683. Partial matrix sum

The matrix of integer a_{ij} is given, where $1 \le i \le n$, $1 \le j \le m$. For all i, j find

$$s_{i,j} = \sum_{k=1,t=1}^{k \le i, t \le j} a_{k,t}.$$

Input. The first line contains the matrix size n, m ($1 \le n, m \le 1000$). Each of the next n lines contains m integers a_{ij} ($1 \le a_{ij} \le 1000$).

Output. Print n rows each containing m numbers S_{ij}.

Sample input

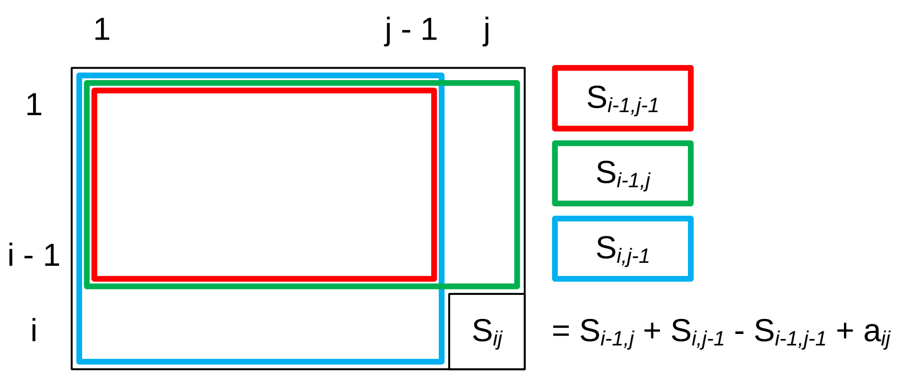
```
3 5
1 2 3 4 5
5 4 3 2 1
2 3 1 5 4
```

Sample output

```
1 3 6 10 15
6 12 18 24 30
8 17 24 35 45
```

Algorithm analysis

Let a be the input array, s be the array of partial sums. We shall fill the array s in ascending order of lines, and the cells in each row – in ascending order of columns. Suppose we calculated this way all the values of s array till s[i][j].



Then s[i][j] = s[i – 1][j] + s[i][j – 1] – s[i – 1][j – 1] + a_{ij}.

Example

Consider for the given example how to calculate the value of s[3][5].

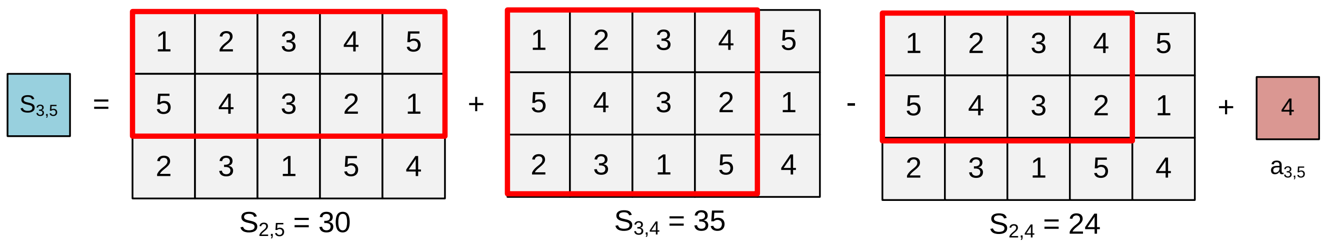
	1	2	3	4	5
1	1	2	3	4	5
2	5	4	3	2	1
3	2	3	1	5	4

a_{ij}

	1	2	3	4	5
1	1	3	6	10	15
2	6	12	18	24	30
3	8	17	24	35	$S_{3,5}$

S_{ij}

s[3][5] = s[2][5] + s[3][4] – s[2][4] + a_{35} = 30 + 35 – 24 + 4 = 45



Exercise

For the given array a_{ij} compute the values of array s_{ij}.

