

TEAMBOOK TRUCHO

POLLOS PRIMO

November 8, 2024

Contents

1	estdatos	3
1.1	MACROS	3
1.2	bit	5
1.3	bit2d	5
1.4	hashtrucho	9
1.5	lazytree	9
1.6	minimalmacros	11
1.7	minsparce	12
1.8	segtreegeneral	13
1.9	segtreeterativo	14
1.10	struclazy	15
1.11	structsegtree	17
2	grafos	18
2.1	SCC	18
2.2	bfsgrillapath	19
2.3	bfspath	20
2.4	bipartitecheck	21
2.5	ciclonegativobf	21
2.6	detectarciclosdirigido	22
2.7	dijkstra	23
2.8	floydwarshall	24
2.9	kruskal	24
2.10	kthsspath	25
2.11	multliavabfs	25
2.12	ssspnegativo	28
2.13	topsort	29
2.14	unionfind	30
3	DP	30
3.1	Knapsack	30
3.2	KnapsackOptimization	31
3.3	LCS	31
3.4	LIS	33
3.5	MCM(MatrixChainMultiplicated)	34
3.6	MCMaplicacionn	34
3.7	OptimalGridTraversal	35
3.8	OptimalGridTraversalOptimization	37
3.9	SOS	37

4	sortingsearchinggreedy	38
4.1	bisecciongeneral	38
4.2	cantdesubarrsumax	38
4.3	cantidaddesumasdistintas	39
4.4	cantsubarraydistintos	40
4.5	cantsubarrdiv	41
4.6	maxsubarraysumentreab	41
4.7	secuencialargaunica	42
4.8	sumade4valroes	43
5	Math	44
5.1	convexHullGrahamScan	44
5.2	convexHullJarvisMarch	46
6	AlgoritmosGeneral	48
6.1	Combinatorics	48
6.2	TernarySearch	49
6.3	TernarySearchReales	50
7	datastructures	51
7.1	FenwikTree	51

1 estdatos

1.1 MACROS

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 // #include <ext/pb_ds/assoc_container.hpp>
4 // #include <ext/pb_ds/assoc_container.hpp>
5 // #include <ext/pb_ds/tree_policy.hpp>
6 // #include <ext/rope>
7 #define int long long
8 #define uset unordered_set
9 #define f first
10 #define sst stringstream
11 #define s second
12 #define umap unordered_map
13 #define mp make_pair
14 #define pb push_back
15 #define sz(x) (int)(x).size()
16 #define all(a) (a).begin(), (a).end()
17 #define rall(a) (a).rbegin(), (a).rend()
18 #define floatigual(a, b) (fabs(a - b) < EPS)
19 #define mod(a) md(a, MOD)
20 #define fore(i, a, n) for(int i = (a); i < (n); i++)
21 #define forb(i, n) for (int i = (n) - 1; i >= 0; i--)
22 #define FORDD(i, a, b) for (int i = (b) - 1; i >= (a); --i)
23 #define techo(a, b) (a / b + (a % b != 0))
24 #define popcount(x) __builtin_popcountll(x);
25 using namespace std;
26 // using namespace __gnu_pbds;
27 // using namespace __gnu_cxx;
28 typedef long double ld;
29 typedef unsigned long long ull;
30 typedef pair<int, int> pii;
31 typedef vector<int> vi;
32 typedef vector<bool> vbool;
33 // typedef
34 // tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>
35 // ordered_set; find_by_order kth largest order_of_key < mt19937
36 // rng(chrono::steady_clock::now().time_since_epoch().count()); rng
37 const int tam = 200010;
38 const int MOD = 1000000007;
39 const int MOD1 = 998244353;
40 const double DINF = 1e100;
41 const double EPS = 1e-9;
42 const double PI = acos(-1);
43 // Modificar la constante para la criba
44 const int constante = 500;
45 vector<bool> criba(constante + 1);
46 vector<int> primos;
47 void eratostenes() {
48     criba[0] = criba[1] = true;
49     for (int i = 2; i <= constante; ++i) {
50         if (!criba[i]) {
51             primos.push_back(i);
52             for (int j = i * i; j <= constante; j += i) {
53                 criba[j] = 1;
54             }
55         }
56     }
57 }
```

```

55     }
56 }
57 }
58 int binpow(int a, int b) {
59     if (b == 0) {
60         return 1;
61     } else if (b == 1) {
62         return a;
63     } else if (b < 0) {
64         return 1 / binpow(a, -b);
65     } else if (b % 2 == 0) {
66         int we = binpow(a, b / 2);
67         return we * we;
68     } else {
69         return a * binpow(a, b - 1);
70     }
71 }
72 int gauss(int n) {
73     int res = (((n % MOD) * ((n + 1) % MOD)) % MOD) / 2;
74     return res;
75 }
76 int expMod(int base, int exponente, int mod) {
77     int res = 1;
78     base %= mod;
79     while (exponente > 0) {
80         if (exponente % 2 == 1) res = (res * base) % mod;
81         exponente >>= 1;
82         base = (base * base) % mod;
83     }
84     return res;
85 }
86 class UnionFind {
87     vector<int> parents;
88     vector<int> sizes;
89
90 public:
91     UnionFind(int size) : parents(size), sizes(size, 1) {
92         for (int i = 0; i < size; i++) {
93             parents[i] = i;
94         }
95     }
96     int find(int x) {
97         return parents[x] == x ? x : (parents[x] = find(parents[x]));
98     }
99     bool join(int x, int y) {
100         int x_root = find(x);
101         int y_root = find(y);
102         if (x_root == y_root) {
103             return false;
104         }
105         if (sizes[x_root] < sizes[y_root]) {
106             swap(x_root, y_root);
107         }
108         sizes[x_root] += sizes[y_root];
109         parents[y_root] = x_root;
110         return true;
111     }
112     bool connected(int x, int y) { return find(x) == find(y); }
113 };

```

```

114
115 void solve() {
116
117 }
118 signed main() {
119     ios::sync_with_stdio(0);
120     cin.tie(0);
121     cout.tie(0);
122     int t; cin >> t; while(t--) solve();
123     solve();
124 }
125 //PLUS ULTRA RECARGADO!!!

```

MACROS

1.2 bit

```

1 struct FenwickTree {
2     vector<int> bit;
3     int n;
4
5     FenwickTree(int n) {
6         this->n = n;
7         bit.assign(n, 0);
8     }
9
10    FenwickTree(vector<int> a) : FenwickTree(a.size()) {
11        for (size_t i = 0; i < a.size(); i++)
12            add(i, a[i]);
13    }
14
15    int sum(int r) {
16        int ret = 0;
17        for (; r >= 0; r = (r & (r + 1)) - 1)
18            ret += bit[r];
19        return ret;
20    }
21
22    int sum(int l, int r) {
23        return sum(r) - sum(l - 1);
24    }
25
26    void add(int idx, int delta) {
27        for (; idx < n; idx = idx | (idx + 1))
28            bit[idx] += delta;
29    }
30 };

```

bit

1.3 bit2d

```

1 /*2D BIT is basically a BIT where each element is another BIT.
2 Updating by adding v on (x, y) means it's effect will be found
3 throughout the rectangle [(x, y), (max_x, max_y)],
4 and query for (x, y) gives you the result of the rectangle
5 [(0, 0), (x, y)], assuming the total rectangle is

```

```

6  [(0, 0), (max_x, max_y)]. So when you query and update on
7  this BIT, you have to be careful about how many times you are
8  subtracting a rectangle and adding it. Simple set union formula
9  works here.
10
11  So if you want to get the result of a specific rectangle
12  [(x1, y1), (x2, y2)], the following steps are necessary:
13
14  Query(x1,y1,x2,y2) = getSum(x2, y2)-getSum(x2, y1-1) -
15                        getSum(x1-1, y2)+getSum(x1-1, y1-1)
16
17  Here 'Query(x1,y1,x2,y2)' means the sum of elements enclosed
18  in the rectangle with bottom-left corner's co-ordinates
19  (x1, y1) and top-right corner's co-ordinates - (x2, y2)
20
21  Constraints -> x1<=x2 and y1<=y2
22
23      /\
24  y /
25      /          -----(x2,y2)
26      /          /      /
27      /          /      /
28      /          /      /
29      /          -----
30      /          (x1,y1)
31      /
32      /----->
33  (0, 0)                                x-->
34
35  In this program we have assumed a square matrix. The
36  program can be easily extended to a rectangular one. */
37
38  #include<bits/stdc++.h>
39  using namespace std;
40
41  #define N 4 // N-->max_x and max_y
42
43  // A structure to hold the queries
44  struct Query
45  {
46      int x1, y1; // x and y co-ordinates of bottom left
47      int x2, y2; // x and y co-ordinates of top right
48  };
49
50  // A function to update the 2D BIT
51  void updateBIT(int BIT[][N+1], int x, int y, int val)
52  {
53      for (; x <= N; x += (x & -x))
54      {
55          // This loop update all the 1D BIT inside the
56          // array of 1D BIT = BIT[x]
57          for (int yy=y; yy <= N; yy += (yy & -yy))
58              BIT[x][yy] += val;
59      }
60      return;
61  }
62
63  // A function to get sum from (0, 0) to (x, y)
64  int getSum(int BIT[][N+1], int x, int y)

```

```

65 {
66     int sum = 0;
67
68     for(; x > 0; x -= x&-x)
69     {
70         // This loop sum through all the 1D BIT
71         // inside the array of 1D BIT = BIT[x]
72         for(int yy=y; yy > 0; yy -= yy&-yy)
73         {
74             sum += BIT[x][yy];
75         }
76     }
77     return sum;
78 }
79
80 // A function to create an auxiliary matrix
81 // from the given input matrix
82 void constructAux(int mat[][N], int aux[][N+1])
83 {
84     // Initialise Auxiliary array to 0
85     for (int i=0; i<=N; i++)
86         for (int j=0; j<=N; j++)
87             aux[i][j] = 0;
88
89     // Construct the Auxiliary Matrix
90     for (int j=1; j<=N; j++)
91         for (int i=1; i<=N; i++)
92             aux[i][j] = mat[N-j][i-1];
93
94     return;
95 }
96
97 // A function to construct a 2D BIT
98 void construct2DBIT(int mat[][N], int BIT[][N+1])
99 {
100     // Create an auxiliary matrix
101     int aux[N+1][N+1];
102     constructAux(mat, aux);
103
104     // Initialise the BIT to 0
105     for (int i=1; i<=N; i++)
106         for (int j=1; j<=N; j++)
107             BIT[i][j] = 0;
108
109     for (int j=1; j<=N; j++)
110     {
111         for (int i=1; i<=N; i++)
112         {
113             // Creating a 2D-BIT using update function
114             // everytime we/ encounter a value in the
115             // input 2D-array
116             int v1 = getSum(BIT, i, j);
117             int v2 = getSum(BIT, i, j-1);
118             int v3 = getSum(BIT, i-1, j-1);
119             int v4 = getSum(BIT, i-1, j);
120
121             // Assigning a value to a particular element
122             // of 2D BIT
123             updateBIT(BIT, i, j, aux[i][j]-(v1-v2-v4+v3));

```

