1

Contents

```
1 Graph Theory
2 Number Theory
2.1 Modulo .
2.4 Exponentiating by Squaring . . . . . . . . . . . . . . . . .
```

Graph Theory

Adjacency List

```
1 vector<int> list[5];
  void Adjacency_List(){
      // initial
      for (int i = 0; i < 5; i++)
7
          list[i].clear();
8
9
      int a, b; // start & end of an edge
10
11
      while (cin >> a >> b)
          list[a].push_back(b);
12
13
          // list[b].push_back(a);
14 }
```

1.2 DFS

```
1 vector<int> G[N];
 bitset<N> vis;
 void dfs(int s) {
      vis[s] = 1;
      for (int t : G[s]) {
6
          if (!vis[i])
7
              dfs(i);
8
```

1.3 BFS

```
1 vector<int> G[N];
2 bitset<N> vis;
3 void bfs(int s) {
       queue<int> q;
       q.push(s);
5
       vis[s] = 1;
6
       while (!q.empty()) {
7
           int v = q.front();
9
           q.pop();
10
           for (int t : G[v]) {
11
                if (!vis[t]) {
                    q.push(t);
12
                    vis[t] = 1;
13
               }
14
15
           }
16
       }
17 }
```

1.4 Disjoint Set and Kruskal

```
1 struct Edge{
  2
         int u, v, w;
  3
         // bool operator < (const Edge &rhs) const {
              return w < rhs.w; }</pre>
  4 };
    vector<int> parent;
    vector<Edge> E;
    bool cmp(Edge edge1, Edge edge2){
         return edge2.w > edge1.w;
  11
3 13
    int find(int x){
         if(parent[x] < 0){
  14
  15
             return x;
  16
  17
         return parent[x] = find(parent[x]);
  18 }
  19
  20
    bool Uni(int a, int b){
  21
         a = find(a);
  22
         b = find(b);
  23
         if(a == b){
  24
             return false;
  25
  26
         if(parent[a] > parent[b]){
  27
             swap(a, b);
  28
         parent[a] = parent[a] + parent[b];
  29
  30
         parent[b] = a;
  31
         return true;
  32 }
  33
    void Kruskal() {
  34
  35
  36
         int cost = 0;
  37
         sort(E.begin(), E.end()); // sort by w
  38
         // sort(E.begin(), E.end(), cmp);
  39
  40
  41
         // two edge in the same tree or not
         for (auto it: E){
  42
             it.s = Find(it.s);
  43
             it.t = Find(it.t);
  45
             if (Uni(it.s, it.t)){
  46
                  cost = cost + it.w;;
  47
  48
         }
  49
    }
  50
  51
    int main(){
  52
  53
         // create N space and initial -1
         parent = vector<int> (N, -1);
  54
  55
  56
         for(i = 0; i < M; i++){
             cin >> u >> v >> w;
  57
             E.push_back({u, v, w});
  58
  59
  60
  61
         Kruskal();
  62
  63
         return 0;
  64 }
```

1.5 Floyd-Warshall

```
1 for (k = 0; k < n; k++)
      for (i = 0; i < n; i++){
2
          for (j = 0; j < n; j++){
3
              w[i][j] = w[j][i] = min(w[i][j],
                  max(w[i][k], w[k][j]));
```

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

1.6 Dijkstra

```
1 struct edge {
2
     int s, t;
     LL d;
3
     edge(){};
5
     edge(int s, int t, LL d) : s(s), t(t), d(d) {}
6 };
7
8 struct heap {
9
    LL d;
     int p; // point
10
     heap(){};
11
12
     heap(LL d, int p) : d(d), p(p) {}
     bool operator<(const heap &b) const { return d >
13
         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
24
25
       for (int i = 0; i < V; i++){
            d[i] = INF;
26
27
28
29
       d[ss] = 0;
30
       p[ss] = -1;
       vis.reset() : Q.push(heap(0, ss));
31
32
       heap x;
33
       while (!Q.empty()){
34
35
36
            x = Q.top();
37
            Q.pop();
            int p = x.p;
38
39
            if (vis[p])
40
41
                continue;
42
            vis[p] = 1;
43
44
            for (int i = 0; i < G[p].size(); i++){</pre>
45
                edge &e = edges[G[p][i]];
46
                if (d[e.t] > d[p] + e.d){
                     d[e.t] = d[p] + e.d;
47
                     p[e.t] = G[p][i];
48
49
                     Q.push(heap(d[e.t], e.t));
50
                }
51
            }
       }
52
53 }
```

2 Number Theory

2.1 Modulo

```
 \cdot \quad (a+b) \operatorname{mod} p = (a \operatorname{mod} p + b \operatorname{mod} p) \operatorname{mod} p   \cdot \quad (a-b) \operatorname{mod} p = (a \operatorname{mod} p - b \operatorname{mod} p + p) \operatorname{mod} p   \cdot \quad (a*b) \operatorname{mod} p = (a \operatorname{mod} p \cdot b \operatorname{mod} p) \operatorname{mod} p   \cdot \quad (a^b) \operatorname{mod} p = ((a \operatorname{mod} p)^b) \operatorname{mod} p   \cdot \quad ((a+b) \operatorname{mod} p + c) \operatorname{mod} p = (a+(b+c)) \operatorname{mod} p
```

```
 \begin{array}{l} \cdot \  \, \left( (a \cdot b) \bmod p \cdot c \right) \bmod p = \left( a \cdot (b \cdot c) \right) \bmod p \\ \\ \cdot \  \, \left( a + b \right) \bmod p = \left( b + a \right) \bmod p \\ \\ \cdot \  \, \left( a \cdot b \right) \bmod p = \left( b \cdot a \right) \bmod p \\ \\ \cdot \  \, \left( (a \cdot b) \bmod p \cdot c \right) = \left( (a \cdot c) \bmod p + (b \cdot c) \bmod p \right) \bmod p \\ \\ \cdot \  \, 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 \\ \\ \cdot \  \, a \equiv b \pmod c, b \equiv d \pmod c \\ \\ \oplus \  \, a \equiv d \pmod c \\ \\ \cdot \  \, \left\{ a \equiv b \pmod m \\ c \equiv d \pmod m \right\} \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

```
1 vector<int> p;
  bitset<MAXN> is_notp;
2
  void PrimeTable(int n) {
       is_notp.reset();
       is_notp[0] = is_notp[1] = 1;
5
6
       for (int i = 2; i <= n; ++i) {
           if (!is_notp[i]){
7
               p.push_back(i);
9
10
           for (int j = 0; j < (int)p.size(); ++j) {</pre>
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

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

2.4 Exponentiating by Squaring

```
1 T pow(int a, int b, int c){ // calculate a ^ b % c
      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 3
```

2.5 Euler

```
1 int Phi(int n){
          Phi(int i);
int ans = n;
for (int i: p) {
    if (i * i > n){
        break;
    ...
 2
                 if (n % i == 0){
                        ans /= i;
ans *= i - 1;
while (n % i == 0){
n /= i;
 9
10
11
12
                 }
13
14
           if (n != 1) {
15
                 ans /= n;
16
                 ans *= n - 1;
17
18
19
           return ans;
20 }
```