


```

5     }
6   }
7 }

```

1.6 Dijkstra

```

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 >
14        b.d; }
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     for (int i = 0; i < V; i++){
25         d[i] = INF;
26     }
27
28     d[ss] = 0;
29     p[ss] = -1;
30     vis.reset() : Q.push(heap(0, ss));
31     heap x;
32
33     while (!Q.empty()){
34
35         x = Q.top();
36         Q.pop();
37         int p = x.p;
38
39         if (vis[p])
40             continue;
41         vis[p] = 1;
42
43         for (int i = 0; i < G[p].size(); i++){
44             edge &e = edges[G[p][i]];
45             if (d[e.t] > d[p] + e.d){
46                 d[e.t] = d[p] + e.d;
47                 p[e.t] = G[p][i];
48                 Q.push(heap(d[e.t], e.t));
49             }
50         }
51     }
52 }
53 }

```

2 Number Theory

2.1 Modulo

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

- $((a \cdot b) \bmod p \cdot c) \bmod p = (a \cdot (b \cdot c)) \bmod p$
- $(a + b) \bmod p = (b + a) \bmod p$
- $(a \cdot b) \bmod p = (b \cdot a) \bmod p$
- $((a + b) \bmod p \cdot c) = ((a \cdot c) \bmod p + (b \cdot c) \bmod p) \bmod p$
- $a \equiv b \pmod{m} \Rightarrow c \cdot m = a - b, c \in \mathbb{Z}$
 $\Rightarrow a \equiv b \pmod{m} \Rightarrow m \mid a - b$
- $a \equiv b \pmod{c}, b \equiv d \pmod{c}$
 $\text{則 } a \equiv d \pmod{c}$
- $\begin{cases} a \equiv b \pmod{m} \\ c \equiv d \pmod{m} \end{cases} \Rightarrow \begin{cases} a \pm c \equiv b \pm d \pmod{m} \\ a \cdot c \equiv b \cdot d \pmod{m} \end{cases}$

2.2 Linear Sieve

```

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

```

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

```

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

```
1 | int Phi(int n){
2 |     int ans = n;
3 |     for (int i: p) {
4 |         if (i * i > n){
5 |             break;
6 |         }
7 |         if (n % i == 0){
8 |             ans /= i;
9 |             ans *= i - 1;
10 |            while (n % i == 0){
11 |                n /= i;
12 |            }
13 |        }
14 |    }
15 |    if (n != 1) {
16 |        ans /= n;
17 |        ans *= n - 1;
18 |    }
19 |    return ans;
20 | }
```