```

124         }
125     }
126
127     return;
128 }
129
130 // A function to answer the queries
131 void answerQueries(Query q[], int m, int BIT[][N+1])
132 {
133     for (int i=0; i<m; i++)
134     {
135         int x1 = q[i].x1 + 1;
136         int y1 = q[i].y1 + 1;
137         int x2 = q[i].x2 + 1;
138         int y2 = q[i].y2 + 1;
139
140         int ans = getSum(BIT, x2, y2)-getSum(BIT, x2, y1-1)-
141                 getSum(BIT, x1-1, y2)+getSum(BIT, x1-1, y1-1);
142
143         printf ("Query(%d, %d, %d, %d) = %d\n",
144                q[i].x1, q[i].y1, q[i].x2, q[i].y2, ans);
145     }
146     return;
147 }
148
149 // Driver program
150 int main()
151 {
152     int mat[N][N] = {{1, 2, 3, 4},
153                      {5, 3, 8, 1},
154                      {4, 6, 7, 5},
155                      {2, 4, 8, 9}};
156
157     // Create a 2D Binary Indexed Tree
158     int BIT[N+1][N+1];
159     construct2DBIT(mat, BIT);
160
161     /* Queries of the form - x1, y1, x2, y2
162     For example the query- {1, 1, 3, 2} means the sub-matrix-
163     y
164     /\
165     3 /      1 2 3 4      Sub-matrix
166     2 /      5 3 8 1      {1,1,3,2}    --->    3 8 1
167     1 /      4 6 7 5
168
169         7 5
170     0 /      2 4 8 9
171         /
172     --/----- 0 1 2 3 ----> x
173
174         Hence sum of the sub-matrix = 3+8+1+6+7+5 = 30
175
176     */
177     Query q[] = {{1, 1, 3, 2}, {2, 3, 3, 3}, {1, 1, 1, 1}};
178     int m = sizeof(q)/sizeof(q[0]);
179
180     answerQueries(q, m, BIT);
181 }

```

6


```

182         return(0);
183     }

```

bit2d

1.4 hashtrucho

```

1 random_device rd;
2 mt19937_64 gen(rd());
3 map<ull, ull> mapping;
4 set<ull> usados = { 0 };
5
6 for (auto &c : v) {
7     ull random;
8     if (!mapping.count(c)) {
9         do { random = gen(); } while (usados.count(random));
10        usados.insert(random);
11        mapping[c] = random;
12    } else {
13        random = mapping[c];
14    }
15    c = random;
16 }
17
18 //buscar los macros para esto

```

hashtrucho

1.5 lazytree

```

1 struct Node {
2     ll mn;
3     ll size = 1;
4
5     Node(ll mn):mn(mn) {
6     }
7 };
8
9 struct Func {
10    ll a = 0;
11 };
12
13 Node e() { // op(x, e()) = x
14     Node a(INT64_MAX);
15     return a;
16 };
17
18 Func id() { // mapping(x, id()) = x
19     Func l = {0};
20     return l;
21 }
22
23 Node op(Node &a, Node &b) { // associative property
24     Node c = e();
25     c.size = a.size + b.size;
26     c.mn = min(a.mn, b.mn);
27     return c;

```

```

28 }
29
30 Node mapping(Node node, Func &lazy) {
31     node.mn += lazy.a;
32     return node;
33 }
34
35 Func composicion(Func &prev, Func &actual) {
36     prev.a = prev.a + actual.a;
37     return prev;
38 }
39
40 struct lazytree {
41     int n;
42     vector<Node> nodes;
43     vector<Func> lazy;
44
45     void init(int nn) {
46         n = nn;
47         int size = 1;
48         while (size < n) {
49             size *= 2;
50         }
51         ll m = size * 2;
52         nodes.assign(m, e());
53         lazy.assign(m, id());
54     }
55
56     void push(int i, int sl, int sr) {
57         nodes[i] = mapping(nodes[i], lazy[i]);
58         if (sl != sr) {
59             lazy[i * 2 + 1] = composicion(lazy[i*2+1], lazy[i]);
60             lazy[i * 2 + 2] = composicion(lazy[i*2+2], lazy[i]);
61         }
62         lazy[i] = id();
63     }
64
65     void apply(int i, int sl, int sr, int l, int r, Func f) {
66         push(i, sl, sr);
67         if (l <= sl && sr <= r) {
68             lazy[i] = f;
69             push(i, sl, sr);
70         } else if (sr < l || r < sl) {
71             // do nothing
72         } else {
73             int mid = (sl + sr) >> 1;
74             apply(i * 2 + 1, sl, mid, l, r, f);
75             apply(i * 2 + 2, mid + 1, sr, l, r, f);
76             nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
77         }
78     }
79
80     void apply(int l, int r, Func f) {
81         assert(l <= r);
82         assert(r < n);
83         apply(0, 0, n - 1, l, r, f);
84     }
85
86     void update(int i, Node node) {
87         //assert(i < n);

```

```

87     update(0, 0, n-1, i, node);
88 }
89
90 void update(int i, int sl, int sr, int pos, Node node) {
91     if (sl <= pos && pos <= sr) {
92         push(i,sl,sr);
93         if (sl == sr) {
94             nodes[i] = node;
95         } else {
96             int mid = (sl + sr) >> 1;
97             update(i * 2 + 1, sl, mid, pos, node);
98             update(i * 2 + 2, mid + 1, sr, pos, node);
99             nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
100         }
101     }
102 }
103
104 Node query(int i, int sl, int sr, int l, int r) {
105     push(i,sl,sr);
106     if (l <= sl && sr <= r) {
107         return nodes[i];
108     } else if (sr < l || r < sl) {
109         return e();
110     } else {
111         int mid = (sl + sr) >> 1;
112         auto a = query(i * 2 + 1, sl, mid, l, r);
113         auto b = query(i * 2 + 2, mid + 1, sr, l, r);
114         return op(a,b);
115     }
116 }
117
118 Node query(int l, int r) {
119     assert(l <= r);
120     assert(r < n);
121     return query(0, 0, n - 1, l, r);
122 }
123 };

```

lazytree

1.6 minimalmacros

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  #define int long long
4  #define f first
5  #define sst stringstream
6  #define s second
7  #define pb push_back
8  #define sz(x) (int)(x).size()
9  #define all(a) (a).begin(), (a).end()
10 #define rall(a) (a).rbegin(), (a).rend()
11 #define fore(i, a, n) for(int i = (a); i < (n); i++)
12 #define forb(i, n) for (int i = (n) - 1; i >= 0; i--)
13 #define popcount(x) __builtin_popcountll(x);
14 typedef pair<int, int> pii;
15 typedef vector<int> vi;
16 const int MOD = 1000000007;

```

```

17 const double EPS = 1e-9;
18 const double PI = acos(-1);
19 const int INF = 1e18;
20 //PLUS ULTRA RECARGADO!!!
21 void solve() {
22
23 }
24 signed main() {
25     ios::sync_with_stdio(0);
26     cin.tie(0);
27     cout.tie(0);
28     int t; cin >> t; while(t--) solve();
29     solve();
30 }
31
32 /*ordered set:
33 #include <ext/pb_ds/assoc_container.hpp>
34 #include <ext/pb_ds/tree_policy.hpp>
35 using namespace __gnu_pbds;
36
37 #define oset tree<ll, null_type, less<ll>, rb_tree_tag,
38     tree_order_statistics_node_update>
39 //find_by_order(k) order_of_key(k)
40 */

```

minimalmacros

1.7 minsparse

```

1 using Type = int;
2 //xd ?????
3 struct min_sparse {
4
5     int log;
6     vector<vector<Type>> sparse;
7
8     void init(vector<Type> &nums) {
9         int n = nums.size();
10        log = 0;
11        while (n) log++, n/=2;
12        n = nums.size();
13        sparse.assign(n, vector<Type>(log, 0));
14        for (int i = 0; i < n; i++) sparse[i][0] = nums[i];
15        for (int l = 1; l < log; l++) {
16            for (int j = 0; j + (1 << l) - 1 < n; j++) {
17                sparse[j][l] = min(sparse[j][l-1], sparse[j+(1 << (l-1))][l-1]);
18            }
19        }
20    }
21
22    Type query(int x, int y) {
23        int n = y - x + 1;
24        int logg = -1;
25        while (n) logg++, n/=2;
26        return min(sparse[x][logg], sparse[y-(1 << logg)+1][logg]);
27    }
28 };

```

1.8 segtreegeneral

```

1  // >>>>>> Implement
2  // Example of a Segment tree of Xor
3  struct Node {
4      ll a = 0;
5  };
6
7  Node e() {
8      Node node;
9      return node;
10 }
11
12 Node op(Node a, Node b) {
13     Node node;
14     node.a = a.a ^ b.a;
15     return node;
16 }
17 // >>>>>> Implement
18
19 struct segtree {
20     vector<Node> nodes;
21     ll n;
22
23     void init(int n) {
24         auto a = vector<Node>(n, e());
25         init(a);
26     }
27
28     void init(vector<Node>& initial) {
29         nodes.clear();
30         n = initial.size();
31         int size = 1;
32         while (size < n) {
33             size *= 2;
34         }
35         nodes.resize(size * 2);
36         build(0, 0, n-1, initial);
37     }
38
39     void build(int i, int sl, int sr, vector<Node>& initial) {
40         if (sl == sr) {
41             nodes[i] = initial[sl];
42         } else {
43             ll mid = (sl + sr) >> 1;
44             build(i*2+1, sl, mid, initial);
45             build(i*2+2, mid+1, sr, initial);
46             nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
47         }
48     }
49
50     void update(int i, int sl, int sr, int pos, Node node) {
51         if (sl <= pos && pos <= sr) {
52             if (sl == sr) {

