Први домаћи задатак – билтен са решењима

1. Androvera

Текст задатака

време	меморија	улаз	излаз
1 s	64 Mb	стандардни улаз	стандардни излаз

1. Сваке године, Београд је домаћин фудбалског дербија (утакмице између Црвене звезде и Партизана). Организатори 144. вечитог дербија су одабрали неколико полицијских инспектора да буду супервизори сваке навијачке фракције током дербија. Супервизори су задужени да помажу навијачима током боравка на стадиону и непосредно пре и након утакмице. Ради безбедности, сви навијачи исте фракције имају једног супервизора- како би заједно били на стадиону. Међутим, један од организатора дербија направио је велику збрку при продаји карата и сваком навијачу при куповини карте је саопштио насумично име супервизора. Ваш задатак је да напишете програм који ће одредити минимални број навијача које организатори морају обавестити (да имају новог супервизора) како би се поправио проблем. При томе важи: да сваки супервизор буде задужен за навијаче из највише једне фракције, као и да сви навијачи из исте фракције имају истог супервизора. Ако постоји више супервизора него фракција навијача, онда неки супервизори могу остати и без задужења.

Улаз: Прва линија стандардног улаза садржи два цела броја **N** ($1 \le N \le 100$) и **M** ($1 \le M \le 15$, $M \le N$), где N је број супервизора и M је број фракција навијача. Наредних N линија описује број навијача за које је задужен сваки супервизор: свака линија садржи M целих бројева Xij ($0 \le Xij \le 10$), где Xij је број навијача из j-те фракције за које је задужен i-ти супервизор (први ред садржи X_{11}, X_{12}, X_{13}, …, X_{1M}, други ред садржи X_{21}, X_{22}, X_{23}, …, X_{2M}, …). Постоји бар 2 навијача из сваке фракције на 144. дербију.

Излаз: Стандардни излаз мора да садржи један цео број: минимални број навијача који ће бити обавештени

Пример

Улаз	Излаз
3 2 4 9 1 0 2 0	5

Објашњење: Први супервизор је задужен за 4 навијача из 1.фракције и 9 навијача из 2.фракције. Други супервизор је задужен за 1 навијача из 1.фракције и 0 навијача из 2.фракције. Трећи супервизор је задужен за 2 навијача из 1.фракције и 0 навијача из 2.фракције. Резултат је да први супервизор буде задужен за 2. фракцију, а трећи супервизор да

буде задужен за 1. фракцију. Зато 4+1 навијача треба обавестити да је трећи супервизор задужен за њих.

Опис решења

Можемо да напишемо рекурентну једначину на овај начин:

$dp(A, B) -$ решење за подскуп A користећи првих B супервизора
$X[A][B] -$ број људи из тима B који су оригинално додељени инспектору A
$bitmask(T)$
$-$ скуп који садржи индексе свих јединица у бинарној репрезентацији целог броја T
$dp(T, K + 1)$

$$= \begin{cases} min\begin{cases} dp\left(T - 2^i, K\right) + \sum_{j=0}^{M-1} X[K+1][j] - X[K+1][i], & \forall i \in bitmask(T) \\ dp(T, K) \end{cases}, & T \neq 0 \\ 0, & T = 0 \end{cases}$$

Дакле, да би добили решење, морамо да изгенеришемо све подскупове скупа који садржи све фракције навијача, да итерирамо кроз све фракције у подскупу и да итерирамо кроз све полицијске инспекторе. Имамо N инспектора, M фракција и 2^M подскупова гракција, те решење са динамичким програмирањем ће имати временску сложеност од O(N * M * 2^M).

Имплементација

```
#include <iostream>
#include <cstring>
using namespace std;

const int INF = 500000;

int main() {
   int N, M;
   int X[100][15];
   int noPeople[15];
   int transformation[100][15];
   memset(noPeople, 0, sizeof(noPeople));
   cin>>N>>M;
   for (int k=0; k<N; k++)
      for (int i=0; i<M; i++){
         cin>>X[k][i];
         noPeople[i] += X[k][i];   //укупан број људи на дербију за фракцију i
      }
   for (int k=0; k<N; k++)
      for (int i=0; i<M; i++)
       transformation[k][i] = noPeople[i] - X[k][i];
//број навијача из фракције i који нису додељени инспектору K
   int dp[32768];
   dp[0] = 0;
   for (int k=1; k<(1<<M); k++)
      dp[k] = INF;
```

```
  for (int k=0; k<N; k++)
    for (int i=(1<<M)-1; i>=0; i--) {
      int tmp = i;
      int exp = 0;
      while (tmp > 0) {
        if (tmp % 2 == 1)
          dp[i] = min(dp[i], dp[i - (1<<exp)] +
                transformation[k][exp]);
        tmp /= 2;
        exp++;
      }
    }
  cout<<dp[(1<<M) - 1]<<endl;
  return 0;
}
```

Најчешће грешке

Идеја 1: Не користити динамичко програмирање, већ израчунавањем максимума по колонама одређивати који навијачи остају код свог супервизора. Ако је у неком елементу *X[k][i];* постигнут максимум у колони i, онда је k. супервизор задужен за фракцију i.

