Contents

1 Graph Theory

1.1 Adjacency List

```cpp
vector<int> list[5];

void Adjacency_List(){

    // initial
    for (int i = 0; i < 5; i++)
        list[i].clear();

    int a, b;    // start & end of an edge

    while (cin >> a >> b)
        list[a].push_back(b);
        // list[b].push_back(a);
}
```

1.2 DFS

```cpp
vector<int> G[N];
bitset<N> vis;
void dfs(int s) {
    vis[s] = 1;
    for (int t : G[s]) {
        if (!vis[i])
            dfs(i);
    }
}
```

1.3 BFS

```cpp
vector<int> G[N];
bitset<N> vis;
void bfs(int s) {
    queue<int> q;
    q.push(s);
    vis[s] = 1;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int t : G[v]) {
            if (!vis[t]) {
                q.push(t);
                vis[t] = 1;
            }
        }
    }
}
```

1.4 Disjoint Set and Kruskal

```cpp
struct Edge{
    int u, v, w;
    // bool operator < (const Edge &rhs) const {
        return w < rhs.w; }
};

vector<int> parent;
vector<Edge> E;

bool cmp(Edge edge1, Edge edge2){
    return edge2.w > edge1.w;
}

int find(int x){
    if(parent[x] < 0){
        return x;
    }
    return parent[x] = find(parent[x]);
}

bool Uni(int a, int b){
    a = find(a);
    b = find(b);
    if(a == b){
        return false;
    }
    if(parent[a] > parent[b]){
        swap(a, b);
    }
    parent[a] = parent[a] + parent[b];
    parent[b] = a;
    return true;
}

void Kruskal() {

    int cost = 0;

    sort(E.begin(), E.end()); // sort by w
    // sort(E.begin(), E.end(), cmp);

    // two edge in the same tree or not
    for (auto it: E){
        it.s = Find(it.s);
        it.t = Find(it.t);
        if (Uni(it.s, it.t)){
            cost = cost + it.w;;
        }
    }
}

int main(){

    // create N space and initial -1
    parent = vector<int> (N, -1);

    for(i = 0; i < M; i++){
        cin >> u >> v >> w;
        E.push_back({u, v, w});
    }

    Kruskal();

    return 0;
}
```

1.5 Floyd-Warshall

```cpp
for (k = 0; k < n; k++){
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            w[i][j] = w[j][i] = min(w[i][j],
                max(w[i][k], w[k][j]));
```

```
5            }
6        }
7 }
```

1.6 Dijkstra

```cpp
1  struct edge {
2    int s, t;
3    LL d;
4    edge(){};
5    edge(int s, int t, LL d) : s(s), t(t), d(d) {}
6  };
7
8  struct heap {
9    LL d;
10   int p; // point
11   heap(){};
12   heap(LL d, int p) : d(d), p(p) {}
13   bool operator<(const heap &b) const { return d >
         b.d; }
14 };
15
16 int d[N], p[N];
17 vector<edge> edges;
18 vector<int> G[N];
19 bitset<N> vis;
20
21 void Dijkstra(int ss){
22     priority_queue<heap> Q;
23     for (int i = 0; i < V; i++){
24         d[i] = INF;
25     }
26     d[ss] = 0;
27     p[ss] = -1;
28     vis.reset() : Q.push(heap(0, ss));
29     heap x;
30     while (!Q.empty()){
31         x = Q.top();
32         Q.pop();
33         int p = x.p;
34         if (vis[p])
35             continue;
36         vis[p] = 1;
37         for (int i = 0; i < G[p].size(); i++){
38             edge &e = edges[G[p][i]];
39             if (d[e.t] > d[p] + e.d){
40                 d[e.t] = d[p] + e.d;
41                 p[e.t] = G[p][i];
42                 Q.push(heap(d[e.t], e.t));
43             }
44         }
45     }
46 }
```

2 Number Theory

2.1 Modulo

- $(a + b) \bmod p = (a \bmod p + b \bmod p) \bmod p$

- $(a - b) \bmod p = (a \bmod p - b \bmod p + p) \bmod p$

- $(a * b) \bmod p = (a \bmod p \cdot b \bmod p) \bmod p$

- $(a^b) \bmod p = ((a \bmod p)^b) \bmod p$

- $((a + b) \bmod p + c) \bmod p = (a + (b + c)) \bmod p$

- $((a \cdot b) \bmod p \cdot c) \bmod p = (a \cdot (b \cdot c)) \bmod p$

- $(a + b) \bmod p = (b + a) \bmod p$

- $(a \cdot b) \bmod p = (b \cdot a) \bmod p$

- $((a + b) \bmod p \cdot c) = ((a \cdot c) \bmod p + (b \cdot c) \bmod p) \bmod p$

- $a \equiv b \pmod{m} \Rightarrow c \cdot m = a - b, c \in \mathbb{Z}$
  $\Rightarrow a \equiv b \pmod{m} \Rightarrow m \mid a - b$

- $a \equiv b \pmod{c}, b \equiv d \pmod{c}$
  則 $a \equiv d \pmod{c}$

- $\begin{cases} a \equiv b \pmod{m} \\ c \equiv d \pmod{m} \end{cases} \Rightarrow \begin{cases} a \pm c \equiv b \pm d \pmod{m} \\ a \cdot c \equiv b \cdot d \pmod{m} \end{cases}$

2.2 Linear Sieve

```cpp
1  vector<int> p;
2  bitset<MAXN> is_notp;
3  void PrimeTable(int n) {
4      is_notp.reset();
5      is_notp[0] = is_notp[1] = 1;
6      for (int i = 2; i <= n; ++i) {
7          if (!is_notp[i]){
8              p.push_back(i);
9          }
10         for (int j = 0; j < (int)p.size(); ++j) {
11             if (i * p[j] > n){
12                 break;
13             }
14             is_notp[i * p[j]] = 1;
15             if (i % p[j] == 0){
16                 break;
17             }
18         }
19     }
20 }
```

2.3 Prime Factorization

```cpp
1  void primeFactorization(int n){
2      for(int i = 0; i < (int)p.size(); i++){
3          if(p[i] * p[i] > n){
4              break;
5          }
6          if(n % p[i]){
7              continue;
8          }
9          cout << p[i] << ' ';
10         while(n % p[i] == 0){
11             n /= p[i];
12         }
13     }
14     if(n != 1){
15         cout << n << ' ';
16     }
17     cout << '\n';
18 }
```

2.4 Exponentiating by Squaring

```cpp
1  T pow(int a, int b, int c){ // calculate a ^ b % c
2      T ans = 1, tmp = a;
3      for (; b; b >>= 1) {
4          if (b & 1){ // b is odd
5              ans = ans * tmp % c;
6          }
7          tmp = tmp * tmp % c;
8      }
9      return ans;
10 }
```

2.5 Euler

```
int Phi(int n){
    int ans = n;
    for (int i: p) {
        if (i * i > n){
            break;
        }
        if (n % i == 0){
            ans /= i;
            ans *= i - 1;
            while (n % i == 0){
                n /= i;
            }
        }
    }
    if (n != 1) {
        ans /= n;
        ans *= n - 1;
    }
    return ans;
}
```