```

```

53         nodes[i] = node;
54     } else {
55         int mid = (sl + sr) >> 1;
56         update(i * 2 + 1, sl, mid, pos, node);
57         update(i * 2 + 2, mid + 1, sr, pos, node);
58         nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
59     }
60 }
61 }
62
63 void update(int pos, Node node) {
64     update(0, 0, n - 1, pos, node);
65 }
66
67 Node query(int i, int sl, int sr, int l, int r) {
68     if (l <= sl && sr <= r) {
69         return nodes[i];
70     } else if (sr < l || r < sl) {
71         return e();
72     } else {
73         int mid = (sl + sr) / 2;
74         auto a = query(i * 2 + 1, sl, mid, l, r);
75         auto b = query(i * 2 + 2, mid + 1, sr, l, r);
76         return op(a, b);
77     }
78 }
79
80 Node query(int l, int r) {
81     return query(0, 0, n - 1, l, r);
82 }
83
84 Node get(int i) {
85     return query(i, i);
86 }
87 };

```

segtreegeneral

1.9 segtreeterativo

```

1  // >>>>>>>> Implement
2  struct Node { //VALOR NEUTRO
3
4      int x = 0; };
5
6  Node e() { return Node(); }
7
8  Node op(Node &a, Node &b) {
9      //GENERALIZAR
10     Node c;
11     c.x = a.x + b.x;
12     return c;
13 }
14 // <<<<<<<<
15
16 struct segtree {
17     vector<Node> t;
18     int n;

```

```

19
20 void init(int n) {
21     t.assign(n * 2, e());
22     this->n = n;
23 }
24
25 void init(vector<Node>& s) {
26     n = s.size();
27     t.assign(n * 2, e());
28     for (int i = 0; i < n; i++) {
29         t[i+n] = s[i];
30     }
31     build();
32 }
33
34 void build() { // build the tree
35     for (int i = n - 1; i > 0; --i) t[i] = op(t[i<<1], t[i<<1|1]);
36 }
37
38 // set value at position p
39 void update(int p, const Node& value) {
40     for (t[p += n] = value; p >>= 1; ) t[p] = op(t[p<<1], t[p<<1|1]);
41 }
42
43 // sum on interval [l, r]
44 Node query(int l, int r) {
45     r++; // make this inclusive
46     Node resl=e(), resr=e(); // nuint element
47     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
48         if (l&1) resl = op(resl, t[l++]);
49         if (r&1) resr = op(t[--r], resr);
50     }
51     return op(resl, resr);
52 }
53
54 Node get(int i) {
55     return query(i, i);
56 }
57 };

```

segtreeterativo

1.10 struclazy

```

1 struct lazytree {
2     int n;
3     vl sum;
4     vl lazySum;
5
6     void init(int nn) {
7         sum.clear();
8         n = nn;
9         int size = 1;
10        while (size < n) {
11            size *= 2;
12        }
13        sum.resize(size * 2);
14        lazySum.resize(size * 2);

```

```

15     }
16
17     void update(int i, int sl, int sr, int l, int r, ll diff) {
18         if (lazySum[i]) {
19             sum[i] += (sr - sl + 1) * lazySum[i];
20             if (sl != sr) {
21                 lazySum[i * 2 + 1] += lazySum[i];
22                 lazySum[i * 2 + 2] += lazySum[i];
23             }
24             lazySum[i] = 0;
25         }
26         if (l <= sl && sr <= r) {
27             sum[i] += (sr - sl + 1) * diff;
28             if (sl != sr) {
29                 lazySum[i * 2 + 1] += diff;
30                 lazySum[i * 2 + 2] += diff;
31             }
32         } else if (sr < l || r < sl) {
33         } else {
34             int mid = (sl + sr) >> 1;
35             update(i * 2 + 1, sl, mid, l, r, diff);
36             update(i * 2 + 2, mid + 1, sr, l, r, diff);
37             sum[i] = sum[i * 2 + 1] + sum[i * 2 + 2];
38         }
39     }
40
41     void update(int l, int r, ll diff) {
42         assert(l <= r);
43         assert(r < n);
44         update(0, 0, n - 1, l, r, diff);
45     }
46
47     ll query(int i, int sl, int sr, int l, int r) {
48         if (lazySum[i]) {
49             sum[i] += lazySum[i] * (sr - sl + 1);
50             if (sl != sr) {
51                 lazySum[i * 2 + 1] += lazySum[i];
52                 lazySum[i * 2 + 2] += lazySum[i];
53             }
54             lazySum[i] = 0;
55         }
56         if (l <= sl && sr <= r) {
57             return sum[i];
58         } else if (sr < l || r < sl) {
59             return 0;
60         } else {
61             int mid = (sl + sr) >> 1;
62             return query(i * 2 + 1, sl, mid, l, r) + query(i * 2 + 2, mid + 1, sr,
63                 l, r);
64         }
65     }
66
67     ll query(int l, int r) {
68         assert(l <= r);
69         assert(r < n);
70         return query(0, 0, n - 1, l, r);
71     }
};

```


1.11 structsegtree

```

1  template <typename T>
2  struct segtree {
3      int n;
4      vector<T> tree;
5      T neutral;
6      void init(int nn, T neutralx) {
7          neutral = neutralx;
8          n = nn;
9          int size = 1;
10         while (size < n) size *= 2;
11         tree.assign(size * 2, neutral);
12     }
13     segtree(int nn, T neutral) { init(nn, neutral); }
14
15     T combine(T a, T b) {
16         //Cambiar por la operacion que se necesite
17         return a xor b;
18     }
19
20     void update(int i, int sl, int sr, int pos, T diff) {
21         if (sl <= pos && pos <= sr) {
22             if (sl == sr) {
23                 tree[i] = diff;
24             } else {
25                 int mid = (sl + sr) / 2;
26                 update(i * 2 + 1, sl, mid, pos, diff);
27                 update(i * 2 + 2, mid + 1, sr, pos, diff);
28                 tree[i] = combine(tree[i * 2 + 1], tree[i * 2 + 2]);
29             }
30         }
31     }
32
33     void update(int pos, T diff) {
34         update(0, 0, n - 1, pos, diff);
35     }
36
37     T query(int i, int sl, int sr, int l, int r) {
38         if (l <= sl && sr <= r) {
39             return tree[i];
40         } else if (sr < l || r < sl) {
41             return neutral;
42         } else {
43             int mid = (sl + sr) / 2;
44             T a = query(i * 2 + 1, sl, mid, l, r);
45             T b = query(i * 2 + 2, mid + 1, sr, l, r);
46             return combine(a, b);
47         }
48     }
49
50     T query(int l, int r) {
51         return query(0, 0, n - 1, l, r);
52     }

```

```
53 };
```

```
structsegtree
```

2 grafos

2.1 SCC

```
1 Dado un grafo dirigido halla las componentes fuertemente conexas (SCC).
2
3 const int inf = 1e9;
4 const int MX = 1e5+5; //Cantidad maxima de nodos
5 vector<int> g[MX]; //Lista de adyacencia
6 stack<int> st;
7 int low[MX], pre[MX], cnt;
8 int comp[MX]; //Almacena la componente a la que pertenece cada nodo
9 int SCC; //Cantidad de componentes fuertemente conexas
10 int n, m; //Cantidad de nodos y aristas
11
12 void tarjan(int u) {
13     low[u] = pre[u] = cnt++;
14     st.push(u);
15
16     for (auto &v : g[u]) {
17         if (pre[v] == -1) tarjan(v);
18         low[u] = min(low[u], low[v]);
19     }
20     if (low[u] == pre[u]) {
21         while (true) {
22             int v = st.top(); st.pop();
23             low[v] = inf;
24             comp[v] = SCC;
25             if (u == v) break;
26         }
27         SCC++;
28     }
29 }
30
31 void init() {
32     cnt = SCC = 0;
33     for (int i = 0; i <= n; i++) {
34         g[i].clear();
35         pre[i] = -1; //no visitado
36     }
37 }
38
39 // example
40 void test_case() {
41     cin >> n >> m;
42     init();
43     rep(i, 0, m) {
44         int x, y;
45         cin >> x >> y;
46         g[x].pb(y);
47     }
48     rep(i, 1, n + 1) {
49         if (pre[i] == -1) {
```

```

50         tarjan(i);
51     }
52 }
53 }

```

SCC

2.2 bfsgrillapath

```

1  int n, m;
2  vector<string> mat(1000 + 10);
3  vector<vector<bool>> visi(1000 + 10, vector<bool>(1000 + 10));
4  //se guarda en camino[i][j] desde donde llego
5  vector<vector<char>> camino(1000 + 10, vector<char>(1000 + 10));
6  vector<pair<int, int>i> direcciones = { { -1, 0 }, { 1, 0 }, { 0, -1 }, { 0, 1 }
7      };
8  vector<char> direchar = { 'U', 'D', 'L', 'R' };
9  bool esval(int nx, int ny) {
10     return nx >= 0 && ny >= 0 && nx < n && ny < m && !visi[nx][ny] &&
11         mat[nx][ny] != '#';
12 }
13 bool bfs(pair<int, int>i ini, pair<int, int>i &fin) {
14     queue<pair<int, int>i> q;
15     q.push(ini);
16     visi[ini.first][ini.second] = true;
17
18     while (!q.empty()) {
19         auto [x, y] = q.front();
20         q.pop();
21
22         for (int i = 0; i < 4; i++) {
23             int nx = x + direcciones[i].first;
24             int ny = y + direcciones[i].second;
25
26             if (esval(nx, ny)) {
27                 q.push({ nx, ny });
28                 visi[nx][ny] = true;
29                 camino[nx][ny] = direchar[i];
30                 if (mat[nx][ny] == 'B') {
31                     fin = { nx, ny };
32                     return true;
33                 }
34             }
35         }
36     }
37     return false;
38 }
39
40 void solve() {
41     cin >> n >> m;
42     pair<int, int>i ini, fin;
43     FOR(i, n) {
44         cin >> mat[i];
45         FOR(j, m) {
46             if (mat[i][j] == 'A') { ini = { i, j }; }
47         }
48     }

```

```

49
50     if (bfs(ini, fin)) {
51         cout << "YES\n";
52         string cam;
53         pair<int, int> i actual = fin;
54         while (mat[actual.first][actual.second] != 'A') {
55             char dirr = camino[actual.first][actual.second];
56             cam += dirr;
57             if (dirr == 'U')
58                 actual.first++;
59             else if (dirr == 'D')
60                 actual.first--;
61             else if (dirr == 'L')
62                 actual.second++;
63             else if (dirr == 'R')
64                 actual.second--;
65         }
66
67         reverse(all(cam));
68         cout << (int)cam.size() << '\n' << cam << '\n';
69     } else {
70         cout << "NO\n";
71     }
72 }

