La tasita

ICPC Notebook

Team: Turistas

Índice general

I Dynamic Programming 3

II Data Structures 15

III Geometry 36

IV Graphs 39

Parte I

Dynamic Programming

1.1. Longest Common Subsequence

```cpp
// https://leetcode.com/problems/longest-common-subsequence/submissions/1815539365/
#include <bits/stdc++.h>
using namespace std;

string s1, s2;
const int n = 1000;
int dp[n][n];

int LCS(int i, int j) {
    if (i < 0 || j < 0) return 0;
    if (dp[i][j] != -1) return dp[i][j];

    if (s1[i] == s2[j]) dp[i][j] = 1 + LCS(i - 1, j - 1);
    else dp[i][j] = max(LCS(i-1,j), LCS(i,j-1));

    return dp[i][j];
}

string LCSReconstruccion() {
    int i = s1.size() - 1;
    int j = s2.size() - 1;
    string ans = "";

    while (i >= 0 && j >= 0) {
        if (s1[i] == s2[j]) {
            ans += s1[i];
            i--, j--;
        }
        else {
            if (i > 0 && (j == 0 || dp[i - 1][j] >= dp[i][j - 1])) i--;
            else j--;
        }
    }

    reverse(ans.begin(), ans.end());
    return ans;
}

int main() {
    memset(dp,-1,sizeof(dp));
    s1 = "abcde";
    s2 = "ace";

    cout << LCS(s1.size() - 1, s2.size() - 1) << endl;
    cout << LCSReconstruccion() << endl;
}
```

1.2. Longest Increasing Subsequence

```
// https://leetcode.com/problems/longest-increasing-subsequence/
    description/
#include <bits/stdc++.h>
using namespace std;

const int max_n = 100000;
int dp[max_n + 1];

int LIS(vector<int> arr) {
    int n = arr.size();
    memset(dp,0,sizeof(dp));

    int res = 0;

    for (int i = 0; i < n; i++) {
        dp[i] = 1;
        int valAc = arr[i];
        for (int j = 0; j < i; j++) {
            int valAn = arr[j];
            if (valAn < valAc) dp[i] = max(dp[i], dp[j] + 1);
        }
        res = max(res, dp[i]);
    }

    return res;
}

vector<int> LISReconstruccion(vector<int> arr) {
    int n = arr.size();
    vector<int> padre(n, -1);

    int res = 0;
    int indiceMejorSecuencia = -1;

    for (int i = 0; i < n; i++) {
        dp[i] = 1;
        int valAc = arr[i];
        for (int j = 0; j < i; j++) {
            int valAn = arr[j];
            if (valAn < valAc && dp[j] + 1 > dp[i]) {
                dp[i] = dp[j] + 1;
                padre[i] = j;
            }
        }

        if (dp[i] > res) {
            res = dp[i];
            indiceMejorSecuencia = i;
        }

    }

    vector<int> secuencia;
    for (int i = indiceMejorSecuencia; i != -1; i = padre[i]) {
        secuencia.push_back(arr[i]);
    }
```

```cpp
    reverse(secuencia.begin(), secuencia.end());

    return secuencia;
}

int main() {
    vector<int> nuevo = {1,3,6,7,9,4,10,5,6};
  cout << LIS(nuevo) << endl;

    vector<int> secuencia = LISReconstruccion(nuevo);
    for (int& val : secuencia) cout << val << " ";
}
```

1.3. Longest Increasing Subsequence Fast

```
// https://leetcode.com/problems/longest-increasing-subsequence/description/
#include <bits/stdc++.h>
using namespace std;

int LISFast(vector<int> arr) {
    int n = arr.size();
    vector<int> res;
    res.push_back(arr[0]);

    for (int i = 1; i < n; i++) {
        if (arr[i] > res.back()) {
            res.push_back(arr[i]);
        } else {
            auto it = lower_bound(res.begin(),res.end(),arr[i]);
            int busca = it - res.begin();
            res[busca] = arr[i];
        }
    }

    return res.size();
}

vector<int> LISFastReconstruccion(vector<int> arr) {
    int n = arr.size();
    vector<int> res;
    vector<int> resIndex;
    vector<int> padre(n, - 1);

    for (int i = 0; i < n; i++) {
        int x = arr[i];
        auto it = lower_bound(res.begin(), res.end(), x);
        int pos = it - res.begin();

        if (it == res.end()) {
            res.push_back(x);
            resIndex.push_back(i);
        } else {
            *it = x;
            resIndex[pos] = i;
        }

        if (pos > 0)
            padre[i] = resIndex[pos - 1];
    }

    vector<int> lis;
    int idx = resIndex.back();
    while (idx != -1) {
        lis.push_back(arr[idx]);
        idx = padre[idx];
    }
    reverse(lis.begin(), lis.end());
    return lis;
}

```

```
int main() {
    vector<int> nuevo = {1,3,6,7,9,4,10,5,6};
  cout << LISFast(nuevo) << endl;

    vector<int> reconstruccion = LISFastReconstruccion(nuevo);
    for (int& val : reconstruccion) cout << val << " ";
}
```

1.4. Maximum Subarray

```
// https://leetcode.com/problems/maximum-subarray/submissions
    /1815546613/
#include <bits/stdc++.h>
using namespace std;

const int max_n = 100001;
int dp[max_n];

int maximumSub(vector<int>& arr) {
    memset(dp,0,sizeof(dp));
    int n = arr.size();

    for (int i = 1; i <= n; i++) {
        int& valor = arr[i - 1];
        dp[i] = max(valor, dp[i - 1] + valor);
    }

    int maximo = -1000000000;
    for (int i = 1; i <= n; i++) {
        maximo = max(maximo, dp[i]);
    }
    return maximo;
}

int main() {
    vector<int> valores = {-2,1,-3,4,-1,2,1,-5,4};
    cout << maximumSub(valores);
}
```

1.5. Knapsack

```cpp
// Source: https://atcoder.jp/contests/dp/tasks/dp_d
#include <bits/stdc++.h>
using namespace std;

const int MAX_W = 100001;
long long dp[MAX_W];
int W = MAX_W;

long long knapsack(const vector<pair<int, long long>>& items) {
    memset(dp, -1, sizeof(dp));
    int n = items.size();

    dp[0] = 0;
    for (int i = 0; i < n; i++) {
        int w = items[i].first;
        long long v = items[i].second;

        for (int j = W; j >= w; j--) {
            if (dp[j - w] != -1) {
                dp[j] = max(dp[j], dp[j - w] + v);
            }
        }
    }

    long long max_v = 0;
    for (int j = 0; j <= W; j++) {
        max_v = max(max_v, dp[j]);
    }
    return max_v;
}

int main() {
  vector<pair<int, long long>> items = {{2,3},{3,5},{6,5}};
    cout << knapsack(items) << endl;
    return 0;
}
```

1.6. Knapsack Binary

```cpp
// https://cses.fi/problemset/task/1159/
#include <bits/stdc++.h>
using namespace std;

const int MAX_W = 100001;
int dp[MAX_W];
int W;

int knapsack(const vector<pair<int, int>>& items) {
    memset(dp, -1, sizeof(dp));
    int n = items.size();
    dp[0] = 0;

    for (int i = 0; i < n; i++) {
        int w = items[i].first;
        int v = items[i].second;

        for (int j = W; j >= w; j--) {
            if (dp[j - w] != -1) {
                dp[j] = max(dp[j], dp[j - w] + v);
            }
        }
    }

    int max_v = 0;
    for (int j = 0; j <= W; j++) {
        max_v = max(max_v, dp[j]);
    }
    return max_v;
}

void res(const vector<int>& pesos, const vector<int>& valores, const vector<int>& copias) {
    int n = pesos.size();
    vector<pair<int, int>> items;
    for (int i = 0; i < n; i++) {
        int k = copias[i];
        int b = 1;
        while (k > 0) {
            int num = min(k, b);
            items.push_back({pesos[i] * num, valores[i] * num});
            k -= num;
            b *= 2;
        }
    }
    cout << knapsack(items) << endl;
}

int main() {
    vector<int> pesos = {2, 6, 3};
    vector<int> valores = {8, 5, 4};
    vector<int> copias = {3, 5, 2};
    res(pesos, valores, copias);
}
```

1.7. Knapsack Re

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAX_N = 1001;
const int MAX_W = 1001;

int N, W;
long long dp[MAX_N][MAX_W];
vector<pair<int, long long>> items;

long long knapsack(int i, int w_rem) {
    if (i == N || w_rem == 0) {
        return 0;
    }
    if (dp[i][w_rem] != -1) {
        return dp[i][w_rem];
    }

    int w = items[i].first;
    long long v = items[i].second;

    long long res = knapsack(i + 1, w_rem);
    if (w_rem >= w) {
        res = max(res, v + knapsack(i + 1, w_rem - w));
    }

    return dp[i][w_rem] = res;
}

vector<pair<int, long long>> knapsackReconstruccion() {
    vector<pair<int, long long>> tomados;
    int w_rem = W;

    for (int i = 0; i < N; i++) {
        int w = items[i].first;
        long long v = items[i].second;

        if (knapsack(i + 1, w_rem) == dp[i][w_rem]) {
            continue;
        }

        else if (w_rem >= w && (v + knapsack(i + 1, w_rem - w) == dp[i][w_rem])) {
            tomados.push_back(items[i]);
            w_rem -= w;
        }
    }
    return tomados;
}


int main() {
    memset(dp, -1, sizeof(dp));
    items = {{3, 30}, {4, 50}, {5, 60}};
    W = 8;
    N = items.size();
    cout << knapsack(0, W) << endl;

    vector<pair<int, long long>> res = knapsackReconstruccion();
    cout << "Items tomados:" << endl;
    for (auto& item : res) {
        cout << "Peso: " << item.first << ", Valor: " << item.second << endl;
    }
}
```

1.8. Knapsack Subset Sum

