Templates

オーナー	Tiểu Phương
タグ	

k-sub (20212)

💡 /* k-subseq bounded weight
a[1], a[2],..., a[n]
k-subseq ≈ k consecutive elements set
weight of a subseq = sum of all elements in this subseq
n, k, A, B
(1 <= n <= 10^5, 1 <= k <= n/2, 1 <= A, B <= 10^5)
Q, nb of k-subseq s.t. their weight w, A <= w <= B

5 3 6 18
2 2 8 8 3

2
*/

```
#define N 100009
int n;      // nb of elements
int a[N];   // the sequence
int k;      // nb of elements in subseq
int A;      // lower bound of weight
int B;      // upper bound of weight
int sum[N];
void naive() {
    int count = 0;
    for (int i = 0; i <= n-k; i++) {
        sum[i] = 0;
        for (int j = 0; j < k; j++) {
            sum[i] += a[i+j];
            if (sum[i] > B) break;
        }
        if (sum[i] >= A && sum[i] <= B) count++;
    }
    cout << count;
}
```

```
// Another improvement using Prefix Sum data structure

int n;      // nb of elements
int a[N];   // the sequence
int k;      // nb of elements in subseq
int A;      // lower bound of weight
int B;      // upper bound of weight
int sum;

bool inRange(int sum) {
    return (sum >= A && sum <= B);
}
void PrefixSum() {
    int count = 0;
    vector<int> prefix(n);
    prefix[0] = a[0];
    for (int i = 1; i < n; i++) {
        prefix[i] = prefix[i-1] + a[i];
    }
    for (int i = 0; i <= n-k; i++) {
        sum = prefix[i+k-1] - (i > 0 ? prefix[i-1] : 0);
        if (inRange(sum)) count++;
    }
    cout << count;
}

int main(int argc, char const *argv[])
{
    cin >> n >> k >> A >> B;
    for (int i = 0; i < n; i++) cin >> a[i];
    PrefixSum();
    return 0;
}
```

```
// An improvement using Sliding Window technique
```

```
#include <iostream>
using namespace std;
#define N 100009
int n;      // nb of elements
int a[N];   // the sequence
int k;      // nb of elements in subseq
int A;      // lower bound of weight
int B;      // upper bound of weight
int sum[N];

bool inRange(int sum) {
    return (sum >= A && sum <= B);
}
void SlidingWindow() {
    int count = 0;
    for (int i = 0; i < k; i++) sum[0] += a[i];
    if (inRange(sum[0])) count++;
    for (int i = 1; i <= n-k; i++) {
        sum[i] = sum[i-1] - a[i-1] + a[i+k-1];
        if (inRange(sum[i])) count++;
    }
    cout << count;
}

int main(int argc, char const *argv[])
{
    cin >> n >> k >> A >> B;
    for (int i = 0; i < n; i++) cin >> a[i];
    SlidingWindow();
    return 0;
}
```

k-combi (20212)

```
/* K-combination sum is equal to m
a[1], a[2], ..., a[n]
k, m
Find Q, nb of choosing k elements s.t. their sum = m

5 3 10
```

```
1 2 3 4 5

2
*/
#include <iostream>
using namespace std;
#define N 20
int k;               // nb of elements chosen
int m;               // the sum
int a[N];            // the array
int n;               // nb of elements in array
int Q;               // nb of choices
int curSum;          // current sum
vector<int> C;       // combination

bool check(int i) {
    return curSum + a[i] <= m;
}

void Ckn(int start, int k, vector<int> & C, int a[]) {
    if (curSum == m) Q++;
    if (k == 0) return;
    for (int i = start; i < n; i++) {
        if (check(i)) {
            C.push_back(a[i]);
            curSum += a[i];
            Ckn(i + 1, k - 1, C, a);
            C.pop_back();
            curSum -= a[i];
        }
    }
}

int main(int argc, char const *argv[])
{
    cin >> n >> k >> m;
    for (int i = 0; i < n; i++) cin >> a[i];
    curSum = 0;
    Q = 0;
    Ckn(0, k, C, a);
    cout << Q;
    return 0;
}
```