```

bfsgrillapath

2.3 bfspath

```

1 void solve() {
2     int n,m;cin>>n>>m;
3     vector<int> g[n+1];
4     for(int i = 0; i<m; i++){
5         int a,b;cin>>a>>b;
6         g[a].push_back(b);
7         g[b].push_back(a);
8     }
9     vector<int> dist(n+1,1e9);
10    vector<int> parent(n+1,-1);
11    queue<int> q;
12    q.push(1);
13    dist[1]=0;
14    while(!q.empty()){
15        int u=q.front();
16        q.pop();
17        for(auto v:g[u]){
18            if(dist[v]>dist[u]+1){
19                dist[v]=dist[u]+1;
20                parent[v]=u;
21                q.push(v);
22            }
23        }
24    }
25    if(dist[n]==1e9){
26        cout<<"IMPOSSIBLE\n";
27        return;
28    }
29    cout<<dist[n]+1<<"\n";

```

```

30     vector<int> ans;
31     int u=n;
32     while(u!=-1){
33         ans.push_back(u);
34         u=parent[u];
35     }
36     reverse(all(ans));
37     for(auto x:ans)cout<<x<<" ";
38     cout<<"\n";
39 }

```

bfspath

2.4 bipartitecheck

```

1  bool check(vector<vector<int>>& g, int n){
2      vector<int> color(n + 1, -1);
3      vector<bool> visi(n + 1, false);
4      for (int i = 1; i <= n; i++) {
5          if (!visi[i]) {
6              queue<int> cola;
7              cola.push(i);
8              color[i] = 1;
9              while (!cola.empty()) {
10                 int act = cola.front();
11                 cola.pop();
12                 if (visi[act]) continue;
13                 visi[act] = true;
14                 for (int &vecino : g[act]) {
15                     if (color[vecino] == color[act]) {
16                         return false;
17                     } else if (!visi[vecino]) {
18                         color[vecino] = 1 - color[act];
19                         cola.push(vecino);
20                     }
21                 }
22             }
23         }
24     }
25     return true;
26 }

```

bipartitecheck

2.5 ciclonegativobf

```

1  // This uses Bellmanford algorithm to find a negative cycle
2  // O(n*m) m=edges, n=nodes
3  void test_case() {
4      ll n, m;
5      cin >> n >> m;
6      vector<ll> dist(n+1);
7      vector<ll> p(n+1);
8      vector<tuple<ll,ll,ll>> edges(m);
9      for (int i=0; i < m; i++) {
10         ll x, y, z;
11         cin >> x >> y >> z;

```

```

12     edges[i] = {x, y, z};
13 }
14
15 ll efe = -1;
16 for (int i = 0; i < n; i++) {
17     efe = -1;
18     for (auto pp : edges) {
19         ll x,y,z;
20         tie(x,y,z) = pp;
21         if (dist[x] + z < dist[y]) {
22             dist[y] = dist[x] + z;
23             p[y] = x;
24             efe = y;
25         }
26     }
27 }
28 if (efe == -1) {
29     cout << "NO\n";
30 } else {
31     cout << "YES\n";
32     ll x = efe;
33     for (int i = 0; i < n; i++) {
34         x = p[x];
35     }
36     vector<ll> cycle;
37     ll y = x;
38     while (cycle.size() == 0 || y != x) {
39         cycle.pb(y);
40         y = p[y];
41     }
42     cycle.pb(x);
43     reverse(all(cycle));
44     for (int i = 0; i < cycle.size(); i++) {
45         cout << cycle[i] << ' ';
46     }
47 }
48 }

```

ciclonegativobf

2.6 detectarciclosdirigido

```

1 vector<vector<int>> adj(2e5+5);
2 vector<int> visited(2e5);
3 bool ok = false; // if cycle was found ok is true
4 vector<int> cycle;
5 void dfs(int x, vector<int> &st) {
6     if (ok || visited[x] == 2) {
7         return;
8     } else if (visited[x] == 1) {
9         cycle.pb(x);
10        while (st.back() != x) {
11            cycle.pb(st.back());
12            st.pop_back();
13        }
14        cycle.pb(x);
15        reverse(aint(cycle));
16        ok = true;

```

```

17         return;
18     }
19     visited[x] = 1;
20     st.pb(x);
21     for (auto y : adj[x]) {
22         dfs(y, st);
23     }
24     st.pop_back();
25     visited[x] = 2;
26 }
27
28 void test_case() {
29     int n, m;
30     cin >> n >> m;
31
32     for (int i = 0; i < m; i++) {
33         int x, y;
34         cin >> x >> y;
35         adj[x].pb(y);
36     }
37
38     vector<int> st;
39     for (int i = 1; i <= n; i++) {
40         dfs(i, st);
41     }
42
43     if (ok) {
44         //haycilo e imprimir
45     }
46 }

```

detectarciclosdirigido

2.7 dijkstra

```

1 vector<int> dijkstra(vector<vector<pair<int, int>>> &grafo, int n, int origen) {
2     vector<int> distancia(n + 1, INF);
3     distancia[origen] = 0;
4     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
5     pq.push({0, origen});
6
7     // vector<int> padre(n + 1);
8     while (!pq.empty()) {
9         auto [peso, nodo] = pq.top();
10        pq.pop();
11        if (distancia[nodo] < peso) {
12            continue;
13        }
14
15        for (auto vecino : grafo[nodo]) {
16            int destino = vecino.first;
17            int costo = vecino.second;
18
19            if (distancia[nodo] + costo < distancia[destino]) {
20                distancia[destino] = distancia[nodo] + costo;
21                pq.push({distancia[destino], destino});
22                // padre[destino] = nodo;

```

```

23     }
24     }
25 }
26
27     return distancia;
28 }

```

dijkstra

2.8 floydwarshall

```

1  const int inf = 1e9;
2  vector<vector<pair<int, int>>> adj;
3  int distance[n][n];
4
5  void floydWarshaint() {
6      for (int i = 0; i < n; i++) {
7          for (int j = 0; j < n; j++) {
8              distance[i][j] = inf;
9          }
10     }
11     for (int i = 0; i < n; i++) {
12         for (auto p : adj[i]) {
13             int b = p.first;
14             int w = p.second;
15             distance[i][b] = w;
16         }
17     }
18     for (int k = 0; k < n; k++) {
19         for (int i = 0; i < n; i++) {
20             for (int j = 0; j < n; j++) {
21                 distance[i][j] = min(distance[i][j], distance[i][k]
22                                     + distance[k][j]);
23             }
24         }
25     }
26 }

```

floydwarshall

2.9 kruskal

```

1  int kruskal(vector<tuple<int,int,int>> &edges, int nodes) {
2      union_find uf(nodes+1);
3      //peso, u v
4      sort(all(edges));
5      // reverse(aint(edges)); // for maxst
6      int answer = 0;
7      for (auto edge : edges) {
8          int cost, a, b;
9          tie(cost, a, b) = edge;
10         if (uf.conectar(a, b))
11             answer += cost;
12     }
13     return answer;
14 }

```


2.10 kthsspath

```

1  // Using djisktra, finds the k shortesth paths from 1 to n
2  // 2    n10    ^5, 1    m210    ^5, 1    weight10    ^9, 1    k10
3  // complexity seems O(k*m)
4  #define P pair<int,int>
5  void test_case() {
6      int n, m, k;
7      cin >> n >> m >> k;
8      vector<int> visited(n+1, 0);
9      vector<vector<pair<int,int>>> adj(n+1);
10     for (int i = 0; i < m; i++) {
11         int a, b, c;
12         cin >> a >> b >> c;
13         adj[a].pb({b, c});
14     }
15     vector<int> ans;
16     priority_queue<P, vector<P>, greater<P>> q;
17     q.push({0, 1});
18     int kk = k;
19     while (q.size()) {
20         int x = q.top().S;
21         int z = q.top().F;
22         q.pop();
23         if (visited[x] >= kk) {
24             continue;
25         }
26         visited[x]++;
27         if (x == n) {
28             ans.pb(z);
29             k--;
30             if (k == 0) break;
31         }
32         for (auto yy : adj[x]) {
33             q.push({yy.S + z, yy.F});
34         }
35     }
36     for (int i = 0; i < ans.size(); i++) {
37         cout << ans[i]<< ' ';
38     }
39 }

```

kthsspath

2.11 multliavabfs

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  #define int long long
4
5  int n, m;
6  vector<pair<int, int>> monstruos;
7  vector<vector<int>> lava(1000 + 10, vector<int>(1000 + 10, INT_MAX));

```

```

8  pair<int, int> inicio, fin;
9  // 0 = U   1 = D   2 = R   3 = L
10 // Estos índices se usan para calc
11 vector<pair<int, int>> mov = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
12 char calc(int x) {
13     char res;
14     switch (x) {
15         case 0:
16             res = 'U';
17             break;
18         case 1:
19             res = 'D';
20             break;
21         case 2:
22             res = 'R';
23             break;
24         case 3:
25             res = 'L';
26             break;
27     }
28     return res;
29 }
30 vector<vector<char>> direcciones(1000 + 10, vector<char>(1000 + 10, '#'));
31 void reconstruircamino() {
32     vector<char> res;
33     auto [x, y] = fin;
34     while (direcciones[x][y] != '$') {
35         char aux = direcciones[x][y];
36         res.push_back(aux);
37         if (aux == 'U') {
38             x++;
39         } else if (aux == 'D') {
40             x--;
41         } else if (aux == 'L') {
42             y++;
43         } else {
44             y--;
45         }
46     }
47     cout << (int)res.size() << '\n';
48     reverse(res.begin(), res.end());
49     for (char& it : res) { cout << it; }
50     cout << '\n';
51 }
52 bool esvalida(int x, int y, int tiempo) {
53     if (x < 0 or y < 0 or x >= n or y >= m) { return false; }
54     if (lava[x][y] <= tiempo) { return false; }
55     return true;
56 }
57
58 bool esSalida(int x, int y, int tiempo) {
59     if (!esvalida(x, y, tiempo)) return false;
60     if (x == 0 or y == 0 or x == n - 1 or y == m - 1) return true;
61     return false;
62 }
63
64 bool bfsDeSalida() {
65     queue<pair<pair<int, int>, int>> q;
66     q.push(make_pair(inicio, 0));

```