```cpp
// https://leetcode.com/problems/tallest-billboard/description/
#include <bits/stdc++.h>
using namespace std;

const int MAX_N = 2501;
int diferencia[2][MAX_N];

int knapsackDiferencia(const vector<int>& arr) {
    int n = arr.size();
    memset(diferencia,-1,sizeof(diferencia));
    diferencia[0][0] = 0;

    for (int i = 0; i < n; i++) {
        int ahora = arr[i];
        int actual = i & 1;
        int siguiente = actual ^ 1;

        fill(diferencia[siguiente], diferencia[siguiente] + MAX_N, -1);

        for (int valor = 0; valor < MAX_N; valor++) {
            if (diferencia[actual][valor] == -1) continue;

            diferencia[siguiente][valor] = max(
                diferencia[siguiente][valor],
                diferencia[actual][valor]
            );

            int nuevaDiferenciaMayor = valor + ahora;
            if (nuevaDiferenciaMayor < MAX_N) {
                diferencia[siguiente][nuevaDiferenciaMayor] = max(
                    diferencia[siguiente][nuevaDiferenciaMayor],
                    diferencia[actual][valor]
                );
            }

            int nuevaDiferenciaMenor = abs(valor - ahora);
            int nuevaAltura = diferencia[actual][valor] + min(valor, ahora);
            diferencia[siguiente][nuevaDiferenciaMenor] = max(
                diferencia[siguiente][nuevaDiferenciaMenor],
                nuevaAltura
            );
        }
    }

    return diferencia[n & 1][0];
}

int main() {
    vector<int> roads = {1,2,3,6};
    cout << knapsackDiferencia(roads) << endl;
}
```

Parte II

Data Structures

2.1. Fenwick Tree

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e5+5;

struct FenwickTree {
  vector<ll> bit;
  ll n;

  FenwickTree(ll n) {
    this->n = n;
    bit.assign(n, 0);
  }

  FenwickTree(vector<ll> const &a) : FenwickTree(a.size()) {
    for (size_t i = 0; i < a.size(); i++)
      add(i, a[i]);
  }

  ll sum(ll r) {
    ll ret = 0;
    for (; r >= 0; r = (r & (r + 1)) - 1)
      ret += bit[r];
    return ret;
  }

  ll sum(ll l, ll r) {
    return sum(r) - sum(l - 1);
  }

  void add(ll idx, ll delta) {
    for (; idx < n; idx = idx | (idx + 1))
      bit[idx] += delta;
  }
};


void solve() {
  int n; cin >> n;
  vector<ll> arr(n);
  for (int i = 0; i < n; i++) {
    cin >> arr[i];
  }
  FenwickTree ft(arr);

  int m;
  cin >> m;
  for (int i = 1; i <= m; i++) {
    int q,l,r;
    cin >> q >> l >> r;
    if (q == 's') {
      cout << ft.sum(l, r) << " ";
    }
    else {
      int delta = r - arr[l];
      arr[l] = r;
      ft.add(l, delta);
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

2.2. Fenwick Tree 2d

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e3+5;

int n, m;
int qn;
char q[10];
int f[MAXN][MAXN];

void update(int x, int y, int delta) {
  for (int i = x; i <= n; i = i | (i + 1))
    for (int j = y; j <= m; j = j | (j + 1))
      f[i][j] += delta;
}

int getSum(int x, int y) {
  int res = 0;
  for (int i = x; i > 0; i = (i & (i + 1)) - 1)
    for (int j = y; j > 0; j = (j & (j + 1)) - 1)
      res += f[i][j];
  return res;
}

int getSum(int xFrom, int xTo, int yFrom, int yTo) {
  return getSum(xTo, yTo) - getSum(xTo, yFrom - 1) - getSum(xFrom - 1, yTo) + getSum(xFrom - 1, yFrom - 1);
}

void solve() {
  cin >> n >> qn;
  m = n;

  for (int i = 1; i <= qn; i++) {
    cin >> q;
    if (q[0] == 'A') {
      int x, y;
      cin >> x >> y;
      update(x, y, 1);
    }
    else {
      int xFrom, xTo, yFrom, yTo;
      cin >> xFrom >> yFrom >> xTo >> yTo;
      if (xFrom > xTo)
        swap(xFrom, xTo);
      if (yFrom > yTo)
        swap(yFrom, yTo);
      cout << getSum(xFrom, xTo, yFrom, yTo);
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

2.3. Treap

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long
#define endl '\n'

const int INF = 1 << 30;
const int MAXN = 1e5+5;

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

int random(int l, int r) {
  return uniform_int_distribution<int>(l, r)(rng);
}

struct node {
  int key;
  int priority;
  node* left;
  node* right;
  int cnt;
  node(int v) : key(v) {
    priority = random(0,1e9);
    left = right = nullptr;
    cnt = 1;
  }
};

int cnt(node* t) {
  return t ? t->cnt : 0;
}

void update(node *t) {
  if (t) t->cnt = 1 + cnt(t->left) + cnt(t->right);
}

node* merge(node* a, node* b) {
  if(a == nullptr) return b;
  if(b == nullptr) return a;
  if(a->priority > b->priority) {
    a->right = merge(a->right, b);
    update(a);
    return a;
  } else {
    b->left = merge(a, b->left);
    update(b);
    return b;
  }
}

pair<node*, node*> split(node *T, int k) {
  if(T == nullptr) return {nullptr,nullptr};
  if(T->key <= k) {
    auto p = split(T->right,k);
    T->right = p.fst;
    update(T);
    return {T, p.snd};
  } else {
    auto p = split(T->left, k);
    T->left = p.snd;
    update(T);
    return {p.fst, T};
  }
}

node* insert(node* T, int x) {
  auto p = split(T,x);
  T = merge(p.fst, new node(x));
  T = merge(T, p.snd);
  return T;
}

node* remove(node* T, int x) {
  auto p = split(T, x);        // p.fst:     x, p.snd: > x
  auto p2 = split(p.fst, x-1); // p2.fst: < x, p2.snd: == x

  // p2.snd contains the node(s) with key = x     we discard it
  return merge(p2.fst, p.snd);
}

bool find(node* T, int x) {
  if (!T) return false;
  if (T->key == x) return true;
  if (x < T->key) return find(T->left, x);
  return find(T->right, x);
}

node* kth(node* t, int k) {
  int left_size = cnt(t->left);
  if (k < left_size) return kth(t->left, k);
  else if (k == left_size) return t;
  else return kth(t->right, k - left_size - 1);
}

void inorder(node* T) {
  if (!T) return;
  inorder(T->left);
  cout << T->key << " ";
  inorder(T->right);
}

void solve() {
  node* root = nullptr;

  root = insert(root, 5);
  root = insert(root, 2);
  root = insert(root, 8);
  root = insert(root, 1);
  root = insert(root, 10);

  cout << "Treap inorder (sorted): ";
  inorder(root); // 1 2 5 8 10
  cout << endl;

  cout << "Find 8? " << (find(root, 8) ? "Yes" : "No") << endl;
  cout << "Find 7? " << (find(root, 7) ? "Yes" : "No") << endl;

  root = remove(root, 5);
  cout << "After removing 5: ";
  inorder(root); // 1 2 8 10
  cout << endl;
}


int main() {
  // ios::sync_with_stdio(false);
  // cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

2.4. Implicit Treap

```cpp
#include <bits/stdc++.h>
using namespace std;

#define endl '\n';

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

int random(int l, int r) {
  return uniform_int_distribution<int>(l, r)(rng);
}

struct node {
  int value;
  int priority;
  node* left;
  node* right;
  int cnt;

  bool rev;
  node(int v) : value(v) {
    priority = random(0, 1e9);
    left = right = nullptr;
    cnt = 1;  // FIX: initialize cnt
  }
};

int cnt(node* t) {
  return t ? t->cnt : 0;
}

// push for reverse or lazy propagation
void push(node* t) {
   if(!t || !t->rev) return;
   t->rev = 0;
   swap(t->left, t->right);
   if(t->left) t->left->rev ^= 1;
   if(t->right) t->right->rev ^= 1;
 }

void update(node* t) {
  if (!t) return;
  t->cnt = 1 + cnt(t->left) + cnt(t->right);
}

node* merge(node* a, node* b) {
  if (!a) return b;
  if (!b) return a;
  if (a->priority > b->priority) {
    push(a);
    a->right = merge(a->right, b);
    update(a);
    return a;
  } else {
    push(b);
    b->left = merge(a, b->left);
    update(b);
    return b;
  }
}