collection (20212)

```
/* Perform Insertion and Delete-Max on a Collection
a[1], a[2],..., a[n]
Queries:
* insert x: adding an element x to collection
* delete-max: delete from collection and return the value of max elemen

10
8 5 7 9 10 4 7 2 2 6
insert 3
delete-max
insert 6
*/
#include <iostream>
#include <queue>
#include <sstream>
using namespace std;
#define N 100000;
int n;                                  // nb of elements
priority_queue<int> a;                  // the collection (max heap impleme
string input;                           // user input
int x;
void insert(int x) {
    a.push(x);
}

int delete_max() {
    int max = a.top();
    a.pop();
    return max;
}

int main(int argc, char const *argv[])
{
    cin >> n;
    int temp;
    for (int i = 0; i < n; i++) {
        cin >> temp;
        a.push(temp);
    }
    cin.ignore();

    while (getline(cin, input) && input != "*") {
        istringstream iss(input);
        string cmd;
        iss >> cmd;
        if (cmd == "insert") {
            iss >> x;
            insert(x);
        } else if (cmd == "delete-max") {
            cout << delete_max() << endl;
        }
    }
    return 0;
}
```

maxsub no 3 adjacents

```
/* Ex 4. Max Sub Sequence, no 3 adjacent elements selected
Given a[1], a[2], ... a[n]
subset, max sum, does not contains 3 adj elements

Input:
[1] n (1 <= n <= 10^5)
[2] a1, a2, ... an (1 <= a[i] <= 10^3)

Output:
[1] sum of subseq

7
4 10 6 6 6 2 5

31
*/
```

```
#include <iostream>
using namespace std;
#define N 100000
int n;              // nb of elements
int a[N];           // the sequence
int s[N];           // solution of subprob

int MaxSub() {
    if (n <= 2) return max(a[0], a[1]);
    if (n == 3) return max(max(a[0], a[1]), a[2]);
    s[0] = a[0];
    s[1] = a[0] + a[1];
    s[2] = max(max(a[0] + a[1], a[1] + a[2]), a[0] + a[2]);
    for (int i = 3; i < n; i++)
        s[i] = max(max(s[i-1], s[i-2] + a[i]), s[i-3] + a[i-1] + a[i]);
    return s[n-1];
}
int main(int argc, char const *argv[])
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    cout << MaxSub();
    return 0;
}
```

binary

```
// Done
#include <iostream>
using namespace std;
char s[100];
int n;
void print_sol(char* s, int n) {
    for (int i = 0; i < n; i++) {
        cout << s[i];
    }
    cout << endl;
}

void TRY() {
    for (int i = 0; i < n; i++) {
        s[i] = '0';
    }
    while (true) {
        print_sol(s, n);
        int i = n - 1;
        while (i >= 0 && s[i] == '1') {
            s[i] = '0';
            i--;
        }
        if (i < 0) {
            break;
        }
        s[i] = '1';
    }
}

int main(int argc, char const *argv[])
{
    cin >> n;
    TRY();
    return 0;
}
```

candy

```
// Done
#include <iostream>
using namespace std;

const int MAXN = 20;
int n;              // number of kids
int m;              // number of candies
int x[MAXN];        // x[i] = nb of candies for kid i
int cur_sum = 0;    // current sum of candies

void print_sol() {
    for (int i = 1; i <= n; i++) {
        cout << x[i] << ' ';
    }
    cout << endl;
}

void TRY(int k) {
    // x1, x2, ..., x[k-1] = cur_sum
    // x[k]: 1 .. m - cur_sum - (n-k)
    int start_val = k != n? 1 : m - cur_sum - (n-k);
    for (int i = start_val; i <= m - cur_sum - (n-k); i++) {
        // for no 2 adjacent kids getting the same nb of candies
        // remember to assign a virtual kid: x[0] = 0;
        // if (x[k-1] == i) continue;
        x[k] = i;
        cur_sum += i;
        if (k == n) print_sol();
        else TRY(k+1);
        cur_sum -= i;
    }
}

int main () {
    cur_sum = 0;
    cin >> n >> m;
    TRY (1);
    return 0;
}
```

nqueen

```
// Done
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
#define N 100
int n;
int x[N];                       // in col i, x[i] is the row in which t
bool candidate(int v, int i) {  // v is the new value to assign to x[i]
    for (int j = 1; j <= i-1; j++) {
        if (x[j] == v) return false;
        if (abs(x[j] - v) == abs(i - j)) return false;
    }
    return true;
}

void solution() {
    for (int i = 1; i<= n; i++) {
        printf("%d ", x[i]);
    }
    printf("\n");
}
void TRY(int i) {
    for (int v = 1; v <= n; v++) {
        if (candidate(v, i)) {
            x[i] = v;
            if (i == n) solution();
            else TRY(i+1);
        }
    }
}
int main(int argc, char const *argv[])
{
    cin >> n;
    TRY(1);
    return 0;
}
```