```

67     direcciones[inicio.first][inicio.second] = '$';
68     while (!q.empty()) {
69         int cx = q.front().first.first;
70         int cy = q.front().first.second;
71         int tiempo = q.front().second;
72         tiempo++;
73         q.pop();
74         for (int i = 0; i < 4; i++) {
75             auto mv = mov[i];
76             int tx = cx + mv.first;
77             int ty = cy + mv.second;
78             if (esSalida(tx, ty, tiempo)) {
79                 direcciones[tx][ty] = calc(i);
80                 fin = { tx, ty };
81                 return true;
82             }
83             if (esvalida(tx, ty, tiempo)) {
84                 direcciones[tx][ty] = calc(i);
85                 lava[tx][ty] = tiempo;
86                 q.push({ { tx, ty }, tiempo });
87             }
88         }
89     }
90     return false;
91 }
92
93 void precalculoLava() {
94     queue<pair<pair<int, int>, int>> q;
95     for (auto m : monstruos) { q.push(make_pair(m, 0)); }
96     while (!q.empty()) {
97         int cx = q.front().first.first;
98         int cy = q.front().first.second;
99         int tiempo = q.front().second;
100         tiempo++;
101         q.pop();
102
103         for (auto mv : mov) {
104             int tx = cx + mv.first;
105             int ty = cy + mv.second;
106             if (esvalida(tx, ty, tiempo)) {
107                 lava[tx][ty] = tiempo;
108                 q.push({ { tx, ty }, tiempo });
109             }
110         }
111     }
112 }
113
114 signed main() {
115     ios_base::sync_with_stdio(false);
116     cin.tie(NULL);
117     cin >> n >> m;
118     for (int i = 0; i < n; ++i) {
119         for (int j = 0; j < m; ++j) {
120             char c;
121             cin >> c;
122             if (c == '#') {
123                 lava[i][j] = 0;
124             } else if (c == 'M') {
125                 lava[i][j] = 0;

```

```

126         monstruos.push_back({ i, j });
127     } else if (c == 'A') {
128         lava[i][j] = 0;
129         inicio = { i, j };
130     } else {
131         lava[i][j] = INT_MAX;
132     }
133 }
134 }
135 }
136 if (inicio.first == 0 or inicio.second == 0 or inicio.first == n - 1 or
137     inicio.second == m - 1) {
138     cout << "YES" << endl;
139     cout << 0;
140     return 0;
141 }
142 precalculoLava();
143
144 if (!bfsDeSalida()) {
145     cout << "NO";
146     return 0;
147 }
148 cout << "YES" << endl;
149 reconstruircamino();
150 }

```

multiavabfs

2.12 ssspnegativo

```

1 // Find the minimum distance from any i to j, with negative weights.
2 // dist[i][j] == -inf, there some negative loop from i to j
3 // dist[i][j] == inf, from i cannot reach j
4 // otherwise the min dist from i to j
5
6 // take care of the max a path from i to j, it has to be less than inf
7 const ll inf = INT32_MAX;
8 void test_case() {
9     ll n, m; // nodes, edges
10    vector<vector<ll>> dist(n, vector<ll>(n, inf));
11    for (int i = 0; i < n; i++) dist[i][i] = 0;
12    for (int i = 0; i < m; i++) {
13        ll a, b, w;
14        cin >> a >> b >> w; // negative weights
15        dist[a][b] = min(dist[a][b], w);
16    }
17    // floyd warshall
18    for (int k = 0; k < n; k++) {
19        for (int i = 0; i < n; i++) {
20            for (int j = 0; j < n; j++) {
21                if (dist[i][k] == inf || dist[k][j] == inf) continue;
22                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
23            }
24        }
25    }
26    // find negative cycles for a node
27    for (int i = 0; i < n; i++) {
28        if (dist[i][i] < 0) dist[i][i] = -inf;

```

```

29     }
30     // find negative cycles between a routes from i to j
31     for (int i = 0; i < n; i++) {
32         for (int j = 0; j < n ; j++) {
33             for (int k = 0; k < n; k++) {
34                 if (dist[k][k] < 0 && dist[i][k] != inf && dist[k][j] != inf) {
35                     dist[i][j] = -inf;
36                 }
37             }
38         }
39     }
40 }

```

ssspnegativo

2.13 topsort

```

1  const int N = 1e5;
2  vector<vector<ll>> adj(N + 10);
3  vector<ll> visited(N +10);
4  bool cycle = false; // reports if doesn't exists a topological sort
5  vector<ll> topo;
6
7  void dfs(ll x) {
8      if (visited[x] == 2) {
9          return;
10     } else if (visited[x] == 1) {
11         cycle = true;
12         return;
13     }
14     visited[x] = 1;
15     for (auto y : adj[x]) {
16         dfs(y);
17     }
18     visited[x] = 2;
19     topo.pb(x);
20 }
21
22 void test_case() {
23     ll n, m;
24     cin >> n >> m;
25     for (int i =0; i < m; i++) {
26         ll x, y;
27         cin >> x >> y;
28         adj[x].pb(y);
29     }
30     for (int i = 1; i <= n; i++) {
31         dfs(i);
32     }
33     reverse(topo.begin(), topo.end());
34     if (cycle) {
35         cout << "IMPOSSIBLE\n";
36     } else {
37         for (int i =0; i < n; i++) {
38             cout << topo[i] << " \n" [i == n - 1];
39         }
40     }
41 }

```

2.14 unionfind

```

1 struct union_find {
2     vi link;
3     vi score;
4     vi size;
5     int n;
6     void init(int nn) {
7         link.resize(nn);
8         score.resize(nn);
9         size.resize(nn);
10        this->n = nn;
11        for (int i = 0; i < n; i++) {
12            link[i] = i;
13            score[i] = 0;
14            size[i] = 1;
15        }
16    }
17    int find(int x) {
18        if (link[x] == x) return x;
19        return (link[x] = find(link[x]));
20    }
21    void group(int a, int b) {
22        int pa = find(a);
23        int pb = find(b);
24        if (pa != pb) {
25            if (score[pa] >= score[pb]) {
26                link[pb] = pa;
27                size[pa] += size[pb];
28                if (score[pa] == score[pb]) score[pa]++;
29            } else {
30                link[pa] = pb;
31                size[pb] += size[pa];
32            }
33        }
34    }
35 };

```

unionfind

3 DP

3.1 Knapsack

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 //dp[i][j] = max(dp[i-1][j], dp[i-1][j-value])
5 void solve(){
6     int n,x; cin>>n>>x;
7     vector<int> v(n);
8     vector<int> w(n);

```

```

9      for(int i = 0; i < n; i++)cin>>w[i];
10     for(int i = 0; i < n; i++)cin>>v[i];
11     vector<vector<int>> dp(n+1, vector<int> (x+1, 0));
12     for(int i = 1; i <= n; i++){
13         for(int j = 0; j <= x; j++){
14             int w1 = w[i-1];
15             int v1 = v[i-1];
16             int t = (j >= w1? dp[i-1][j-w1]+v1:0);
17             int nt = dp[i-1][j];
18             dp[i][j] = max(t, nt);
19         }
20     }
21     cout<<dp[n][x]<<endl;
22 }
23
24 signed main() {
25     solve();
26     return 0;
27 }

```

Knapsack

3.2 KnapsackOptimization

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  //Knapsack Optimization memory O(2*W)
5  void solve(){
6      int n,x; cin>>n>>x;
7      vector<int> v(n);
8      vector<int> w(n);
9      for(int i = 0; i < n; i++)cin>>w[i];
10     for(int i = 0; i < n; i++)cin>>v[i];
11     vector<int> ant(x+1, 0);
12     for(int i = 1; i <= n; i++){
13         vector<int> act(x+1, 0);
14         for(int j = 0; j <= x; j++){
15             int w1 = w[i-1];
16             int v1 = v[i-1];
17             int t = (j >= w1? ant[j-w1]+v1:0); //dp[i-1][j-value]
18             int nt = ant[j]; //dp[i-1][j]
19             act[j]= max(t, nt);
20         }
21         ant = act;
22     }
23     cout<<ant[x]<<endl;
24 }
25
26 signed main() {
27     solve();
28     return 0;
29 }

```

KnapsackOptimization

3.3 LCS

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int const N = 1000;
5  int memo[N][N];
6  pair<int, int> path[N+1][N+1];
7  //-1 no calculated
8  int dp(string &s1, string &s2, int l, int r){
9      if(l >= (int)s1.size() || r >= (int)s2.size()){
10         path[l][r] = {-1, -1}; return 0;
11     }
12     int &a = memo[l][r];
13     if(a == -1){
14         a = 0;
15         if(s1[l] == s2[r]){
16             path[l][r] = {l+1, r+1};
17             a = dp(s1,s2,l+1, r+1)+1;
18         }else{
19             int op1 = dp(s1,s2,l+1,r);
20             int op2 = dp(s1,s2,l,r+1);
21             if(op1 > op2) path[l][r] = {l+1,r};
22             else path[l][r] = {l, r+1};
23             a = max(op1, op2);
24         }
25     }
26     return a;
27 }
28
29 // Construct answer after DP
30 string construct(string &s1, string &s2) {
31     int i, j;
32     i = j = 0;
33     string ans;
34     while (i != -1 && j != -1) {
35         if (s1[i] == s2[j]) {
36             ans.push_back(s1[i]);
37         }
38         pair<int, int> aux = path[i][j];
39         i = aux.first;
40         j = aux.second;
41     }
42     return ans;
43 }
44
45
46 signed main(){
47     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
48     string s1,s2;cin>>s1>>s2;
49     memset(memo, -1, sizeof(memo));
50     cout<<dp(s1,s2,0,0)<<endl;
51     cout<<construct(s1, s2)<<endl;
52     return 0;
53 }

