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;
// Kruskal (MST) 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;
```

```cpp
    }
    bool operator<(const Edge &e) const {
        return wt < e.wt;
    }
};

const int maxN = 100000 + 5;  // maxN個節點
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;
    }
}

int m, n, ta, tb, tc, weight;

int main() {
    while (~scanf("%d %d", &m, &n)) {
        for (int i = 0; i < n; i++) {
            scanf("%d %d %d", &ta, &tb, &tc);
            edges.push_back({ta, tb, tc});
        }
        sort(edges.begin(), edges.end());
        for (int i = 0; i <= m; i++) {
            parent[i] = -1;
        }
        weight = 0;
        for (auto e : edges) {
            ta = do_find(e.v);
            tb = do_find(e.w);
            if (ta != tb) {
                weight += e.wt;
                do_union(ta, tb);
            }
        }
        printf("%d\n", weight);
    }
    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;
```

```cpp
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

```cpp
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);
```

```cpp
18 vector<int> distTo(maxN);   //到節點i的權重
19 bool hasNegtiveCycle;
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     hasNegtiveCycle = 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 (!hasNegtiveCycle) {
67         for (int i = 0; i < V; i++) {
68             printf("%d ", distTo[i]);
69         }
70         printf("\n");
71     } else {
72         printf("Has Negtive Cycle.");
73     }
74     return 0;
75 }
```

3 DataStructure

3.1 BitIndexTree

```cpp
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  }
```