tsp

```
// TSP: Branch and Bound
#include <iostream>
using namespace std;

int n;               // number of cities [1..n]
int c[100][100];     // cost matrix
int cmin = 1000000; // minimum element of the cost matrix, exclusive of
int x[100];          // current solution
int visited[100];    // marking array: visited[i] = 1 if city i is visit
int f;               // current cost
int fopt;            // optimal cost
```

```
void input() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; i++) {
            cin >> c[i][j];
        }
    }
}
void find_cmin () {
    for (int i = 1; i <= n; i++) {
        for (int j = i+1; j <= n; i++) {
            if (c[i][j] < cmin) cmin = c[i][j];
        }
    }
}
void TRY(int k) {
    for (int v = 1; v <= n; v++) {
        x[k] = v;
        visited[v] = 1;
        f += c[x[k-1]][k];
        if (k == n) {
            if (f + c[x[n]][1] < fopt) {
                fopt = f + c[x[n]][1];
            }
        }
        else {
            if (f + (n-k+1)*cmin < fopt) {
                TRY(k+1);
            }
        }
        f -= c[x[k-1]][k];
        visited[v] = 0;
    }
}
int main(int argc, char const *argv[])
{
    input();
    find_cmin();
    for (int i = 1; i <= n; i++) visited[i] = 0;
    f = 0;
    fopt = 0;
    x[1] = 1;
    visited[1] = 1;
    TRY(2);
    return 0;
}
```

maxsubseq (DC)

```
// Done
#include <iostream>
using namespace std;
#define N 100000
int n;          // size of array
int a[N];       // the sequence

int MaxLeftMid(int i, int j) {
    int maxLM = a[j];
    int s = 0;
    for (int k = j; k >= i; k--) {
        s += a[k];
        maxLM = max(maxLM, s);
    }
    return maxLM;
}

int MaxRightMid(int i, int j) {
    int maxRM = a[i];
    int s = 0;
    for (int k = i; k <= j; k++) {
        s += a[k];
        maxRM = max(maxRM, s);
    }
    return maxRM;
}

int SubSeqMax(int i, int j) {
    if (i == j) return a[i];
    int mid = (i + j) / 2;
    int wL = SubSeqMax(i, mid);
    int wR = SubSeqMax(mid+1, j);
    int maxLM = MaxLeftMid(i, mid);
    int maxRM = MaxRightMid(mid+1, j);
    int wM = maxLM + maxRM;
    return max(max(wL, wR), wM);
}

int main(int argc, char const *argv[])
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    int ans = SubSeqMax(0, n);
    cout << ans;
    return 0;
}

// IN: 7
// -16 7 -3 0 -1 5 -4
// OUT: 8
```

fib (DP)

```
#include <iostream>
#include <map>
using namespace std;
int n;

// Recursion, no memory
int fib(int n) {
    if (n <= 2) return 1;
    int res = fib(n-2) + fib(n-1);
    return res;
}

// Recursion w/ memory, using map structure (O(log n) retrieval)
map<int, int> Mem;
int mapfib(int n) {
    if (n <= 2)                         return 1;
    if (Mem.find(n) != Mem.end())   return Mem[n];
    int res = mapfib(n-2) + mapfib(n-1);
    Mem[n] = res;
    return res;
}

// Recursion w/ memory, using array structure (O(1) retrieval)

int iMem[1001];

int arrfib(int n) {
    if (n <= 2)         return 1;
    if (iMem[n] != -1)  return iMem[n];
    int res = arrfib(n-2) + arrfib(n-1);
    iMem[n] = res;
    return res;
}

int main(int argc, char const *argv[])
{
    printf("n = ");
    cin >> n;
    for (int i = 1; i <= 1000; i++)
        iMem[i] = -1;                   // for mapfib(n)
    printf("The %d-th number in Fibonacci sequence\n", n);
    printf("fib(n)\t\tmapfib(n)\tarrfib(n)\t\n");
    cout << fib(n) << "\t\t" << mapfib(n) << "\t\t" << arrfib(n) << "\n
    return 0;
}
```