	1	2	3	4	5
1	2	4	3	5	6
2	4	3	2	1	1
3	5	4	6	3	3
4	2	1	1	4	2

a_{ij}

	1	2	3	4	5
1					
2					
3					
4					

S_{ij}

Algorithm realization

Declare two two-dimensional arrays.

```
#define MAX 1001
int a[MAX][MAX], s[MAX][MAX];
```

Read the input data.

```
scanf("%d %d",&n,&m);
for(i = 1; i <= n; i++)
for(j = 1; j <= m; j++)
  scanf("%d",&a[i][j]);
```

Calculate the partial sums.

```
memset(s,0,sizeof(s));
for(i = 1; i <= n; i++)
for(j = 1; j <= m; j++)
  s[i][j] = s[i][j-1] + s[i-1][j] - s[i-1][j-1] + a[i][j];
```

Print the resulting array of partial sums.

```
for(i = 1; i <= n; i++)
{
  for(j = 1; j <= m; j++)
    printf("%d ",s[i][j]);
  printf("\n");
}
```

Algorithm realization on fly

Read the current value of a_{ij}, calculate and print on fly the partial sum of S_{ij}. In this case we do not need to keep the input array.

```
#include <stdio.h>
#include <string.h>
#define MAX 1001

int i, j, n, m, value;
int mas[MAX][MAX];

int main(void)
{
  scanf("%d %d",&n,&m);
  memset(mas,0,sizeof(mas));
  for(i = 1; i <= n; i++)
  {
    for(j = 1; j <= m; j++)
    {
      scanf("%d",&value);
      mas[i][j] = mas[i][j-1] + mas[i-1][j] - mas[i-1][j-1] + value;
      printf("%d ",mas[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

877. Puzzle multiplication

In multiplication puzzle play with a number of cards, each of which contains one positive integer. During a turn a player removes one card from the series and gets the number of points equal to the product number on the cleaned card and the numbers on the cards, lying immediately to the left and right of it. Are not allowed to remove the first and last card number. After the last course in the series remains the only two cards.

Goal of the game – to remove cards in such a manner as to minimize the total number of points.

For example, if the cards contain numbers 10, 1, 50, 20 and 5, the player can take the card with the number 1, then 20 and 50, get points

$$10 * 1 * 50 + 50 * 20 * 5 + 10 * 50 * 5 = 500 + 5000 + 2500 = 8000$$

If he took the cards in reverse order, i.e. 50, then 20, then 1, the number of points would be:

$$1 * 50 * 20 + 1 * 20 * 5 + 10 * 1 * 5 = 1000 + 100 + 50 = 1150.$$

Input. The first line is the number of cards n ($3 \le n \le 100$), the second – n numbers on the cards. The numbers on the cards are integers from 1 to 100.

Output. Print a single integer – the minimum possible number of points.

Sample input

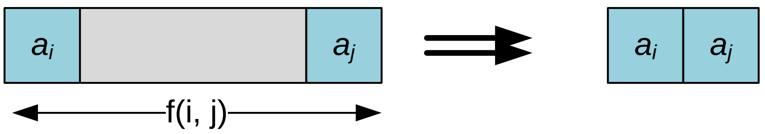
```
6
10 1 50 50 20 5
```

Sample output

