;
53         if (dis[info.v] > info.wt) {
54             dis[info.v] = info.wt;
55         }
56         for (auto e : g[info.v]) {
57             if (!visied[e.v]) {
58                 pq.push({e.v, dis[info.v] + e.wt});
59             }
60         }
61     }
62 }
63
64 int m, n, ta, tb, tc;
65 int main() {
66     ios::sync_with_stdio(0);
67     cin.tie(0);
68     while (cin >> m >> n) {
69         init();
70         while (n-- > 0) {
71             cin >> ta >> tb >> tc;
72             g[ta].push_back({tb, tc});
73             g[tb].push_back({ta, tc});
74         }
75         dijkstra(1); //從1號節點開始
76     }
77     return 0;
78 }

```

2.3 BellmanFord

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //節點從0開始(適用於無向圖)
4 //備註:如果圖為無向圖且包含負權邊 則必定有負權環
5 struct Edge {
6     int f, t, wt;
7     Edge() {}
8     Edge(int a, int b, int c) {
9         f = a;
10        t = b;
11        wt = c;
12    }
13 };
14

```

```

15 int V, E; //節點與邊數
16 const int maxN = 100000; //最多maxN個節點
17 vector<vector<Edge>> G(maxN);
18 vector<int> distTo(maxN); //到節點i的權重
19 bool hasNegativeCycle;
20 Edge e;
21
22 void init() {
23     for (int i = 0; i < V; i++) {
24         G[i].clear();
25         distTo[i] = 0x3f3f3f;
26     }
27 }
28
29 bool detectHasCycle() {
30     for (int i = 0; i < V; i++) {
31         for (int j = 0; j < G[i].size(); j++) {
32             e = G[i][j];
33             if (distTo[e.f] + e.wt < distTo[e.t]) {
34                 return true;
35             }
36         }
37     }
38     return false;
39 }
40
41 void bellmanFord(int s) { //從s點開始
42     distTo[s] = 0;
43     //執行節點-1次鬆弛
44     for (int pass = 1; pass < V; pass++) {
45         for (int i = 0; i < V; i++) {
46             for (int j = 0; j < G[i].size(); j++) {
47                 e = G[i][j];
48                 if (distTo[e.f] + e.wt < distTo[e.t]) {
49                     distTo[e.t] = distTo[e.f] + e.wt;
50                 }
51             }
52         }
53     }
54     //檢測負權環
55     hasNegativeCycle = detectHasCycle();
56 }
57
58 int main() {
59     scanf("%d %d", &V, &E);
60     init();
61     for (int i = 0; i < E; i++) {
62         scanf("%d %d %d", &e.f, &e.t, &e.wt);
63         G[e.f].push_back(e);
64     }
65     bellmanFord(0); //從節點0開始
66     if (!hasNegativeCycle) {
67         for (int i = 0; i < V; i++) {
68             printf("%d ", distTo[i]);
69         }
70         printf("\n");
71     } else {
72         printf("Has Negative Cycle.");
73     }
74     return 0;
75 }

```

3 DataStructure

3.1 BitIndexTree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // bit陣列索引從1開始
5 const int maxN = 100000 + 5; // bit容量

```

```

6 | const int dataSize = 5;           //資料大小
   |     arr[0,5)---->bit[1,5]
7 | int bit[maxN];
8 |
9 | int query(int x) {
10 |     // query prefix sum in BIT
11 |     int ret = 0;
12 |     while (x) {
13 |         ret += bit[x];
14 |         x -= x & (-x);
15 |     }
16 |     return ret;
17 | }
18 |
19 | //更新bit[x]的值
20 | void update(int x, int d) {
21 |     while (x <= dataSize) {
22 |         bit[x] += d;
23 |         x += x & (-x);
24 |     }
25 | }
26 |
27 | // 區間和 [l,r]
28 | int rSum(int l, int r) {
29 |     return query(r) - query(l - 1);
30 | }
31 |
32 | int main() {
33 |     memset(bit, 0, sizeof(bit));
34 |     int arr[dataSize] = {1, 2, 3, 4, 5};
35 |     for (int i = 0; i < dataSize; i++) {
36 |         update(i + 1, arr[i]); //
   |             arr[i]放bit[i+1]的位置
37 |     }
38 |     printf("%d\n", rSum(2, 4)); // arr[2,4]=2+3+4
39 |     return 0;
40 | }

```