lcs (DP)

```
// Done
/* Denote X = X1, X2, …, Xn, a subsequence of X is created by removing
Given two sequences X = X1, X2, …, Xn and Y = Y1, Y2, …, Ym.
Find a common subsequence of X and Y such that the length is the longes
Input
Line 1: two positive integers n and m (1 <= n,m <= 10000)
Line 2: n integers X1, X2, …, Xn
Line 3:  m integers Y1, Y2, …, Ym
Output
Length of the longest subsequence of X and Y
```

```
Example
Input
7 10
3 7 2 5 1 4 9
4 3 2 3 6 1 5 4 9 7
Output
5 */
#include <iostream>
using namespace std;
#define SIZE 10001
int n, m;                       // nb of elements of X, Y
int X[SIZE], Y[SIZE];           // the sequences
int dp[SIZE][SIZE];             // dp[i][j] is length of LCS of X[1..i], Y[

int main(int argc, char const *argv[])
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> X[i];
        dp[i][0] = 0;
    }
    for (int j = 1; j <= m; j++) {
        cin >> Y[j];
        dp[0][j] = 0;
    }

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (X[i] == Y[j]) dp[i][j] = dp[i-1][j-1] + 1;
            else dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
        }
    }
    cout << dp[n][m];
    return 0;
}
```

lis (DP)

```
// Done
#include <iostream>
```

```
#include <cstring>
using namespace std;
int A[100000];        // the sequence A
int iMem[100000];     // iMem[i]: the result of subproblem i
int n;               // size of A


int LIS(int i) {
    int res = 1;        // the result of subproblem
    if (iMem[i] != -1) return iMem[i];
    for (int j = 1; j < i; j++) {
        if (A[j] < A[i]) {  // A[j] will only count as a candidate for
            res = max(res, 1 + LIS(j));
        }
    }
    iMem[i] = res;
    return res;
}
int main(int argc, char const *argv[])
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> A[i];
    memset(iMem, -1, sizeof(iMem));
    int ans = 0;    // final global answer
    // int pos = 0;    // mark the end position of LIS
    for (int i = 1; i <= n; i++) {
        LIS(i);
    }
    for (int i = 1; i <= n; i++) {
        ans = max(ans, iMem[i]);
        // if (ans == iMem[i]) pos = i;
    }
    cout << ans;
    return 0;
}
```

maze

```
// Done
#include <iostream>
```

```
#include <queue>
using namespace std;
typedef pair<int, int> ii;      // pair of integers
const int maxN = 999 + 100;     // maximum number of rows and columns
const int oo = 1e9 + 7;         // infinity
int n;                          // number of rows
int m;                          // number of columns
int r;                          // starting row
int c;                          // starting column
int a[maxN][maxN];              // the maze
int d[maxN][maxN];              // distance from (r, c) to (i, j)
int dx[] = {-1, 0, 1, 0},       // direction vectors
    dy[] = {0, 1, 0, -1};       // direction vectors
queue<ii> q;                    // queue for BFS
int solve() {
    q.push(ii(r, c));
    d[r][c] = 0;
    a[r][c] = 1;
    while (!q.empty())
    {
        ii u = q.front();
        q.pop();
        for (int i = 0; i < 4; i++) {
            int x = dx[i] + u.first;
            int y = dy[i] + u.second;
            if (x < 1 || x > m || y < 1 || y > n)
                return d[u.first][u.second] + 1;
            if (a[x][y] != 1) {
                d[x][y] = d[u.first][u.second] + 1;
                q.push(ii(x, y));
                a[x][y] = 1;
            }
        }
    }
    return -1;
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    // printf("Enter m, n, r, c (separated by a space): ");
    cin >> m >> n >> r >> c;
    // printf("Enter the maze matrix: \n");
    for (int i = 1; i <= m; i++)
        for (int j = 1; j<= n; j++)
            cin >> a[i][j];
    int ans = solve();
    // cout << "ans = " << ans << "\n";
    cout << ans;
    return 0;
}

// SAMPLE INPUT
// 8 12 5 6
// 1 1 0 0 0 0 1 0 0 0 0 1
// 1 0 0 0 1 1 0 1 0 0 1 1
// 0 0 1 0 0 0 0 0 0 0 0 0
// 1 0 0 0 0 0 1 0 0 1 0 1
// 1 0 0 1 0 0 0 0 0 1 0 0
// 1 0 1 0 1 0 0 0 1 0 1 0
// 0 0 0 0 1 0 1 0 0 0 0 0
// 1 0 1 1 0 1 1 1 0 1 0 1
// OUTPUT
// 7
```