```
3650
```

Algorithm analysis

Put the input sequence of cards into array a[0, …, n – 1]. Let f(i, j) be the smallest number of points that can be obtained by removing all cards strictly inside the segment [a_i, …, a_j] (except for the extreme two a_i and a_j, the extreme cards cannot be removed by the problem statement). Store this value in dp[i][j].

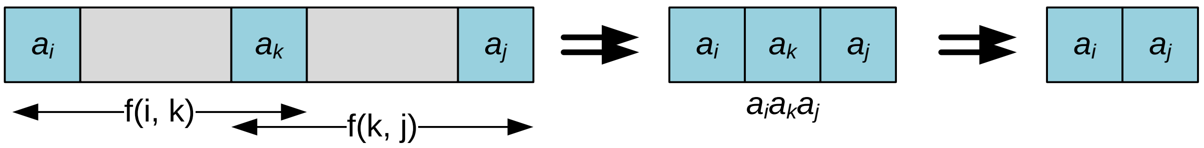


Then the answer to the problem will be f(0, n – 1).

Put the values INF = ∞ into the cells of dp array, initialize f(i, i) = dp[i][i] = 0, f(i, i + 1) = dp[i][i + 1] = 0.

Suppose that when removing numbers inside the segment [a_i, …, a_j], the last will be removed a_k ($i < k < j$). Moreover, it will add to the total sum the term a_i a_k a_j. But in order for a_k to be adjacent to a_i and a_j at the last step of choosing a card, it is necessary to remove all cards inside the segments [a_i, …, a_k] and [a_k, …, a_j]. I.e

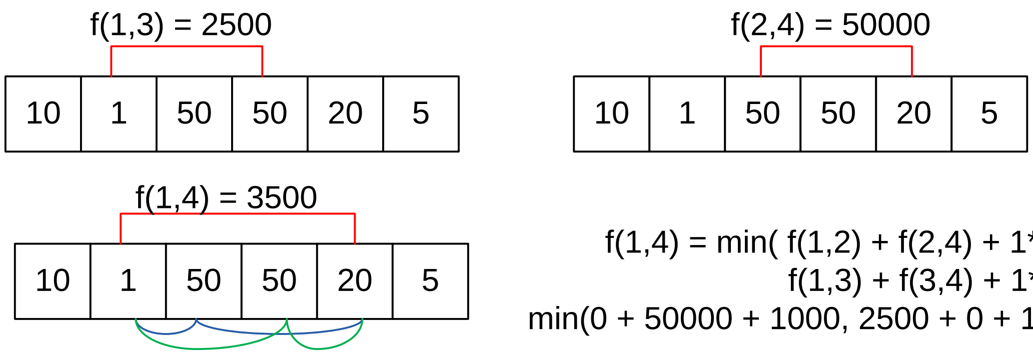
$$f(i, j) = \min_{i<k<j}\left(f(i,k) + f(k,j) + a_i a_k a_j\right)$$



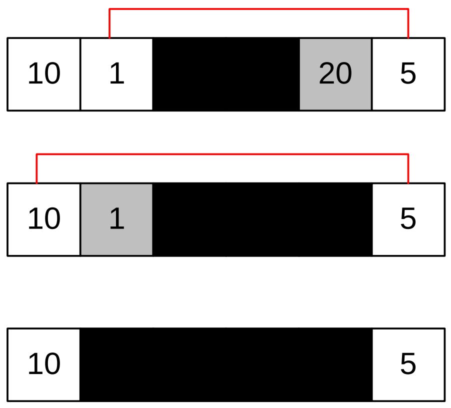
For example,

$$f(i, i + 2) = f(i, i + 1) + f(i + 1, i + 2) + a_i * a_{i+1} * a_{i+2} = a_i * a_{i+1} * a_{i+2}$$

Example



f(1,4) = min(f(1,2) + f(2,4) + 1*50*20,
f(1,3) + f(3,4) + 1*50*20) =
min(0 + 50000 + 1000, 2500 + 0 + 1000) = 3500



f(1,5) = f(1,4) + f(4,5) + 1*20*5 = 3500 + 0 + 100 = 3600

f(0,5) = f(0,1) + f(1,5) + 10*1*5 = 0 + 3600 + 50 = 3650

Algorithm realization

Declare the arrays.

```
#define MAX 110
#define INF 0x3F3F3F3F
int dp[MAX][MAX], a[MAX];
```

Function ***f*** returns the smallest number of points that can be obtained by removing all cards strictly inside the segment [a_i, …, a_j] (except for the extreme two a_i и a_j, the extreme cards cannot be removed by the problem statement).

```
int f(int i, int j)
{
  if (dp[i][j] == INF)
    for (int k = i + 1; k < j; k++)
      dp[i][j] = min(dp[i][j], f(i,k) + f(k,j) + a[i] * a[k] * a[j]);
  return dp[i][j];
}
```

The main part of the program. Read the input data. Initialize the dp array.

```
scanf("%d",&n);
memset(dp,0x3F,sizeof(dp));
for(i = 0; i < n; i++)
{
  scanf("%d",&a[i]);
  dp[i][i] = 0;
  dp[i][i+1] = 0;
}
```

Compute and print the answer.

