Анализ на задачите от състезанието по програмиране на **18 декември 2011 г.**

A. Системата MIU

(Николай Киров)

Задачата е техническа, няма никакви „подводни камъни”.

B. Делене на половина

(Николай Киров)

Задачата е динамично програмиране, от книгата на Наков, и др. - Братска подялба [8.2.2].

C. Повтарящи се числа

(Николай Киров)

Най-лесната задача – в масив от 100000 елемента се брои колко пъти е прочетено съответното на индекса число и после се търси с максимален елемент в масива с < в права посока.

D. Гигантско микадо

(Антония Ташева)

Задачата има ограничение на броя горни клечки (1000), което подсказва, че може да се поддържа динамичен списък от тях. Всяка поредна клечка се добавя автоматично в този

списък и от него се премахват всички предишни клечки (отсечки), с които последната има вътрешно пресичане. И така до края. Извеждат се запаметените индекси на клечките в списъка.

E. Единици

(Емил Келеведжиев)

Моделираме процеса на „ръчното умножение" на числото 111...111 със себе си. Така трябва да намерим сумата по колонки (с евентуални преноси, движейки се отдясно-наляво) в таблицата:

```
         111...111
        111...111
       111...111
  …
  111...111
 111...111
111...111
```

Броят на единиците в колонките отдясно-наляво е 1, 2, 3, ..., n–1, n, n–1, n–2, ..., 3, 2, 1. В програмата **ones_ver1.cpp** има два цикъла, в които индексната променлива се променя от 1 до n и от n+1 до 2*n–1, с което се запълват елементите на масива `d`, като се отчита броя на единиците и стойността на преноса, записван в променливата `c`.

Програмата **ones.cpp** не използва масив, а отчита закономерността, с която се редуват получените суми в колонките.

```
//ones_ver1.cpp
#include<iostream>
using namespace std;
int d[1000100]; int n;

void run ()
{
  memset(d,0,sizeof(d)); cin >> n;
  int c=0;
  for(int i=1;i<=n;i++)
  { int v = i+c; d[i]= v%10; c=v/10;}
```

```
  int a=n;
  for(int i=n+1;i<=2*n-1;i++)
  { a--; int v=a+c; d[i]=v%10; c=v/10;}
  for(int i=2*n-1;i>=1;i--) cout << d[i];
  cout << endl;
}
int main()
{
 int nt; cin >> nt;
 for(int i=1;i<=nt; i++) run();
}
```

F. Низове

(Емил Келеведжиев)

Разглеждаме всички пермутации на елементите 1,2,..,n и ги подреждаме лексикографски в таблица с n стълба и n! реда. Първите (n–1)! от тези редове започват с 1, следващите (n–1)! реда започват с 2 и т.н. Вътре в групата на първите (n–1)! реда, първите (n–2)! реда имат 2 във втория стълб на таблицата, следващите (n–2)! реда имат 3 във втория стълб и т.н.

За да намерим в кой ред на таблицата се намира пермутацията p[1], p[2], p[3], … p[n], трябва да отброим (p[1]–1) пъти по (n–1)! реда, след това намаляваме с 1 тези елементи на последователността p[2], p[3], …, p[n], които са по-големи от p[1]. Сега отброяваме (p[2]–1) пъти по (n–2)! и извършваме подобна трансформация с p[3], …, p[n], като намаляваме с 1 елементите, които са по-големи от p[2]. След това отброяваме (p[3]–1) пъти по (n–3)! реда и т.н., повтаряме докато достигнем изчерпим редицата p[1], p[2], p[3], …, p[n].

Сумата на всички отброявания дава номера на търсеното място.

G. Цифри

(Велислав Николов)

Задачата е давана на 2-ри рунд на COCI през ноември 2010. Оригиналното условие и анализа може да прочетете тук (CONTEST #2, зад 2).

H. Бира

(Велислав Николов)

Задачата бе е давана на бронзовата дивизия в USACO през януари 2008. Оригиналния анализ може да прочетете тук, а условието й тук.

I. Зайци

(Велислав Николов)

Задачата се решава по следния начин:

1. Преизчисляване на всички числата на Фибоначи и добавянето им в индексно дърво **`allFibTree`** (то е най подходяща структура за актуализация на стойност на дадена позиция и запитвания за сумата на числа в даден интервал), използвайки подхода на динамичното оптимиране. Понеже тези числа растат изключително бързо, се налага и използването на големи числа.

2. За всеки тестов пример копираме в отделно индексно дърво **`tree`** вече изчислените стойности от **`allFibTree`**.

2.1. За всяка заявка тип **add** актуализираме **`tree`**, а за всяка **get** извличаме от **`tree`** сумата на елементите в зададения интервал. И двата вида заявки обработваме за логаритмично време.

2.2. Преобразуваме отговора в съответната бройна система и взимаме последните му S символа, което не е най-оптималното.

Аз лично се възползвах от класа, за големи числа в **Java** – **`BigInteger`**. Той предоставя и функционалност за четене и записване на числа в K-на бройна система.

Отбора на СУ използваха по едно индексно дърво за всяка бройна система (общо 34) и държаха в съответното дърво само последните 20 символа от преобразувания отговор в K-ична бройна система.

J. Отпуски

(Велислав Николов)

Задачата бе е давана на сребърната дивизия в USACO през ноември 2008. Оригиналния анализ може да прочетете тук, а условието й тук.