rmq

```
// Done
#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

const int MAXN = 1000001;    // maximum number of elements
int n;                      // number of elements
int M[30][MAXN];            // M[j][i] = index of the minimum element i
int A[MAXN];                // the array

// Building the segment tree
void preprocessing() {
    for (int j = 0; (1 << j) <= n; j++) {
        for (int i = 0; i < n; i++)
            M[j][i] = -1;
    }
    for (int i = 0; i < n; i++)
        M[0][i] = i;
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 0; i + (1 << j) - 1 < n; i++) {
            if (A[M[j - 1][i]] < A[M[j - 1][i + (1 << (j - 1))]])
                M[j][i] = M[j - 1][i];
            else
                M[j][i] = M[j - 1][i + (1 << (j - 1))];
        }
    }
}

// Querying the segment tree
int rmq(int i, int j) {
    int k = log2(j - i + 1);
    int p2k = (1 << k); // 2^k
    if (A[M[k][i]] <= A[M[k][j - p2k + 1]]) {
        return M[k][i];
    } else {
        return M[k][j - p2k + 1];
    }
}

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> A[i];
    }
    preprocessing();
    int ans = 0;
    int m;
    cin >> m;
    for (int i = 0; i < m; i++) {
        int I, J;
        scanf("%d %d", &I, &J);
        ans += A[rmq(I, J)];
    }
    cout << ans;
    return 0;
}
```

subrec

```
// Done
#include <iostream>
#include <stack>
#include <vector>

using namespace std;
const int N = 1e3 + 1;
int a[N][N];                    // the matrix
int n;                          // number of rows
int m;                          // number of columns
int ans;
long long h[N];                 // height of the histogram
stack<long long> S;             // stack for computing the histogram
vector<long long> V;            // vector for computing the histogram
long long L[N], R[N];           // left and right boundaries of the histogr

void input() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            cin >> a[i][j];
        }
    }
}
long long compute() {
    h[0] = -1;
    h[m+1] = -1;
    V.clear();
    for (int i = 1; i <= m+1; i++)
    {
        while (!V.empty() && h[i] < h[V[V.size()-1]])
        {
            R[V[V.size()-1]] = i;
            V.pop_back();
        }
        V.push_back(i);
    }
    for (int i = m; i >= 0; i--)
    {
        while (!V.empty() && h[i] < h[V[V.size()-1]])
        {
            L[V[V.size()-1]] = i;
            V.pop_back();
        }
        V.push_back(i);
    }
    unsigned long long ans = 0;
    for (int i = 1; i <= m; i++)
    {
        unsigned long long c = (R[i] - L[i] - 1) * h[i];
        ans = max(ans, c);
    }
    return ans;
}
void solve() {
    long long ans = 0;
    for (int i = 0; i <= m; i++)
        h[i] = 0;
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= m; j++)
        {
            if (a[i][j] == 0)
                h[j] = 0;
            else
                h[j]++;
        }
        long long t = compute();
        if (t > ans) ans = t;
    }
    cout << ans << endl;
}
```

```
int main(int argc, char const *argv[])
{
    input();
    solve();
    return 0;
}

//SAMPLE INPUT
// 4 4
// 0 1 1 1
// 1 1 1 0
// 1 1 0 0
// 1 1 1 0
//OUTPUT
//6
```