```
printf("%d\n",f(0,n-1));
```

7447. Cut a string

You are given a string *s*. It is allowed to take any two same neighbor symbols of this string and delete them. This operation you can do while possible. At the beginning you can choose any symbols from string and delete them. Determine the minimum number of symbols you can delete at the beginning, so that you get the empty string after performing allowed operations.

Input. One string *s* of length no more than 100.

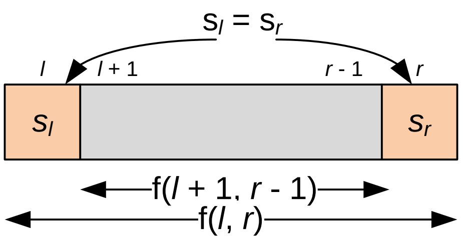
Output. Print the minimum number of symbols you should delete at the beginning.

Sample input	**Sample output**
`abacdeec`	`2`

Algorithm analysis

Let dp[*l*][*r*] = f(*l*, *r*) be the minimum number of characters that should be removed from the substring $s_l..s_r$, so that later, as a result of applying operations (removing identical adjacent characters), an empty string can be obtained. Then:

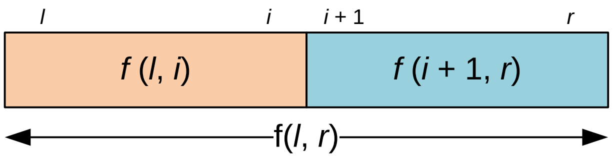
- f(*l*, *r*) = 0, if *l* > *r*;
- f(*l*, *l*) = 1, single character should be removed at the beginning;
- f(*l*, *r*) = f(*l* + 1, *r* – 1), if s[*l*] = s[*r*]. If the leftmost and the rightmost characters are the same, then the inner part should be deleted, after which these characters become adjacent and they can be deleted by applying the operation;



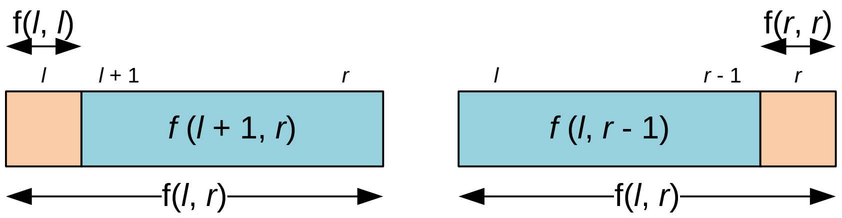
- f(*l*, *r*) = 1 + f(*l* + 1, *r*), if we remove the first character;
- f(*l*, *r*) = 1 + f(*l*, *r* – 1), if we remove the last character;

However, the last two conditions can be included in the following: to solve the problem on the segment [*l*; *r*] let’s solve the problem separately on segments [*l*; *i*] and [*i* + 1; *r*] (1 ≤ *i* < *r*) and take the smallest result:

$$f(l, r) = \min_{1 \le i < r} \left(f(l, i) + f(i + 1, r) \right)$$



For example, the case of removing the first character from the string is equivalent to *i* = *l* (then f(*l*, *l*) = 1), and the case of removing the last character is equivalent to *i* = *r* – 1 (f (*r*, *r*) = 1).

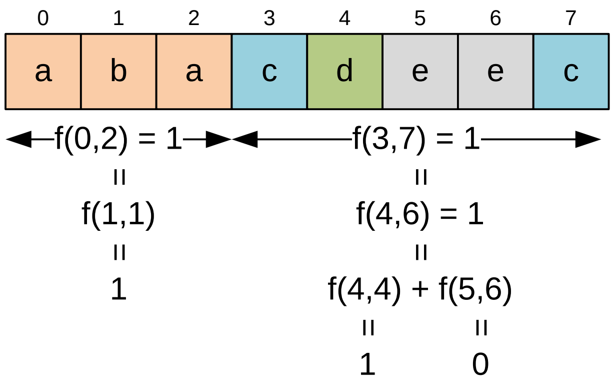


The answer to the problem is dp[0][n – 1] = f(0, n – 1), where n is the length of the input string.

Example

Consider the sample given in the problem statement. We have:

$$f(0, 7) = f(0, 2) + f(3, 7) = 1 + 1 = 2$$



Algorithm realization

Declare the arrays.

```
#define MAX 101
#define INF 0x3F3F3F3F
int dp[MAX][MAX];
string s;
```

Let f(l, r) be the solution of the problem on the segment [l; r].

```
int f(int l, int r)
{
  if (l > r) return 0;
  if (l == r) return 1;
  if (dp[l][r] != INF) return dp[l][r];
  int &res = dp[l][r];

  if (s[l] == s[r])
    res = min(res, f(l + 1, r - 1));

  for (int i = l; i < r; i++)
    res = min(res, f(l, i) + f(i + 1, r));
  return res;
}
```

The main part of the program. Read the line.