// 0-index
pair<node*, node*> split(node* T, int k, int add = 0) {
  if (!T) return {nullptr, nullptr};
  push(T);
  int cur_key = add + cnt(T->left);
  if (k <= cur_key) {
    auto p = split(T->left, k, add);
    T->left = p.second;
    update(T);
    return {p.first, T};
  } else {
    auto p = split(T->right, k, add + 1 + cnt(T->left));
    T->right = p.first;
    update(T);
    return {T, p.second};
  }
}

node* insert(node* T, int pos, int value) {
  auto p = split(T, pos);
  T = merge(p.first, new node(value));
  T = merge(T, p.second);
  return T;
}

node* remove(node* T, int pos) {
  auto p1 = split(T, pos);        // [0..pos-1], [pos..end]
  auto p2 = split(p1.second, 1);  // [pos], [pos+1..end]
                                  // delete p2.first; // optional, free memory
  return merge(p1.first, p2.second);
}

void inorder(node* T) {
  if (!T) return;
  inorder(T->left);
  cout << T->value << " ";
  inorder(T->right);
}

node* kth(node* t, int k) {
  push(t);
  int left_size = cnt(t->left);
  if (k < left_size) return kth(t->left, k);
  else if (k == left_size) return t;
  else return kth(t->right, k - left_size - 1);
}

node* update_range(node* t, int l, int r, int delta) {
  auto p = split(t, r + 1);
  auto p2 = split(p.first, l);

  if (p2.second) {
    p2.second->lazy += delta;  // mark range for update
    push(p2.second);           // optionally apply now (or can defer)
    update(p2.second);         // recompute cnt/sum
  }

  return merge(merge(p2.first, p2.second), p.second);
}

void solve() {
  node* root = nullptr;

  root = insert(root, 0, 10);
  root = insert(root, 1, 20);
  root = insert(root, 2, 30);
  cout << "Initial array: ";
  inorder(root); cout << "\n";

  root = insert(root, 1, 15);
  cout << "After inserting 15 at pos 1: ";
  inorder(root); cout << endl;

  root = remove(root, 2);
  cout << "After removing element at pos 2: ";
  inorder(root); cout << endl;
}

int main() {
  solve();
}
```

2.5. Ordered Set

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<
T,
  null_type,
  less<T>,                 // use less_equal<T> if you want to allow duplicates (see note below)
  rb_tree_tag,
  tree_order_statistics_node_update>;

// --- Example usage ---
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  ordered_set<int> s;

  s.insert(10);
  s.insert(20);
  s.insert(30);
  s.insert(40);

  // 1        Get element by index (0-based)
  cout << "Element at index 2: " << *s.find_by_order(2) << "\n"; // -> 30

  // 2        Get index of an element
  cout << "Index of 30: " << s.order_of_key(30) << "\n"; // -> 2

  // 3        Lower bound behavior
  cout << "Index where 25 would be inserted: " << s.order_of_key(25) << "\n"; // -> 2

  // 4        Iterate through all elements
  for (auto x : s) cout << x << " ";
  cout << "\n";

  // 5        Check if an element exists
  if (s.find(20) != s.end())
    cout << "20 exists in set\n";
}
```

2.6. Segment Tree

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e5+5;

int n, t[4*MAXN];

void build(vector<int> a, int v, int tl, int tr) {
  if (tl == tr) {
    t[v] = a[tl];
  } else {
    int tm = tl + (tr - tl) / 2;
    build(a, v*2, tl, tm);
    build(a, v*2+1, tm+1, tr);
    t[v] = t[v*2] + t[v*2+1];
  }
}

void update(int v, int tl, int tr, int pos, int new_val) {
  if (tl == tr) {
    t[v] = new_val;
  } else {
    int tm = tl + (tr - tl) / 2;
    if (pos <= tm)
      update(v*2, tl, tm, pos, new_val);
    else
      update(v*2+1, tm+1, tr, pos, new_val);
    t[v] = t[v*2] + t[v*2+1];
  }
}

int sum(int v, int tl, int tr, int l, int r) {
  if (l > r)
    return 0;
  if (l == tl && r == tr) {
    return t[v];
  }
  int tm = (tl + tr) / 2;
  return sum(v*2, tl, tm, l, min(r, tm))
    + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}


int main(){
  vector<int> arr = {1,1,1,1,1};
  int n = arr.size();
  build(arr, 1, 0, n-1);
  cout << sum(1,0,n-1,0,4) << endl; // 5
  cout << sum(1,0,n-1,0,1) << endl; // 2
  cout << sum(1,0,n-1,1,4) << endl; // 4

  update(1, 0, n-1, 4, 0);

  cout << sum(1,0,n-1,0,4) << endl; // 4
  cout << sum(1,0,n-1,0,1) << endl; // 2
  cout << sum(1,0,n-1,1,4) << endl; // 3
  return 0;
}
```

2.7. Segment Tree Summary

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e5+5;

struct Summary {
  ll gcd;
  ll cnt;
  Summary(Summary left, Summary right){
    gcd = __gcd(left.gcd,right.gcd);
    cnt = (gcd==left.gcd?left.cnt:0) + (gcd==right.gcd?right.cnt:0);
  }
  Summary(ll gcd, ll cnt) : gcd(gcd), cnt(cnt) {}
  Summary() : gcd(0), cnt(0) {} //elemento neutro
};

struct Node {
  Summary value;
  int start, end;
  Node* left;
  Node* right;
  ll lazy;

  Node(int start, int end) :
    start(start), end(end), value(0,0), lazy(0), left(nullptr), right(nullptr) {}
};

Node* build(vector<ll> &arr, int l, int r){
  Node* node = new Node(l,r);
  if(l == r) {
    node->value = Summary(arr[l],1);
    return node;
  }

  int m = (l + r) / 2;
  node->left = build(arr, l, m);
  node->right = build(arr, m+1, r);
  node->value = Summary(node->left->value, node->right->value);
  return node;
}

Summary query(Node* root, int start, int end) {
  if(start > end) return Summary();
  if(root->start == start && root->end == end)
    return root->value;

  int m = (root->start+root->end)/2;
  Summary ans(query(root->left,start,min(m,end)), query(root->right,max(
   m+1,start),end));
  return ans;
}

void solve() {
  int n; cin >> n;
  vector<ll> arr(n);
  for(int i = 0; i < n; i++) {
    cin >> arr[i];
  }
  Node* root = build(arr, 0, n-1);
  int m; cin >> m;
  while(m--) {
    int l,r;
    cin >> l >> r;
    l--,r--;
    Summary ans = query(root, l, r);
    cout << (r - l + 1) - (ans.cnt)  << endl;
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

2.8. Persistent Segment Tree

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e5+5;

struct Node {
  ll value;
  int start, end;
  Node* left;
  Node* right;
  ll lazy;

  Node(int start, int end) :
    start(start), end(end), value(0), lazy(0), left(nullptr), right(nullptr) {}
};

ll aggregate_function(ll left, ll right){
  return left + right;
}

Node* build(vector<ll> &arr, int l, int r){
  Node* node = new Node(l,r);
  if(l == r) {
    node->value = arr[l];
    return node;
  }

  int m = (l + r) / 2;
  node->left = build(arr, l, m);
  node->right = build(arr, m+1, r);
  node->value = aggregate_function(node->left->value, node->right->value);
  return node;
}

void pushDown(Node* node) {
  if(node->lazy != 0) {
    if(node->left) {
      int leftRange = node->left->end - node->left->start + 1;
      node->left->value += node->lazy * leftRange;
      node->left->lazy += node->lazy;
    }
    if(node->right) {
      int rightRange = node->right->end - node->right->start + 1;
      node->right->value += node->lazy * rightRange;
      node->right->lazy += node->lazy;
    }
    node->lazy = 0;
  }
}

ll query(Node* root, int start, int end) {
  if(start > end) return 0;
  if(root->start == start && root->end == end)
    return root->value;

  pushDown(root);
  int m = (root->start+root->end)/2;
  ll ans = aggregate_function(query(root->left,start,min(m,end)), query(root->right,max(m+1,start),end));
  return ans;
}

Node* persistent_update(Node* root, int pos, ll value) {
  Node* node = new Node(root->start, root->end);
  if(root->start == root->end) {
    node->value = value;
    return node;
  }
  int m = (root->start + root->end) / 2;
  if(pos <= m) {
    node->left = persistent_update(root->left, pos, value);
    node->right = root->right;
  } else {
    node->right = persistent_update(root->right, pos, value);
    node->left = root->left;
  }
  node->value = aggregate_function(node->left->value, node->right->value);
  return node;
}

void update(Node* root, int pos, ll value) {
  if(root->start == root->end) {
    root->value = value;
    return;
  }
  int m = (root->start + root->end) / 2;
  if(pos <= m) {
    update(root->left, pos, value);
  } else {
    update(root->right, pos, value);
  }
  root->value = aggregate_function(root->left->value, root->right->value);
}

void update_range(Node* node, int l, int r, ll delta) {
  if(l > r) return;
  if(l <= node->start && node->end <= r) {
    int rangeLength = node->end - node->start + 1;
    node->value += delta * rangeLength;
    node->lazy += delta;
    return;
  }
  pushDown(node);
  int m = (node->start + node->end) / 2;
  if(l <= m)
    update_range(node->left, l, min(r, m), delta);
  if(r > m)
    update_range(node->right, max(l, m+1), r, delta);
  node->value = aggregate_function(node->left->value, node->right->value);
}

void solve() {
  vector<ll> arr = {1,1,1,1,1,1};
  int n = arr.size();
  Node* root = build(arr,0,n-1);
  cout << query(root,0,n-1) << endl;
  Node* mod1 = persistent_update(root,0,2);
  cout << query(mod1,0,n-1) << endl;
  Node* mod2 = persistent_update(mod1,0,10);
  cout << query(mod2,0,n-1) << endl;

  cout << endl;
  update_range(root, 0, 2, 1);
  cout << query(root,0,n-1) << endl;
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

2.9. Sparse Table

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e5+5;

int MAXL;
int n;

int arr[MAXN];
vector<vector<int>> st; // sparse table

void preprocess() {
  MAXL = ceil(log2(n));

  st.assign(n, vector<int>(MAXL + 1, INF));

  for(int i = 0; i < n; i++)
    st[i][0] = arr[i];

  for(int j = 1; j <= MAXL; j++) {
    for (int i = 0; i + (1 << j) - 1 < n; i++) {
      st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
  }
}


void solve() {
  cin >> n;
  for(int i = 0; i < n; i++) {
    cin >> arr[i];
  }

  preprocess();

  int q; cin >> q;
  while(q--){
    int l,r;
    cin >> l >> r;
    int k = floor(log2(r - l + 1));
    cout << min(st[l][k], st[r - (1 << k) + 1][k]) << endl;
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

Parte III

Geometry

3.1. Convex Hull