telco

```
// PARTIAL
/*
Write a C program to perform some queries on a telco data (comming from
The first block of data consists of lines (terminated by a line contain
    call <from_number> <to_number> <date> <from_time> <end_time>
which is a call from the phone number <from_number> to a phone number <
<from_number> and <to_number> are string of 10 characters (a phone numb
<date> is under the form YYYY-MM-DD (for example 2022-10-21)
<from_time> and <to_time> are under the form hh:mm:ss (for example, 10:

The second block consists of queries (terminated by a line containing #
?check_phone_number: print to stdout (in a new line) value 1 if no phon
?number_calls_from <phone_number>: print to stdout (in a new line) the
?number_total_calls: print to stdout (in a new line) the total number o
?count_time_calls_from <phone_number>: print to stdout (in a new line)

Example
Input
call 0912345678 0132465789 2022-07-12 10:30:23 10:32:00
call 0912345678 0945324545 2022-07-13 11:30:10 11:35:11
call 0132465789 0945324545 2022-07-13 11:30:23 11:32:23
call 0945324545 0912345678 2022-07-13 07:30:23 07:48:30
```

```
#
?check_phone_number
?number_calls_from 0912345678
?number_total_calls
?count_time_calls_from 0912345678
?count_time_calls_from 0132465789
#

Output
1
2
4
398
120
*/

#include <iostream>
#include <map>
using namespace std;
bool checkPhone (string s) {
    if (s.length() != 10) return false;
    for (int i = 0; i < s.length(); i++) {
        if (!(s[i] >= '0' && s[i] <= '9')) return false;
    }
    return true;
}
int countTime(string ftime, string etime) {
    int startTime   = 3600 * ((ftime[0] - '0') * 10 + ftime[1] - '0') +
                        60 * ((ftime[3] - '0') * 10 + ftime[4] - '0') +
                             ((ftime[6] - '0') * 10 + ftime[7] - '0');
    int endTime     = 3600 * ((etime[0] - '0') * 10 + etime[1] - '0') +
                        60 * ((etime[3] - '0') * 10 + etime[4] - '0') +
                             ((etime[6] - '0') * 10 + etime[7] - '0');
    return endTime - startTime;
}
map <string, int> numberCalls, timeCall;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(NULL);
    cout.tie(NULL);
    string type;
    int totalCalls = 0;
    int incorrectPhone = 0;
    do {
        cin >> type;
        if (type == "#") continue;
        ++totalCalls;
        string fnum, tnum, date, ftime, etime;
        cin >> fnum >> tnum >> date >> ftime >> etime;
        if (!checkPhone(fnum) || !checkPhone(tnum)) ++incorrectPhone;
        numberCalls[fnum]++;
        timeCall[fnum] += countTime(ftime, etime);
    } while (type != "#");

    do
    {
        cin >> type;
        if (type == "#") continue;
        if (type == "?check_phone_number") {
            if (incorrectPhone == 0) cout << 1 << endl;
            else                     cout << 0 << endl;
        } else if (type == "?number_calls_from") {
            string phone;
            cin >> phone;
            cout << numberCalls[phone] << endl;
        } else if (type == "?count_time_call_from") {
            string phone;
            cin >> phone;
            cout << timeCall[phone] << endl;
        }
    } while (type != "#");

    return 0;
}
```

bca

```
/* PARTIAL
At the beginning of the semester, the head of a computer science depart
```

```
The department D has m teachers T={1,2,...,m} and n courses C={1,2,...,
Each teacher t ∈ T has a preference list which is a list of courses he/
We known a list of pairs of conflicting two courses that cannot be assi
The load of a teacher is the number of courses assigned to her/him.
How to assign n courses to m teacher such that:
    * each course assigned to a teacher is in his/her preference list,
    * no two conflicting courses are assigned to the same teacher,
    * the maximal load is minimal.
Input
The input consists of following lines
Line 1: contains two integer m and n (1 ≤ m ≤ 10, 1 ≤ n ≤ 30)
Line i+1: contains an positive integer k and k positive integers indica
Line m+2: contains an integer k
Line i+m+2: contains two integer i and j indicating two conflicting cou
Output
The output contains a unique number which is the maximal load of the te
Example
Input
4 12
5 1 3 5 10 12
5 9 3 4 8 12
6 1 2 3 4 9 7
7 1 2 3 5 6 10 11
25
1 2
1 3
1 5
2 4
2 5
2 6
3 5
3 7
3 10
4 6
4 9
5 6
5 7
5 8
6 8
6 9
```

```
7 8
7 10
7 11
8 9
8 11
8 12
9 12
10 11
11 12

Output
3
*/
#include <iostream>
#include <vector>
using namespace std;
#define N 50                    // (variable length array declaration not a
int m;                          // number of teachers
int n;                          // number of courses
int x[N];                       // i-th course is taught by teacher x[i]
int res;                        // minimum load
int load[N];                    // current load
bool conflict[N][N];            // course conflict pairs
vector<int> T[N];               // (preference list) list of teachers can b
void input() {
    cin >> m >> n;
    for (int i = 1; i <= m; i++) {
        int k;
        cin >> k;
        for (int t = 1; t <= k; t++) {
            int c;
            cin >> c;
            T[c].push_back(t);
        }
    }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            conflict[i][j] = false;
    int k;
    cin >> k;
    for (int r = 1; r <= k; r++) {
        int i, j;
        cin >> i >> j;
        conflict[i][j] = true;
        conflict[j][i] = true;
    }
}
bool check (int t, int k) {
    for (int i = 1; i <= k-1; i++) {
        if (conflict[i][k] && x[i] == t)
            return false;
    }
    return true;
}

void solution() {
    int maxLoad = 0;
    for (int t = 1; t <= m; t++) {
        maxLoad = max(maxLoad, load[t]);
    }
    if (maxLoad < res) res = maxLoad;
}

void TRY(int k) {
    for (int i = 0; i < T[k].size(); i++) {
        int t = T[k][i];
        if (check(t, k)) {
            x[k] = t;   // assign course k to teacher t
            load[t] += 1;
            if (k == n) solution();
            else {
                if (load[t] <= res)
                    TRY(k + 1);
            }
            load[t] -= 1;
        }
    }
}

int main () {
    input();
    for (int t = 1; t <= m; t++)
        load[t] = 0;
    res = 1e9;
    TRY(1);
    cout << res;
    return 0;
}
```

