

La tasita

ICPC Notebook

Team: Turistas

Índice general

I	Dynamic Programming	3
1.1.	Longest Common Subsequence	4
1.2.	Longest Increasing Subsequence	5
1.3.	Longest Increasing Subsequence Fast	7
1.4.	Maximum Subarray	9
1.5.	Knapsack	10
1.6.	Knapsack Binary	11
1.7.	Knapsack Re	12
1.8.	Knapsack Subset Sum	14
II	Data Structures	15
2.1.	Fenwick Tree	16
2.2.	Fenwick Tree 2d	18
2.3.	Treap	20
2.4.	Implicit Treap	23
2.5.	Ordered Set	26
2.6.	Segment Tree	27
2.7.	Segment Tree Summary	29
2.8.	Persistent Segment Tree	31
2.9.	Sparse Table	34
III	Geometry	36
3.1.	Convex Hull	37
IV	Graphs	39
4.1.	2 Sat	40
4.2.	Bellman Ford	43
4.3.	Dijkstra	45
4.4.	Floyd Warshall	47
4.5.	Ford Fulkerson	48
4.6.	Functional Graph	51
4.7.	Hungarian	54
4.8.	Kosaraju	56
4.9.	Kruskal	59
4.10.	Heavy Light Descomposition	61
4.11.	LCA	64

4.12. Topological Sort	66
4.13. Centroid Descomposition	68
V Math	71
5.1. FFT	72
5.2. NTT	74
5.3. Binary Exponentiation	76
5.4. Matrix Exponentiation	77
VI Strings	79
6.1. KMP Automaton	80
6.2. Hashing	82

Parte I

Dynamic Programming

1.1. Longest Common Subsequence

```
1 // https://leetcode.com/problems/longest-common-subsequence/submissions/1815539365/
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 string s1, s2;
6 const int n = 1000;
7 int dp[n][n];
8
9 int LCS(int i, int j) {
10     if (i < 0 || j < 0) return 0;
11     if (dp[i][j] != -1) return dp[i][j];
12
13     if (s1[i] == s2[j]) dp[i][j] = 1 + LCS(i - 1, j - 1);
14     else dp[i][j] = max(LCS(i-1,j), LCS(i,j-1));
15
16     return dp[i][j];
17 }
18
19 string LCSReconstruccion() {
20     int i = s1.size() - 1;
21     int j = s2.size() - 1;
22     string ans = "";
23
24     while (i >= 0 && j >= 0) {
25         if (s1[i] == s2[j]) {
26             ans += s1[i];
27             i--, j--;
28         }
29         else {
30             if (i > 0 && (j == 0 || dp[i - 1][j] >= dp[i][j - 1])) i--;
31             else j--;
32         }
33     }
34
35     reverse(ans.begin(), ans.end());
36     return ans;
37 }
38
39 int main() {
40     memset(dp, -1, sizeof(dp));
41     s1 = "abcde";
42     s2 = "ace";
43
44     cout << LCS(s1.size() - 1, s2.size() - 1) << endl;
45     cout << LCSReconstruccion() << endl;
46 }
```

1.2. Longest Increasing Subsequence

```
1 // https://leetcode.com/problems/longest-increasing-subsequence/  
2 // description/  
3 #include <bits/stdc++.h>  
4 using namespace std;  
5  
6 const int max_n = 100000;  
7 int dp[max_n + 1];  
8  
9 int LIS(vector<int> arr) {  
10     int n = arr.size();  
11     memset(dp, 0, sizeof(dp));  
12  
13     int res = 0;  
14     for (int i = 0; i < n; i++) {  
15         dp[i] = 1;  
16         int valAc = arr[i];  
17         for (int j = 0; j < i; j++) {  
18             int valAn = arr[j];  
19             if (valAn < valAc) dp[i] = max(dp[i], dp[j] + 1);  
20         }  
21         res = max(res, dp[i]);  
22     }  
23  
24     return res;  
25 }  
26  
27 vector<int> LISReconstruccion(vector<int> arr) {  
28     int n = arr.size();  
29     vector<int> padre(n, -1);  
30  
31     int res = 0;  
32     indiceMejorSecuencia = -1;  
33  
34     for (int i = 0; i < n; i++) {  
35         dp[i] = 1;  
36         int valAc = arr[i];  
37         for (int j = 0; j < i; j++) {  
38             int valAn = arr[j];  
39             if (valAn < valAc && dp[j] + 1 > dp[i]) {  
40                 dp[i] = dp[j] + 1;  
41                 padre[i] = j;  
42             }  
43         }  
44  
45         if (dp[i] > res) {  
46             res = dp[i];  
47             indiceMejorSecuencia = i;  
48         }  
49     }  
50  
51     vector<int> secuencia;  
52     for (int i = indiceMejorSecuencia; i != -1; i = padre[i]) {  
53         secuencia.push_back(arr[i]);  
54     }  
55 }
```

```

56     reverse(secuencia.begin(), secuencia.end());
57
58     return secuencia;
59 }
60
61 int main() {
62     vector<int> nuevo = {1,3,6,7,9,4,10,5,6};
63     cout << LIS(nuevo) << endl;
64
65     vector<int> secuencia = LISReconstruccion(nuevo);
66     for (int& val : secuencia) cout << val << " ";
67 }

```

1.3. Longest Increasing Subsequence Fast

```
1 // https://leetcode.com/problems/longest-increasing-subsequence/  
2 // description/  
3 #include <bits/stdc++.h>  
4 using namespace std;  
5  
6 int LISFast(vector<int> arr) {  
7     int n = arr.size();  
8     vector<int> res;  
9     res.push_back(arr[0]);  
10  
11     for (int i = 1; i < n; i++) {  
12         if (arr[i] > res.back()) {  
13             res.push_back(arr[i]);  
14         } else {  
15             auto it = lower_bound(res.begin(), res.end(), arr[i]);  
16             int busca = it - res.begin();  
17             res[busca] = arr[i];  
18         }  
19     }  
20  
21     return res.size();  
22 }  
23  
24 vector<int> LISFastReconstruccion(vector<int> arr) {  
25     int n = arr.size();  
26     vector<int> res;  
27     vector<int> resIndex;  
28     vector<int> padre(n, -1);  
29  
30     for (int i = 0; i < n; i++) {  
31         int x = arr[i];  
32         auto it = lower_bound(res.begin(), res.end(), x);  
33         int pos = it - res.begin();  
34  
35         if (it == res.end()) {  
36             res.push_back(x);  
37             resIndex.push_back(i);  
38         } else {  
39             *it = x;  
40             resIndex[pos] = i;  
41         }  
42  
43         if (pos > 0)  
44             padre[i] = resIndex[pos - 1];  
45     }  
46  
47     vector<int> lis;  
48     int idx = resIndex.back();  
49     while (idx != -1) {  
50         lis.push_back(arr[idx]);  
51         idx = padre[idx];  
52     }  
53     reverse(lis.begin(), lis.end());  
54     return lis;  
55 }
```



```
56 int main() {  
57     vector<int> nuevo = {1,3,6,7,9,4,10,5,6};  
58     cout << LISFast(nuevo) << endl;  
59  
60     vector<int> reconstruccion = LISFastReconstruccion(nuevo);  
61     for (int& val : reconstruccion) cout << val << " ";  
62 }
```

1.4. Maximum Subarray

```
1 // https://leetcode.com/problems/maximum-subarray/submissions  
  //1815546613/  
2 #include <bits/stdc++.h>  
3 using namespace std;  
4  
5 const int max_n = 100001;  
6 int dp[max_n];  
7  
8 int maximumSub(vector<int>& arr) {  
9     memset(dp,0,sizeof(dp));  
10    int n = arr.size();  
11  
12    for (int i = 1; i <= n; i++) {  
13        int& valor = arr[i - 1];  
14        dp[i] = max(valor, dp[i - 1] + valor);  
15    }  
16  
17    int maximo = -1000000000;  
18    for (int i = 1; i <= n; i++) {  
19        maximo = max(maximo, dp[i]);  
20    }  
21    return maximo;  
22 }  
23  
24 int main() {  
25     vector<int> valores = {-2,1,-3,4,-1,2,1,-5,4};  
26     cout << maximumSub(valores);  
27 }
```

1.5. Knapsack

```
1 // Source: https://atcoder.jp/contests/dp/tasks/dp\_d
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int MAX_W = 100001;
6 long long dp[MAX_W];
7 int W = MAX_W;
8
9 long long knapsack(const vector<pair<int, long long>>& items) {
10     memset(dp, -1, sizeof(dp));
11     int n = items.size();
12
13     dp[0] = 0;
14     for (int i = 0; i < n; i++) {
15         int w = items[i].first;
16         long long v = items[i].second;
17
18         for (int j = W; j >= w; j--) {
19             if (dp[j - w] != -1) {
20                 dp[j] = max(dp[j], dp[j - w] + v);
21             }
22         }
23     }
24
25     long long max_v = 0;
26     for (int j = 0; j <= W; j++) {
27         max_v = max(max_v, dp[j]);
28     }
29     return max_v;
30 }
31
32 int main() {
33     vector<pair<int, long long>> items = {{2,3},{3,5},{6,5}};
34     cout << knapsack(items) << endl;
35     return 0;
36 }
```

1.6. Knapsack Binary

```
1 // https://cses.fi/problemset/task/1159/
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int MAX_W = 100001;
6 int dp[MAX_W];
7 int W;
8
9 int knapsack(const vector<pair<int, int>>& items) {
10     memset(dp, -1, sizeof(dp));
11     int n = items.size();
12     dp[0] = 0;
13
14     for (int i = 0; i < n; i++) {
15         int w = items[i].first;
16         int v = items[i].second;
17
18         for (int j = W; j >= w; j--) {
19             if (dp[j - w] != -1) {
20                 dp[j] = max(dp[j], dp[j - w] + v);
21             }
22         }
23     }
24
25     int max_v = 0;
26     for (int j = 0; j <= W; j++) {
27         max_v = max(max_v, dp[j]);
28     }
29     return max_v;
30 }
31
32 void res(const vector<int>& pesos, const vector<int>& valores, const
vector<int>& copias) {
33     int n = pesos.size();
34     vector<pair<int, int>> items;
35     for (int i = 0; i < n; i++) {
36         int k = copias[i];
37         int b = 1;
38         while (k > 0) {
39             int num = min(k, b);
40             items.push_back({pesos[i] * num, valores[i] * num});
41             k -= num;
42             b *= 2;
43         }
44     }
45     cout << knapsack(items) << endl;
46 }
47
48 int main() {
49     vector<int> pesos = {2, 6, 3};
50     vector<int> valores = {8, 5, 4};
51     vector<int> copias = {3, 5, 2};
52     res(pesos, valores, copias);
53 }
```

