

## Contents

### 1 Graph Theory

1.1 Adjacency List . . . . .	1
1.2 DFS . . . . .	1
1.3 BFS . . . . .	1
1.4 Disjoint Set and Kruskal . . . . .	1
1.5 Floyd-Warshall . . . . .	2
1.6 Dijkstra . . . . .	2

### 2 Number Theory

2.1 Modulo . . . . .	2
2.2 Linear Sieve . . . . .	2
2.3 Prime Factorization . . . . .	3
2.4 Exponentiating by Squaring . . . . .	3
2.5 Euler . . . . .	3

### 3 Dynamic Programming

3.1 Fibonacci . . . . .	3
3.2 Pascal Triangle . . . . .	3
3.3 Robot . . . . .	3
3.4 Max Interval Sum . . . . .	3
3.5 Max Area . . . . .	3
3.6 LCS . . . . .	4
3.7 0-1 Bag . . . . .	4
3.8 Infinite Bag . . . . .	4
3.9 Tree . . . . .	4

## 1 Graph Theory

### 1.1 Adjacency List

```

1 vector<int> list[5];
2
3 void Adjacency_List(){
4
5     // initial
6     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)
12         list[a].push_back(b);
13         // list[b].push_back(a);
14 }

```

### 1.2 DFS

```

1 vector<int> G[N];
2 bitset<N> vis;
3 void dfs(int s) {
4     vis[s] = 1;
5     for (int t : G[s]) {
6         if (!vis[t])
7             dfs(t);
8     }
9 }

```

### 1.3 BFS

```

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

```

```

13         vis[t] = 1;
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 {
4         return w < rhs.w; }
5 };
6 vector<int> parent;
7 vector<Edge> E;
8
9 bool cmp(Edge edge1, Edge edge2){
10     return edge2.w > edge1.w;
11 }
12
13 int find(int x){
14     if(parent[x] < 0){
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     }
29     parent[a] = parent[a] + parent[b];
30     parent[b] = a;
31     return true;
32 }
33
34 void Kruskal() {
35
36     int cost = 0;
37
38     sort(E.begin(), E.end()); // sort by w
39     // sort(E.begin(), E.end(), cmp);
40
41     // two edge in the same tree or not
42     for (auto it: E){
43         it.s = Find(it.s);
44         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
54     parent = vector<int> (N, -1);
55
56     for(i = 0; i < M; i++){
57         cin >> u >> v >> w;
58         E.push_back({u, v, w});
59     }
60
61     Kruskal();
62
63     return 0;
64 }

```

## 1.5 Floyd-Warshall

```

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

```

## 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 >
        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     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

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

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

### 2.2 Linear Sieve

```

1 vector<int> p;
2 bitset<MAXN> is_notp;
3 void PrimeTable(int n){
4
5     is_notp.reset();
6     is_notp[0] = is_notp[1] = 1;
7
8     for (int i = 2; i <= n; ++i){
9         if (!is_notp[i]){
10             p.push_back(i);
11         }
12         for (int j = 0; j < (int)p.size(); ++j){
13             if (i * p[j] > n){
14                 break;
15             }
16             is_notp[i * p[j]] = 1;
17
18             if (i % p[j] == 0){
19                 break;
20             }
21         }
22     }
23 }
24 }

```

### 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 }

```

# 3 Dynamic Programming

## 3.1 Fibonacci

```

1 // f(n) = f(n - 1) + f(n - 2)
2 // f(0) = 0, f(1) = 1
3 int dp[30];
4 int f(int n){
5     if (dp[n] != -1){
6         return dp[n];
7     }
8     return dp[n] = f(n - 1) + f(n - 2);
9 }
10
11 int main(){
12     memset(dp, -1, sizeof(dp));
13     dp[0] = 0;
14     dp[1] = 1;
15     cout << f(25) << '\n';
16 }

```

## 3.2 Pascal Triangle