```

LCS

3.4 LIS

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4
5 vector<int> lis(vector<int> &v){
6     int n = v.size();
7     vector<int> dp;
8     dp.push_back(-1e9);
9     vector<int> curr(n);
10    for (int i = 0; i < n; i++) {
11        int l = 0, r = dp.size()-1;
12        int pos = dp.size();
13        while (l <= r) {
14            int m = l + (r-l) / 2;
15            if(dp[m] >= v[i]){
16                pos = m;
17                r = m - 1;
18            }else{
19                l = m + 1;
20            }
21        }
22        curr[i] = pos;
23        if (pos == dp.size()) {
24            dp.push_back(v[i]);
25        } else {
26            dp[pos] = v[i];
27        }
28    }
29    vector<int> ans;
30    int x = dp.size() - 1;
31    for (int i = n - 1; i >= 0; i--) {
32        if (curr[i] == x) {
33            ans.push_back(v[i]);
34            x--;
35        }
36    }
37    reverse(ans.begin(), ans.end());
38    return ans;
39 }
40
41
42 signed main() {
43     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
44     int n; cin>>n; vector<int> arr(n);
45     for(int i = 0; i < n; i++) cin>>arr[i];
46     vector<int> ans = lis(arr);
47     cout<<ans.size()<<endl;
48     for(auto x: ans){
49         cout<<x<<" ";
50     }
51     return 0;
52 }
```

LIS

3.5 MCM(MatrixChainMultiplicated)

```
1 #include<bits/stdc++.h>
2 #define int long long
3 using namespace std;
4
5 int const N = 105;
6 int memo[N][N];
7 int dp(vector<int> &arr, int l, int r){
8     if(l+1 == r) return 0; //one matrix
9     if(l+2 == r) return arr[l]*arr[l+1]*arr[l+2]; //two matrix
10    int &a = memo[l][r];
11    if(a == -1){
12        a = 1e17;
13        for(int i = l+1; i < r; i++){
14            int precio = arr[l]*arr[i]*arr[r];
15            a = min(a, dp(arr,l,i)+dp(arr,i,r)+precio);
16        }
17    }
18    return a;
19 }
20
21 signed main(){
22     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
23     int n; cin>>n;
24     vector<int> arr(n);
25     for(int i = 0; i < n; i++) cin>>arr[i];
26     memset(memo, -1, sizeof(memo));
27     cout<<dp(arr,0, n-1);
28     return 0;
29 }
```

MCM(MatrixChainMultiplicated)

3.6 MCMMaplicationn

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int memo1[501][501];
5 int memo2[501][501];
6
7 int dp(int l, int r, vector<int> &arr){
8     if(l+1 == r && arr[r] != arr[l]) return 0;
9     if(l+1 == r && arr[r] == arr[l]) return arr[l]+1;
10    if(l == r) return arr[l];
11    int &a = memo1[l][r];
12    if(a == -1){
13        a = 0;
14        for(int i = l; i < r; i++){
15            memo1[l][i] = dp(l, i, arr);
16            memo1[i+1][r] = dp(i+1, r, arr);
17            if(memo1[l][i] == memo1[i+1][r] && memo1[l][i] != 0){
18                memo1[l][r] = memo1[l][i]+1;
19            }
20        }
21    }
22    return a;
23 }
```

```

23 }
24
25 int dp2(int l, int r, vector<int> &arr){
26     if (l == r) return 1;
27     if (memo2[l][r] != -1) return memo2[l][r];
28     if (memo1[l][r] != 0) return memo2[l][r] = 1;
29     int mini = r - l + 1;
30     for (int i = l; i < r; ++i) {
31         mini = min(mini, dp2(l, i, arr) + dp2(i + 1, r, arr));
32     }
33     return memo2[l][r] = mini;
34 }
35
36 signed main(){
37     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
38     memset(memo1, -1, sizeof(memo1));
39     memset(memo2, -1, sizeof(memo2));
40     int n; cin>>n;
41     vector<int>arr(n+1);
42     for(int i = 1; i <= n; i++)cin>>arr[i];
43     dp(1, n, arr);
44     cout<<dp2(1, n, arr);
45     return 0;
46 }

```

MCMApplicationn

3.7 OptimalGridTraversal

```

1  #include<bits/stdc++.h>
2
3  using namespace std;
4  int const MAX_N = 100;
5  int const MAX_M = 100;
6  int G[MAX_N][MAX_M];
7  int n,m;
8  int memo[MAX_N][MAX_M];
9
10 int f(int i, int j){
11     int &a = memo[i][j];
12     if(a != -1) return a;
13     if(i == n-1 && j == m-1) a = G[i][j];
14     else{
15         a = 0;
16         if(i < n-1) a = max(a, G[i][j]+f(i+1,j));
17         if(j < m-1) a = max(a, G[i][j]+f(i,j+1));
18     }
19     return a;
20 }
21
22 //llamar despues de llamar a la funcion f
23 string ans = "";
24 void caminoOptimo(int i, int j){
25     if(i == n-1 && j == m-1) return;
26     if(i < n-1 && memo[i][j] == G[i][j]+memo[i+1][j]){
27         ans+='A'; //hacia abajo
28         caminoOptimo(i+1,j);
29     }else{

```

```

30         ans += 'D'; //hacia derecha
31         caminoOptimo(i,j+1);
32     }
33 }
34 //igualmente despues de llamar a la funcion f
35 int cont[MAX_N][MAX_M];
36 int cant(){
37     for(int i = n-1; i >= 0; --i){
38         for(int j = m-1; j>= 0; --j){
39             if(i == n-1 && j == m-1) cont[i][j] = 1;
40             else{
41                 cont[i][j] = 0;
42                 if(i < n-1 && memo[i][j] == G[i][j]+memo[i+1][j])
43                     cont[i][j] += cont[i+1][j];
44                 if(j < m-1 && memo[i][j] == G[i][j]+memo[i][j+1])
45                     cont[i][j] += cont[i][j+1];
46             }
47         }
48     }
49     return cont[0][0];
50 }
51
52 string sol = "";
53 //usa el contar solucion y la funcion f
54 void getk(int i, int j, int k) {
55     if (i == n-1 && j == m-1) return;
56     if (i < n-1 && memo[i][j] == G[i][j] + memo[i+1][j]){
57         if(cont[i+1][j] >= k){
58             sol += 'A';
59             getk(i+1, j, k);
60             return;
61         }
62         k -= cont[i+1][j];
63     }
64     if (j < m-1 && memo[i][j] == G[i][j] + memo[i][j+1]) {
65         sol += 'D';
66         getk(i, j+1, k);
67     }
68 }
69 signed main(){
70     memset(memo, -1, sizeof(memo));
71     cin>>n>>m;
72     for(int i = 0; i < n; i++){
73         for(int j = 0; j < m; j++){
74             cin>>G[i][j];
75         }
76     }
77     cout<<f(0,0)<<endl;
78     caminoOptimo(0,0);
79     cout<<ans<<endl;
80     cout<<cant()<<endl;
81     getk(0,0,3);
82     cout<<sol<<endl;
83 }

```

OptimalGridTraversal

3.8 OptimalGridTraversalOptimization

```
1 #include<bits/stdc++.h>
2
3 using namespace std;
4 int const MAX_N = 100;
5 int const MAX_M = 100;
6 int G[MAX_N][MAX_M];
7 int n,m;
8 int memo[2][MAX_M];
9
10 int calc_dp_optimized(){
11     for(int i = n-1; i>= 0; i--){
12         for(int j = m-1; j >= 0; --j){
13             if(i == n-1 && j == m-1) memo[i%2][j] = G[i][j];
14             else{
15                 memo[i%2][j] = 0;
16                 if(i < n-1){
17                     memo[i%2][j] = max(memo[i%2][j], G[i][j]+memo[1-i%2][j]);
18                 }
19                 if(j < m-1){
20                     memo[i%2][j] = max(memo[i%2][j], G[i][j]+memo[i%2][j+1]);
21                 }
22             }
23         }
24     }
25     return memo[0][0];
26 }
27
28 signed main(){
29     memset(memo, -1, sizeof(memo));
30     cin>>n>>m;
31     for(int i = 0; i < n; i++){
32         for(int j = 0; j < m; j++){
33             cin>>G[i][j];
34         }
35     }
36     cout<<calc_dp_optimized()<<endl;
37 }
```

OptimalGridTraversalOptimization

3.9 SOS

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void sos() {
5     int n; cin>>n;
6     vector<int> arr(1<<n);
7     for(int i = 0; i <(1<<n); i++) cin>>arr[i];
8     int sos[1 << n] = {0};
9     for (int x = 0; x < (1 << n); x++) {
10         sos[x] = arr[0];
11         for (int i = x; i > 0; i = (i - 1) & x)
12             sos[x] += arr[i];
13     }
14 }
```

```

15     for(int i = 0; i <(1<<n); i++) {
16         cout<<sos[i]<< " ";
17     }
18 }
19
20 signed main(){
21     sos();
22     return 0;
23 }

```

SOS

4 sortingsearchinggreedy

4.1 bisecciongeneral

```

1 double bisection(double a, double b) {
2     if (eval(a) * eval(b) >= 0) {
3         // no establecemos bien los limites de evaluacion
4         return -INF;
5     }
6
7     double c = a;
8     while ((b - a) >= EPS) {
9         // punto medio:
10        c = (a + b) / 2;
11
12        // si encontramos justo la solucion
13        if (eval(c) == 0.0)
14            break;
15
16        // biseccion:
17        else if (eval(c) * eval(a) < 0)
18            b = c;
19        else
20            a = c;
21    }
22    // el valor de una raiz esta en c
23    return c;
24 }

```

bisecciongeneral

4.2 cantdesubarrsumax

```

1 int cantdesubarrsumaequis(vector<int>& arr, int x, int n)
2 {
3     // Map to store the frequency of prefix sums
4     map<int, int> prefSums;
5
6     prefSums[0] = 1;
7     int pref = 0;
8     int cnt = 0;
9
10    // Calculate the prefix sum at every index, and find the
11    // count of subarrays with sum = pref - x

```

```

12     for (int i = 0; i < n; ++i) {
13         pref += arr[i];
14         cnt += (prefSums[pref - x]);
15         prefSums[pref]++;
16     }
17     return cnt;
18 }

```

cantdesubarrsumax

4.3 cantidaddesumasdistintas