1.7. Knapsack Re

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAX_N = 1001;
5 const int MAX_W = 1001;
6
7 int N, W;
8 long long dp[MAX_N][MAX_W];
9 vector<pair<int, long long>> items;
10
11 long long knapsack(int i, int w_rem) {
12     if (i == N || w_rem == 0) {
13         return 0;
14     }
15     if (dp[i][w_rem] != -1) {
16         return dp[i][w_rem];
17     }
18
19     int w = items[i].first;
20     long long v = items[i].second;
21
22     long long res = knapsack(i + 1, w_rem);
23     if (w_rem >= w) {
24         res = max(res, v + knapsack(i + 1, w_rem - w));
25     }
26
27     return dp[i][w_rem] = res;
28 }
29
30 vector<pair<int, long long>> knapsackReconstruccion() {
31     vector<pair<int, long long>> tomados;
32     int w_rem = W;
33
34     for (int i = 0; i < N; i++) {
35         int w = items[i].first;
36         long long v = items[i].second;
37
38         if (knapsack(i + 1, w_rem) == dp[i][w_rem]) {
39             continue;
40         }
41
42         else if (w_rem >= w && (v + knapsack(i + 1, w_rem - w) == dp[i][
43 w_rem])) {
44             tomados.push_back(items[i]);
45             w_rem -= w;
46         }
47     }
48     return tomados;
49 }
50
51 int main() {
52     memset(dp, -1, sizeof(dp));
53     items = {{3, 30}, {4, 50}, {5, 60}};
54     W = 8;
55     N = items.size();
```

```
56     cout << knapsack(0, W) << endl;
57
58     vector<pair<int, long long>> res = knapsackReconstruccion();
59     cout << "Items tomados:" << endl;
60     for (auto& item : res) {
61         cout << "Peso: " << item.first << ", Valor: " << item.second <<
endl;
62     }
63 }
```

1.8. Knapsack Subset Sum

```
1 // https://leetcode.com/problems/tallest-billboard/description/
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int MAX_N = 2501;
6 int diferencia[2][MAX_N];
7
8 int knapsackDiferencia(const vector<int>& arr) {
9     int n = arr.size();
10    memset(diferencia, -1, sizeof(diferencia));
11    diferencia[0][0] = 0;
12
13    for (int i = 0; i < n; i++) {
14        int ahora = arr[i];
15        int actual = i & 1;
16        int siguiente = actual ^ 1;
17
18        fill(diferencia[siguiente], diferencia[siguiente] + MAX_N, -1);
19
20        for (int valor = 0; valor < MAX_N; valor++) {
21            if (diferencia[actual][valor] == -1) continue;
22
23            diferencia[siguiente][valor] = max(
24                diferencia[siguiente][valor],
25                diferencia[actual][valor]
26            );
27
28            int nuevaDiferenciaMayor = valor + ahora;
29            if (nuevaDiferenciaMayor < MAX_N) {
30                diferencia[siguiente][nuevaDiferenciaMayor] = max(
31                    diferencia[siguiente][nuevaDiferenciaMayor],
32                    diferencia[actual][valor]
33                );
34            }
35
36            int nuevaDiferenciaMenor = abs(valor - ahora);
37            int nuevaAltura = diferencia[actual][valor] + min(valor,
38                ahora);
39            diferencia[siguiente][nuevaDiferenciaMenor] = max(
40                diferencia[siguiente][nuevaDiferenciaMenor],
41                nuevaAltura
42            );
43        }
44    }
45
46    return diferencia[n & 1][0];
47 }
48
49 int main() {
50     vector<int> roads = {1,2,3,6};
51     cout << knapsackDiferencia(roads) << endl;
52 }
```

Parte II

Data Structures

2.1. Fenwick Tree

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e5+5;
12
13 struct FenwickTree {
14     vector<ll> bit;
15     ll n;
16
17     FenwickTree(ll n) {
18         this->n = n;
19         bit.assign(n, 0);
20     }
21
22     FenwickTree(vector<ll> const &a) : FenwickTree(a.size()) {
23         for (size_t i = 0; i < a.size(); i++)
24             add(i, a[i]);
25     }
26
27     ll sum(ll r) {
28         ll ret = 0;
29         for (; r >= 0; r = (r & (r + 1)) - 1)
30             ret += bit[r];
31         return ret;
32     }
33
34     ll sum(ll l, ll r) {
35         return sum(r) - sum(l - 1);
36     }
37
38     void add(ll idx, ll delta) {
39         for (; idx < n; idx = idx | (idx + 1))
40             bit[idx] += delta;
41     }
42 };
43
44
45 void solve() {
46     int n; cin >> n;
47     vector<ll> arr(n);
48     for (int i = 0; i < n; i++) {
49         cin >> arr[i];
50     }
51     FenwickTree ft(arr);
52
53     int m;
54     cin >> m;
55     for (int i = 1; i <= m; i++) {
56         int q,l,r;
```

```

57     cin >> q >> l >> r;
58     if (q == 's') {
59         cout << ft.sum(l, r) << " ";
60     }
61     else {
62         int delta = r - arr[l];
63         arr[l] = r;
64         ft.add(l, delta);
65     }
66 }
67 }
68
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t = 1;
73     //cin >> t;
74     while(t--)
75         solve();
76 }

```

2.2. Fenwick Tree 2d

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e3+5;
12
13 int n, m;
14 int qn;
15 char q[10];
16 int f[MAXN][MAXN];
17
18 void update(int x, int y, int delta) {
19     for (int i = x; i <= n; i = i | (i + 1))
20         for (int j = y; j <= m; j = j | (j + 1))
21             f[i][j] += delta;
22 }
23
24 int getSum(int x, int y) {
25     int res = 0;
26     for (int i = x; i > 0; i = (i & (i + 1)) - 1)
27         for (int j = y; j > 0; j = (j & (j + 1)) - 1)
28             res += f[i][j];
29     return res;
30 }
31
32 int getSum(int xFrom, int xTo, int yFrom, int yTo) {
33     return getSum(xTo, yTo) - getSum(xTo, yFrom - 1) - getSum(xFrom - 1,
34         yTo) + getSum(xFrom - 1, yFrom - 1);
35 }
36
37 void solve() {
38     cin >> n >> qn;
39     m = n;
40
41     for (int i = 1; i <= qn; i++) {
42         cin >> q;
43         if (q[0] == 'A') {
44             int x, y;
45             cin >> x >> y;
46             update(x, y, 1);
47         }
48         else {
49             int xFrom, xTo, yFrom, yTo;
50             cin >> xFrom >> yFrom >> xTo >> yTo;
51             if (xFrom > xTo)
52                 swap(xFrom, xTo);
53             if (yFrom > yTo)
54                 swap(yFrom, yTo);
55             cout << getSum(xFrom, xTo, yFrom, yTo);
56         }
57     }
```

```
56     }
57 }
58
59 int main() {
60     ios::sync_with_stdio(false);
61     cin.tie(nullptr);
62     int t = 1;
63     //cin >> t;
64     while(t--)
65         solve();
66 }
```

2.3. Treap

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9 #define endl '\n'
10
11 const int INF = 1 << 30;
12 const int MAXN = 1e5+5;
13
14 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
15
16 int random(int l, int r) {
17     return uniform_int_distribution<int>(l, r)(rng);
18 }
19
20 struct node {
21     int key;
22     int priority;
23     node* left;
24     node* right;
25     int cnt;
26     node(int v) : key(v) {
27         priority = random(0,1e9);
28         left = right = nullptr;
29         cnt = 1;
30     }
31 };
32
33 int cnt(node* t) {
34     return t ? t->cnt : 0;
35 }
36
37 void update(node *t) {
38     if (t) t->cnt = 1 + cnt(t->left) + cnt(t->right);
39 }
40
41 node* merge(node* a, node* b) {
42     if(a == nullptr) return b;
43     if(b == nullptr) return a;
44     if(a->priority > b->priority) {
45         a->right = merge(a->right, b);
46         update(a);
47         return a;
48     } else {
49         b->left = merge(a, b->left);
50         update(b);
51         return b;
52     }
53 }
54
55 pair<node*, node*> split(node *T, int k) {
56     if(T == nullptr) return {nullptr, nullptr};
```

```

57     if(T->key <= k) {
58         auto p = split(T->right,k);
59         T->right = p.fst;
60         update(T);
61         return {T, p.snd};
62     } else {
63         auto p = split(T->left, k);
64         T->left = p.snd;
65         update(T);
66         return {p.fst, T};
67     }
68 }
69
70 node* insert(node* T, int x) {
71     auto p = split(T,x);
72     T = merge(p.fst, new node(x));
73     T = merge(T, p.snd);
74     return T;
75 }
76
77 node* remove(node* T, int x) {
78     auto p = split(T, x);           // p.fst:      x, p.snd: > x
79     auto p2 = split(p.fst, x-1);    // p2.fst: < x, p2.snd: == x
80
81     // p2.snd contains the node(s) with key = x      we discard it
82     return merge(p2.fst, p.snd);
83 }
84
85 bool find(node* T, int x) {
86     if (!T) return false;
87     if (T->key == x) return true;
88     if (x < T->key) return find(T->left, x);
89     return find(T->right, x);
90 }
91
92 node* kth(node* t, int k) {
93     int left_size = cnt(t->left);
94     if (k < left_size) return kth(t->left, k);
95     else if (k == left_size) return t;
96     else return kth(t->right, k - left_size - 1);
97 }
98
99 void inorder(node* T) {
100     if (!T) return;
101     inorder(T->left);
102     cout << T->key << " ";
103     inorder(T->right);
104 }
105
106 void solve() {
107     node* root = nullptr;
108
109     root = insert(root, 5);
110     root = insert(root, 2);
111     root = insert(root, 8);
112     root = insert(root, 1);
113     root = insert(root, 10);
114

```

```

115     cout << "Treap inorder (sorted): ";
116     inorder(root); // 1 2 5 8 10
117     cout << endl;
118
119     cout << "Find 8? " << (find(root, 8) ? "Yes" : "No") << endl;
120     cout << "Find 7? " << (find(root, 7) ? "Yes" : "No") << endl;
121
122     root = remove(root, 5);
123     cout << "After removing 5: ";
124     inorder(root); // 1 2 8 10
125     cout << endl;
126 }
127
128
129 int main() {
130     // ios::sync_with_stdio(false);
131     // cin.tie(nullptr);
132     int t = 1;
133     //cin >> t;
134     while(t--)
135         solve();
136 }

```

2.4. Implicit Treap

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define endl '\n';
5
6 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
7
8 int random(int l, int r) {
9     return uniform_int_distribution<int>(l, r)(rng);
10 }
11
12 struct node {
13     int value;
14     int priority;
15     node* left;
16     node* right;
17     int cnt;
18
19     bool rev;
20     node(int v) : value(v) {
21         priority = random(0, 1e9);
22         left = right = nullptr;
23         cnt = 1; // FIX: initialize cnt
24     }
25 };
26
27 int cnt(node* t) {
28     return t ? t->cnt : 0;
29 }
30
31 // push for reverse or lazy propagation
32 void push(node* t) {
33     if(!t || !t->rev) return;
34     t->rev = 0;
35     swap(t->left, t->right);
36     if(t->left) t->left->rev ^= 1;
37     if(t->right) t->right->rev ^= 1;
38 }
39
40 void update(node* t) {
41     if (!t) return;
42     t->cnt = 1 + cnt(t->left) + cnt(t->right);
43 }
44
45 node* merge(node* a, node* b) {
46     if (!a) return b;
47     if (!b) return a;
48     if (a->priority > b->priority) {
49         push(a);
50         a->right = merge(a->right, b);
51         update(a);
52         return a;
53     } else {
54         push(b);
55         b->left = merge(a, b->left);
56         update(b);
```



