

Contents

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

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     //long
23     for (long long i = 3; i < MAXN; i += 2) {
24         if (primes[i]) {
25             //如果之前還未被刪掉 才做篩法
26             for (long long j = i * i; j < MAXN; j += i) {
27                 //從i*i開始(因為i*2,i*3...都被前面處理完了)
28                 primes[j] = false;
29             }
30         }
31     }
32     //搜集所有質數
33     for (int i = 0; i < MAXN; i++) {
34         if (primes[i]) {
35             p.emplace_back(i);
36         }
37     }
38 }

```

2 Graph

2.1 Kruskal

```

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

```

```

10        wt = c;
11    }
12    bool operator<(const Edge &e) const {
13        return wt < e.wt;
14    }
15 };
16
17 const int maxN = 100000 + 5; // 最多maxN個節點
18 int V, E; // 有V個節點E條邊
19 int parent[maxN];
20 vector<Edge> edges;
21
22 int do_find(int p) {
23     while (parent[p] >= 0) {
24         p = parent[p];
25     }
26     return p;
27 }
28
29 void do_union(int p, int q) {
30     if (parent[p] > parent[q]) {
31         parent[q] += parent[p];
32         parent[p] = q;
33     } else {
34         parent[p] += parent[q];
35         parent[q] = p;
36     }
37 }
38
39 void init() {
40     edges.clear();
41     for (int i = 0; i < V; i++) {
42         parent[i] = -1;
43     }
44 }
45
46 int kruskal() {
47     sort(edges.begin(), edges.end());
48     int mstWeight = 0;
49     int pRoot, qRoot;
50     for (auto e : edges) {
51         pRoot = do_find(e.v);
52         qRoot = do_find(e.w);
53         if (pRoot != qRoot) {
54             mstWeight += e.wt;
55             do_union(pRoot, qRoot);
56         }
57     }
58     return mstWeight;
59 }
60
61 int main() {
62     int ta, tb, tc;
63     while (~scanf("%d %d", &V, &E)) {
64         init();
65         for (int i = 0; i < E; i++) {
66             scanf("%d %d %d", &ta, &tb, &tc);
67             edges.push_back({ta, tb, tc});
68         }
69         printf("%d\n", kruskal());
70     }
71     return 0;
72 }

```

2.2 Dijkstra

```

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

```

```

10     }
11     Edge() {}
12 };
13 struct Info {
14     int v;
15     int wt;
16     Info(int a, int b) : v(a), wt(b) {}
17     Info() {}
18
19     bool operator<(const Info &i) const {
20         return wt > i.wt;
21     }
22 };
23
24 const int maxN = 100000 + 5; // 最多maxN個節點
25 int V, E; // 有V個節點E條邊
26 vector<Edge> g[maxN];
27 vector<bool> visied(maxN);
28 vector<int> dis(maxN);
29 priority_queue<Info> pq;
30
31 void init() {
32     for (int i = 0; i < V; i++) {
33         g[i].clear();
34         visied[i] = false;
35         dis[i] = 0x3f3f3f;
36     }
37     while (!pq.empty()) {
38         pq.pop();
39     }
40 }
41
42 void dijkstra(int s) {
43     Info info;
44     dis[s] = 0;
45     visied[s] = true;
46     pq.push({s, 0});
47
48     while (!pq.empty()) {
49         info = pq.top();
50         pq.pop();
51         visied[info.v] = true;
52         if (dis[info.v] > info.wt) {
53             dis[info.v] = info.wt;
54         }
55         for (auto e : g[info.v]) {
56             if (!visied[e.v]) {
57                 pq.push({e.v, dis[info.v] + e.wt});
58             }
59         }
60     }
61 }
62
63 int main() {
64     int ta, tb, tc;
65     while (~scanf("%d %d", &V, &E)) {
66         init();
67         while (E--) {
68             scanf("%d %d %d", &ta, &tb, &tc);
69             g[ta].push_back({tb, tc});
70             g[tb].push_back({ta, tc});
71         }
72         dijkstra(1); // 從1號節點開始
73     }
74     return 0;
75 }

```

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
7   arr[0,5)---->bit[1,5]
8 int bit[maxN];
9
10 int query(int x) {
11     // query prefix sum in BIT
12     int ret = 0;
13     while (x) {
14         ret += bit[x];
15         x -= x & (-x);
16     }
17     return ret;
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]); //
37         // arr[i]放bit[i+1]的位置
38     }
39     printf("%d\n", rSum(2, 4)); // arr[2,4]=2+3+4
40     return 0;
41 }

```