```cpp
#include <bits/stdc++.h>

using namespace std;

struct point {
  double x, y;
  point() { x = y = 0; }
  point(double _x, double _y) : x(_x), y(_y) {}
  bool operator == (point const& t) const {
    return x == t.x && y == t.y;
  }
};

int orientation(point a, point b, point c) {
  double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
  if (v < 0) return -1; // clockwise
  if (v > 0) return +1; // counter-clockwise
  return 0;
}

bool cw(point a, point b, point c, bool include_collinear) {
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}
bool collinear(point a, point b, point c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<point>& a, bool include_collinear = false) {
  point p0 = *min_element(a.begin(), a.end(), [](point a, point b) {
      return make_pair(a.y, a.x) < make_pair(b.y, b.x);
      });
  sort(a.begin(), a.end(), [&p0](const point& a, const point& b) {
      int o = orientation(p0, a, b);
      if (o == 0)
      return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
      < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
      return o < 0;
      });
  if (include_collinear) {
    int i = (int)a.size()-1;
    while (i >= 0 && collinear(p0, a[i], a.back())) i--;
    reverse(a.begin()+i+1, a.end());
  }

  vector<point> st;
  for (int i = 0; i < (int)a.size(); i++) {
    while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
      st.pop_back();
    st.push_back(a[i]);
  }

  if (include_collinear == false && st.size() == 2 && st[0] == st[1])
    st.pop_back();

  a = st;
}

double area(const vector<point>& fig) {
  double res = 0;
  for (unsigned i = 0; i < fig.size(); i++) {
    point p = i ? fig[i - 1] : fig.back();
    point q = fig[i];
    res += (p.x - q.x) * (p.y + q.y);
  }
  return fabs(res) / 2;
}


// example use case
void solve() {
  double n,r;
  cin >> n >> r;

  vector<point> coords(n);
  for(int i = 0; i < n; i++) {
    cin >> coords[i].x >> coords[i].y;
  }

  auto dist = [](point &a, point&b) {
    return sqrt(pow(a.x-b.x,2)+pow(a.y-b.y,2));
  };

  const double PI = 3.141516;
  double cir = 2 * PI * r;
  convex_hull(coords);

  double per = 0;
  coords.push_back(coords[0]);
  for(int i = 1; i < coords.size(); i++) {
    per +=  dist(coords[i-1],coords[i]);
  }

  int ans = round(per+cir);
  cout << ans << endl;
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

Parte IV

Graphs

4.1. 2 Sat

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long
#define endl '\n'

const int INF = 1 << 30;
const int MAXN = 1e5+5;

struct TwoSatSolver {
  int n_vars;
  int n_vertices;
  vector<vector<int>> adj, adj_t;
  vector<bool> used;
  vector<int> order, comp;
  vector<bool> assignment;

  TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 * n_vars), adj(n_vertices), adj_t(n_vertices), used(n_vertices), order(), comp(n_vertices, -1), assignment(n_vars) {
    order.reserve(n_vertices);
  }
  void dfs1(int v) {
    used[v] = true;
    for (int u : adj[v]) {
      if (!used[u])
        dfs1(u);
    }
    order.push_back(v);
  }

  void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : adj_t[v]) {
      if (comp[u] == -1)
        dfs2(u, cl);
    }
  }

  bool solve_2SAT() {
    order.clear();
    used.assign(n_vertices, false);
    for (int i = 0; i < n_vertices; ++i) {
      if (!used[i])
        dfs1(i);
    }

    comp.assign(n_vertices, -1);
    for (int i = 0, j = 0; i < n_vertices; ++i) {
      int v = order[n_vertices - i - 1];
      if (comp[v] == -1)
        dfs2(v, j++);
    }

    assignment.assign(n_vars, false);
    for (int i = 0; i < n_vertices; i += 2) {
      if (comp[i] == comp[i + 1])
        return false;
      assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
  }

  void add_disjunction(int a, bool na, int b, bool nb) {
    // na and nb signify whether a and b are to be negated
    a = 2 * a ^ na;
    b = 2 * b ^ nb;
    int neg_a = a ^ 1;
    int neg_b = b ^ 1;
    adj[neg_a].push_back(b);
    adj[neg_b].push_back(a);
    adj_t[b].push_back(neg_a);
    adj_t[a].push_back(neg_b);
  }

};

void solve() {
  int n,m;
  cin >> n >> m;

  TwoSatSolver solver(m);
  for(int i = 0; i < n; i++) {
    int topping1, topping2;
    char op1, op2;
    cin >> op1 >> topping1 >> op2 >> topping2;
    bool no_good1 = op1 == '-';
    bool no_good2 = op2 == '-';
    topping1--,topping2--;
    solver.add_disjunction(topping1,no_good1, topping2, no_good2);
  }

  bool poss = solver.solve_2SAT();
  if(!poss) {
    cout << "IMPOSSIBLE" << endl;
    return;
  }

  for(int i = 0; i < m; i++) {
    if(solver.assignment[i]) {
      cout << "+ ";
    } else {
      cout << "- ";
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

4.2. Bellman Ford

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const ll INF = LONG_LONG_MAX;
const int MAXN = 1e5+5;

struct Edge {
  int u, v;
  ll weight;
};

int n,m;
vector<Edge> edges;

// Bellman-Ford algorithm to detect negative cycles and find shortest paths
bool bellman(int src, vector<ll> &dist) {
  dist.assign(n + 1, INF);
  dist[src] = 0;
  for(int i = 2; i <= n; i++)
  {
    for (int j = 0; j < m; j++)
    {
      int u = edges[j].u;
      int v = edges[j].v;
      ll weight = edges[j].weight;
      if (dist[u]!=INF && dist[u] + weight < dist[v])
        dist[v] = dist[u] + weight;
    }
  }
  for (int i = 0; i < m; i++)
  {
    int u = edges[i].u;
    int v = edges[i].v;
    ll weight = edges[i].weight;
    // True if neg-cylce exists
    if (dist[u]!=INF && dist[u] + weight < dist[v])
      return true;
  }
  return false;
}

int main() {
  cin >> n >> m;
  edges.resize(m);

  for (int i = 0; i < m; i++) {
    int a, b;
    ll w;
    cin >> a >> b >> w;
    edges[i] = {a, b, w};
  }

  vector<ll> dist;

  bool hasNegativeCycle = bellman(1, dist);

  if (hasNegativeCycle) {
    cout << "negative-weight cycle\n";
  } else {
    if (dist[n] == INF) {
      cout << "no path from 1 to " << n << "\n";
    } else {
      cout << "shortest path from 1 to " << n << " is: " << dist[n] << "\n";
    }
  }
}
```

4.3. Dijkstra

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e5+5;

int n,m;
vector<pair<int, int>> adj[MAXN];

void dijkstra(int s, vector<int> & d, vector<int> & p) {
  d.assign(n+1, INF);
  p.assign(n+1, -1);
  vector<bool> visited(n+1,0);

  d[s] = 0;
  using pii = pair<int, int>;
  priority_queue<pii, vector<pii>, greater<pii>> q;
  q.push({0, s});
  while (!q.empty()) {
    int v = q.top().second;
    q.pop();
    if(visited[v]) continue;
    visited[v] = 1;

    for (auto edge : adj[v]) {
      int to = edge.first;
      int len = edge.second;

      if (d[v] + len < d[to]) {
        d[to] = d[v] + len;
        p[to] = v;
        q.push({d[to], to});
      }
    }
  }
}

vector<int> restore_path(int s, int t, vector<int> const& p) {
  vector<int> path;

  for (int v = t; v != s; v = p[v])
    path.push_back(v);
  path.push_back(s);

  reverse(path.begin(), path.end());
  return path;
}


int main()
```

```
{
  cin >> n >> m;
  // adj.resize(n+1);
  while(m--) {
    int a,b,w;
    cin >> a >> b >> w;
    adj[a].push_back({b,w});
  }

  int str=1, target=n;

  vector<int> distance, path;

  dijkstra(str, distance,path);
  cout << distance[n] << '\n';
}
```