```

1  //cantidad de sumas distintas de exactamente k de longitud (equivalente a cantidad
   de sumas ordenadas)
2  int dp(int n, int k, const vector<int>& arr) {
3      int m = arr.size();
4
5      // Crear una tabla DP para almacenar los resultados
6      vector<vector<vector<int>>> dp(n + 1, vector<vector<int>>(k + 1, vector<int>(m
   + 1, 0)));
7
8      // Caso base: Si x == 0 y k == 0, hay exactamente 1 forma (no seleccionar
   ning n n mero)
9      for (int i = 0; i <= m; i++) {
10         dp[0][0][i] = 1; // 1 forma de hacer suma 0 con 0 elementos
11     }
12
13     // Rellenar la tabla iterativamente
14     for (int x = 1; x <= n; x++) {
15         for (int kk = 1; kk <= k; kk++) {
16             for (int i = 0; i < m; i++) {
17                 // Opci n 1: No tomar el n mero arr[i], simplemente tomamos el
   valor de la fila anterior
18                 dp[x][kk][i + 1] = dp[x][kk][i];
19
20                 // Opci n 2: Tomar el n mero arr[i], restamos arr[i] de x y
   reducimos k
21                 if (x >= arr[i]) {
22                     dp[x][kk][i + 1] += dp[x - arr[i]][kk - 1][i];
23                 }
24             }
25         }
26     }
27     // La respuesta final est en dp[n][k][m], que es la forma de sumar n con k
   n meros usando todos los numeros
28     return dp[n][k][m];
29 }
30
31 //reduccion de memoria (NO COMPROBADO)
32 int dp(int n, int k, const vector<int>& arr) {
33     int m = arr.size();
34
35     // Crear una tabla 2D de DP: dp[x][k] donde x es la suma y k es el n mero de
   elementos seleccionados
36     vector<vector<int>> dp(n + 1, vector<int>(k + 1, 0));
37
38     // Caso base: hay exactamente 1 forma de hacer suma 0 con 0 elementos
39

```

```

40 dp[0][0] = 1;
41
42 // Rellenar la tabla iterativamente
43 for (int i = 0; i < m; i++) { // Iteramos sobre cada elemento en arr
44     // Iteramos hacia atr s para no sobrescribir los valores de la misma
45     // iteraci n
46     for (int x = n; x >= arr[i]; x--) { // Solo consideramos sumas donde x >=
47         // arr[i]
48         for (int kk = k; kk >= 1; kk--) { // Solo consideramos hasta k
49             // elementos
50             dp[x][kk] += dp[x - arr[i]][kk - 1];
51         }
52     }
53 }
54
55 // La respuesta final est en dp[n][k], que es la forma de sumar n con k
56 // elementos
57 return dp[n][k];
58 }

```

cantidaddesumasdistintas

4.4 cantsubarraydistintos

```

1 int cantsubarraydistinct(int* arr, int N, int K) {
2     // left and right pointers to mark the start and end of
3     // the sliding window
4     int left = 0, right = 0;
5     // Variable to count how many different numbers we have
6     // in the window
7     int distinct_count = 0;
8     // Variable to store the final result
9     int result = 0;
10
11     // Map to keep track of how many times each number
12     // appears in the window
13     unordered_map<int, int> frequency;
14
15     // Slide the window tiint the window tiint the right
16     // pointer does not reach the end of the array
17     while (right < N) {
18         // Check if the current number is new or if its
19         // count is zero
20         if (frequency.find(arr[right]) == frequency.end() ||
21             frequency[arr[right]] == 0)
22             distinct_count++;
23
24         // Update the count of the current number
25         frequency[arr[right]]++;
26
27         // If there are more than K distinct numbers, shrink
28         // the window from the left
29         while (distinct_count > K) {
30             // Decrease the count of the number going out of
31             // the window
32             frequency[arr[left]]--;
33             // If its count becomes zero, then there wiint be
34             // one less distinct number in the window

```



```

35         if (frequency[arr[left]] == 0) distinct_count--;
36         // Move the left pointer to the right to shrink
37         // the window
38         left++;
39     }
40
41     // Calculate the number of subarrays that end at the
42     // current position
43     result += right - left + 1;
44
45     // Move the right edge of the window to the right to
46     // expand it
47     right++;
48 }
49 // Return the result
50 return result;
51 }

```

cantsubarraydistintos

4.5 cantsubarrdiv

```

1 // Function to count the number of subarrays divisible by n
2 int solve(vector<int>& arr, int n)
3 {
4     // Map to store the frequency of prefix sums % n
5     map<int, int> contResid;
6
7     contResid[0] += 1;
8     int residuo = 0;
9     int cnt = 0;
10
11     // Iterate over aint the index and add the count of
12     // subarrays with sum divisible by n
13     for (int i = 0; i < n; ++i) {
14         // Since arr[i] can be negative, we add n to the
15         // residuo to avoid negative residuos
16         residuo = ((residuo + arr[i]) % n + n) % n;
17         cnt += contResid[residuo];
18         contResid[residuo] += 1;
19     }
20     return cnt;
21 }

```

cantsubarrdiv

4.6 maxsubarraysumentreab

```

1 void MaximumSubarraySumentreab(int N, int A, int B, vector<int>& arr) {
2     // Initialize a deque to store indices in increasing
3     // order of prefix sum values
4     deque<int> dq;
5     // Initialize a prefixSum array to store cumulative sums
6     vector<int> prefixSum(N + 1);
7     // Initialize the answer to track the maximum sum
8     int ans = intONG_MIN;
9     // Calculate cumulative sums

```

```

10     for (int i = 1; i <= N; i++) {
11         prefixSum[i] += prefixSum[i - 1] + arr[i - 1];
12     }
13     // Loop through the first (B-1) indices to initialize
14     // deque
15     for (int i = 1; i < B; i++) {
16         // Maintain deque in increasing order of prefix sum
17         // values
18         while (!dq.empty() && prefixSum[dq.front()] <= prefixSum[i]) {
19             dq.pop_front();
20         }
21         dq.push_front(i);
22     }
23     // Loop through each starting index i from 0 to (n-a)
24     for (int i = 0; i <= (N - A); i++) {
25         // Maintain deque in increasing order of prefix sum
26         // values
27         while (i + B <= N && !dq.empty() &&
28             prefixSum[dq.front()] <= prefixSum[i + B]) {
29             dq.pop_front();
30         }
31         // Push the right end index to the front of deque
32         if (i + B <= N) dq.push_front(i + B);
33         // If the index of maximum element outside the
34         // current window , pop elements from the back of
35         // the deque until the back index(index of maximum
36         // element) is within the current window.
37         while (!dq.empty() && dq.back() < (A + i)) { dq.pop_back(); }
38         // Update the answer by taking the maximum of the
39         // current answer and the difference between the
40         // prefix sum at the back(maximum element) of the
41         // deque and the prefix sum at index i
42         ans = max(ans, prefixSum[dq.back()] - prefixSum[i]);
43     }
44     // Print the final answer
45     cout << ans << "\n";
46 }

```

maxsubarraysumentreab

4.7 secuencialargaunica

```

1 // la longitud del subarray con numeros sucesivos mas larga (sin repetir
2 // numeros)
3 int LI-SUBARRAYsinrepetir(int n, vector<int> arr) {
4     int l = 0, ans = 0;
5     // mapa para guardar la ultima ocurrencia de un numero
6     map<int, int> mp;
7
8     // two pointers
9     for (int r = 0; r < n; r++) {
10         // Si el elemento actual no esta en el mapa
11         if (mp.find(arr[r]) == mp.end())
12             mp.insert({ arr[r], r });
13         else {
14             // if el numero actual esta en el mapa y esta en la ventana
15             if (mp[arr[r]] >= l) l = mp[arr[r]] + 1;
16             // actualizar la ultima ocurrencia del caracter en el indice actual

```

```

17         mp[arr[r]] = r;
18     }
19     //maximizar la respuesta
20     ans = max(ans, r - l + 1);
21 }
22 return ans;
23 }

```

secuencialargaunica

4.8 sumade4valroes

```

1  // function to find a quadruplet whose sum = X
2  void suma4(vector<int> &arr, int X, int N) {
3      // vector to store the values along with their indices
4      vector<vector<int>> vec(N, vector<int>(2));
5
6      for (int i = 0; i < N; i++) {
7          vec[i][0] = arr[i];
8          vec[i][1] = i + 1;
9      }
10
11     // Sort the vector in increasing order of the values
12     sort(vec.begin(), vec.end());
13
14     // Iterate for aint possible values of first element
15     for (int ptr1 = 0; ptr1 < N - 3; ptr1++) {
16         // Iterate for aint possible values of second element
17         for (int ptr2 = ptr1 + 1; ptr2 < N - 2; ptr2++) {
18             // Maintain two pointers for the third and
19             // fourth element
20             int ptr3 = ptr2 + 1;
21             int ptr4 = N - 1;
22             while (ptr3 < ptr4) {
23                 int currentSum =
24                     vec[ptr1][0] + vec[ptr2][0] + vec[ptr3][0] + vec[ptr4][0];
25                 if (currentSum == X) {
26                     cout << vec[ptr1][1] << " " << vec[ptr2][1] << " "
27                         << vec[ptr3][1] << " " << vec[ptr4][1] << "\n";
28                     return;
29                 }
30                 // Decrease the currentSum by moving ptr4 to
31                 // ptr4 - 1
32                 else if (currentSum > X) {
33                     ptr4--;
34                 } else if (currentSum < X) {
35                     ptr3++;
36                 }
37             }
38         }
39     }
40     cout << "IMPOSSIBLE";
41 }

```

sumade4valroes

5 Math

5.1 convexHullGrahamScan

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  struct Point {
5      int x, y;
6  };
7
8  // to the first point Used in compare function of qsort()
9  Point p0;
10
11 // A utility function to find next to top in a stack
12 Point nextToTop(stack<Point> &S) {
13     Point p = S.top();
14     S.pop();
15     Point res = S.top();
16     S.push(p);
17     return res;
18 }
19
20 // A utility function to swap two points
21 void swap(Point &p1, Point &p2) {
22     Point temp = p1;
23     p1 = p2;
24     p2 = temp;
25 }
26
27 // A utility function to return square of distance
28 // between p1 and p2
29 int distSq(Point p1, Point p2) {
30     return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
31 }
32
33 // To find orientation of ordered triplet (p, q, r).
34 // The function returns following values
35 // 0 --> p, q and r are collinear
36 // 1 --> Clockwise
37 // 2 --> Counterclockwise
38 int orientation(Point p, Point q, Point r) {
39     int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
40
41     if (val == 0)
42         return 0; // collinear
43     return (val > 0) ? 1 : 2; // clock or counterclock wise
44 }
45
46 // A function used by library function qsort() to sort an array of
47 // points with respect to the first point
48 int compare(const void *vp1, const void *vp2) {
49     Point *p1 = (Point *)vp1;
50     Point *p2 = (Point *)vp2;
51
52     // Find orientation
53     int o = orientation(p0, *p1, *p2);
54     if (o == 0)
```

```

55         return (distSq(p0, *p2) >= distSq(p0, *p1)) ? -1 : 1;
56
57     return (o == 2) ? -1 : 1;
58 }
59
60 // Prints convex hull of a set of n points.
61 void convexHull(Point points[], int n) {
62     // Find the bottommost point
63     int ymin = points[0].y, min = 0;
64     for (int i = 1; i < n; i++) {
65         int y = points[i].y;
66
67         // Pick the bottom-most or choose the left
68         // most point in case of tie
69         if ((y < ymin) || (ymin == y && points[i].x < points[min].x))
70             ymin = points[i].y, min = i;
71     }
72
73     // Place the bottom-most point at first position
74     swap(points[0], points[min]);
75
76     // Sort n-1 points with respect to the first point.
77     // A point p1 comes before p2 in sorted output if p2
78     // has larger polar angle (in counterclockwise
79     // direction) than p1
80     p0 = points[0];
81     qsort(&points[1], n - 1, sizeof(Point), compare);
82
83     // If two or more points make same angle with p0,
84     // Remove all but the one that is farthest from p0
85     // Remember that, in above sorting, our criteria was
86     // to keep the farthest point at the end when more than
87     // one points have same angle.
88     int m = 1; // Initialize size of modified array
89     for (int i = 1; i < n; i++) {
90         // Keep removing i while angle of i and i+1 is same
91         // with respect to p0
92         while (i < n - 1 && orientation(p0, points[i], points[i + 1]) == 0)
93             i++;
94
95         points[m] = points[i];
96         m++; // Update size of modified array
97     }
98
99     // If modified array of points has less than 3 points,
100    // convex hull is not possible
101    if (m < 3)
102        return;
103
104    // Create an empty stack and push first three points
105    // to it.
106    stack<Point> S;
107    S.push(points[0]);
108    S.push(points[1]);
109    S.push(points[2]);
110
111    // Process remaining n-3 points
112    for (int i = 3; i < m; i++) {
113        // Keep removing top while the angle formed by