```
cin >> s;
memset(dp,0x3F,sizeof(dp));
```

Compute and print the answer f(0, n – 1), where n is the length of string s.

```
printf("%d\n",f(0, s.size() - 1));
```

2585. Profit

The cows have opened a new business, and Farmer John wants to see how well they are doing. The business has been running for n ($1 \le n \le 100000$) days, and every day i the cows recorded their net profit P_i ($-1000 \le P_i \le 1000$).

Farmer John wants to find the largest total profit that the cows have made during any consecutive time period. (Note that a consecutive time period can range in length from one day through n days.) Help him by writing a program to calculate the largest sum of consecutive profits.

Input. First line contains a single integer n. Each of the next n lines contains a single integer P_i.

Output. Print the value of the maximum sum of profits for any consecutive time period.

Sample input

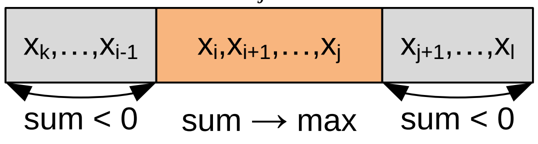
```
7
-3
4
9
-2
-5
8
-3
```

Sample output

```
14
```

Algorithm analysis

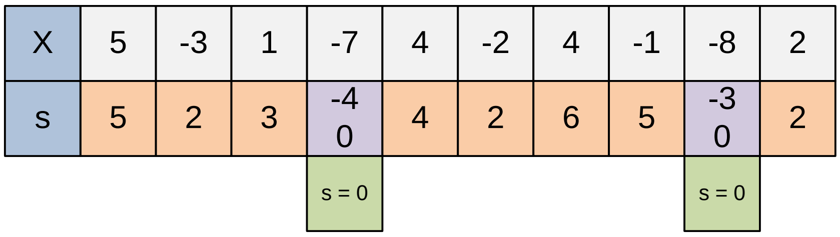
In the problem we need to find a subsequence of consecutive numbers that will have the maximum possible sum among all possible such subsequences. If the maximum sum is attained on subsequence $x_i, x_{i+1}, \dots, x_j$, then for any k, $1 \le k < i$ and l, $j < l \le n$, the sum of elements $x_k, \dots, x_{i-1}$ and $x_{j+1}, \dots, x_l$ will be negative.



Kadane's algorithm. Move through the array from left to right and accumulate the current partial sum in the variable s. If at some moment s turns out to be negative, then we assign $s = 0$. The maximum of all values of the variable s during the passage through the array will be the answer to the problem.

Example

Consider a sequence X given below. Construct the partial sums. The current value of the partial sum is set to zero when the current sum becomes less than zero and we start counting the sum from the next number. The maximum value among all partial sums is 6, which is the answer. The required subsequence is 4, -2, 4.



X	5	-3	1	-7	4	-2	4	-1	-8	2
s	5	2	3	-4 0	4	2	6	5	-3 0	2
				s = 0					s = 0	

Algorithm realization

Read the length of the sequence *n*. Reset the value of the maximum profit *max* and the current partial sum *s*.

```
scanf("%d",&n);
s = 0; max = 0;
```

Read *n* integers. Each input integer *Number* is added to the current sum *s* and the current value of the profit *max* is recalculated. If at some step the value of *s* becomes negative, then we set it to zero and continue the process.

```
for(i = 0; i < n; i++)
{
  scanf("%d",&Number);
  s += Number;
  if (s > max) max = s;
  if (s < 0) s = 0;
}
```

Print the answer.