```

57     return b;
58 }
59 }
60
61 // 0-index
62 pair<node*, node*> split(node* T, int k, int add = 0) {
63     if (!T) return {nullptr, nullptr};
64     push(T);
65     int cur_key = add + cnt(T->left);
66     if (k <= cur_key) {
67         auto p = split(T->left, k, add);
68         T->left = p.second;
69         update(T);
70         return {p.first, T};
71     } else {
72         auto p = split(T->right, k, add + 1 + cnt(T->left));
73         T->right = p.first;
74         update(T);
75         return {T, p.second};
76     }
77 }
78
79 node* insert(node* T, int pos, int value) {
80     auto p = split(T, pos);
81     T = merge(p.first, new node(value));
82     T = merge(T, p.second);
83     return T;
84 }
85
86 node* remove(node* T, int pos) {
87     auto p1 = split(T, pos);           // [0..pos-1], [pos..end]
88     auto p2 = split(p1.second, 1);     // [pos], [pos+1..end]
89                                         // delete p2.first; // optional, free
90     memory
91     return merge(p1.first, p2.second);
92 }
93
94 void inorder(node* T) {
95     if (!T) return;
96     inorder(T->left);
97     cout << T->value << " ";
98     inorder(T->right);
99 }
100
101 node* kth(node* t, int k) {
102     push(t);
103     int left_size = cnt(t->left);
104     if (k < left_size) return kth(t->left, k);
105     else if (k == left_size) return t;
106     else return kth(t->right, k - left_size - 1);
107 }
108
109 node* update_range(node* t, int l, int r, int delta) {
110     auto p = split(t, r + 1);
111     auto p2 = split(p.first, l);
112
113     if (p2.second) {
114         p2.second->lazy += delta; // mark range for update

```

```

114     push(p2.second);           // optionally apply now (or can defer)
115     update(p2.second);        // recompute cnt/sum
116 }
117
118     return merge(merge(p2.first, p2.second), p.second);
119 }
120
121 void solve() {
122     node* root = nullptr;
123
124     root = insert(root, 0, 10);
125     root = insert(root, 1, 20);
126     root = insert(root, 2, 30);
127     cout << "Initial array: ";
128     inorder(root); cout << "\n";
129
130     root = insert(root, 1, 15);
131     cout << "After inserting 15 at pos 1: ";
132     inorder(root); cout << endl;
133
134     root = remove(root, 2);
135     cout << "After removing element at pos 2: ";
136     inorder(root); cout << endl;
137 }
138
139 int main() {
140     solve();
141 }

```

2.5. Ordered Set

```
1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 using namespace std;
5 using namespace __gnu_pbds;
6
7 template <class T>
8 using ordered_set = tree<
9 T,
10 null_type,
11 less<T>, // use less_equal<T> if you want to allow
12         duplicates (see note below)
13 rb_tree_tag,
14 tree_order_statistics_node_update>;
15
16 // --- Example usage ---
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20
21     ordered_set<int> s;
22
23     s.insert(10);
24     s.insert(20);
25     s.insert(30);
26     s.insert(40);
27
28     // 1      Get element by index (0-based)
29     cout << "Element at index 2: " << *s.find_by_order(2) << "\n"; // -> 30
30
31     // 2      Get index of an element
32     cout << "Index of 30: " << s.order_of_key(30) << "\n"; // -> 2
33
34     // 3      Lower bound behavior
35     cout << "Index where 25 would be inserted: " << s.order_of_key(25) <<
36         "\n"; // -> 2
37
38     // 4      Iterate through all elements
39     for (auto x : s) cout << x << " ";
40     cout << "\n";
41
42     // 5      Check if an element exists
43     if (s.find(20) != s.end())
44         cout << "20 exists in set\n";
45 }
```

2.6. Segment Tree

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e5+5;
12
13 int n, t[4*MAXN];
14
15 void build(vector<int> a, int v, int tl, int tr) {
16     if (tl == tr) {
17         t[v] = a[tl];
18     } else {
19         int tm = tl + (tr - tl) / 2;
20         build(a, v*2, tl, tm);
21         build(a, v*2+1, tm+1, tr);
22         t[v] = t[v*2] + t[v*2+1];
23     }
24 }
25
26 void update(int v, int tl, int tr, int pos, int new_val) {
27     if (tl == tr) {
28         t[v] = new_val;
29     } else {
30         int tm = tl + (tr - tl) / 2;
31         if (pos <= tm)
32             update(v*2, tl, tm, pos, new_val);
33         else
34             update(v*2+1, tm+1, tr, pos, new_val);
35         t[v] = t[v*2] + t[v*2+1];
36     }
37 }
38
39 int sum(int v, int tl, int tr, int l, int r) {
40     if (l > r)
41         return 0;
42     if (l == tl && r == tr) {
43         return t[v];
44     }
45     int tm = (tl + tr) / 2;
46     return sum(v*2, tl, tm, l, min(r, tm))
47         + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
48 }
49
50
51 int main(){
52     vector<int> arr = {1,1,1,1,1};
53     int n = arr.size();
54     build(arr, 1, 0, n-1);
55     cout << sum(1,0,n-1,0,4) << endl; // 5
56     cout << sum(1,0,n-1,0,1) << endl; // 2
```

```
57     cout << sum(1,0,n-1,1,4) << endl; // 4
58
59     update(1, 0, n-1, 4, 0);
60
61     cout << sum(1,0,n-1,0,4) << endl; // 4
62     cout << sum(1,0,n-1,0,1) << endl; // 2
63     cout << sum(1,0,n-1,1,4) << endl; // 3
64     return 0;
65 }
```

2.7. Segment Tree Summary

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e5+5;
12
13 struct Summary {
14     ll gcd;
15     ll cnt;
16     Summary(Summary left, Summary right){
17         gcd = __gcd(left.gcd, right.gcd);
18         cnt = (gcd==left.gcd?left.cnt:0) + (gcd==right.gcd?right.cnt:0);
19     }
20     Summary(ll gcd, ll cnt) : gcd(gcd), cnt(cnt) {}
21     Summary() : gcd(0), cnt(0) {} //elemento neutro
22 };
23
24 struct Node {
25     Summary value;
26     int start, end;
27     Node* left;
28     Node* right;
29     ll lazy;
30
31     Node(int start, int end) :
32         start(start), end(end), value(0,0), lazy(0), left(nullptr), right(
33         nullptr) {}
34 };
35
36 Node* build(vector<ll> &arr, int l, int r){
37     Node* node = new Node(l,r);
38     if(l == r) {
39         node->value = Summary(arr[l],1);
40         return node;
41     }
42
43     int m = (l + r) / 2;
44     node->left = build(arr, l, m);
45     node->right = build(arr, m+1, r);
46     node->value = Summary(node->left->value, node->right->value);
47     return node;
48 }
49
50 Summary query(Node* root, int start, int end) {
51     if(start > end) return Summary();
52     if(root->start == start && root->end == end)
53         return root->value;
54
55     int m = (root->start+root->end)/2;
56     Summary ans(query(root->left, start, min(m,end)), query(root->right, max(
```

```

        m+1,start),end));
56     return ans;
57 }
58
59 void solve() {
60     int n; cin >> n;
61     vector<ll> arr(n);
62     for(int i = 0; i < n; i++) {
63         cin >> arr[i];
64     }
65     Node* root = build(arr, 0, n-1);
66     int m; cin >> m;
67     while(m--) {
68         int l,r;
69         cin >> l >> r;
70         l--,r--;
71         Summary ans = query(root, l, r);
72         cout << (r - l + 1) - (ans.cnt) << endl;
73     }
74 }
75
76 int main() {
77     ios::sync_with_stdio(false);
78     cin.tie(nullptr);
79     int t = 1;
80     //cin >> t;
81     while(t--)
82         solve();
83 }

```

2.8. Persistent Segment Tree

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e5+5;
12
13 struct Node {
14     ll value;
15     int start, end;
16     Node* left;
17     Node* right;
18     ll lazy;
19
20     Node(int start, int end) :
21         start(start), end(end), value(0), lazy(0), left(nullptr), right(
22         nullptr) {}
23 };
24
25 ll aggregate_function(ll left, ll right){
26     return left + right;
27 }
28
29 Node* build(vector<ll> &arr, int l, int r){
30     Node* node = new Node(l,r);
31     if(l == r) {
32         node->value = arr[l];
33         return node;
34     }
35
36     int m = (l + r) / 2;
37     node->left = build(arr, l, m);
38     node->right = build(arr, m+1, r);
39     node->value = aggregate_function(node->left->value, node->right->value
40     );
41     return node;
42 }
43
44 void pushDown(Node* node) {
45     if(node->lazy != 0) {
46         if(node->left) {
47             int leftRange = node->left->end - node->left->start + 1;
48             node->left->value += node->lazy * leftRange;
49             node->left->lazy += node->lazy;
50         }
51         if(node->right) {
52             int rightRange = node->right->end - node->right->start + 1;
53             node->right->value += node->lazy * rightRange;
54             node->right->lazy += node->lazy;
55         }
56     }
57     node->lazy = 0;
58 }
```



```

55     }
56 }
57
58 ll query(Node* root, int start, int end) {
59     if(start > end) return 0;
60     if(root->start == start && root->end == end)
61         return root->value;
62
63     pushDown(root);
64     int m = (root->start+root->end)/2;
65     ll ans = aggregate_function(query(root->left, start, min(m, end)), query(
66         root->right, max(m+1, start), end));
67     return ans;
68 }
69
70 Node* persistent_update(Node* root, int pos, ll value) {
71     Node* node = new Node(root->start, root->end);
72     if(root->start == root->end) {
73         node->value = value;
74         return node;
75     }
76     int m = (root->start + root->end) / 2;
77     if(pos <= m) {
78         node->left = persistent_update(root->left, pos, value);
79         node->right = root->right;
80     } else {
81         node->right = persistent_update(root->right, pos, value);
82         node->left = root->left;
83     }
84     node->value = aggregate_function(node->left->value, node->right->value);
85     return node;
86 }
87
88 void update(Node* root, int pos, ll value) {
89     if(root->start == root->end) {
90         root->value = value;
91         return;
92     }
93     int m = (root->start + root->end) / 2;
94     if(pos <= m) {
95         update(root->left, pos, value);
96     } else {
97         update(root->right, pos, value);
98     }
99     root->value = aggregate_function(root->left->value, root->right->value);
100 }
101
102 void update_range(Node* node, int l, int r, ll delta) {
103     if(l > r) return;
104     if(l <= node->start && node->end <= r) {
105         int rangeLength = node->end - node->start + 1;
106         node->value += delta * rangeLength;
107         node->lazy += delta;
108         return;
109     }
110     pushDown(node);