4.4. Floyd Warshall

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e3+5;

int n;
ll d[MAXN][MAXN];

void floyd_warshall() {
  for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
        if (d[i][k] + d[k][j] < d[i][j])
          d[i][j] = d[i][k] + d[k][j];
}

void solve() {
  int s, t;
  cin >> n;
  cin >> s >> t;
  for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
      cin >> d[i][j];
      if(d[i][j] == -1) d[i][j] = INF;
    }
  }

  if(d[s][t] == INF)
    cout << "-1\n";
  else
   cout << d[s][t] << '\n';
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

4.5. Ford Fulkerson

```cpp
// max flow algorithm

#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())

const int INF = 1e9;

struct graph {
  typedef long long flow_type;
  struct edge {
    int src, dst;
    flow_type capacity, flow;
    size_t rev;
  };
  int n;
  vector<vector<edge>> adj;

  graph(int n) : n(n), adj(n) { }

  void add_edge(int src, int dst, flow_type capacity) {
    adj[src].push_back({src, dst, capacity, 0, adj[dst].size()});
    adj[dst].push_back({dst, src, 0, 0, adj[src].size() - 1});
  }

  int max_flow(int s, int t) {
    vector<bool> visited(n);
    function<flow_type(int, flow_type)> augment = [&](int u, flow_type cur) {
      if (u == t) return cur;
      visited[u] = true;
      for (auto &e : adj[u]) {
        if (!visited[e.dst] && e.capacity > e.flow) {
          flow_type f = augment(e.dst, min(e.capacity - e.flow, cur));
          if (f > 0) {
            e.flow += f;
            adj[e.dst][e.rev].flow -= f;
            return f;
          }
        }
      }
      return flow_type(0);
    };
    for (int u = 0; u < n; ++u)
      for (auto &e : adj[u]) e.flow = 0;

    flow_type flow = 0;
    while (1) {
      fill(all(visited), false);
      flow_type f = augment(s, INF);
      if (f == 0) break;
      flow += f;
    }
    return flow;
  }

  // test function to understand the code
  // Function to print the residual graph
  void print_residual_graph() {
    cout << "Residual Graph:" << endl;
    for (int u = 0; u < n; ++u) {
      for (const auto &e : adj[u]) {
        if (e.capacity > 0) {  // Ignore reverse edges with 0 original capacity
          cout << "Edge from " << e.src << " to " << e.dst
            << " | Capacity: " << e.capacity
            << " | Flow: " << e.flow
            << " | Residual Capacity: " << (e.capacity - e.flow) << endl;
        }
      }
    }
    cout << "------------------------" << endl;
  }

  void print_mincut_edges() {
    // Step 1: Find reachable nodes from the source in the residual graph
    vector<bool> visited(n, false);
    queue<int> q;
    q.push(0);  // Assuming 0 is the source node
    visited[0] = true;

    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (const auto &e : adj[u]) {
        if (!visited[e.dst] && e.capacity > e.flow) {  // Residual capacity > 0
          visited[e.dst] = true;
          q.push(e.dst);
        }
      }
    }

    // Step 2: Print edges that cross the cut
    for (int u = 0; u < n; ++u) {
      if (!visited[u]) continue;
      for (const auto &e : adj[u]) {
        if (!visited[e.dst] && e.capacity > 0) {
          cout << u + 1 << " " << e.dst + 1 << '\n';  // Output edge in 1-based index
        }
      }

    }
  }


};
```

```cpp

int main() {
  int n,m;
  cin >> n >> m;
  graph g(n);
  for (int i = 0; i < m; ++i) {
    int u, v, w;
    cin >> u >> v >> w;
    g.add_edge(u, v, w);
  }
  cout << g.max_flow(0,n-1) << endl;
  g.print_residual_graph();
}
```

4.6. Functional Graph

```cpp
#include <bits/stdc++.h>

using namespace std;

#define endl ’\n’

const int MAXN = 2e5+5;
const int LOG = 22;

int up[MAXN][LOG];
bool visited[MAXN];
bool on_stack[MAXN];
int next_node[MAXN];
int cycle[MAXN];
int cycle_length[MAXN];
int depth[MAXN];

int cycle_idx[MAXN];

void mark_cycle(int u, int id) {
  cycle[u] = id;
  depth[u] = 0;
  cycle_length[id]++;
  cycle_idx[u] = cycle_length[id];
  if(!cycle[next_node[u]])
    mark_cycle(next_node[u], id);
}

void dfs(int u) {
  visited[u] = 1;
  on_stack[u] = 1;

  int v = next_node[u];
  if(on_stack[v]) {
    mark_cycle(v, v);
  } else if(!visited[v]){
    dfs(v);
  }

  if(!cycle[u])
    depth[u] = depth[v] + 1;

  on_stack[u] = 0;
}

int jumpto(int u, int k) {
  int v = u;
  for(int i = 0; i < LOG; i++) {
    if((1 << i) & k) {
      v = up[v][i];
    }
  }
  return v;
}