Тест пример
2 2
10 8
6 2

По тој идеји, супервизор 1. је задужен за фракцију 1, а супервизор 2 за фракцију 2. Укупан број навијача који мења супервизора је 8+6=14. Али, оптималније је да број навијача који мења супервизора буде 10+2=12.

Идеја 2: Изабрати највећи број у матрици као ону фракцију која остаје код тог супервизора. Остале навијаче из те фракције пребацити код њега и потом из даљег разматрања елиминисати колону и ред у којој се налазио дати број.

Међутим, то решење не ради на следећем тест примеру:

2 2
30 20
20 0

Та идеја као решење даје 40, јер се максимум матрице 30 бира као онај број навијача који не мења 1. супервизора, те онда морају да 20+20 навијача промене супервизора. Али, ако се пребаци само 30 навијача из 1. фракције код 2. супервизора добија се да укупно решење је 30 што је мање од 40.

2. Болеро

Текст задатака

vreme	memorija	ulaz	izlaz
1 s	1000 Mb	standardni ulaz	standardni izlaz

Дат је низ целих бројева дужине N. Треба одредити најдужи строго растући подниз, не обавезно узастопних елемената датог низа.

Улаз

У првом реду се налази број N, а у следећем реду се налази N целих бројева

Излаз

Потребно је исписати дужину најдужег растућег подниза датог низа.

Ограничења

$0 < N < 100\,000$

$0 < a[i] < 2\,000\,000\,000$

Пример

Улаз	Излаз
7 1 3 2 5 9 7 6	4

Објашњење примера

Поднизови највеће дужине су:

1 2 5 7
1 2 5 9
1 2 5 6
1 3 5 7
1 3 5 9
1 3 5 6

Опис решења

Користимо динамичко програмирање.

Мора се водити рачуна за сваку дужину K, који је до тад познат подниз дужине K са најмањим последњим елементом. Тада се за следећи елемент a[i + 1] у улазном низу нађе први подниз који чији највећи елемент је већи (или једнак) од a[i + 1], и замени се са a[i + 1]. У случају да не постоји, то значи да може a[i + 1] да се стави на крају до тада најдужег подниза и да ће бит подниз који је већи за један.

Имплементација

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
int main()
```

```
{
   int n;
    scanf("%d", &n);
   vector<int> a;
   a.resize(n);

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

   vector<int> krajevi;
  krajevi.push_back(a[0]);
   for(int i = 1; i < n; i++)
   {
      vector<int>::iterator pozicija = lower_bound(krajevi.begin(), krajevi.end(), a[i]);
      if (pozicija == krajevi.end())
      {
         // trazeni element je manji od prvog
         krajevi.push_back(a[i]);
         continue;
      }

      *pozicija = a[i];
   }

   printf("%d", static_cast<int>(krajevi.size()));
}
```

Најчешће грешке

3. Cantante

Текст задатака

vreme	memorija	ulaz	izlaz
1 s	1000 Mb	standardni ulaz	standardni izlaz

Дат је низ бројева. Потребно је направити програм који ће израчунати дужину најдужег подниза (не нужно узастопних елемената) датог низа са својством да је збир бројева у то м поднизу дељив задатим бројем M.

Улаз

У првој линији улаза налази се природан број M, мањи или једнак 1 000. У другој линији улаза налази се природан број N, мањи или једнак 10 000, који означава број елемената д атог низа. У следећих N редова налазе се елементи низа, у свакој линији по један број. Ел ементи низа су природни бројеви, мањи или једнаки 10 000.

Излаз
Програм исписује дужину траженог подниза.
Ограничења
M < 1 000
N < 10 000
a[i] < 10 000
Пример
Улаз Излаз
13 3
5
17
3
7
15
2
Објашњење примера
Подниз који се тражи је (17, 7, 15)

Опис решења

Проблем се решава динамичким програмирањем, тако што се памти за сваку дужину низа који су најдужи поднизови који дају сваки остатак при сабирању и дељењу са M. Такав низ се ажурира са сваким новим учитаним елементом.

Имплементација

```
#include <cstdio>
#include <vector>
using namespace std;
int main()
{
   int m;
   int n;
   scanf("%d%d", &m, &n);

   vector<int> preth(m, -1);
   vector<int> tek(m);
   preth[0] = 0;

   for(int i = 0; i < n; i++)
   {
      int clan;
      scanf("%d", &clan);
      clan = clan % m;
      for (int j = 0; j < m; j++)
      {
         if (preth[j] == -1)
         {
            tek[(j + clan) % m] = preth[(j + clan) % m];
         }
         else
         {
            tek[(j + clan) % m] = std::max(preth[(j + clan) % m], preth[j] + 1);
```