```

```

110     int m = (node->start + node->end) / 2;
111     if(l <= m)
112         update_range(node->left, l, min(r, m), delta);
113     if(r > m)
114         update_range(node->right, max(l, m+1), r, delta);
115     node->value = aggregate_function(node->left->value, node->right->value
116 );
117 }
118 void solve() {
119     vector<ll> arr = {1,1,1,1,1,1};
120     int n = arr.size();
121     Node* root = build(arr,0,n-1);
122     cout << query(root,0,n-1) << endl;
123     Node* mod1 = persistent_update(root,0,2);
124     cout << query(mod1,0,n-1) << endl;
125     Node* mod2 = persistent_update(mod1,0,10);
126     cout << query(mod2,0,n-1) << endl;
127
128     cout << endl;
129     update_range(root, 0, 2, 1);
130     cout << query(root,0,n-1) << endl;
131 }
132
133 int main() {
134     ios::sync_with_stdio(false);
135     cin.tie(nullptr);
136     int t = 1;
137     //cin >> t;
138     while(t--)
139         solve();
140 }

```

2.9. Sparse Table

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e5+5;
12
13 int MAXL;
14 int n;
15
16 int arr[MAXN];
17 vector<vector<int>> st; // sparse table
18
19 void preprocess() {
20     MAXL = ceil(log2(n));
21
22     st.assign(n, vector<int>(MAXL + 1, INF));
23
24     for(int i = 0; i < n; i++)
25         st[i][0] = arr[i];
26
27     for(int j = 1; j <= MAXL; j++) {
28         for (int i = 0; i + (1 << j) - 1 < n; i++) {
29             st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
30         }
31     }
32 }
33
34
35 void solve() {
36     cin >> n;
37     for(int i = 0; i < n; i++) {
38         cin >> arr[i];
39     }
40
41     preprocess();
42
43     int q; cin >> q;
44     while(q--){
45         int l,r;
46         cin >> l >> r;
47         int k = floor(log2(r - l + 1));
48         cout << min(st[l][k], st[r - (1 << k) + 1][k]) << endl;
49     }
50 }
51
52 int main() {
53     ios::sync_with_stdio(false);
54     cin.tie(nullptr);
55     int t = 1;
56     //cin >> t;
```

```
57 while(t--)  
58     solve();  
59 }
```

Parte III

Geometry

3.1. Convex Hull

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct point {
6     double x, y;
7     point() { x = y = 0; }
8     point(double _x, double _y) : x(_x), y(_y) {}
9     bool operator == (point const& t) const {
10         return x == t.x && y == t.y;
11     }
12 };
13
14 int orientation(point a, point b, point c) {
15     double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
16     if (v < 0) return -1; // clockwise
17     if (v > 0) return +1; // counter-clockwise
18     return 0;
19 }
20
21 bool cw(point a, point b, point c, bool include_collinear) {
22     int o = orientation(a, b, c);
23     return o < 0 || (include_collinear && o == 0);
24 }
25
26 bool collinear(point a, point b, point c) { return orientation(a, b, c)
27     == 0; }
28
29 void convex_hull(vector<point>& a, bool include_collinear = false) {
30     point p0 = *min_element(a.begin(), a.end(), [](point a, point b) {
31         return make_pair(a.y, a.x) < make_pair(b.y, b.x);
32     });
33     sort(a.begin(), a.end(), [&p0](const point& a, const point& b) {
34         int o = orientation(p0, a, b);
35         if (o == 0)
36             return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
37                 < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
38         return o < 0;
39     });
40     if (include_collinear) {
41         int i = (int)a.size()-1;
42         while (i >= 0 && collinear(p0, a[i], a.back())) i--;
43         reverse(a.begin()+i+1, a.end());
44     }
45
46     vector<point> st;
47     for (int i = 0; i < (int)a.size(); i++) {
48         while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i],
49             include_collinear))
50             st.pop_back();
51         st.push_back(a[i]);
52     }
53
54     if (include_collinear == false && st.size() == 2 && st[0] == st[1])
55         st.pop_back();
56
57     a = st;
```

```

55 }
56
57 double area(const vector<point>& fig) {
58     double res = 0;
59     for (unsigned i = 0; i < fig.size(); i++) {
60         point p = i ? fig[i - 1] : fig.back();
61         point q = fig[i];
62         res += (p.x - q.x) * (p.y + q.y);
63     }
64     return fabs(res) / 2;
65 }
66
67
68 // example use case
69 void solve() {
70     double n,r;
71     cin >> n >> r;
72
73     vector<point> coords(n);
74     for(int i = 0; i < n; i++) {
75         cin >> coords[i].x >> coords[i].y;
76     }
77
78     auto dist = [](point &a, point&b) {
79         return sqrt(pow(a.x-b.x,2)+pow(a.y-b.y,2));
80     };
81
82     const double PI = 3.141516;
83     double cir = 2 * PI * r;
84     convex_hull(coords);
85
86     double per = 0;
87     coords.push_back(coords[0]);
88     for(int i = 1; i < coords.size(); i++) {
89         per += dist(coords[i-1],coords[i]);
90     }
91
92     int ans = round(per+cir);
93     cout << ans << endl;
94 }
95
96 int main() {
97     ios::sync_with_stdio(false);
98     cin.tie(0);
99     int t = 1;
100     //cin >> t;
101     while(t--)
102         solve();
103 }

```

Parte IV

Graphs

4.1. 2 Sat

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9 #define endl '\n'
10
11 const int INF = 1 << 30;
12 const int MAXN = 1e5+5;
13
14 struct TwoSatSolver {
15     int n_vars;
16     int n_vertices;
17     vector<vector<int>> adj, adj_t;
18     vector<bool> used;
19     vector<int> order, comp;
20     vector<bool> assignment;
21
22     TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 * n_vars),
23         adj(n_vertices), adj_t(n_vertices), used(n_vertices), order(), comp(
24             n_vertices, -1), assignment(n_vars) {
25         order.reserve(n_vertices);
26     }
27
28     void dfs1(int v) {
29         used[v] = true;
30         for (int u : adj[v]) {
31             if (!used[u])
32                 dfs1(u);
33         }
34         order.push_back(v);
35     }
36
37     void dfs2(int v, int cl) {
38         comp[v] = cl;
39         for (int u : adj_t[v]) {
40             if (comp[u] == -1)
41                 dfs2(u, cl);
42         }
43     }
44
45     bool solve_2SAT() {
46         order.clear();
47         used.assign(n_vertices, false);
48         for (int i = 0; i < n_vertices; ++i) {
49             if (!used[i])
50                 dfs1(i);
51         }
52
53         comp.assign(n_vertices, -1);
54         for (int i = 0, j = 0; i < n_vertices; ++i) {
55             int v = order[n_vertices - i - 1];
56             if (comp[v] == -1)
57                 dfs2(v, j++);
58         }
59     }
60 }
```

```

55     }
56
57     assignment.assign(n_vars, false);
58     for (int i = 0; i < n_vertices; i += 2) {
59         if (comp[i] == comp[i + 1])
60             return false;
61         assignment[i / 2] = comp[i] > comp[i + 1];
62     }
63     return true;
64 }
65
66 void add_disjunction(int a, bool na, int b, bool nb) {
67     // na and nb signify whether a and b are to be negated
68     a = 2 * a ^ na;
69     b = 2 * b ^ nb;
70     int neg_a = a ^ 1;
71     int neg_b = b ^ 1;
72     adj[neg_a].push_back(b);
73     adj[neg_b].push_back(a);
74     adj_t[b].push_back(neg_a);
75     adj_t[a].push_back(neg_b);
76 }
77
78 };
79
80 void solve() {
81     int n,m;
82     cin >> n >> m;
83
84     TwoSatSolver solver(m);
85     for(int i = 0; i < n; i++) {
86         int topping1, topping2;
87         char op1, op2;
88         cin >> op1 >> topping1 >> op2 >> topping2;
89         bool no_good1 = op1 == '-';
90         bool no_good2 = op2 == '-';
91         topping1--, topping2--;
92         solver.add_disjunction(topping1, no_good1, topping2, no_good2);
93     }
94
95     bool poss = solver.solve_2SAT();
96     if(!poss) {
97         cout << "IMPOSSIBLE" << endl;
98         return;
99     }
100
101     for(int i = 0; i < m; i++) {
102         if(solver.assignment[i]) {
103             cout << "+ ";
104         } else {
105             cout << "- ";
106         }
107     }
108 }
109
110 int main() {
111     ios::sync_with_stdio(false);
112     cin.tie(nullptr);

```

```
113  int t = 1;
114  //cin >> t;
115  while(t-->0)
116      solve();
117 }
```

4.2. Bellman Ford

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const ll INF = LONG_LONG_MAX;
11 const int MAXN = 1e5+5;
12
13 struct Edge {
14     int u, v;
15     ll weight;
16 };
17
18 int n,m;
19 vector<Edge> edges;
20
21 // Bellman-Ford algorithm to detect negative cycles and find shortest paths
22 bool bellman(int src, vector<ll> &dist) {
23     dist.assign(n + 1, INF);
24     dist[src] = 0;
25     for(int i = 2; i <= n; i++)
26     {
27         for (int j = 0; j < m; j++)
28         {
29             int u = edges[j].u;
30             int v = edges[j].v;
31             ll weight = edges[j].weight;
32             if (dist[u]!=INF && dist[u] + weight < dist[v])
33                 dist[v] = dist[u] + weight;
34         }
35     }
36     for (int i = 0; i < m; i++)
37     {
38         int u = edges[i].u;
39         int v = edges[i].v;
40         ll weight = edges[i].weight;
41         // True if neg-cycle exists
42         if (dist[u]!=INF && dist[u] + weight < dist[v])
43             return true;
44     }
45     return false;
46 }
47
48 int main() {
49     cin >> n >> m;
50     edges.resize(m);
51
52     for (int i = 0; i < m; i++) {
53         int a, b;
54         ll w;
55         cin >> a >> b >> w;
```

```

56     edges[i] = {a, b, w};
57 }
58
59 vector<ll> dist;
60
61 bool hasNegativeCycle = bellman(1, dist);
62
63 if (hasNegativeCycle) {
64     cout << "negative-weight cycle\n";
65 } else {
66     if (dist[n] == INF) {
67         cout << "no path from 1 to " << n << "\n";
68     } else {
69         cout << "shortest path from 1 to " << n << " is: " << dist[n] << "
70         \n";
71     }
72 }