cbus

```
// Done
#include <iostream>
using namespace std;
#define MAX 30
int n;                              // number of passengers/ requests per s
int len;                            // 2*n, total elements in solution repr
int capacity;                       // number of available bus seats
int C[2*MAX+1][2*MAX+ 1];           // travelling distance matrix
bool appear[MAX];                   // marking traveled points
int load;                           // number of current passengers on bus
int x[MAX];                         // current solution
int f;                              // current cost
int f_best;                         // best cost
int x_best[MAX];                    // best solution
int cmin;                           // smallest element (diagonal excluded)
void input() {
    cin >> n >> capacity;
    len = 2*n;
    cmin = 1000000;
    for (int i = 0; i <= len; i++)
        for (int j = 0; j <= len; j++) {
            cin >> C[i][j];
            if (i != j && cmin > C[i][j]) cmin = C[i][j];
        }
}

bool check(int v, int k) {
    if (appear[v] == 1) return 0;
    if (v > n) {
        if (!appear[v-n]) return 0;
    } else {
        if (load + 1 > capacity) return 0;
    }
    return 1;
}

void solution() {
    if (f + C[x[len]][0] < f_best) {
        f_best = f + C[x[len]][0];
        for (int i = 0; i <= len; i++)
            x_best[i] = x[i];
    }
}

void TRY(int k) {
    for (int v = 1; v <= len; v++) {
        if (check(v, k)) {
            x[k] = v;
            f += C[x[k-1]][x[k]];
            if (v <= n) load += 1;
            else        load -= 1;
            appear[v] = 1;
            if (k == len) solution();
            else {
                if (f + (len+1-k) * cmin < f_best)
                    TRY(k+1);
            }
            if (v <= n) load -= 1;
            else        load += 1;
            appear[v] = 0;
            f -= C[x[k-1]][x[k]];
        }
    }
}

void solve() {
    load = 0;
    f = 0;
    f_best = 1000000;
    for (int i = 1; i <= len; i++)
        appear[i] = 0;
    x[0] = 0;   // starting point
    TRY(1);
    printf("%d", f_best);
}

void print() {
    for (int i = 0; i <= len; i++)
        printf("%d ", x_best[i]);
}

int main()
{
    input();
    solve();
    return 0;
}
```

equation

```
#include <iostream>
using namespace std;
#define MAX 50
int n;                    // number of positive integers
int M;                    // sum (RHS) of the equation
int a[MAX];               // coefficients
int t[MAX];               // temporary array
int X[MAX];               // solution
int f;                    // accumulated sum of the solution
int ans;                // number of positive solutions

void input() {
    cin >> n >> M;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
}
void solution() {
    if (f == M)
        ans++;
}

void TRY(int k) {
    int v;
    for (v = 1; v <= (M - f - (t[n] - t[k])) / a[k]; v++) {
        X[k] = v;
        f += a[k] * X[k];
        if (k == n) solution();
        else TRY(k+1);
        f -= a[k] * X[k];
    }
}

void solve() {
    f = 0;
    t[1] = a[1];
    for (int i = 2; i <= n; i++)
        t[i] = t[i-1] + a[i];
    ans = 0;
    TRY(1);
    cout << ans;
}

int main(int argc, char const *argv[])
{
    input();
    solve();
    return 0;
}
```

