Идеи за решаване на задачите

Задача 1. Игра за богати – 2

Тази задача е предложена от Евгений Василев,
катедра Информатика, НПМГ „Акад. Л. Чакалов" – София

Ако в кръга няма празна касичка, решението е лесно. Взимате по 2 левчета от касичката с най-малък номер докато я изпразните. Ако в нея има нечетно количество левчета, взимате по 2 докато остане последното – него го прехвърляте в сметката заедно с 1 левче от съседната касичка. Действайки така е сигурно, че винаги можете да изпразните касичката с най-малък номер. Повтаряте действието за останалите касички докато не прехвърлите всичко в сметката. За последната касичка е възможно да остане 1 самотно левче - ще трябва да прехвърлите и него, макар и с непълна транзакция. Т.е. решението е частното на сумата от парите в касичките с 2 плюс остатъка.

Ако изначално има празни касички, в кръга са се обособили сегменти съседни празни и непразни касички. Трябва да се пресметнат транзакциите за всеки сегмент от непразни съседни касички и да се сумират. Разглежданата особеност позволява да реализираме алгоритъма без да използваме масив за парите във всички касички. Изчисляваме сумата за поредния сегмент непразни касички „в движение", докато четем данните. Като приключи сегментът (поредната касичка е с 0 лева), пресмятаме транзакциите за него и ги добавяме към общия брой. Трябва да се съобрази, че заради кръговата подредба на касичките е възможно първият прочетен сегмент от непразни касички да е продължение на последния! Това се случва ако първата и последната прочетени касички са непразни. Достатъчно е да запомним тези данни, както и четността на сумите в касичките от първия и последния сегмент. Ако сумите са били нечетни, това означава, че сме отчели 2 непълни транзакции, но тъй като става дума всъщност за един сегмент, двете непълни транзакции се обединяват в една пълна и трябва да коригираме (намалим с 1) транзакциите.

Кодът на програмата, написан на езика C++

```
#include <cstdio>

int n,                // брой касички
    current, prev,    // парите в текущата и предишната касичка
    trans,            // транзакции
    cs,               // колко поредици от непразни касички
    sum,              // сумата в текущата поредица непразни
    first,            // парите в първата касичка в редицата
    odd_first_sum;    // сумата в първата поредица е нечетна (1)
int main() {
  scanf("%d %d",&n, &prev);
  first=sum=prev;
  for ( ; --n ; ) {
    scanf("%d",&current);
    if (current) sum += (prev=current);
    else if (prev) {
      trans+=(sum+1)>>1;
      if (cs++ == 0) odd_first_sum = sum & 1;
      prev=sum=0;
```

```
    }
  }
  if (prev) {
    trans+=(sum+1)>>1;
    if (first && odd_first_sum && (sum & 1)) trans--;
  }
  printf("%d\n",trans);
  return 0;
}
```

Задача 2. Квадрати

Тази задача е предложена от Кристиян Цъклев,
8 клас, СМГ „Паисий Хилендарски”,
Разширен национален отбор по информатика (младша възраст) – 2010

Нека за базова точка на всеки специален квадрат приемем горния ляв връх. Можем да обходим с 2 вложени цикъла всички възможни позиции върху листа, върху които може да се разположат базови точки. Ако в дадена позиция за базова точка има червена точка, проверяваме дали може да се построи специален квадрат първо със страна t=1 cm, после – с 2, 3, 4 и т.н. до пределната възможна стойност на t. Построяването на специални квадрати е възможно, ако и в позициите на останалите 3 върха има червени точки. Самите позиции на останалите върхове лесно се изчисляват чрез координатите на базовата точка и големината на страната. Ако открием, че може да се построи специален квадрат, трябва да проверим дали лицето му не е рекордно голямо и в такъв случай да го запомним като търсен отговор.

Както се вижда от описанието и приложения код необходим ни е един външен цикъл, с който да местим едната координата на базовата точка (ще се изпъни 200 пъти), един вложен цикъл за местене на другата координата (той се изпълнява 200 пъти за всяка итерация на външния цикъл) и един още „по-вложен” цикъл за промяна големината на специалния квадрат в дадена позиция (той се изпълнява различен брой пъти за различни позиции на базовата точка, но приблизителната средна оценка е половината от широчината на страната на листа, т.е. 100 пъти средно за всяка базова точка). Или общият брой итерации на алгоритъма е пропорционален на широчината на страната на листа повдигната на трета степен ($200.200.200/2=1/2.200^3=C.200^3$) – към 4 милиона при конкретните ограничения на задачата. Нищо работа за съвременните компютри, дори и със съвсем скромни характеристики.

Ако листът хартия беше с 10 пъти по-голяма страна, то тогава щяхме да имаме големи ядове с бързодействието на програмата ни: ще са ни необходими към 4 милиарда изпълнения на най-вътрешния цикъл, в чието тяло се върши основната работа на алгоритъма. За такива ограничения на данните ще трябва да търсим по-ефективен алгоритъм – оставям Ви да се поровите из нета или да питате ръководителите си в търсенето му ☺

Кодът на програмата, написан на езика C++

```
#include<iostream>
using namespace std;
int n,x,y,total,maxx=-1;
bool coord[201][201];
int main()
```

```
{
     cin>>n;
     for(int i=0;i<n;i=i+1)
     {
          cin>>x>>y;
          coord[x][y]=true;
     }
     for(int i=0;i<200;i=i+1)
     {
          for(int j=0;j<200;j=j+1)
          {
               if(coord[i][j])
               {
                    int t=1;
                    while((i+t<=200)&&(j+t<=200))
                    {
                         if((coord[i][j+t])&&(coord[i+t][j+t])&&(coord[i+t][j]))
                         {
                              total=total+1;
                              if(t*t>maxx)
                              {
                                   maxx=t*t;
                              }
                         }
                         t=t+1;
                    }
               }
          }
     }
     cout<<total<<" "<<maxx<<"\n";
     return 0;
}
```

