# 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++){
```

```
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
23     priority_queue<heap> Q;
24
25     for (int i = 0; i < V; i++){
26         d[i] = INF;
27     }
28
29     d[ss] = 0;
30     p[ss] = -1;
31     vis.reset() : Q.push(heap(0, ss));
32     heap x;
33
34     while (!Q.empty()){
35
36         x = Q.top();
37         Q.pop();
38         int p = x.p;
39
40         if (vis[p])
41             continue;
42         vis[p] = 1;
43
44         for (int i = 0; i < G[p].size(); i++){
45             edge &e = edges[G[p][i]];
46             if (d[e.t] > d[p] + e.d){
47                 d[e.t] = d[p] + e.d;
48                 p[e.t] = G[p][i];
49                 Q.push(heap(d[e.t], e.t));
50             }
51         }
52     }
53 }
```

# 2  Number Theory

## 2.1  Modulo

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

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

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

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

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

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

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

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

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

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

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

```
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
17             is_notp[i * p[j]] = 1;
18
19             if (i % p[j] == 0){
20                 break;
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  // trens : 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] <<
                    '\n';
39        }
40        cout << ans << '\n';
41 }
```