```

4.3. Dijkstra

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e5+5;
12
13 int n,m;
14 vector<pair<int, int>> adj[MAXN];
15
16 void dijkstra(int s, vector<int> & d, vector<int> & p) {
17     d.assign(n+1, INF);
18     p.assign(n+1, -1);
19     vector<bool> visited(n+1,0);
20
21     d[s] = 0;
22     using pii = pair<int, int>;
23     priority_queue<pii, vector<pii>, greater<pii>> q;
24     q.push({0, s});
25     while (!q.empty()) {
26         int v = q.top().second;
27         q.pop();
28         if(visited[v]) continue;
29         visited[v] = 1;
30
31         for (auto edge : adj[v]) {
32             int to = edge.first;
33             int len = edge.second;
34
35             if (d[v] + len < d[to]) {
36                 d[to] = d[v] + len;
37                 p[to] = v;
38                 q.push({d[to], to});
39             }
40         }
41     }
42 }
43
44 vector<int> restore_path(int s, int t, vector<int> const& p) {
45     vector<int> path;
46
47     for (int v = t; v != s; v = p[v])
48         path.push_back(v);
49     path.push_back(s);
50
51     reverse(path.begin(), path.end());
52     return path;
53 }
54
55
56 int main()
```

```

57 {
58     cin >> n >> m;
59     // adj.resize(n+1);
60     while(m--) {
61         int a,b,w;
62         cin >> a >> b >> w;
63         adj[a].push_back({b,w});
64     }
65
66     int str=1, target=n;
67
68     vector<int> distance, path;
69
70     dijkstra(str, distance, path);
71     cout << distance[n] << '\n';
72 }

```

4.4. Floyd Warshall

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e3+5;
12
13 int n;
14 ll d[MAXN][MAXN];
15
16 void floyd_warshall() {
17     for (int k = 1; k <= n; k++)
18         for (int i = 1; i <= n; i++)
19             for (int j = 1; j <= n; j++)
20                 if (d[i][k] + d[k][j] < d[i][j])
21                     d[i][j] = d[i][k] + d[k][j];
22 }
23
24 void solve() {
25     int s, t;
26     cin >> n;
27     cin >> s >> t;
28     for(int i = 1; i <= n; i++) {
29         for(int j = 1; j <= n; j++) {
30             cin >> d[i][j];
31             if(d[i][j] == -1) d[i][j] = INF;
32         }
33     }
34
35     if(d[s][t] == INF)
36         cout << "-1\n";
37     else
38         cout << d[s][t] << '\n';
39 }
40
41 int main() {
42     ios::sync_with_stdio(false);
43     cin.tie(nullptr);
44     int t = 1;
45     //cin >> t;
46     while(t--)
47         solve();
48 }
```


4.5. Ford Fulkerson

```
1 // max flow algorithm
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 #define fst first
8 #define snd second
9 #define all(c) ((c).begin()), ((c).end())
10
11 const int INF = 1e9;
12
13 struct graph {
14     typedef long long flow_type;
15     struct edge {
16         int src, dst;
17         flow_type capacity, flow;
18         size_t rev;
19     };
20     int n;
21     vector<vector<edge>> adj;
22
23     graph(int n) : n(n), adj(n) { }
24
25     void add_edge(int src, int dst, flow_type capacity) {
26         adj[src].push_back({src, dst, capacity, 0, adj[dst].size()});
27         adj[dst].push_back({dst, src, 0, 0, adj[src].size() - 1});
28     }
29
30     int max_flow(int s, int t) {
31         vector<bool> visited(n);
32         function<flow_type(int, flow_type)> augment = [&](int u, flow_type
33         cur) {
34             if (u == t) return cur;
35             visited[u] = true;
36             for (auto &e : adj[u]) {
37                 if (!visited[e.dst] && e.capacity > e.flow) {
38                     flow_type f = augment(e.dst, min(e.capacity - e.flow, cur));
39                     if (f > 0) {
40                         e.flow += f;
41                         adj[e.dst][e.rev].flow -= f;
42                         return f;
43                     }
44                 }
45             }
46             return flow_type(0);
47         };
48         for (int u = 0; u < n; ++u)
49             for (auto &e : adj[u]) e.flow = 0;
50
51         flow_type flow = 0;
52         while (1) {
53             fill(all(visited), false);
54             flow_type f = augment(s, INF);
55             if (f == 0) break;
56             flow += f;
57         }
58         return flow;
59     }
60 }
```

```

56     }
57     return flow;
58 }
59
60 // test function to understand the code
61 // Function to print the residual graph
62 void print_residual_graph() {
63     cout << "Residual Graph:" << endl;
64     for (int u = 0; u < n; ++u) {
65         for (const auto &e : adj[u]) {
66             if (e.capacity > 0) { // Ignore reverse edges with 0 original
capacity
67                 cout << "Edge from " << e.src << " to " << e.dst
68                     << " | Capacity: " << e.capacity
69                     << " | Flow: " << e.flow
70                     << " | Residual Capacity: " << (e.capacity - e.flow) << endl
71             ;
72         }
73     }
74     cout << "-----" << endl;
75 }
76
77 void print_mincut_edges() {
78     // Step 1: Find reachable nodes from the source in the residual
graph
79     vector<bool> visited(n, false);
80     queue<int> q;
81     q.push(0); // Assuming 0 is the source node
82     visited[0] = true;
83
84     while (!q.empty()) {
85         int u = q.front();
86         q.pop();
87         for (const auto &e : adj[u]) {
88             if (!visited[e.dst] && e.capacity > e.flow) { // Residual
capacity > 0
89                 visited[e.dst] = true;
90                 q.push(e.dst);
91             }
92         }
93     }
94
95     // Step 2: Print edges that cross the cut
96     for (int u = 0; u < n; ++u) {
97         if (!visited[u]) continue;
98         for (const auto &e : adj[u]) {
99             if (!visited[e.dst] && e.capacity > 0) {
100                 cout << u + 1 << " " << e.dst + 1 << '\n'; // Output edge in
1-based index
101             }
102         }
103     }
104 }
105 }
106
107 };
108

```

```

109
110 int main() {
111     int n,m;
112     cin >> n >> m;
113     graph g(n);
114     for (int i = 0; i < m; ++i) {
115         int u, v, w;
116         cin >> u >> v >> w;
117         g.add_edge(u, v, w);
118     }
119     cout << g.max_flow(0,n-1) << endl;
120     g.print_residual_graph();
121 }

```

4.6. Functional Graph

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define endl '\n'
6
7 const int MAXN = 2e5+5;
8 const int LOG = 22;
9
10 int up[MAXN][LOG];
11 bool visited[MAXN];
12 bool on_stack[MAXN];
13 int next_node[MAXN];
14 int cycle[MAXN];
15 int cycle_length[MAXN];
16 int depth[MAXN];
17
18 int cycle_idx[MAXN];
19
20 void mark_cycle(int u, int id) {
21     cycle[u] = id;
22     depth[u] = 0;
23     cycle_length[id]++;
24     cycle_idx[u] = cycle_length[id];
25     if(!cycle[next_node[u]])
26         mark_cycle(next_node[u], id);
27 }
28
29 void dfs(int u) {
30     visited[u] = 1;
31     on_stack[u] = 1;
32
33     int v = next_node[u];
34     if(on_stack[v]) {
35         mark_cycle(v, v);
36     } else if(!visited[v]){
37         dfs(v);
38     }
39
40     if(!cycle[u])
41         depth[u] = depth[v] + 1;
42
43     on_stack[u] = 0;
44 }
45
46 int jumpto(int u, int k) {
47     int v = u;
48     for(int i = 0; i < LOG; i++) {
49         if((1 << i) & k) {
50             v = up[v][i];
51         }
52     }
53     return v;
54 }
55
56
```

```

57 int distcycle(int u, int v) {
58     if(cycle_idx[u] < cycle_idx[v]) {
59         return cycle_idx[v] - cycle_idx[u];
60     }
61     int turn = (cycle_length[cycle[u]] - cycle_idx[u] + 1) % cycle_length[
        cycle[u]];
62     return turn + cycle_idx[v] - 1;
63 }
64
65 void solve() {
66     int n,q;
67     cin >> n >> q;
68
69     for(int i = 1; i <= n; i++) {
70         cin >> next_node[i];
71         up[i][0] = next_node[i];
72     }
73
74     for(int j = 1; j < LOG; j++) {
75         for(int i = 1; i <= n; i++) {
76             up[i][j] = up[up[i][j-1]][j-1];
77         }
78     }
79
80     for(int i = 1; i <= n; i++) {
81         if(!visited[i]) dfs(i);
82     }
83
84     while(q--) {
85         int u,v;
86         cin >> u >> v;
87
88         int cyu = cycle[u];
89         int cyv = cycle[v];
90
91         if(cyu == 0) {
92             cyu = cycle[jumpto(u,depth[u])];
93         }
94         if(cyv == 0) {
95             cyv = cycle[jumpto(v,depth[v])];
96         }
97
98         int a = u;
99         int b = v;
100
101         if(cyu != cyv) {
102             cout << -1 << endl;
103             continue;
104         }
105
106         if(cycle[u] == 0 && cycle[v] == 0) {
107             if(depth[u] < depth[v]) {
108                 cout << -1 << endl;
109                 continue;
110             }
111             int diff = depth[u] - depth[v];
112             if(jumpto(u,diff) != v) {
113                 cout << -1 << endl;

```

```

114     } else {
115         cout << diff << endl;
116     }
117     continue;
118 }
119
120 if(cycle[u] && cycle[v]) {
121     cout << distcycle(u,v) << endl;
122     continue;
123 }
124
125 if(cycle[u] == 0) {
126     int steps = depth[u];
127     steps += distcycle(jumpsto(u,depth[u]), v);
128     cout << steps << endl;
129     continue;
130 }
131
132 if(cycle[v] == 0) {
133     cout << -1 <<endl;
134     continue;
135 }
136 }
137 }
138
139 int main() {
140     ios::sync_with_stdio(false);
141     cin.tie(nullptr);
142     int t = 1;
143     //cin >> t;
144     while(t--)
145         solve();
146 }

```

4.7. Hungarian

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9 #define ms(a, b) memset(a, b, sizeof(a))
10
11 const int INF = 1 << 30;
12 const int MAXN = 25;
13 const int MAXM = 25;
14
15 // hungarian weighted matching algo
16 // finds the MAX cost of MAX matching, to find mincost, add edges as
   // negatives
17 // Nodes are indexed from 1 on both sides
18 template<typename T>
19 struct KuhnMunkras { // n for left, m for right
20     int n, m, match[MAXM];
21     T g[MAXN][MAXM], lx[MAXN], ly[MAXM], slack[MAXM];
22     bool vx[MAXN], vy[MAXM];
23
24     void init(int n_, int m_) {
25         ms(g, 0); n = n_, m = m_;
26     }
27
28     void add(int u, int v, T w) {
29         g[u][v] = w;
30     }
31
32     bool find(int x) {
33         vx[x] = true;
34         for (int y = 1; y <= m; ++y) {
35             if (!vy[y]) {
36                 T delta = lx[x] + ly[y] - g[x][y];
37                 if (delta == 0) {
38                     vy[y] = true;
39                     if (match[y] == 0 || find(match[y])) {
40                         match[y] = x;
41                         return true;
42                     }
43                 } else slack[y] = min(slack[y], delta);
44             }
45         }
46         return false;
47     }
48
49     T matching() { // maximum weight matching
50         fill(lx + 1, lx + 1 + n, numeric_limits<T>::lowest());
51         ms(ly, 0);
52         ms(match, 0);
53         for (int i = 1; i <= n; ++i) {
54             for (int j = 1; j <= m; ++j) lx[i] = max(lx[i], g[i][j]);
55         }
```

```

56     for (int k = 1; k <= n; ++k) {
57         fill(slack + 1, slack + 1 + m, numeric_limits<T>::max());
58         while (true) {
59             ms(vx,0);
60             ms(vy,0);
61             if (find(k)) break;
62             else {
63                 T delta = numeric_limits<T>::max();
64                 for (int i = 1; i <= m; ++i) {
65                     if (!vy[i]) delta = min(delta, slack[i]);
66                 }
67                 for (int i = 1; i <= n; ++i) {
68                     if (vx[i]) lx[i] -= delta;
69                 }
70                 for (int i = 1; i <= m; ++i) {
71                     if (vy[i]) ly[i] += delta;
72                     if (!vy[i]) slack[i] -= delta;
73                 }
74             }
75         }
76     }
77     T result = 0;
78     for (int i = 1; i <= n; ++i) result += lx[i];
79     for (int i = 1; i <= m; ++i) result += ly[i];
80     return result;
81 }
82 };
83
84 void solve() {
85     int n, m;
86     cin >> n >> m;
87
88     KuhnMunkras<double> km;
89     km.init(n, m);
90
91     for (int i = 1; i <= n; ++i) {
92         for (int j = 1; j <= m; ++j) {
93             double c;
94             cin >> c;
95             km.add(i,j, -c);
96         }
97     }
98
99     double result = km.matching();
100    double total_cost = -result;
101    cout << total_cost << '\n';
102 }
103
104
105 int main() {
106     ios::sync_with_stdio(false);
107     cin.tie(nullptr);
108     int t = 1;
109     //cin >> t;
110     while(t--)
111         solve();
112 }