```
printf("%d\n",max);
```

1618. The Longest Common Subsequence

Two sequences of integers are given. Find the length of their longest common subsequence (the subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements).

Input. First line contains the length *n* ($1 \le n \le 1000$) of the first sequence. Second line contains elements of the first sequence – integers no more than 10000 by absolute value. Third line contains the length of the second sequence *m* ($1 \le m \le 1000$). Fourth line contains elements of the second sequence – integers no more than 10000 by absolute value.

Output. Print the length of the longest common subsequence, or 0 if it does not exist.

Sample input
```
3
1 2 3
4
2 1 3 5
```

Sample output
```
2
```

Algorithm analysis

A ***subsequence*** of a sequence is a set of elements that appear in left-to-right order, but not necessarily consecutively. A subsequence can be derived from a sequence only by deletion of some elements.

For example, consider the sequence {2, 1, 3, 5}. Then

- {1, 5}, {2}, {2, 3, 5} are subsequences;
- {5, 1}, {2, 3, 1} are not subsequences;

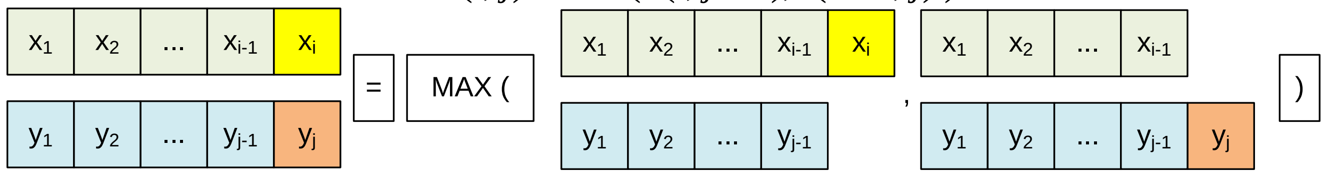
A ***common subequence*** of two sequences is a subsequence that appears in both sequences. A ***longest common subequence*** (***lcs***) is a common subsequence of maximal length.

For example, the longest common subequence of {1, 2, 3} and {2, 1, 3, 5} can be {1, 3} or {2, 3}. The length of lcs is 2.

Let f(i, j) be the length of the longest common subsequence of sequences $x_1x_2\ldots x_i$ and $y_1y_2\ldots y_j$.

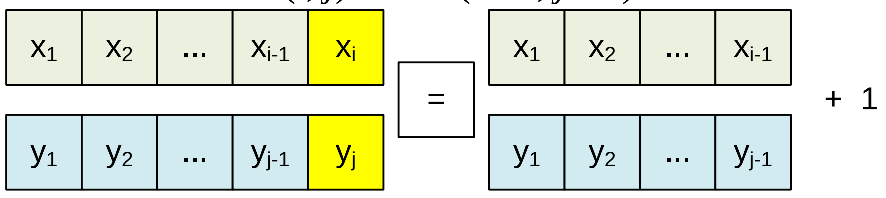
If $x_i \neq y_j$, then we find ***lcs*** among $x_1x_2\ldots x_i$ and $y_1y_2\ldots y_{j-1}$, and also among $x_1x_2\ldots x_{i-1}$ and $y_1y_2\ldots y_j$. Return the biggest value:

$$f(i, j) = \max(\ f(i, j-1),\ f(i-1, j)\ )$$



If $x_i = y_j$, then we find ***lcs*** among $x_1x_2\ldots x_{i-1}$ and $y_1y_2\ldots y_{j-1}$:

$$f(i, j) = 1 + f(i-1, j-1)$$



If one of the sequences is empty, then their lcs is empty:

$$f(0, j) = f(i, 0) = 0$$

Let’s summarize the recurrence relation:

$$f(i,j)=\begin{cases}\max(f(i,j-1),f(i-1,j)), x_i \neq y_j\\ f(i-1,j-1)+1, x_i = y_j\\ 0, i=0 \; or \; j=0\end{cases}$$

The values f(*i*, *j*) will be stored in array m[0..1000, 0..1000], where m[0][*i*] = m[*i*][0] = 0. Each next line of array m[*i*][*j*] is calculated through the previous one. Therefore, to find the answer, it is enough to keep in memory only two lines of length 1000.

Example

Let X = *abcdgh*, Y = *aedfhr*. The the longest common subsequence is *adh*, its length equals to f(6, 6) = 3.

f(i, j)		X	a	b	c	d	g	h
		0	1	2	3	4	5	6
Y	0	0	0	0	0	0	0	0
a	1	0	1(a)	1	1	1	1	1
e	2	0	1	1	1	1	1	1
d	3	0	1	1	1	2(d)	2	2
f	4	0	1	1	1	2	2	2
h	5	0	1	1	1	2	2	3(h)
r	6	0	1	1	1	2	2	3

f(6, 6) = max(f(6, 5), f(5, 6)) = max(2, 3) = 3, because Y[6] = *r* ≠ *h* = X[6].
f(5, 6) = 1 + f(4, 5) = 1 + 2 = 3, because Y[5] = *h* = X[6].

Algorithm realization

Arrays x and y contain input sequences, *n* and *m* are their lengths. Array mas contains two last lines of dynamic calculations.