```

```
int distcycle(int u, int v) {
  if(cycle_idx[u] < cycle_idx[v]) {
    return cycle_idx[v] - cycle_idx[u];
  }
  int turn = (cycle_length[cycle[u]] - cycle_idx[u] + 1) % cycle_length[cycle[u]];
  return turn + cycle_idx[v] - 1;
}

void solve() {
  int n,q;
  cin >> n >> q;

  for(int i = 1; i <= n; i++) {
    cin >> next_node[i];
    up[i][0] = next_node[i];
  }

  for(int j = 1; j < LOG; j++) {
    for(int i = 1; i <= n; i++) {
      up[i][j] = up[up[i][j-1]][j-1];
    }
  }

  for(int i = 1; i <= n; i++) {
    if(!visited[i]) dfs(i);
  }

  while(q--) {
    int u,v;
    cin >> u >> v;

    int cyu = cycle[u];
    int cyv = cycle[v];

    if(cyu == 0) {
      cyu = cycle[jumpto(u,depth[u])];
    }
    if(cyv == 0) {
      cyv = cycle[jumpto(v,depth[v])];
    }

    int a = u;
    int b = v;

    if(cyu != cyv) {
      cout << -1 << endl;
      continue;
    }

    if(cycle[u] == 0 && cycle[v] == 0) {
      if(depth[u] < depth[v]) {
        cout << -1 << endl;
        continue;
      }
      int diff = depth[u] - depth[v];
      if(jumpto(u,diff) != v) {
        cout << -1 << endl;
      } else {
        cout << diff << endl;
      }
      continue;
    }

    if(cycle[u] && cycle[v]) {
      cout << distcycle(u,v) << endl;
      continue;
    }

    if(cycle[u] == 0) {
      int steps = depth[u];
      steps += distcycle(jumpto(u,depth[u]), v);
      cout << steps << endl;
      continue;
    }

    if(cycle[v] == 0) {
      cout << -1 <<endl;
      continue;
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

4.7. Hungarian

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long
#define ms(a, b) memset(a, b, sizeof(a))

const int INF = 1 << 30;
const int MAXN = 25;
const int MAXM = 25;

// hungarian weighted matching algo
// finds the MAX cost of MAX matching, to find mincost, add edges as negatives
// Nodes are indexed from 1 on both sides
template<typename T>
struct KuhnMunkras { // n for left, m for right
  int n, m, match[MAXM];
  T g[MAXN][MAXM], lx[MAXN], ly[MAXM], slack[MAXM];
  bool vx[MAXN], vy[MAXM];

  void init(int n_, int m_) {
    ms(g,0); n = n_, m = m_;
  }

  void add(int u, int v, T w) {
    g[u][v] = w;
  }

  bool find(int x) {
    vx[x] = true;
    for (int y = 1; y <= m; ++y) {
      if (!vy[y]) {
        T delta = lx[x] + ly[y] - g[x][y];
        if (delta==0) {
          vy[y] = true;
          if (match[y] == 0 || find(match[y])) {
            match[y] = x;
            return true;
          }
        } else slack[y] = min(slack[y], delta);
      }
    }
    return false;
  }

  T matching() { // maximum weight matching
    fill(lx + 1, lx + 1 + n, numeric_limits<T>::lowest());
    ms(ly,0);
    ms(match,0);
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= m; ++j) lx[i] = max(lx[i], g[i][j]);
    }
    for (int k = 1; k <= n; ++k) {
      fill(slack + 1, slack + 1 + m, numeric_limits<T>::max());
      while (true) {
        ms(vx,0);
        ms(vy,0);
        if (find(k)) break;
        else {
          T delta = numeric_limits<T>::max();
          for (int i = 1; i <= m; ++i) {
            if (!vy[i]) delta = min(delta, slack[i]);
          }
          for (int i = 1; i <= n; ++i) {
            if (vx[i]) lx[i] -= delta;
          }
          for (int i = 1; i <= m; ++i) {
            if (vy[i]) ly[i] += delta;
            if (!vy[i]) slack[i] -= delta;
          }
        }
      }
    }
    T result = 0;
    for (int i = 1; i <= n; ++i) result += lx[i];
    for (int i = 1; i <= m; ++i) result += ly[i];
    return result;
  }
};

void solve() {
  int n, m;
  cin >> n >> m;

  KuhnMunkras<double> km;
  km.init(n, m);

  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      double c;
      cin >> c;
      km.add(i,j, -c);
    }
  }

  double result = km.matching();
  double total_cost = -result;
  cout << total_cost << ’\n’;
}


int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

4.8. Kosaraju

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long
#define ms(a, b) memset(a, b, sizeof(a))

const int INF = 1 << 30;
const int MAXN = 25;
const int MAXM = 25;

vector<bool> visited; // keeps track of which vertices are already
    visited

// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector when dfs leaves
    it.
void dfs(int v, vector<vector<int>> const& adj, vector<int> &output) {
  visited[v] = true;
  for (auto u : adj[v])
    if (!visited[u])
      dfs(u, adj, output);
  output.push_back(v);
}

// input: adj -- adjacency list of G
// output: components -- the strongly connected components in G
// output: adj_cond -- adjacency list of G^SCC (by root vertices)
void strongly_connected_components(vector<vector<int>> const& adj,
    vector<vector<int>> &components,
    vector<vector<int>> &adj_cond) {
  int n = (int)adj.size();
  components.clear();
  adj_cond.clear();

  // 1) Order the vertices by finish time in a first DFS.
  vector<int> order;
  visited.assign(n, false);
  for (int i = 0; i < n; i++) {
    if (!visited[i]) {
      dfs(i, adj, order);
    }
  }

  // 2) Reverse all edges to get the transpose graph.
  vector<vector<int>> adj_rev(n);
  for (int v = 0; v < n; v++) {
    for (int u : adj[v]) {
      adj_rev[u].push_back(v);
    }
  }

  // 3) Do a second DFS in descending order of finish times (from step
```

```
   1).
  visited.assign(n, false);
  reverse(order.begin(), order.end());

  // this array will store the "root" (representative) of each SCC
  vector<int> roots(n, -1);

  for (auto v : order) {
    if (!visited[v]) {
      vector<int> component;
      dfs(v, adj_rev, component);
      // store the newly found component
      components.push_back(component);
      // the root is the minimum vertex in that component (arbitrary
   choice)
      int root = *min_element(begin(component), end(component));
      for (auto u : component) {
        roots[u] = root;
      }
    }
  }

  // 4) Build the condensation graph (SCC DAG).
  adj_cond.assign(n, {});
  for (int v = 0; v < n; v++) {
    for (auto u : adj[v]) {
      if (roots[v] != roots[u]) {
        adj_cond[roots[v]].push_back(roots[u]);
      }
    }
  }
}

void solve() {

  int n = 5, m = 5;
  vector<vector<int>> adj(n);

  vector<pair<int,int>> edges = {
    {0,1},
    {1,2},
    {2,0},
    {1,3},
    {3,4}
  };

  for (auto &e : edges) {
    adj[e.first].push_back(e.second);
  }

  vector<vector<int>> components, cond_graph;
  strongly_connected_components(adj, components, cond_graph);

  for (auto &comp : components) {
    sort(comp.begin(), comp.end());
  }

  cout << "Number of SCCs: " << (int)components.size() << "\n";
```

```cpp
  for (int i = 0; i < (int)components.size(); i++) {
    cout << "SCC " << i << ": ";
    for (auto &v : components[i]) {
      cout << v << " ";
    }
    cout << "\n";
  }

  cout << "Condensation graph adjacency list:\n";
  // We'll print only non-empty adjacency to see edges
  for (int i = 0; i < (int)cond_graph.size(); i++) {
    if (!cond_graph[i].empty()) {
      cout << "root " << i << ": ";
      for (auto &v : cond_graph[i]) {
        cout << v << " ";
      }
      cout << "\n";
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  //cin >> t;
  while(t--)
    solve();
}
```

4.9. Kruskal

```cpp
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define endl '\n'

struct Edge {
  ll u, v, weight;
  bool operator<(Edge const& other) const {
    return weight < other.weight;
  }
};

struct DSU {
  vector<int> parent, size;

  DSU(int n) {
    parent.resize(n);
    size.assign(n, 1);
    iota(parent.begin(), parent.end(), 0);
  }

  int find(int v) {
    if (v == parent[v]) return v;
    return parent[v] = find(parent[v]);
  }

  bool unite(int a, int b) {
    a = find(a);
    b = find(b);
    if (a == b) return false;
    if (size[a] < size[b]) swap(a, b);
    parent[b] = a;
    size[a] += size[b];
    return true;
  }
};

ll kruskal(int n, vector<Edge>& edges, vector<Edge>& result) {
  sort(edges.begin(), edges.end());
  DSU dsu(n);
  ll cost = 0;

  for (auto& e : edges) {
    if (dsu.unite(e.u, e.v)) {
      cost += e.weight;
      result.push_back(e);
    }
  }
  return cost;
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  int n, m;
  cin >> n >> m;

  vector<Edge> edges;
  edges.reserve(m);

  for (int i = 0; i < m; i++) {
    ll u, v, w;
    cin >> u >> v >> w;
    u--, v--;
    edges.push_back({u, v, w});
  }

  vector<Edge> result;
  ll total_cost = kruskal(n, edges, result);

  if(result.size() == n-1) {
    cout << total_cost << endl;
  } else {
    cout << "IMPOSSIBLE" << endl;
  }

  // Optional: print MST edges
  // for (auto &e : result)
  //     cout << e.u + 1 << " " << e.v + 1 << " " << e.weight << endl;
}
```

4.10. Heavy Light Descomposition

```cpp
#include <bits/stdc++.h>

using namespace std;


template <class T> class MaxSegmentTree {
  private:
  // const T DEFAULT = std::numeric_limits<T>().max();
  const T DEFAULT = 0;
  int len;
  vector<T> segtree;
  public:
  MaxSegmentTree(int len) : len(len), segtree(len * 2, DEFAULT) {}
  void set(int ind, T val) {
    ind += len;
    segtree[ind] = val;
    for (; ind > 1; ind /= 2) {
      segtree[ind / 2] = std::max(segtree[ind], segtree[ind ^ 1]);
    }
  }

  T range_max(int start, int end) {
    T max = DEFAULT;
    for (start += len, end += len; start < end; start /= 2, end /= 2) {
      if (start % 2 == 1) { max = std::max(max, segtree[start++]); }
      if (end % 2 == 1) { max = std::max(max, segtree[--end]); }
    }
    return max;
  }

};


template <class T, bool VALS_IN_EDGES> class HLD {
  private:
  int N, R, tim = 0;  // n, root node, time
  vector<vector<int>> adj;
  vector<int> par, siz, depth, rt, pos;  // parent, size, depth, root, position arrays
  MaxSegmentTree<T> segtree;              // Modify as needed
  /** Compute the size of each subtree and set parent-child relationship
   * Subtree of node v corresponds to segment [ pos[v], pos[v] + sz[v] ) */

  void dfs_sz(int v) {
    if (par[v] != -1) adj[v].erase(find(adj[v].begin(), adj[v].end(), par[v]));
    for (int &u : adj[v]) {
      par[u] = v, depth[u] = depth[v] + 1;
      dfs_sz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }

  }
  /** Assign positions for nodes
```

```cpp
    * Path from v to the last vertex in ascending heavy path
      corresponds to [ pos[rt[v]], pos[v] ] */

  void dfs_hld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u);
      dfs_hld(u);
    }
  }
  /** process all heavy path and combine their results */
  template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
      if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
      op(pos[rt[v]], pos[v]);
    }

    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u] + VALS_IN_EDGES, pos[v]);
  }
  public:
  HLD(vector<vector<int>> adj_, int _R)
      : N(adj_.size()), R(_R), adj(adj_), par(N, -1), siz(N, 1), depth(N), rt(N),
        pos(N), segtree(N)  // modify as needed
  {
    rt[R] = R;
    dfs_sz(R);
    dfs_hld(R);
  }

  T query_path(int u, int v) {
    T res = 0;  // default value, modify depending on problem
    process(u, v, [&](int l, int r) {
      res = max(res, segtree.range_max(l, r + 1));  // modify depending on problem
    });
    return res;
  }
  void modify_node(int u, T val) { segtree.set(pos[u], val); }
};


int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  int n, q;
  cin >> n >> q;
  vector<int> v(n);
  vector<vector<int>> adj(n);
  for (int i = 0; i < n; i++) { cin >> v[i]; }
  for (int i = 0; i < n - 1; i++) {
    int a, b;
    cin >> a >> b;
    --a, --b;
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  HLD<int, 0> H(adj, 0);
  for (int i = 0; i < n; i++) { H.modify_node(i, v[i]); }
  while (q--) {
    int type, s, a, b, x;
    cin >> type;
    if (type == 1) {
      cin >> s >> x;
      --s;
      H.modify_node(s, x);
    } else if (type == 2) {
      cin >> a >> b;
      --a, --b;
      cout << H.query_path(a, b) << " ";
    }
  }

}
```

