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...都被前面處理完了)
                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;
```

```cpp
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

```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;
```

```cpp
7      Edge() {}
8      Edge(int a, int b, int c) {
9          f = a;
10         t = b;
11         wt = c;
12     }
13 };
14
15 const int maxN = 100000;    //最多 maxN 個節點
16 int V, E;                   //有 V 個節點 E 條邊
17 vector<vector<Edge>> G(maxN);
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      while (~scanf("%d %d", &V, &E)) {
60          init();
61          for (int i = 0; i < E; i++) {    //無向圖
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      }
75      return 0;
76 }
```

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最大容量
int bit[maxN];
//放入的資料量
int dataSize;

//更新bit[x]的值(加d)
void update(int x, int d) {
    while (x <= dataSize) {
        bit[x] += d;
        x += x & (-x);
    }
}
//前綴和[1,x]
int query(int x) {
    int ret = 0;
    while (x) {
        ret += bit[x];
        x -= x & (-x);
    }
    return ret;
}

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

int main() {
    memset(bit, 0, sizeof(bit));
    vector<int> v = {0x3f3f3f, 1, 2, 3, 4, 5};
    dataSize = v.size() - 1;                 //
        v[0]不使用
    for (int i = 1; i < v.size(); i++) {   //放入v[1,5]
        update(i, v[i]);
    }
    printf("%d\n", rSum(2, 4));   // v[2,4]=2+3+4=9
    return 0;
}
```