```

```

114         // points next-to-top, top, and points[i] makes
115         // a non-left turn
116         while (S.size() > 1 &&
117                orientation(nextToTop(S), S.top(), points[i]) != 2)
118             S.pop();
119         S.push(points[i]);
120     }
121 }
122 void solve() {
123     Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
124                      {0, 0}, {1, 2}, {3, 1}, {3, 3}};
125     int n = sizeof(points) / sizeof(points[0]);
126     convexHull(points, n);
127 }
128
129 signed main() {
130     std::ios::sync_with_stdio(false);
131     cin.tie(0);
132     int t;
133
134     t = 1;
135     // memset(dp, -1, sizeof(dp));
136     // cin >> t;
137     while (t--) {
138         solve();
139     }
140 }
141 void solve() {
142     //
143 }
144
145 signed main() {
146     std::ios::sync_with_stdio(false);
147     cin.tie(0);
148     int t;
149
150     t = 1;
151     // memset(dp, -1, sizeof(dp));
152     // cin >> t;
153     while (t--) {
154         solve();
155     }
156 }

```

convexHullGrahamScan

5.2 convexHullJarvisMarch

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 /**
6  .------.
7  /
8  /

```

```

9  /
10 /
11 /
12 /
13 '-----',
14 **/
15
16 // A C++ program to find convex hull of a set of points. Refer
17 // https://www.geeksforgeeks.org/orientation-3-ordered-points/
18 // for explanation of orientation()
19
20 struct Point {
21     int x, y;
22 };
23
24 // To find orientation of ordered triplet (p, q, r).
25 // The function returns following values
26 // 0 --> p, q and r are collinear
27 // 1 --> Clockwise
28 // 2 --> Counterclockwise
29 int orientation(Point p, Point q, Point r) {
30     int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
31
32     if (val == 0)
33         return 0; // collinear
34     return (val > 0) ? 1 : 2; // clock or counterclock wise
35 }
36
37 // Prints convex hull of a set of n points.
38 void convexHull(Point points[], int n) {
39     // There must be at least 3 points
40     if (n < 3)
41         return;
42
43     // Initialize Result
44     vector<Point> hull;
45
46     // Find the leftmost point
47     int l = 0;
48     for (int i = 1; i < n; i++)
49         if (points[i].x < points[l].x)
50             l = i;
51
52     // Start from leftmost point, keep moving counterclockwise
53     // until reach the start point again. This loop runs O(h)
54     // times where h is number of points in result or output.
55     int p = l, q;
56     do {
57         // Add current point to result
58         hull.push_back(points[p]);
59
60         // Search for a point 'q' such that orientation(p, q,
61         // x) is counterclockwise for all points 'x'. The idea
62         // is to keep track of last visited most counterclock-
63         // wise point in q. If any point 'i' is more counterclock-

```

```

64     // wise than q, then update q.
65     q = (p + 1) % n;
66     for (int i = 0; i < n; i++) {
67         // If i is more counterclockwise than current q, then
68         // update q
69         if (orientation(points[p], points[i], points[q]) == 2)
70             q = i;
71     }
72
73     // Now q is the most counterclockwise with respect to p
74     // Set p as q for next iteration, so that q is added to
75     // result 'hull'
76     p = q;
77
78     } while (p != 1); // While we don't come to first point
79
80     // Print Result
81     for (int i = 0; i < hull.size(); i++)
82         cout << "(" << hull[i].x << ", " << hull[i].y << ")\n";
83 }
84
85 // Driver program to test above functions
86 int main() {
87     Point points[] = {{0, 3}, {2, 2}, {1, 1}, {2, 1}, {3, 0}, {0, 0}, {3, 3}};
88     int n = sizeof(points) / sizeof(points[0]);
89     convexHull(points, n);
90     return 0;
91 }

```

convexHullJarvisMarch

6 AlgoritmosGeneral

6.1 Combinatorics

```

1  #include <bits/stdc++.h>
2  #define int long long
3
4  using namespace std;
5
6  const int MOD = 1e9 + 7;
7  const int N = 200005;
8  int fact[N];
9  int invfact[N];
10
11 int binpow(int a, int b, int m) {
12     int res = 1;
13     while (b > 0) {
14         if (b & 1) {
15             res = (res * a) % m;
16         }
17         a = (a * a) % m;
18         b >>= 1;
19     }
20     return res;
21 }
22

```



```

23 int inversoFermat(int a, int m){
24     return binpow(a, m-2, m);
25 }
26
27 void procesar(){
28     fact[0] = fact[1] = 1;
29     invfact[0] = invfact[1] = inversoFermat(1, MOD);
30     for(int i = 2; i < N; i++){
31         fact[i] = i*fact[i-1]%MOD;
32         invfact[i] = inversoFermat(fact[i], MOD);
33     }
34 }
35
36 int nCk(int n, int k){
37     if(k == n) return 1;
38     if(k > n) return 0;
39     int res = fact[n] * invfact[n-k] % MOD * invfact[k] % MOD;
40     return res;
41 }
42
43 signed main(){
44     int n,k; cin>>n>>k;
45     procesar();
46     cout<<nCk(n,k);
47 }

```

Combinatorics

6.2 TernarySearch

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  /*
4   *   *
5   *
6   *
7   * */
8
9  // ar vector que contiene a la funcion
10
11 void ternarySearchMaximo() {
12     int l = 0;
13     r = n - 1, m1, m2;
14     while (r - l > 2) {
15         m1 = l + (r - l) / 3;
16         m2 = r - (r - l) / 3;
17
18         int fm1 = ar[m1]; // resultado funcion evaluada en m1
19         int fm2 = ar[m2];
20
21         if (fm1 == fm2) {
22             l = m1;
23             r = m2;
24
25         } else if (fm1 < fm2) {
26             l = m1;
27         } else {
28             r = m2;

```

```

29     }
30 }
31 int ans = INT_MIN;
32 for (int i = l; i <= r; i++) {
33     if (ar[i] > ans) {
34         ans = ar[i];
35     }
36 }
37 }
38 void ternarySearchMin() {
39     int l = 0;
40     r = n - 1, m1, m2;
41     while (r - l > 2) {
42         m1 = l + (r - l) / 3;
43         m2 = r - (r - l) / 3;
44
45         int fm1 = ar[m1]; // resultado funcion evaluada en m1
46         int fm2 = ar[m2];
47
48         if (fm1 == fm2) {
49             l = m1;
50             r = m2;
51
52         } else if (fm1 < fm2) {
53             l = m1;
54         } else {
55             r = m2;
56         }
57     }
58     int ans = INT_MAX;
59     for (int i = l; i <= r; i++) {
60         if (ar[i] < ans) {
61             ans = ar[i];
62         }
63     }
64 }

```

TernarySearch

6.3 TernarySearchReales

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  /**
6  .------.
7  /
8  /
9  /
10 /
11 /

```

```

12 /
13 ,-----,
14 **/
15
16 double f(double x) {
17     return 0;
18     // escribir la funcion;
19 }
20
21 void ternarySearchMaximo() {
22     double eps = 1e-6;
23     int l = -100, r = 100, m1, m2;
24     while (r - l > 2) {
25         m1 = l + (r - l) / 3;
26         m2 = r - (r - l) / 3;
27
28         int fm1 = ar[m1]; // resultado funcion evaluada en m1
29         int fm2 = ar[m2];
30
31         if (fm1 == fm2) {
32             l = m1;
33             r = m2;
34
35         } else if (fm1 < fm2) {
36             l = m1;
37         } else {
38             r = m2;
39         }
40     }
41     int ans = INT_MIN;
42     for (int i = l; i <= r; i++) {
43         if (ar[i] > ans) {
44             ans = ar[i];
45         }
46     }
47 }
48 i

```

TernarySearchReales

7 datastructures

7.1 FenwickTree

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4
5 int lsb(int x) {return x & -x;}
6 template <typename T>
7 struct FenwickTree{
8     //indexado a 1
9     vector<T> bit;
10     FenwickTree(int N): bit(N+1, 0) {};
11     void add(int i, T val){
12         while(i < bit.size()){

```

```

13         bit[i] = (bit[i]+val)%MOD;
14         i += lsb(i);
15     }
16 }
17 T sum(int i){
18     T ret = 0;
19     while(i > 0){
20         ret = (ret + bit[i]) %MOD;
21         i -= lsb(i);
22     }
23     return ret;
24 }
25 T sum(int l, int r){
26     return sum(r)-sum(l-1);
27 }
28 };
29
30 signed main(){
31     return 0;
32 }

```

FenwickTree