```


4.8. Kosaraju

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9 #define ms(a, b) memset(a, b, sizeof(a))
10
11 const int INF = 1 << 30;
12 const int MAXN = 25;
13 const int MAXM = 25;
14
15 vector<bool> visited; // keeps track of which vertices are already
    visited
16
17 // runs depth first search starting at vertex v.
18 // each visited vertex is appended to the output vector when dfs leaves
    it.
19 void dfs(int v, vector<vector<int>> const& adj, vector<int> &output) {
20     visited[v] = true;
21     for (auto u : adj[v])
22         if (!visited[u])
23             dfs(u, adj, output);
24     output.push_back(v);
25 }
26
27 // input: adj -- adjacency list of G
28 // output: components -- the strongly connected components in G
29 // output: adj_cond -- adjacency list of G^SCC (by root vertices)
30 void strongly_connected_components(vector<vector<int>> const& adj,
31     vector<vector<int>> &components,
32     vector<vector<int>> &adj_cond) {
33     int n = (int)adj.size();
34     components.clear();
35     adj_cond.clear();
36
37     // 1) Order the vertices by finish time in a first DFS.
38     vector<int> order;
39     visited.assign(n, false);
40     for (int i = 0; i < n; i++) {
41         if (!visited[i]) {
42             dfs(i, adj, order);
43         }
44     }
45
46     // 2) Reverse all edges to get the transpose graph.
47     vector<vector<int>> adj_rev(n);
48     for (int v = 0; v < n; v++) {
49         for (int u : adj[v]) {
50             adj_rev[u].push_back(v);
51         }
52     }
53
54     // 3) Do a second DFS in descending order of finish times (from step
```

```

1).
55 visited.assign(n, false);
56 reverse(order.begin(), order.end());
57
58 // this array will store the "root" (representative) of each SCC
59 vector<int> roots(n, -1);
60
61 for (auto v : order) {
62     if (!visited[v]) {
63         vector<int> component;
64         dfs(v, adj_rev, component);
65         // store the newly found component
66         components.push_back(component);
67         // the root is the minimum vertex in that component (arbitrary
        // choice)
68         int root = *min_element(begin(component), end(component));
69         for (auto u : component) {
70             roots[u] = root;
71         }
72     }
73 }
74
75 // 4) Build the condensation graph (SCC DAG).
76 adj_cond.assign(n, {});
77 for (int v = 0; v < n; v++) {
78     for (auto u : adj[v]) {
79         if (roots[v] != roots[u]) {
80             adj_cond[roots[v]].push_back(roots[u]);
81         }
82     }
83 }
84 }
85
86 void solve() {
87
88     int n = 5, m = 5;
89     vector<vector<int>> adj(n);
90
91     vector<pair<int,int>> edges = {
92         {0,1},
93         {1,2},
94         {2,0},
95         {1,3},
96         {3,4}
97     };
98
99     for (auto &e : edges) {
100         adj[e.first].push_back(e.second);
101     }
102
103     vector<vector<int>> components, cond_graph;
104     strongly_connected_components(adj, components, cond_graph);
105
106     for (auto &comp : components) {
107         sort(comp.begin(), comp.end());
108     }
109
110     cout << "Number of SCCs: " << (int)components.size() << "\n";

```

```

111 for (int i = 0; i < (int)components.size(); i++) {
112     cout << "SCC " << i << ": ";
113     for (auto &v : components[i]) {
114         cout << v << " ";
115     }
116     cout << "\n";
117 }
118
119 cout << "Condensation graph adjacency list:\n";
120 // We'll print only non-empty adjacency to see edges
121 for (int i = 0; i < (int)cond_graph.size(); i++) {
122     if (!cond_graph[i].empty()) {
123         cout << "root " << i << ": ";
124         for (auto &v : cond_graph[i]) {
125             cout << v << " ";
126         }
127         cout << "\n";
128     }
129 }
130 }
131
132 int main() {
133     ios::sync_with_stdio(false);
134     cin.tie(nullptr);
135     int t = 1;
136     //cin >> t;
137     while(t--)
138         solve();
139 }

```

4.9. Kruskal

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define ll long long
5 #define endl '\n'
6
7 struct Edge {
8     ll u, v, weight;
9     bool operator<(Edge const& other) const {
10         return weight < other.weight;
11     }
12 };
13
14 struct DSU {
15     vector<int> parent, size;
16
17     DSU(int n) {
18         parent.resize(n);
19         size.assign(n, 1);
20         iota(parent.begin(), parent.end(), 0);
21     }
22
23     int find(int v) {
24         if (v == parent[v]) return v;
25         return parent[v] = find(parent[v]);
26     }
27
28     bool unite(int a, int b) {
29         a = find(a);
30         b = find(b);
31         if (a == b) return false;
32         if (size[a] < size[b]) swap(a, b);
33         parent[b] = a;
34         size[a] += size[b];
35         return true;
36     }
37 };
38
39 ll kruskal(int n, vector<Edge>& edges, vector<Edge>& result) {
40     sort(edges.begin(), edges.end());
41     DSU dsu(n);
42     ll cost = 0;
43
44     for (auto& e : edges) {
45         if (dsu.unite(e.u, e.v)) {
46             cost += e.weight;
47             result.push_back(e);
48         }
49     }
50     return cost;
51 }
52
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56 }
```

```

57  int n, m;
58  cin >> n >> m;
59
60  vector<Edge> edges;
61  edges.reserve(m);
62
63  for (int i = 0; i < m; i++) {
64      ll u, v, w;
65      cin >> u >> v >> w;
66      u--, v--;
67      edges.push_back({u, v, w});
68  }
69
70  vector<Edge> result;
71  ll total_cost = kruskal(n, edges, result);
72
73  if(result.size() == n-1) {
74      cout << total_cost << endl;
75  } else {
76      cout << "IMPOSSIBLE" << endl;
77  }
78
79  // Optional: print MST edges
80  // for (auto &e : result)
81  //     cout << e.u + 1 << " " << e.v + 1 << " " << e.weight << endl;
82 }

```

4.10. Heavy Light Descomposition

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5
6 template <class T> class MaxSegmentTree {
7     private:
8         // const T DEFAULT = std::numeric_limits<T>().max();
9         const T DEFAULT = 0;
10        int len;
11        vector<T> segtree;
12    public:
13        MaxSegmentTree(int len) : len(len), segtree(len * 2, DEFAULT) {}
14        void set(int ind, T val) {
15            ind += len;
16            segtree[ind] = val;
17            for (; ind > 1; ind /= 2) {
18                segtree[ind / 2] = std::max(segtree[ind], segtree[ind ^ 1]);
19            }
20        }
21
22        T range_max(int start, int end) {
23            T max = DEFAULT;
24            for (start += len, end += len; start < end; start /= 2, end /= 2) {
25                if (start % 2 == 1) { max = std::max(max, segtree[start++]); }
26                if (end % 2 == 1) { max = std::max(max, segtree[--end]); }
27            }
28            return max;
29        }
30    };
31
32
33
34 template <class T, bool VALS_IN_EDGES> class HLD {
35     private:
36         int N, R, tim = 0; // n, root node, time
37         vector<vector<int>> adj;
38         vector<int> par, siz, depth, rt, pos; // parent, size, depth, root,
39                                           // position arrays
40         MaxSegmentTree<T> segtree; // Modify as needed
41         /** Compute the size of each subtree and set parent-child relationship
42          * Subtree of node v corresponds to segment [ pos[v], pos[v] + sz[v] )
43          */
44
45     void dfs_sz(int v) {
46         if (par[v] != -1) adj[v].erase(find(adj[v].begin(), adj[v].end(),
47             par[v]));
48         for (int &u : adj[v]) {
49             par[u] = v, depth[u] = depth[v] + 1;
50             dfs_sz(u);
51             siz[v] += siz[u];
52             if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
53         }
54     }
55
56     /** Assign positions for nodes
```

```

54     * Path from v to the last vertex in ascending heavy path
55     corresponds to [ pos[rt[v]], pos[v] ] */
56
57 void dfs_hld(int v) {
58     pos[v] = tim++;
59     for (int u : adj[v]) {
60         rt[u] = (u == adj[v][0] ? rt[v] : u);
61         dfs_hld(u);
62     }
63 }
64 /** process all heavy path and combine their results */
65 template <class B> void process(int u, int v, B op) {
66     for (; rt[u] != rt[v]; v = par[rt[v]]) {
67         if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
68         op(pos[rt[v]], pos[v]);
69     }
70
71     if (depth[u] > depth[v]) swap(u, v);
72     op(pos[u] + VALS_IN_EDGES, pos[v]);
73 }
74 public:
75 HLD(vector<vector<int>> adj_, int _R)
76     : N(adj_.size()), R(_R), adj(adj_), par(N, -1), siz(N, 1), depth(N),
77       rt(N), segtree(N) // modify as needed
78 {
79     rt[R] = R;
80     dfs_sz(R);
81     dfs_hld(R);
82 }
83
84 T query_path(int u, int v) {
85     T res = 0; // default value, modify depending on problem
86     process(u, v, [&](int l, int r) {
87         res = max(res, segtree.range_max(l, r + 1)); // modify depending
88         on problem
89     });
90     return res;
91 }
92 void modify_node(int u, T val) { segtree.set(pos[u], val); }
93 };
94
95 int main() {
96     ios_base::sync_with_stdio(false);
97     cin.tie(0);
98
99     int n, q;
100     cin >> n >> q;
101     vector<int> v(n);
102     vector<vector<int>> adj(n);
103     for (int i = 0; i < n; i++) { cin >> v[i]; }
104     for (int i = 0; i < n - 1; i++) {
105         int a, b;
106         cin >> a >> b;
107         --a, --b;
108         adj[a].push_back(b);
109         adj[b].push_back(a);

```