Задача 3. Твърдо „нье" на Фибоначи

Тази задача е предложена от Павлин Пеев,
ПМГ „Гео Милев" – Стара Загора
(варианти на задачата са давани и на други състезания)

Трябва да елиминираме първите *n* числа от редицата на „нефибоначиевите" за да достигнем до отговора – последното елиминирано от *n*-те. Започвайки от 4-тото фибоначиево число, всеки две последователни фибоначиеви числа *prevfib* (предишно, по-малко) и *currentfib* (текущо, по-голямо) ограждат поредица (порция) от *m* „нефибоначиеви". Очевидно *m*=*currentfib* – *prevfib* – 1. Ако *n* > *m*, елиминираме всичките *m* „нефибоначиеви" от текущата поредица (намаляваме *n* с *m*), и се премесваме в следващата поредица „нефибоначиеви" (за нея трябва да пресметнем нови стойности на *currentfib*, *prevfib* и *m*). Ако *n* ≤ *m*, елиминираме само *n* от поредицата „нефибоначиеви". Последното елиминирано – отговорът, е *prevfib* + *n*.

Кодът на програмата, написан на езика C++

```
#include <iostream>
using namespace std;
int main(){
  unsigned long long n, prevfib=3, currentfib=5;
  for (cin >> n; prevfib + n >= currentfib; ) {
    n-=currentfib-prevfib-1;
    currentfib+=prevfib;
```

```
    prevfib=currentfib-prevfib;
  }
  cout << prevfib+n << endl;
  return 0;
}
```

Задача 4. Клингони

Тази задача е предложена от Марий Йонов,
студент във ФМИ на СУ „Св. Климент Охридски"

Гледайки пределната стойност за *N* е много съмнително дали ще успеем да пазим в паметта всички съобщения – в най-лошия случай ще ни трябват 21.10^6 символа (байта), а повечето състезателни проверяващи системи прилагат някакви ограничения за използвана памет от програмите Ви, дори и да не са посочени явно. Последното май не го знаехте, нали? Е, и аз не го знаех ☹, докато на едно състезание уж перфектното ми решение получи 0 точки заради ***ml*** (memory limit).

Да помните: винаги трябва да сте нащрек ако ползваната памет надхвърли 15-20 MB!

Та – тъй, да минем нататък. След като не можем да пазим всичките съобщения, отпадат вариантите с поддържане на `map`, `set` или други подобни „хубавиня", дето ги има в библиотеката STL. Не можем да приложим и грубото решение да прочетем целия вход, да го сортираме и накрая да преброим двойките еднакви съседни съобщения и само това дето няма еднакво съседно е търсеното.

Задачата се решава чрез последователно еднократно (!) обработване на всяко едно поредно съобщение от входа, като обработката води до промяна на един низ – текущ отговор. Първоначално той съдържа 20 символа ***0***. Поредният низ от входа е своеобразен модификатор, който ***X***(итро) ***O***(бмислено) ***R***(ъчка) стройността на текущия отговор по следния начин: ***1*** на позиция *i* в низа-модификатор „сръчква" стойността в позиция *i* на текущия отговор към противоположната ѝ – ако е била ***1***, става ***0*** и обратно. Ако в позиция *i* на низа-модификатор има ***0***, стойността в позиция ***i*** на отговора не се променя. Какво ни дава това и къде всъщност е хитрината на такова ръчкане (операция)? Следната таблица илюстрира ефекта от няколко последователни прилагания на ***XOR*** за символите от дадена позиция

№ на низа	Начална стойност	1	2	3	4	Крайна стойност
Пореден низ (модификатор)		***1***	***0***	***1***	***0***	
Текущ отговор	***0***	***0***	***1***	***1***	***0***	***0***
Текущ отговор (след ***XOR***)		***1***	***1***	***0***	***0***	

Понеже единиците в низа-модификатор обръщат стойността в съответната позиция на текущия отговор е ясно, че четен брой обръщания на стойността ще доведат до първоначалното състояние на отговора (ефектът на „двойно преобърнатата палачинка") и само единствения недублиран низ-модификатор от входа ще доведе до

промяна на текущия отговор (по същество промяната е откопиране на низа-модификатор върху текущия отговор). При това е без значение в какъв ред се появяват низовете-модификатори на входа, в което лесно може да се убедим, ако разгледаме няколко варианта с разменен ред на низовете от предходната таблица (по-долу е показан един от тях):

№ на низа	Начална стойност	1 (предишен 2)	2 (предишен 1)	3 (предишен 4)	4 (предишен 3)	Крайна стойност
Пореден низ (модификатор)		***0***	***1***	***0***	***1***	
Текущ отговор	***0***	***0***	***0***	***1***	***1***	***0***
Текущ отговор (след ***XOR***)		***0***	***1***	***1***	***0***	

В таблиците може да откриете и една много полезна интерпретация на ***XOR*** при практическа реализация на код: *различни стойности в текущия отговор и низа-модификатор (операндите на операцията) дават резултат **1**, еднаквите стойност – резултат **0**.*

Кодът на програмата, написан на езика C++

```
#include <cstdio>

int main () {
  int n;
  char curr[21], ans[21] = "00000000000000000000";

  for (scanf ("%d\n", &n); n; n--) {
    scanf ("%s\n", curr);
    for (int j = 0; j < 20; ++j)
      if ( ans[j] == curr[j] ) ans[j] = '0';
      else ans[j] = '1';
  }

  printf ("%s\n", ans);
  return 0;
}
```