disjoint

```
// Done
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define N 100005
int n;          // number of time intervals
pair<int, int> T[N];
```

```
int result;

bool compare(const pair<int, int>& a, const pair <int, int>& b) {
    return a.second < b.second;
}

void input() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> T[i].first >> T[i].second;
}

void greedy() {
    int lastEndTime = -1;       // the endtime of last chosen segment
    int count = 0;
    sort(T + 1, T + n + 1, compare);
    for (int i = 1; i <= n; i++) {
        if (T[i].first > lastEndTime) {
            count++;
            lastEndTime = T[i].second;
        }
    }
    cout << count;
}
int main(int argc, char const *argv[])
{
    input();
    greedy();
    return 0;
}

/*
6
0 10
3 7
6 14
9 11
12 15
17 19

4
```

```
*/
```

inversion

```
// Done
/*
• Given a sequence of integers a1, a2,…, an. A pair (I, j) is called an
• Compute the number Q of inversions
• Input
• Line 1: contains a positive integer n ( 1 <= n <= 106 )
• Line 2: contains a1, a2,…, an ( 0 <= ai <= 106 )
• Output
• Write the value Q module 109 + 7
6
3 2 4 5 6 1

6
*/
#include <iostream>
#include <algorithm>
using namespace std;
#define N 1000006
#define MOD 1000000007
int n;
int a[N];
int temp[N];

int merge(int left, int mid, int right) {
    int i = left, j = mid+1, k = left, inv_count = 0; // k is index for
    while ((i <= mid) && (j <= right)) {
        if (a[i] <= a[j]) temp[k++] = a[i++];
        else {
            temp[k++] = a[j++];
            inv_count = (inv_count + (mid - i + 1)) % MOD;
        }
    }
    while (i <= mid) temp[k++] = a[i++];
    while (j <= right) temp[k++] = a[j++];
    copy(temp+left, temp+right+1, a+left);      // copy merged elements
    return inv_count;
}

int mergeSort(int left, int right) {
    int mid, inv_count = 0;
    if (right > left) {
        mid = (right + left) / 2;
        inv_count = (inv_count + mergeSort(left, mid)) % MOD;
        inv_count = (inv_count + mergeSort(mid + 1, right)) % MOD;
        inv_count = (inv_count + merge(left, mid, right)) % MOD;
    }
    return inv_count;
}

int main(int argc, char const *argv[])
{
    ios_base::sync_with_stdio(0); cin.tie(NULL); cout.tie(NULL); // opt
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cout << mergeSort(1, n);
    return 0;
}

/// @brief A naive implementation, complexity O(n^2)
/// @return count
int naive() {
    int count = 0;
    for (int i = 0; i < n-1; i++) {
        for (int j = i; j < n; j++) {
            if (a[i] > a[j]) count++;
        }
    }
    return count;
}
```

maxdist

```
// Done
/*
● Given N elements (2≤ N ≤100,000) on a straight line at positions x1,…
● The distance of a subset of N elements is defined to be the minimum d
```

```
• Find the subset of N given elements containing exactly C elements suc
• Input
• The first line contains a positive integer T (1 <= T <= 20) which is
• Subsequent lines are T test cases with the following format:
• Line 1: Two space-separated integers: N and C
• Lines 2: contains x1, x2 , . . , xn
• Output
• For each test case output one integer: the distance of the subset fou
*/
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
#define N 100000
int T;                  // number of tests in test set T
int n;                  // number of elements
int c;                  // size of a subset
int a[N];               // set of points
int result;
bool check(int distance) {
    int pcount = 1;
    int i = 0, j = 1;
    while (i < n) {
        while (j < n && a[j] - a[i] < distance) ++j;
        if (j < n) pcount ++;
        if (pcount >= c) return true;
        i = j;
        j = i+1;
    }
    return false;
}
int MaxDistance() {
    int left = 0, right = a[n-1] - a[0];
    while (left <= right) {
        int mid = (left + right) / 2;
        if (check(mid)) left = mid + 1;
        else right = mid - 1;
    }
    return right;
}
```

```
int main(int argc, char const *argv[])
{
    cin >> T;
    while (T--) {
        cin >> n >> c;
        for (int i = 0; i < n; i++) cin >> a[i];
        sort(a, a + n);
        cout << MaxDistance() << endl;
    }
    return 0;
}
```