```

110 }
111
112 HLD<int, 0> H(adj, 0);
113 for (int i = 0; i < n; i++) { H.modify_node(i, v[i]); }
114 while (q--) {
115     int type, s, a, b, x;
116     cin >> type;
117     if (type == 1) {
118         cin >> s >> x;
119         --s;
120         H.modify_node(s, x);
121     } else if (type == 2) {
122         cin >> a >> b;
123         --a, --b;
124         cout << H.query_path(a, b) << " ";
125     }
126 }
127
128 }

```


4.11. LCA

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define endl '\n'
5 using ll = long long;
6
7 int n, l;
8 vector<vector<pair<int,int>>> adj;
9 int timer;
10 vector<int> tin, tout;
11 vector<vector<int>> up;
12 vector<ll> dist; // distance from root to node
13
14 void dfs(int v, int p)
15 {
16     tin[v] = ++timer;
17     up[v][0] = p;
18
19     for (int i = 1; i <= l; ++i)
20         up[v][i] = up[up[v][i-1]][i-1];
21
22     for (auto [u, w] : adj[v]) {
23         if (u != p) {
24             dist[u] = dist[v] + w;
25             dfs(u, v);
26         }
27     }
28
29     tout[v] = ++timer;
30 }
31
32 bool is_ancestor(int u, int v) {
33     return tin[u] <= tin[v] && tout[u] >= tout[v];
34 }
35
36 int lca(int u, int v)
37 {
38     if (is_ancestor(u, v)) return u;
39     if (is_ancestor(v, u)) return v;
40     for (int i = l; i >= 0; --i)
41         if (!is_ancestor(up[u][i], v))
42             u = up[u][i];
43     return up[u][0];
44 }
45
46 ll distance(int u, int v)
47 {
48     int ancestor = lca(u, v);
49     return dist[u] + dist[v] - 2 * dist[ancestor];
50 }
51
52 void preprocess(int root)
53 {
54     tin.resize(n);
55     tout.resize(n);
56     dist.assign(n, 0);
```

```

57     timer = 0;
58     l = ceil(log2(n));
59     up.assign(n, vector<int>(l + 1));
60     dfs(root, root);
61 }
62
63 int main() {
64     ios::sync_with_stdio(false);
65     cin.tie(nullptr);
66
67     cin >> n;
68     adj.assign(n, {});
69
70     for (int i = 0; i < n - 1; i++) {
71         int u, v, w;
72         cin >> u >> v >> w;
73         --u; --v;
74         adj[u].push_back({v, w});
75         adj[v].push_back({u, w});
76     }
77
78     preprocess(0); // root = 0
79
80     int q; cin >> q;
81     while (q--) {
82         int u, v;
83         cin >> u >> v;
84         --u; --v;
85         cout << distance(u, v) << endl;
86     }
87 }

```

4.12. Topological Sort

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define fst first
6 #define snd second
7 #define all(c) ((c).begin()), ((c).end())
8 #define ll long long
9
10 const int INF = 1 << 30;
11 const int MAXN = 1e5+5;
12
13 vector<int> adj[MAXN];
14 vector<int> topo;
15
16 int n,m;
17
18 void dfs(int u, vector<int> &visited) {
19     visited[u] = 1;
20     for (int &v : adj[u]) {
21         if (!visited[v])
22             dfs(v, visited);
23         else if(visited[v] == 1){
24             cout << "IMPOSSIBLE\n";
25             exit(0);
26         }
27     }
28     visited[u] = 2;
29     topo.push_back(u);
30 }
31
32 void topological_sort() {
33     topo.clear();
34     vector<int> visited(n+1,0);
35
36     for (int i = 1; i <= n; i++)
37         if (!visited[i])
38             dfs(i, visited);
39
40     reverse(topo.begin(), topo.end());
41 }
42
43 void solve() {
44     cin >> n >> m;
45     for(int i = 0; i <= n; i++) adj[i].clear();
46     while(m--) {
47         int a,b;
48         cin >> a >> b;
49         adj[a].push_back(b);
50     }
51     topological_sort();
52     for (int x : topo) cout << x << " ";
53     cout << "\n";
54 }
55
56 int main() {
```

```
57  int t = 1;
58  // cin >> t;
59  while(t-->0)
60      solve();
61 }
```

4.13. Centroid Descomposition

```
1 #include "bits/stdc++.h"
2 using namespace std;
3
4 using ll = long long;
5 using ld = long double;
6 #define sz(v) ((int)((v).size()))
7
8 const int MAXN = 100005;
9
10 int n;
11 vector<int> adj[MAXN];
12 int sub[MAXN];
13 bool removed[MAXN];
14 int parCentroid[MAXN];
15 vector<vector<pair<int,int>>> elements(MAXN);
16
17 vector<vector<pair<int,int>>> sufix(MAXN);
18 #define linea() cerr << "-----" << '\n'
19 int dfs_size(int u, int p) {
20     sub[u] = 1;
21     for (int v : adj[u]) {
22         if (v == p || removed[v]) continue;
23         sub[u] += dfs_size(v, u);
24     }
25     return sub[u];
26 }
27
28 int find_centroid(int u, int p, int total) {
29     for (int v : adj[u]) {
30         if (v == p || removed[v]) continue;
31         if (sub[v] > total / 2)
32             return find_centroid(v, u, total);
33     }
34     return u;
35 }
36 void bfs(int node) {
37     map<int, int> dist;
38     queue<int> q;
39     q.push(node);
40     elements[node].push_back({node, 0});
41     while (!q.empty()) {
42         auto l = q.front();
43         q.pop();
44         for (auto& x : adj[l]) {
45             if (dist[x] == 0 && !removed[x] && x != node) {
46                 dist[x] = dist[l] + 1;
47                 q.push(x);
48                 elements[node].push_back({x, dist[x]});
49             }
50         }
51     }
52
53     sort(elements[node].begin(), elements[node].end());
54
55     sufix[node].resize(sz(elements[node]) + 1);
56 }
```

```

57     pair<int, int> otro {1e9, 1e9};
58     for (int i = sz(elements[node]) - 1; i >= 0; i--) {
59         if (otro.second > elements[node][i].second) {
60             otro = elements[node][i];
61         }
62         else if (otro.second == elements[node][i].second) {
63             if (otro.first > elements[node][i].first) otro = elements[
node][i];
64         }
65         sufik[node][i] = otro;
66     }
67 }
68 void decompose(int u, int p = -1) {
69     int total = dfs_size(u, -1);
70     int c = find_centroid(u, -1, total);
71     bfs(c);
72     removed[c] = true;
73
74     parCentroid[c] = (p == -1 ? c : p); // root centroid points to
itself
75
76     // Example: hook custom logic here (before recursing)
77     // e.g., preprocess distances from c to other nodes in its component
78
79     for (int v : adj[c]) {
80         if (!removed[v])
81             decompose(v, c);
82     }
83 }
84 int bs(vector<pair<int, int>>& v, int target) {
85     int low = 0, high = sz(v) - 1;
86     int ans = -1;
87     while (low <= high) {
88         int mid = low + (high - low) / 2;
89         if (v[mid].first > target) {
90             ans = mid;
91             high = mid - 1;
92         }
93         else low = mid + 1;
94     }
95     return ans;
96 }
97 int dist(vector<pair<int, int>>& v, int target) {
98     int low = 0, high = sz(v) - 1;
99     int ans = -1;
100    while (low <= high) {
101        int mid = low + (high - low) / 2;
102        if (v[mid].first == target) {
103            return mid;
104        }
105        else if (v[mid].first < target) low = mid + 1;
106        else high = mid - 1;
107    }
108    return -1;
109 }
110 int32_t main() {
111     ios::sync_with_stdio(false);
112     cin.tie(nullptr);

```

```

113     cin >> n;
114     for (int i = 0; i < n - 1; i++) {
115         int u, v;
116         cin >> u >> v;
117         adj[u].push_back(v);
118         adj[v].push_back(u);
119     }
120
121     decompose(1);
122     for (int i = 1; i < n; i++) {
123         int aux = i;
124         int ori = aux;
125         pair<int, int> res = {1e9, 1e9};
126         bool ya = false;
127         do {
128             int it = bs(elements[aux], ori);
129             if (it == -1) {
130                 aux = parCentroid[aux];
131                 linea();
132                 continue;
133             }
134             pair<int, int> ans = sufix[aux][it];
135             int it2 = dist(elements[aux], ori);
136             ans.second += elements[aux][it2].second;
137             if (ans.second < res.second) {
138                 res = ans;
139             }
140             else if (ans.second == res.second) {
141                 if (ans.first < res.first) {
142                     res = ans;
143                 }
144             }
145             aux = parCentroid[aux];
146             linea();
147             if (ya) break;
148             if (parCentroid[aux] == aux) ya = true;
149         } while (true);
150         cout << res.first << ' ';
151     }
152     cout << n << '\n';
153
154     return 0;
155 }

```

Parte V

Math

5.1. FFT

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define ll long long
5 const double PI = acos(-1);
6
7 namespace FFT {
8     static vector<int> rev;
9     static vector<complex<double>> raices{{0,0},{1,0}};
10
11     void fft(vector<complex<double>>& a, bool invertido) {
12         int n = a.size();
13
14         if (rev.size() != n) {
15             int k = __builtin_ctz(n); // log2(n)
16             rev.assign(n, 0);
17             for (int i = 0; i < n; i++) {
18                 rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (k - 1));
19             }
20         }
21
22         for (int i = 0; i < n; i++) {
23             if (i < rev[i]) swap(a[i], a[rev[i]]);
24         }
25
26         if (raices.size() < n) {
27             int k = __builtin_ctz(raices.size());
28             raices.resize(n);
29             while ((1 << k) < n) {
30                 double angulo = 2 * PI / (1 << (k + 1));
31                 for (int i = 1 << (k - 1); i < (1 << k); i++) {
32                     raices[2 * i] = raices[i];
33                     double ang = angulo * (2 * i + 1 - (1 << k));
34                     raices[2 * i + 1] = complex<double>(cos(ang), sin(
35 ang));
36                 }
37                 k++;
38             }
39
40             for (int len = 1; len < n; len = len << 1) {
41                 for (int i = 0; i < n; i += 2 * len) {
42                     for (int j = 0; j < len; j++) {
43                         complex<double> u = a[i + j];
44                         complex<double> v = a[i + j + len] * raices[len + j
45 ];
46
47                         a[i + j] = u + v;
48                         a[i + j + len] = u - v;
49                     }
50                 }
51             }
52
53             if (invertido) {
54                 reverse(a.begin() + 1, a.end());
55                 for (auto &x : a) x /= n;
56             }
57         }
58     }
59 }
```

```

55     }
56 } // namespace FFT
57
58 vector<ll> multiplicacion_polinomio(const vector<ll>& A, const vector<ll>
    & B) {
59     int n1 = A.size(), n2 = B.size();
60     if (n1 == 0 || n2 == 0) return vector<ll>();
61
62     int n = 1;
63     while (n < n1 + n2 - 1) n = n << 1;
64     vector<complex<double>> fa(n), fb(n);
65     for (int i = 0; i < n1; i++) fa[i] = complex<double>(A[i],0);
66     for (int i = 0; i < n2; i++) fb[i] = complex<double>(B[i],0);
67
68     FFT::fft(fa, false);
69     FFT::fft(fb, false);
70     for (int i = 0; i < n; i++) fa[i] *= fb[i];
71     FFT::fft(fa, true);
72
73     vector<ll> res(n1 + n2 - 1);
74     for (int i = 0; i < res.size(); i++) {
75         res[i] = llround(fa[i].real()); // Redondeo
76     }
77     return res;
78 }
79
80
81 int main() {
82     vector<ll> A = {1, 2, 3}; // 1 + 2x + 3x^2
83     vector<ll> B = {2, 0, 1}; // 2 + 0x + 1x^2
84     vector<ll> C = multiplicacion_polinomio(A, B);
85     return 0;
86 }

