Contents

1 Graph Theory

1.1 Adjacency List

```
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

```
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

```
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

```
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

```
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]));
```

1.6 Dijkstra

```
struct edge {
  int s, t;
  LL d;
  edge(){};
  edge(int s, int t, LL d) : s(s), t(t), d(d) {}
};

struct heap {
  LL d;
  int p; // point
  heap(){};
  heap(LL d, int p) : d(d), p(p) {}
  bool operator<(const heap &b) const { return d >
      b.d; }
};

int d[N], p[N];
vector<edge> edges;
vector<int> G[N];
bitset<N> vis;

void Dijkstra(int ss){

    priority_queue<heap> Q;

    for (int i = 0; i < V; i++){
        d[i] = INF;
    }

    d[ss] = 0;
    p[ss] = -1;
    vis.reset() : Q.push(heap(0, ss));
    heap x;

    while (!Q.empty()){

        x = Q.top();
        Q.pop();
        int p = x.p;

        if (vis[p])
            continue;
        vis[p] = 1;

        for (int i = 0; i < G[p].size(); i++){
            edge &e = edges[G[p][i]];
            if (d[e.t] > d[p] + e.d){
                d[e.t] = d[p] + e.d;
                p[e.t] = G[p][i];
                Q.push(heap(d[e.t], e.t));
            }
        }
    }
}
```

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

2.2 Linear Sieve

```
vector<int> p;
bitset<MAXN> is_notp;
void PrimeTable(int n){

    is_notp.reset();
    is_notp[0] = is_notp[1] = 1;

    for (int i = 2; i <= n; ++i){
        if (!is_notp[i]){
            p.push_back(i);
        }
        for (int j = 0; j < (int)p.size(); ++j){
            if (i * p[j] > n){
                break;
            }

            is_notp[i * p[j]] = 1;

            if (i % p[j] == 0){
                break;
            }
        }
    }
}
```

2.3 Prime Factorization

```
void primeFactorization(int n){
    for(int i = 0; i < (int)p.size(); i++){
        if(p[i] * p[i] > n){
            break;
        }
        if(n % p[i]){
            continue;
        }
        cout << p[i] << ' ';
        while(n % p[i] == 0){
            n /= p[i];
        }
    }
    if(n != 1){
        cout << n << ' ';
    }
    cout << '\n';
}
```

2.4 Exponentiating by Squaring

```
T pow(int a, int b, int c){ // calculate a ^ b % c
    T ans = 1, tmp = a;
    for (; b; b >>= 1) {
        if (b & 1){ // b is odd
            ans = ans * tmp % c;
        }
        tmp = tmp * tmp % c;
    }
  return ans;
}
```

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;
}
```