4.11. LCA

```
#include <bits/stdc++.h>
using namespace std;

#define endl '\n'
using ll = long long;

int n, l;
vector<vector<pair<int,int>>> adj;
int timer;
vector<int> tin, tout;
vector<vector<int>> up;
vector<ll> dist; // distance from root to node

void dfs(int v, int p)
{
  tin[v] = ++timer;
  up[v][0] = p;

  for (int i = 1; i <= l; ++i)
    up[v][i] = up[up[v][i-1]][i-1];

  for (auto [u, w] : adj[v]) {
    if (u != p) {
      dist[u] = dist[v] + w;
      dfs(u, v);
    }
  }

  tout[v] = ++timer;
}

bool is_ancestor(int u, int v) {
  return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
  if (is_ancestor(u, v)) return u;
  if (is_ancestor(v, u)) return v;
  for (int i = l; i >= 0; --i)
    if (!is_ancestor(up[u][i], v))
      u = up[u][i];
  return up[u][0];
}

ll distance(int u, int v)
{
  int ancestor = lca(u, v);
  return dist[u] + dist[v] - 2 * dist[ancestor];
}

void preprocess(int root)
{
  tin.resize(n);
  tout.resize(n);
  dist.assign(n, 0);
```

```cpp
  timer = 0;
  l = ceil(log2(n));
  up.assign(n, vector<int>(l + 1));
  dfs(root, root);
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  cin >> n;
  adj.assign(n, {});

  for (int i = 0; i < n - 1; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    --u; --v;
    adj[u].push_back({v, w});
    adj[v].push_back({u, w});
  }

  preprocess(0); // root = 0

  int q; cin >> q;
  while (q--) {
    int u, v;
    cin >> u >> v;
    --u; --v;
    cout << distance(u, v) << endl;
  }
}
```

4.12. Topological Sort

```cpp
#include <bits/stdc++.h>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define ll long long

const int INF = 1 << 30;
const int MAXN = 1e5+5;

vector<int> adj[MAXN];
vector<int> topo;

int n,m;

void dfs(int u, vector<int> &visited) {
  visited[u] = 1;
  for (int &v : adj[u]) {
    if (!visited[v])
      dfs(v, visited);
    else if(visited[v] == 1){
      cout << "IMPOSSIBLE\n";
      exit(0);
    }
  }
  visited[u] = 2;
  topo.push_back(u);
}

void topological_sort() {
  topo.clear();
  vector<int> visited(n+1,0);

  for (int i = 1; i <= n; i++)
    if (!visited[i])
      dfs(i, visited);

  reverse(topo.begin(), topo.end());
}

void solve() {
  cin >> n >> m;
  for(int i = 0; i <= n; i++) adj[i].clear();
  while(m--) {
    int a,b;
    cin >> a >> b;
    adj[a].push_back(b);
  }
  topological_sort();
  for (int x : topo) cout << x << " ";
  cout << "\n";
}

int main() {
  int t = 1;
  // cin >> t;
  while(t--)
    solve();
}
```

4.13. Centroid Descomposition

```cpp
#include "bits/stdc++.h"
using namespace std;

using ll = long long;
using ld = long double;
#define sz(v) ((int)((v).size()))

const int MAXN = 100005;

int n;
vector<int> adj[MAXN];
int sub[MAXN];
bool removed[MAXN];
int parCentroid[MAXN];
vector<vector<pair<int,int>>> elements(MAXN);

vector<vector<pair<int,int>>> sufix(MAXN);
#define linea() cerr << "---------------------------" << '\n'
int dfs_size(int u, int p) {
    sub[u] = 1;
    for (int v : adj[u]) {
        if (v == p || removed[v]) continue;
        sub[u] += dfs_size(v, u);
    }
    return sub[u];
}

int find_centroid(int u, int p, int total) {
    for (int v : adj[u]) {
        if (v == p || removed[v]) continue;
        if (sub[v] > total / 2)
            return find_centroid(v, u, total);
    }
    return u;
}
void bfs(int node) {
    map<int, int> dist;
    queue<int> q;
    q.push(node);
    elements[node].push_back({node, 0});
    while (!q.empty()) {
        auto l = q.front();
        q.pop();
        for (auto& x : adj[l]) {
            if (dist[x] == 0 && !removed[x] && x != node) {
                dist[x] = dist[l] + 1;
                q.push(x);
                elements[node].push_back({x, dist[x]});
            }
        }
    }

    sort(elements[node].begin(), elements[node].end());

    sufix[node].resize(sz(elements[node]) + 1);

    pair<int, int> otro {1e9, 1e9};
    for (int i = sz(elements[node]) - 1; i >= 0; i--) {
        if (otro.second > elements[node][i].second) {
            otro = elements[node][i];
        }
        else if (otro.second == elements[node][i].second) {
            if (otro.first > elements[node][i].first) otro = elements[node][i];
        }
        sufix[node][i] = otro;
    }
}
void decompose(int u, int p = -1) {
    int total = dfs_size(u, -1);
    int c = find_centroid(u, -1, total);
    bfs(c);
    removed[c] = true;

    parCentroid[c] = (p == -1 ? c : p); // root centroid points to itself

    // Example: hook custom logic here (before recursing)
    // e.g., preprocess distances from c to other nodes in its component

    for (int v : adj[c]) {
        if (!removed[v])
            decompose(v, c);
    }
}
int bs(vector<pair<int, int>>& v, int target) {
    int low = 0, high = sz(v) - 1;
    int ans = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (v[mid].first > target) {
            ans = mid;
            high = mid - 1;
        }
        else low = mid + 1;
    }
    return ans;
}
int dist(vector<pair<int, int>>& v, int target) {
    int low = 0, high = sz(v) - 1;
    int ans = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (v[mid].first == target) {
            return mid;
        }
        else if (v[mid].first < target) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
int32_t main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> n;
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    decompose(1);
    for (int i = 1; i < n; i++) {
        int aux = i;
        int ori = aux;
        pair<int, int> res = {1e9, 1e9};
        bool ya = false;
        do {
            int it = bs(elements[aux], ori);
            if (it == -1) {
                aux = parCentroid[aux];
                linea();
                continue;
            }
            pair<int, int> ans = sufix[aux][it];
            int it2 = dist(elements[aux], ori);
            ans.second += elements[aux][it2].second;
            if (ans.second < res.second) {
                res = ans;
            }
            else if (ans.second == res.second) {
                if (ans.first < res.first) {
                    res = ans;
                }
            }
            aux = parCentroid[aux];
            linea();
            if (ya) break;
            if (parCentroid[aux] == aux) ya = true;
        } while (true);
        cout << res.first << ' ';
    }
    cout << n << '\n';

    return 0;
}
```

Parte V

Math

5.1. FFT

```cpp
#include <bits/stdc++.h>
using namespace std;

#define ll long long
const double PI = acos(-1);

namespace FFT {
    static vector<int> rev;
    static vector<complex<double>> raices{{0,0},{1,0}};

    void fft(vector<complex<double>>& a, bool invertido) {
        int n = a.size();

        if (rev.size() != n) {
            int k = __builtin_ctz(n); // log2(n)
            rev.assign(n, 0);
            for (int i = 0; i < n; i++) {
                rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (k - 1));
            }
        }

        for (int i = 0; i < n; i++) {
            if (i < rev[i]) swap(a[i], a[rev[i]]);
        }

        if (raices.size() < n) {
            int k = __builtin_ctz(raices.size());
            raices.resize(n);
            while ((1 << k) < n) {
                double angulo = 2 * PI / (1 << (k + 1));
                for (int i = 1 << (k - 1); i < (1 << k); i++) {
                    raices[2 * i] = raices[i];
                    double ang = angulo * (2 * i + 1 - (1 << k));
                    raices[2 * i + 1] = complex<double>(cos(ang), sin(ang));
                }
                k++;
            }
        }

        for (int len = 1; len < n; len = len << 1) {
            for (int i = 0; i < n; i += 2 * len) {
                for (int j = 0; j < len; j++) {
                    complex<double> u = a[i + j];
                    complex<double> v = a[i + j + len] * raices[len + j];
                    a[i + j] = u + v;
                    a[i + j + len] = u - v;
                }
            }
        }

        if (invertido) {
            reverse(a.begin() + 1, a.end());
            for (auto &x : a) x /= n;
        }
    }
} // namespace FFT

vector<ll> multiplicacion_polinomio(const vector<ll>& A, const vector<ll>& B) {
    int n1 = A.size(), n2 = B.size();
    if (n1 == 0 || n2 == 0) return vector<ll>();

    int n = 1;
    while (n < n1 + n2 - 1) n = n << 1;
    vector<complex<double>> fa(n), fb(n);
    for (int i = 0; i < n1; i++) fa[i] = complex<double>(A[i],0);
    for (int i = 0; i < n2; i++) fb[i] = complex<double>(B[i],0);

    FFT::fft(fa, false);
    FFT::fft(fb, false);
    for (int i = 0; i < n; i++) fa[i] *= fb[i];
    FFT::fft(fa, true);

    vector<ll> res(n1 + n2 - 1);
    for (int i = 0; i < res.size(); i++) {
        res[i] = llround(fa[i].real()); // Redondeo
    }
    return res;
}


int main() {
    vector<ll> A = {1, 2, 3}; // 1 + 2x + 3x^2
    vector<ll> B = {2, 0, 1}; // 2 + 0x + 1x^2
    vector<ll> C = multiplicacion_polinomio(A, B);
    return 0;
}
```