```

5.2. NTT

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define ll long long
5
6 namespace NTT {
7     // MOD debe ser un primo de la forma  $c * 2^k + 1$ 
8     //  $998244353 = 119 * 2^{23} + 1$ . Ra z primitiva  $G = 3$ .
9     const ll MOD = 998244353;
10    const ll G = 3; // Ra z primitiva para MOD
11
12    ll fast_pow(ll a, ll b) {
13        ll res = 1;
14        a %= MOD;
15        while (b > 0) {
16            if (b & 1) res = (res * a) % MOD;
17            a = (a * a) % MOD;
18            b >>= 1;
19        }
20        return res;
21    }
22
23    ll inv(ll a) {
24        return fast_pow(a, MOD - 2);
25    }
26
27    static vector<int> rev;
28    static vector<ll> w;
29
30    void precompute(int n) {
31        if (rev.size() == n) return;
32        int k = __builtin_ctz(n);
33        rev.assign(n, 0);
34        for (int i = 0; i < n; i++) {
35            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (k - 1));
36        }
37
38        w.resize(n);
39        w[0] = 1;
40        ll g_n = fast_pow(G, (MOD - 1) / n);
41        for (int i = 1; i < n; i++) {
42            w[i] = (w[i - 1] * g_n) % MOD;
43        }
44    }
45
46    void fft(vector<ll>& a, bool invertido) {
47        int n = a.size();
48        precompute(n);
49
50        for (int i = 0; i < n; i++) {
51            if (i < rev[i]) swap(a[i], a[rev[i]]);
52        }
53
54        for (int len = 1; len < n; len <= 1) {
55            int step = n / (2 * len);
56            for (int i = 0; i < n; i += 2 * len) {
```

```

57         for (int j = 0; j < len; j++) {
58             ll u = a[i + j];
59             ll v = (a[i + j + len] * w[j * step]) % MOD;
60
61             a[i + j] = (u + v) % MOD;
62             a[i + j + len] = (u - v + MOD) % MOD;
63         }
64     }
65 }
66
67     if (invertido) {
68         reverse(a.begin() + 1, a.end());
69         ll n_inv = inv(n);
70         for (auto &x : a) {
71             x = (x * n_inv) % MOD;
72         }
73     }
74 }
75 } // namespace NTT
76
77 vector<ll> multiplicacion_polinomio_mod(const vector<ll>& A, const
78     vector<ll>& B) {
79     int n1 = A.size(), n2 = B.size();
80     if (n1 == 0 || n2 == 0) return vector<ll>();
81
82     int n = 1;
83     while (n < n1 + n2 - 1) n = n << 1;
84
85     vector<ll> fa(n), fb(n);
86     for (int i = 0; i < n1; i++) fa[i] = (A[i] % NTT::MOD + NTT::MOD) %
NTT::MOD;
87     for (int i = 0; i < n2; i++) fb[i] = (B[i] % NTT::MOD + NTT::MOD) %
NTT::MOD;
88
89     NTT::fft(fa, false);
90     NTT::fft(fb, false);
91
92     for (int i = 0; i < n; i++) {
93         fa[i] = (fa[i] * fb[i]) % NTT::MOD;
94     }
95
96     NTT::fft(fa, true);
97
98     fa.resize(n1 + n2 - 1);
99     return fa;
100 }
101
102 int main() {
103     vector<ll> A = {1, 2, 3}; // 1 + 2x + 3x^2
104     vector<ll> B = {2, 0, 1}; // 2 + 0x + 1x^2
105     vector<ll> C = multiplicacion_polinomio_mod(A, B);
106     return 0;
107 }

```

5.3. Binary Exponentiation

```
1 #include <bits/stdc++.h>
2
3 #define ll long long
4
5 ll binpow(ll a, ll b) {
6     if (b == 0)
7         return 1;
8     ll res = binpow(a, b / 2);
9     if (b % 2)
10         return res * res * a;
11     else
12         return res * res;
13 }
14
15 ll binpowIt(ll a, ll b) {
16     ll res = 1;
17     while (b > 0) {
18         if (b & 1)
19             res = res * a;
20         a = a * a;
21         b >>= 1;
22     }
23     return res;
24 }
25
26 int power(ll x, ll y, ll M)
27 {
28     if (y == 0)
29         return 1;
30
31     int p = power(x, y / 2, M) % M;
32     p = (p * p) % M;
33
34     return (y % 2 == 0) ? p : (x * p) % M;
35 }
36 // inverse modular power(A, M - 2, M)
```

5.4. Matrix Exponentiation

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define ll long long
6 const int MOD = 1e9+7;
7
8 struct Matrix {
9     int n;
10    vector<vector<ll>> mat;
11
12    Matrix(int size) : n(size), mat(size, vector<ll>(size)) {}
13
14    // Identity matrix
15    static Matrix identity(int size) {
16        Matrix I(size);
17        for (int i = 0; i < size; i++)
18            I.mat[i][i] = 1;
19        return I;
20    }
21
22    // Matrix multiplication
23    Matrix operator*(const Matrix& other) const {
24        Matrix product(n);
25        for (int i = 0; i < n; ++i)
26            for (int k = 0; k < n; ++k)
27                for (int j = 0; j < n; ++j) {
28                    product.mat[i][j] += mat[i][k] * other.mat[k][j];
29                    product.mat[i][j] %= MOD;
30                }
31        return product;
32    }
33
34    // Matrix exponentiation
35    Matrix pow(ll power) const {
36        Matrix result = identity(n);
37        Matrix base = *this;
38
39        while (power > 0) {
40            if (power & 1)
41                result = result * base;
42            base = base * base;
43            power >>= 1;
44        }
45        return result;
46    }
47 };
48
49 ll fibonacci(int n) {
50     if (n == 0) return 0;
51     Matrix base(2);
52     base.mat = {{1, 1}, {1, 0}};
53     base = base.pow(n - 1);
54     return base.mat[0][0]; // F(n)
55 }
56
```

```
57 int main() {  
58     for(int i = 0; i < 25; i++)  
59         cout << fibonacci(i) << " ";  
60 }
```

Parte VI

Strings

6.1. KMP Automaton

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define ll long long
6 #define endl '\n'
7
8 vector<int> prefix_function(string s) {
9     int n = (int)s.length();
10    vector<int> pi(n);
11    for (int i = 1; i < n; i++) {
12        int j = pi[i-1];
13        while (j > 0 && s[i] != s[j])
14            j = pi[j-1];
15        if (s[i] == s[j])
16            j++;
17        pi[i] = j;
18    }
19    return pi;
20 }
21
22 void compute_automaton(string s, vector<vector<int>>& aut) {
23     s += '#';
24     int n = s.size();
25     vector<int> pi = prefix_function(s);
26     aut.assign(n, vector<int>(26));
27     for (int i = 0; i < n; i++) {
28         for (int c = 0; c < 26; c++) {
29             if (i > 0 && 'a' + c != s[i])
30                 aut[i][c] = aut[pi[i-1]][c];
31             else
32                 aut[i][c] = i + ('a' + c == s[i]);
33         }
34     }
35 }
36
37 const int MAXN = 1e4+5;
38 const int MAXK = 1e3+5;
39 const int MOD = 1e9+7;
40
41 int n,k;
42 string pattern;
43 vector<vector<int>> aut;
44
45 long long binpow(long long a, long long b)
46 {
47     long long result = 1;
48     while (b) {
49         if (b & 1)
50             result = (result * a) % MOD;
51         a = (a * a) % MOD;
52         b >>= 1;
53     }
54     return result;
55 }
56 }
```

```

57
58 ll dp[MAXN][MAXK];
59 ll fdp(int pos, int state) {
60     bool found = state == pattern.size();
61     if(pos == n) {
62         return found;
63     }
64
65     ll &val = dp[pos][state];
66     if(val != -1) return val;
67
68     val = 0;
69
70     if(found) {
71         ll l = n-pos;
72         return val = binpow(k, l);
73     }
74
75     for(int i = 0; i < k; i++) {
76         val = (val + fdp(pos+1, aut[state][i])) % MOD;
77     }
78
79     return val;
80 }
81
82 void solve() {
83     memset(dp, -1, sizeof dp);
84     cin >> n >> k >> pattern;
85
86     compute_automaton(pattern, aut);
87
88     cout << fdp(0,0) << endl;
89 }
90
91 int main() {
92     solve();
93 }

```

6.2. Hashing

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define ll long long
6
7 const int MAXN = 1e5+5;
8 const int BASE = 31;
9 const int M = 1e9 + 7;
10
11 int pot[MAXN];
12
13 void compute_powers(int n) {
14     int p_pow = 1;
15     for (int i = 0; i < n; i++) {
16         pot[i] = p_pow;
17         p_pow = (p_pow * BASE) % M;
18     }
19 }
20
21 vector<ll> compute_hash(string const& s) {
22     int n = s.size();
23     vector<ll> hash_values(n + 5);
24     hash_values[0] = 0;
25     for (int i = 1; i <= n; i++) {
26         hash_values[i] = (hash_values[i - 1] * BASE + s[i - 1]) % M;
27     }
28     return hash_values;
29 }
30
31 ll get_hash(vector<ll> &hash_values, int a, int b) {
32     return (hash_values[b] - (hash_values[a - 1] * pot[b - a + 1]) % M +
33     M) % M;
34 }
35
36 int main() {
37     string word = "ALLEY";
38     compute_powers(word.size());
39     vector<ll> hash = compute_hash(word);
40
41     for (int i = 1; i <= word.size(); i++) {
42         cout << hash[i] << ' ';
43     }
44     cout << endl;
45     for (int i = 1; i <= word.size(); i++) {
46         cout << word[i - 1] << ": " << get_hash(hash, i, i) << endl;
47     }
48
49     return 0;
50 }
```