```

1 // init: f(i, 0) = f(i, i) = 1
2 // tren: f(i, j) = f(i - 1, j) + f(i - 1, j - 1)
3 int main(){
4     int dp[30][30];
5     memset(dp, 0, sizeof(dp));
6     for (int i = 0; i < 30; ++i){
7         dp[i][0] = dp[i][i] = 1;
8     }
9     for (int i = 1; i < 30; ++i){
10        for (int j = 1; j < 30; ++j){

```

```

11            dp[i][j] = dp[i - 1][j] + dp[i - 1][j - 1];
12        }
13    }
14 }

```

## 3.3 Robot

```

1 // f(1, j) = f(i, 1) = 1
2 // f(i, j) = f(i - 1, j) + f(i, j - 1)
3 int dp[105][105];
4 dp[1][1] = 1;
5 for (int i = 1; i <= 100; ++i){
6     for (int j = 1; j <= 100; ++j){
7         if (i + 1 <= 100) dp[i + 1][j] += dp[i][j];
8         if (j + 1 <= 100) dp[i][j + 1] += dp[i][j];
9     }
10 }

```

## 3.4 Max Interval Sum

```

1 // No Limit
2 int ans = A[1];
3 sum[1] = dp[1] = A[1];
4
5 for (int i = 2; i <= n; ++i){
6     sum[i] = A[i] + sum[i - 1];
7     dp[i] = min(dp[i - 1], sum[i]);
8     ans = max(ans, sum[i] - dp[i - 1]);
9 }
10
11 // length <= L
12 int a[15] = {0, 6, -8, 4, -10, 7, 9, -6, 4, 5, -1};
13 int sum[15];
14
15 int main(){
16     int L = 3, ans = 0;
17     for (int i = 1; i <= 10; ++i)
18     {
19         sum[i] = a[i] + sum[i - 1];
20     }
21     deque<int> dq;
22     dq.push_back(0);
23     for (int i = 1; i <= 10; ++i){
24         if (i - dq.front() > L){
25             dq.pop_front();
26         }
27         ans = max(ans, sum[i] - sum[dq.front()]);
28         while (!dq.empty() && sum[i] < sum[dq.back()]){
29             dq.pop_back();
30         }
31         dq.push_back(i);
32     }
33     cout << ans << '\n';
34 }

```

## 3.5 Max Area

```

1 const int N = 25;
2
3 int main(){
4     int n;
5     cin >> n;
6     vector<int> H(n + 5), L(n + 5), R(n + 5);
7     for (int i = 0; i < n; ++i){
8         cin >> H[i];
9     }
10    stack<int> st;
11    // calculate R[]
12    for (int i = 0; i < n; ++i){
13        while (!st.empty() && H[st.top()] > H[i]){

```

```

14     R[st.top()] = i - 1;
15     st.pop();
16 }
17     st.push(i);
18 }
19 while (!st.empty()){
20     R[st.top()] = n - 1;
21     st.pop();
22 }
23 // calculate L[]
24 for (int i = n - 1; i >= 0; --i){
25     while (!st.empty() && H[st.top()] > H[i]){
26         L[st.top()] = i + 1;
27         st.pop();
28     }
29     st.push(i);
30 }
31 while (!st.empty()){
32     L[st.top()] = 0;
33     st.pop();
34 }
35 int ans = 0;
36 for (int i = 0; i < n; ++i){
37     ans = max(ans, H[i] * (R[i] - L[i] + 1));
38     cout << i << ' ' << L[i] << ' ' << R[i] <<
39         '\n';
40 }
41 cout << ans << '\n';
42 }

```

### 3.6 LCS