5.2. NTT

```cpp
#include <bits/stdc++.h>
using namespace std;

#define ll long long

namespace NTT {
    // MOD debe ser un primo de la forma c * 2^k + 1
    // 998244353 = 119 * 2^23 + 1. Ra z primitiva G = 3.
    const ll MOD = 998244353;
    const ll G = 3; // Ra z primitiva para MOD

    ll fast_pow(ll a, ll b) {
        ll res = 1;
        a %= MOD;
        while (b > 0) {
            if (b & 1) res = (res * a) % MOD;
            a = (a * a) % MOD;
            b >>= 1;
        }
        return res;
    }

    ll inv(ll a) {
        return fast_pow(a, MOD - 2);
    }

    static vector<int> rev;
    static vector<ll> w;

    void precompute(int n) {
        if (rev.size() == n) return;
        int k = __builtin_ctz(n);
        rev.assign(n, 0);
        for (int i = 0; i < n; i++) {
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (k - 1));
        }

        w.resize(n);
        w[0] = 1;
        ll g_n = fast_pow(G, (MOD - 1) / n);
        for(int i = 1; i < n; i++) {
            w[i] = (w[i - 1] * g_n) % MOD;
        }
    }

    void fft(vector<ll>& a, bool invertido) {
        int n = a.size();
        precompute(n);

        for (int i = 0; i < n; i++) {
            if (i < rev[i]) swap(a[i], a[rev[i]]);
        }

        for (int len = 1; len < n; len <<= 1) {
            int step = n / (2 * len);
            for (int i = 0; i < n; i += 2 * len) {
                for (int j = 0; j < len; j++) {
                    ll u = a[i + j];
                    ll v = (a[i + j + len] * w[j * step]) % MOD;

                    a[i + j] = (u + v) % MOD;
                    a[i + j + len] = (u - v + MOD) % MOD;
                }
            }
        }

        if (invertido) {
            reverse(a.begin() + 1, a.end());
            ll n_inv = inv(n);
            for (auto &x : a) {
                x = (x * n_inv) % MOD;
            }
        }
    }
} // namespace NTT

vector<ll> multiplicacion_polinomio_mod(const vector<ll>& A, const vector<ll>& B) {
    int n1 = A.size(), n2 = B.size();
    if (n1 == 0 || n2 == 0) return vector<ll>();

    int n = 1;
    while (n < n1 + n2 - 1) n = n << 1;

    vector<ll> fa(n), fb(n);
    for (int i = 0; i < n1; i++) fa[i] = (A[i] % NTT::MOD + NTT::MOD) % NTT::MOD;
    for (int i = 0; i < n2; i++) fb[i] = (B[i] % NTT::MOD + NTT::MOD) % NTT::MOD;

    NTT::fft(fa, false);
    NTT::fft(fb, false);

    for (int i = 0; i < n; i++) {
        fa[i] = (fa[i] * fb[i]) % NTT::MOD;
    }

    NTT::fft(fa, true);

    fa.resize(n1 + n2 - 1);
    return fa;
}


int main() {
    vector<ll> A = {1, 2, 3}; // 1 + 2x + 3x^2
    vector<ll> B = {2, 0, 1}; // 2 + 0x + 1x^2
    vector<ll> C = multiplicacion_polinomio_mod(A, B);
    return 0;
}
```

5.3. Binary Exponentiation

```cpp
#include <bits/stdc++.h>

#define ll long long

ll binpow(ll a, ll b) {
    if (b == 0)
        return 1;
    ll res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}

ll binpowIt(ll a, ll b) {
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}

int power(ll x, ll y, ll M)
{
    if (y == 0)
        return 1;

    int p = power(x, y / 2, M) % M;
    p = (p * p) % M;

    return (y % 2 == 0) ? p : (x * p) % M;
}
// inverse modular power(A, M - 2, M)
```

5.4. Matrix Exponentiation

```cpp
#include <bits/stdc++.h>

using namespace std;

#define ll long long
const int MOD = 1e9+7;

struct Matrix {
  int n;
  vector<vector<ll>> mat;

  Matrix(int size) : n(size), mat(size, vector<ll>(size)) {}

  // Identity matrix
  static Matrix identity(int size) {
    Matrix I(size);
    for (int i = 0; i < size; i++)
      I.mat[i][i] = 1;
    return I;
  }

  // Matrix multiplication
  Matrix operator*(const Matrix& other) const {
    Matrix product(n);
    for (int i = 0; i < n; ++i)
      for (int k = 0; k < n; ++k)
        for (int j = 0; j < n; ++j) {
          product.mat[i][j] += mat[i][k] * other.mat[k][j];
          product.mat[i][j] %= MOD;
        }
    return product;
  }

  // Matrix exponentiation
  Matrix pow(ll power) const {
    Matrix result = identity(n);
    Matrix base = *this;

    while (power > 0) {
      if (power & 1)
        result = result * base;
      base = base * base;
      power >>= 1;
    }
    return result;
  }
};

ll fibonacci(int n) {
  if (n == 0) return 0;
  Matrix base(2);
  base.mat = {{1, 1}, {1, 0}};
  base = base.pow(n - 1);
  return base.mat[0][0]; // F(n)
}

```

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

Parte VI

Strings

6.1. KMP Automaton

```cpp
#include <bits/stdc++.h>

using namespace std;

#define ll long long
#define endl '\n'

vector<int> prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i-1];
    while (j > 0 && s[i] != s[j])
      j = pi[j-1];
    if (s[i] == s[j])
      j++;
    pi[i] = j;
  }
  return pi;
}

void compute_automaton(string s, vector<vector<int>>& aut) {
  s += '#';
  int n = s.size();
  vector<int> pi = prefix_function(s);
  aut.assign(n, vector<int>(26));
  for (int i = 0; i < n; i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i])
        aut[i][c] = aut[pi[i-1]][c];
      else
        aut[i][c] = i + ('a' + c == s[i]);
    }
  }
}

const int MAXN = 1e4+5;
const int MAXK = 1e3+5;
const int MOD = 1e9+7;


int n,k;
string pattern;
vector<vector<int>> aut;

long long binpow(long long a, long long b)
{
    long long result = 1;
    while (b) {
        if (b & 1)
            result = (result * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return result;
}
```

```cpp

ll dp[MAXN][MAXK];
ll fdp(int pos, int state) {
  bool found = state == pattern.size();
  if(pos == n) {
    return  found;
  }

  ll &val = dp[pos][state];
  if(val != -1) return val;

  val = 0;

  if(found) {
    ll l = n-pos;
    return val = binpow(k, l);
  }

  for(int i = 0; i < k; i++) {
    val = (val + fdp(pos+1,aut[state][i])) % MOD;
  }

  return val;
}

void solve() {
  memset(dp, -1, sizeof dp);
  cin >> n >> k >> pattern;

  compute_automaton(pattern, aut);

  cout << fdp(0,0) << endl;
}

int main() {
  solve();
}
```

6.2. Hashing

```cpp
#include <bits/stdc++.h>

using namespace std;

#define ll long long

const int MAXN = 1e5+5;
const int BASE = 31;
const int M = 1e9 + 7;

int pot[MAXN];

void compute_powers(int n) {
    int p_pow = 1;
    for (int i = 0; i < n; i++) {
        pot[i] = p_pow;
        p_pow = (p_pow * BASE) % M;
    }
}

vector<ll> compute_hash(string const& s) {
    int n = s.size();
    vector<ll> hash_values(n + 5);
    hash_values[0] = 0;
    for (int i = 1; i <= n; i++) {
        hash_values[i] = (hash_values[i - 1] * BASE + s[i - 1]) % M;
    }
    return hash_values;
}

ll get_hash(vector<ll> &hash_values, int a, int b) {
    return (hash_values[b] - (hash_values[a - 1] * pot[b - a + 1]) % M + M) % M;
}

int main() {
    string word = "ALLEY";
    compute_powers(word.size());
    vector<ll> hash = compute_hash(word);

    for (int i = 1; i <= word.size(); i++) {
        cout << hash[i] << ' ';
    }
    cout << endl;
    for (int i = 1; i <= word.size(); i++) {
        cout << word[i - 1] << ": " << get_hash(hash, i, i) << endl;
    }

    return 0;
}
```