```
#define SIZE 1010
int x[SIZE], y[SIZE], mas[2][SIZE];
```

Main part of the program. Read input sequences to arrays, starting from the first index. Then read the data into x[1..*n*] and y[1..*m*].

```
scanf("%d",&n);
for(i = 1; i <= n; i++) scanf("%d",&x[i]);
scanf("%d",&m);
for(i = 1; i <= m; i++) scanf("%d",&y[i]);
```

Fill array mas with zeroes. Dynamically calculate the values f(*i*, *j*). Initially mas[0][*j*] contains the values f(0, *j*). Then put into mas[1][*j*] the values f(1, *j*). Since to calculate f(2, *j*) it is enough to have the values of the previous row of mas array, the values of f(2, *j*) can be stored in mas [0][*j*], the values of f(3, *j*) in mas [1][*j*] and so on.

```
memset(mas,0,sizeof(mas));
for(i = 1; i <= n; i++)
for(j = 1; j <= m; j++)
  if (x[i] == y[j])
    mas[i%2][j] = 1 + mas[(i+1)%2][j-1];
  else
    mas[i%2][j] = max(mas[(i+1)%2][j],mas[i%2][j-1]);
```

Print the answer, that is located in the cell mas[*n*][*m*]. Take the first argument modulo 2.

```
printf("%d\n",mas[n%2][m]);
```

Algorithm realization – recursion

```
#include <stdio.h>
#include <string.h>
#define SIZE 1002

int x[SIZE], y[SIZE], dp[SIZE][SIZE];
int n, m, i, j, res;

int max(int i, int j)
{
  return (i > j) ? i : j;
}

int lcs(int *x, int *y, int m, int n)
{
  if (m == 0 || n == 0)
    return 0;
  if (dp[m][n] != -1) return dp[m][n];

  if (x[m] == y[n])
    return dp[m][n] = 1 + lcs(x, y, m - 1, n - 1);
  else
    return dp[m][n] = max(lcs(x, y, m, n - 1), lcs(x, y, m - 1, n));
}

int main(void)
{
  scanf("%d", &n);
  for (i = 1; i <= n; i++) scanf("%d", &x[i]);
  scanf("%d", &m);
  for (i = 1; i <= m; i++) scanf("%d", &y[i]);

  memset(dp, -1, sizeof(dp));
  res = lcs(x, y, n, m);
  printf("%d\n", res);
  return 0;
}
```

988. Subsequence

Given a sequence, find the length of the largest strictly increasing subsequence.

Input. First line contains the length n ($1 \le n \le 1000$) of the sequence. The second line contains the sequence itself. All numbers are integers not exceeding 10000 by absolute value.

Output. Print the maximum length of strictly increasing subsequence.

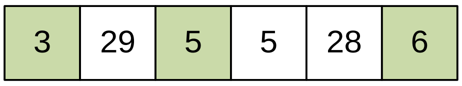
Sample input	Sample output
6 3 29 5 5 28 6	3

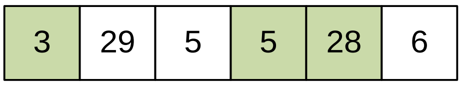
Algorithm analysis

In this problem you must find the length of the longest increasing subsequence (LIS).

Example

For the sample given, LIS can be {3, 5, 6} or {3, 5, 28}. The length of the LIS is 3.





Algorithm realization

Store the input sequence in the array x. Declare an auxiliary array lis.

```
#define MAX 1001
int x[MAX], lis[MAX];
```

The main part of the program. Read the input sequence.

```
scanf("%d",&n);
for(i = 0; i < n; i++) scanf("%d",&x[i]);
```

Carry out n iterations of the algorithm.

```
for (len = i = 0; i < n; i++)
{
  pos = lower_bound(lis,lis+len,x[i]) - lis;
  if (pos < len) lis[pos] = x[i]; else lis[len++] = x[i];
}
```

Print the length of LIS.

```
printf("%d\n",len);
```