```
        }
      }
      preth.swap(tek);
    }

    printf("%d\n", preth[0]);
}
```

Најчешће грешке

4. **Danube**

Текст задатака

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
0,7 s	64 MB	standardni ulaz	standardni izlaz

Дат је низ са N природних бројева. Дозвољена Вам је следећа операција трансформације низа: убацивање тачно једног целог броја у овај низ. Можете изабрати позицију убаченог елемента и његову вредност. Ову операцију треба извршити са циљем да максимизирате дужину најдужег подниза који се састоји од **узастопних вредности елемената** датог низа у строго растућем поретку. За потребе овог задатка, сматрамо да подниз настаје из низа изостављењем неких елемената низа без измене редоследа елемената. Нпр. подниз низа 11, 10, 12, 15 јесте подниз 11, 12, 15, али низ 10, 11, 15 није подниз низа 11, 10, 12, 15.
Ограничења: $1 \leq N \leq 300000$, сви чланови низа су мањи од 1 000 001
Улаз:

4

1 1 2 2

Излаз:

3

Убацити 3 на 4. позицију: 1 1 2 **3** 2

Добијамо жељени најдужи растући подниз узастопних вредности: **1** 1 **2** **3** 2

Улаз:

5

13 14 12 14 16

Излаз:

4

Убацити 15 на 3. позицију: 13 14 **15** 12 14 16

Добијамо жељени најдужи растући подниз узастопних вредности: **13 14 15** 11 14 **16**

Улаз:

6

1002 1001 1002 1003 1005 1007

Излаз:

5

Убацити 1004 на 5. позицију: 1002 1001 1002 1003 **1004** 1005 1007

Добијамо жељени најдужи растући подниз узастопних вредности: 1002 **1001 1002 1003 1004 1005** 1007

Уочите да ако нпр. убацимо вредност 1006 на 6. позицију, онда добијамо следећи растући подниз узастопних вредности: 1002 1001 1002 1003 **1005 1006 1007**

Опис решења

Означимо улазни низ са ***v***.

Пролазом кроз низ одржавамо ДП стање:

- dp[x][0] = максимална дужина подниза који се завршава у елементу чија вредност је једнака ***x*** ако не убацујемо додатни елемент
- dp[x][1] = максимална дужина подниза који се завршава у елементу чија вредност је једнака ***x*** ако је убачен додатни елемент

Пролазом кроз низ ***v***, морамо да ажурирамо следеће вредности за члан v[i]:

- dp[v[i]][0] = 1 + dp[v[i]-1][0]
- dp[v[i]][1] = max(1 + dp[v[i]-1][1], 2 + dp[v[i]-2][0])

Имплементација

```
#include <cstring>
#include <iostream>

using namespace std;

const int MaxVrednost = 1000001;

int N; //dimenzija niza
int DP[MaxVrednost + 5][2];
int rezultatDuzina;

int main() {
  cin >> N;
  while (N--) {
    int x;
```

```
    cin >> x;
    ++x;
    rezultatDuzina = max(rezultatDuzina, (DP[x][0] = DP[x - 1][0] + 1) + 1);
    rezultatDuzina = max(rezultatDuzina, DP[x][1] = max(DP[x - 1][1] + 1, DP[x - 2][0] + 2));
  }
  cout << rezultatDuzina << "\n";
  return 0;
}
```

Најчешће грешке

5. Епимено

Текст задатака

Опис решења

Имплементација

Најчешће грешке

6. Felicita

Текст задатака

Опис решења

Имплементација

Најчешће грешке

САБИРАЊЕ 2 ЦЕЛА БРОЈА

ПРОВЕРА ЗНАЊА ОСНОВНИХ улазно-излазних наредби

C++:

```
#include <cstdio>
#include <iostream>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
}
```

Java:

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Main {
    public static void main(String[] args) throws java
```

```
        .lang.Exception {
        Scanner in = new Scanner(System.in);
        System.out.println(in.nextInt() + in.nextInt());
    }
}
```

Python 3

```
def read_number():
    data = get_chuck()
    try:
        return int(data)
    except ValueError:
        return float(data)


def input_generator():
    while 1:
        data = list(input().split(' '))
        for number in data:
            if len(number) > 0:
                yield(number)


input_parser = input_generator()
def get_chuck():
    global input_parser
    return next(input_parser)


a = read_number()
b = read_number()
print(a + b)
```

C#:

```
using System;


public class Solution
{
```

```
        public static void Main()
    {
        string[] tokens = System.Console.ReadLine().Split();

        //Parse element 0
        int a = int.Parse(tokens[0]);

        //Parse element 1
        int b = int.Parse(tokens[1]);

        System.Console.WriteLine((a + b));
        }
}
```