```

1 // init: dp[i][0] = dp[0][i] = 0
2 // tren: dp[i][j] =
3 // if a[i] = b[j]
4 // dp[i - 1][j - 1] + 1
5 // else
6 // max(dp[i - 1][j], dp[i][j - 1])
7 // LIS
8 // init: dp[0] = 0
9 // tren: dp[i] = max{dp[j] | j < i and A[j] <
10 // A[i]} + 1
11 // LIS → LCS (嚴格遞增)
12 // A 為原序列, B = sort(A)
13 // 對 A, B 做 LCS
14 // LCS → LIS (數字重複、有數字在 B 裡面不在 A 裡面)
15 // A, B 為原本的兩序列
16 // 對 A 序列作編號轉換, 將轉換規則套用在 B
17 // 對 B 做 LIS
18 int dp[a.size() + 1][b.size() + 1];
19 for(int i = 0; i <= a.size(); i++){
20     dp[i][0] = 0;
21 }
22 for(int i = 0; i <= b.size(); i++){
23     dp[0][i] = 0;
24 }
25 for(int i = 1; i <= a.size(); i++){
26     for(int j = 1; j <= b.size(); j++){
27         if(a[i - 1] == b[j - 1]){
28             dp[i][j] = dp[i - 1][j - 1] + 1;
29         }
30         else{
31             dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
32         }
33     }
34 }
35 return 0;
36 }

```

### 3.7 0-1 Bag

```

1 // 不放: 重量和價值不變
2 // to f(i, j) = f(i - 1, j)
3 // 放: 重量 + w_i, 價值 + v_i
4 // to f(i, j) = f(i - 1, j - w_i) + v_i
5 // tren: f(i, j) = max(f(i - 1, j), f(i - 1, j - w_i)
6 // + v_i)
7 int dp[MXN + 1][MXW + 1];
8 memset(dp, 0, sizeof(dp));
9 for (int i = 1; i <= MXN; ++i){
10     for (int j = 0; j < w[i]; ++j){
11         dp[i][j] = dp[i - 1][j];
12     }
13     for (int j = w[i]; j <= MXW; ++j){
14         dp[i][j] = max(dp[i - 1][j - w[i]] + v[i],
15             dp[i - 1][j]);
16     }
17 }
18 cout << dp[MXN][MXW] << '\n';

```

### 3.8 Infinite Bag

```

1 // f(i, j) = max(f(i - 1, j), f(i - 1, j - wi) + vi,
2 // f(i, j - wi) + vi)
3 // coin chage
4 // 最少幾枚能湊成 M 元
5 // f(i, j) = min(f(i - 1, j), f(i - 1, j - ci) + 1, f(i, j - ci) + 1)
6 // 多少種能湊成 M 元
7 // f(i, j) = f(i - 1, j) + f(i, j - ci)
8 int dp[MXW];
9 memset(dp, -INF, sizeof(dp));
10 dp[0] = 0;
11 for (int i = 0; i < N; ++i){
12     for (int j = w[i]; j <= MXW; ++j){
13         dp[j] = max(dp[j - w[i]] + v[i], dp[j]);
14     }
15 }

```

### 3.9 Tree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MXV = 15;
4 vector<int> G[MXV];
5 int high[MXV][2];
6 int ans[MXV], height[MXV];
7
8 void dfs(int u){
9     height[u] = 1;
10     for (int v : G[u]){
11         dfs(v);
12         height[u] = max(height[u], height[v] + 1);
13         if (high[u][0] == 0 || height[high[u][0]] <
14             height[v]){
15             high[u][1] = high[u][0];
16             high[u][0] = v;
17         }
18         else if (high[u][1] == 0 ||
19             height[high[u][1]] < height[v]){
20             high[u][1] = v;
21         }
22     }
23 }
24 void dfs2(int u, int legnth){
25     ans[u] = height[high[u][0]] +
26         max(height[high[u][1]], legnth) + 1;
27     for (int v : G[u]){
28         if (v == high[u][0]){
29             dfs2(v, max(height[high[u][1]], legnth) +
30                 1);
31         }
32         else{
33             dfs2(v, legnth);
34         }
35     }
36 }

```

```
30         dfs2(v, max(height[high[u][0]], legnth) +
31             1);
32     }
33 }
34
35 int main(){
36     int n;
37     cin >> n;
38     for (int i = 1; i < n; ++i){
39         int x, y;
40         cin >> x >> y;
41         G[x].emplace_back(y);
42     }
43     dfs(1);
44     dfs2(1, 0);
45     for (int i = 1; i <= n; ++i){
46         cout << ans[i] << '\n';
47     }
48 }
```