Идеи за решаване на задачите

Задача 1. Разстояние

Прочитаме в масив данните за текущите височини на дърветата. След това четем за всяко дърво колко ще е прирастът му и го добавяме към стойността на съответния елемент на масива. След като се изчислят височините след една година, остава да се намери кои са най-ниското и най-високото дървета. Търсенето се прави по стандартния начин: проверяваме всеки елемент на масива дали не е „по-добър" от някаква текуща рекордна стойност и ако е така, коригираме рекордната стойност да съвпада с тази на проверявания в момента елемент. В края на проверките рекордната стойност съвпада наистина с най-добрата в масива. В случая е полезно да търсим не самата рекордна стойност, а позицията на елемента от масива с такава стойност. Така по позициите на най-високото и най-ниското дърво можем да определим разстоянието между тях. Не трябва да се забравя, че при *N* дърветата междините между двойки съседни дървета са *N*-1. От тук следва, че търсеното разстояние е:

$$\left|индекс_{най-ниско} - индекс_{най-високо}\right| \cdot \frac{L}{N-1}$$

Допълнително е важно да се съобрази, че в горната формула всички величини са цели числа и ако се програмира директно, компилаторът ще приложи целочислена аритметика и резултатът ще е цяло число. За да принудим компилатора да използва аритметика с плаваща запетая е достатъчно да приложим един трик – да изпишем единицата в знаменателя като 1.0 Така в израза вече се появяват операнди, които са реални числа и резултатът също се изчислява като реално. Има и други начини да се постигне този ефект, но тях ще Ви ги разкажат ръководителите Ви или по-опитните Ви съученици.

Кодът на програмата, написан на езика C++

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

int main() {
  int h[1000],                  // височини на дърветата
      min_h_idx=0, max_h_idx=0, // номер (индекс) на най-ниското и най-високото
                                //след 1 год.
      N, L,                     // брой дървета и разстояние между крайните
      i,                        // брояч в цикли for
      j;                        // годишен прираст на поредното дърво

  cin >> L >> N;
  for (i=0; i<N; i++ ) {
    cin >> h[i];
  }

  for (i=0; i<N; i++ ) {
    cin >> j;
    h[i] += j;
    if (h[i] < h[min_h_idx]) min_h_idx = i;
    if (h[i] > h[max_h_idx]) max_h_idx = i;
  }
```

```
  i = max_h_idx - min_h_idx;    // най-високото може да е след най-ниското:
                                //  отрицателно i
  if (i<0) i = -i;              // ако е така - обръщаме знака му

  cout << i*L/(N-1.0) << endl;

  return 0;
}
```

Задача 2. Кебапчета

Чета броя на кебапчета за поредния ден и ако денят е нещастен (кебапчетата са нечетен брой) запомням 0 в променливата, предназначена за съхраняване на отговора на задачата. Ако денят е щастлив, има две възможности: Може това да е денят с най много кебапчета до момента измежду последните щастливи дни – запомням тази бройка в променливата-отговор. Или пък току що прочетеният брой не е „по-щастлив" от някой предишен – миии... хапвам си ги кебапченцата и марш към следващия ден да търся по-голямо щастие ☺
Стоплихте ли, а? Само 3 променливи ползвам: текущия брой кебапчета, броят кебапчета от „най-щастливия"ден и... някаква циклова променлива, дето използвам да местя дните.

Това е! Тичам, че има да пиша задачка за следващия кръг –... за наденички!

Кодът на програмата, написан на езика C++

```
#include<iostream>
using namespace std;
int main(){
    long long dni, tekkebapcheta, maxkebapcheta;
    cin>>dni;
    maxkebapcheta=0;
    for(; dni>0; dni=dni-1){
        cin>>tekkebapcheta;
        if(tekkebapcheta%2!=0){
            maxkebapcheta=0;
            }
        else if(maxkebapcheta<tekkebapcheta){
            maxkebapcheta=tekkebapcheta;
        }
    }
    cout<<maxkebapcheta<<"\n";
    return 0;
}
```

Задача 3. Бърничкане из кода

Нека предположим, че в N-те числа можем да намерим търсените M числа. Очевидно търсеното може да започва от 0-вото число на кода, от 1-то, от 2-то и т.н. до ...
Всъщност кое е последното число от N-те, което може да е начало на търсената поредица? Като поразмислим малко стигаме до заключението, че това е (N-M)-то число! Всяко по-задно число от N-те не може да е начало на търсената поредица, просто защото до края на редицата няма да са останали необходимите M числа. И така:

```
За всяка позиция в N-те числа, започвайки от 0-вата и завършваики с (N-M)тата:
     Проверяваме дали е начало на M-те търсени числа
```

Нещо такова:

```
for (start_pos = 0; start_pos <= N-M; start_pos++) {
  //Тук проверяваме M последователни числа
}
```

Самата проверка се извършва позиция по позиция докато числата от търсената поредица съвпадат с анализираните числа в кода, които започват от позиция start_pos. Ето така:

```
for (ofset = 0; ofset < M; ofset++) {
  if (число_от_кода[start_pos+offset] != число_от_търсената[offset]) break;
}
```

Работата на този цикъл приключва по 2 възможни начина: естествен край, при който не са открити разлики и в този случай ofset има стойност M или чрез преждевременно прекъсване при откриване на разлика (останалите сравнения са ненужни) и тогава ofset е различно от M. Значи:

ofset == M – намерена е търсената поредица на позиция start_pos, може да отпечатим резултата, че даже и да приключим веднага програмата

ofset != M – на позиция start_pos не се намира търсената поредица – отиваме да я търсим в следващата позиция.

Ако не открием търсената поредица числа на никоя от възможните позиции в кода – отпечатваме -1.

Кодът на програмата, написан на езика C++

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

int c[1000],         // кодът
    magic_seq[10];   // търсената специална поредица

int main() {

  int i, j, m, n, start_pos, ofset;

  cin >> m >> n;
  for (i=0; i<m; i++ ) cin >> magic_seq[i];
  for (i=0; i<n; i++ ) cin >> c[i];

  n = n-m;

  for (start_pos=0; start_pos<=n; start_pos++){
    for (ofset=0; ofset<m; ofset++)
      if (c[start_pos+ofset] != magic_seq[ofset]) break;
    if (ofset==m) {
      cout << start_pos << endl;
      return 0;
    }
  }
  cout << -1 << endl;
  return 0;
}
```

Задача 4.

Четем 49 времена на събуждане и ги записваме в масив. Да съобразим, че форматът на входните данни е доста неудобен за работа защото е комбинация от две числа – превръщаме входните данни в минути. Така ще работим само с едно число за времената. И после постъпваме по следния начин:

```
За всяка минута от денонощието:
  Проверяваме всеки апартамент ако се буди в тази минута – събуждаме и съседни
```

Нещо такова:

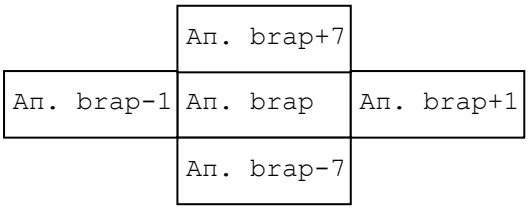
```
for(brmin=0;brmin<24*60+14*7;brmin=brmin+1){
        for(brap=0;brap<49;brap=brap+1){
          // Тук проверяваме дали се будим в brmin и трябва ли да се будят
          // съседи
        }
}
```

Сега остава да измислим как да правим проверката и как да будим съседи, ако е нужно. Самата проверка е лесна:

```
if(subujdane[brap]==brmin){
  // будим съседи
}
```

Буденето на съседите е свързано с определяне на номерата на апартаментите, които ще будим. Виждаме, че апартаменти с номера 1, 8, 15, ..., 43 нямат ляв съсед. Аналогично, тези с номера 7, 14, 21, ..., 49 нямат десен съсед, тези с номера по-малки от 8 нямат долен съсед, а с номера по-големи от 42 – нямат горен. Въоръжени с това наблюдение ще правим опит за събуждане на съседи в 4 посоки и за всяка от посоките трябва да проверим дали съществува съсед. Освен това има значение дали поредния апартамент се е събудил от собствения си будилник или е събуден от съседска кавга. Когато е събуден от собствен будилник, съседите му се будят в същия момент, а когато е събуден от съседска кавга съседите се будят след 7 минути. За да знаем типа на събуждане ни е необходим още един масив, в чиито елементи ще записваме тази информация за съответния апартамент. Първоначално в масива ще запишем, че всеки апартамент се буди от собствен часовник, но в последствие ще променяме това, ако се установи, че някой апартамент се буди принудително от комшии. За целта си избираме една стойност да означава събуждане от будилник (напр. 0) и друга стойност за събуждане от комшия(напр. 1).

Тънкият момент тук е, че когато от будилник се събудят и съседи, няма как да се обработи разпространение на събуждането към „по-леви” или „по-долни” апартаменти при кавгата, която ще вдигнат събудените съседи от будилника. И наистина, нека погледнем схемата по-долу: ако се налага да коригираме събуждането на съседите на апартамент$_{brap}$ то при следващата итерация на цикъла за поредната минута проверката дали да се буди даден апартамент ще пропадне за апартамент$_{brap-7}$ и апартамент$_{brap-1}$ (brmin вече ще е с 1 повече) и така няма да може да се стигне до събуждане на съседите им. Такъв проблем не съществува за апартаменти с по-голям номер от brap – през тях ще се премине при следващите итерации на вътрешния цикъл в рамките на текущата итерация за външния цикъл.



За да разрешим този проблем може да изхитруваме: когато коригираме времената на събуждане на съседните апартаменти да съвпадат с позвъняване на будилник, намаляваме текущото време с 1 минута и прекратяваме вътрешния цикъл. Като излезнем във външния цикъл по естествен начин променливата за време ще се увеличи с 1 и на практика ще се повтори итерацията от начало (за същата brmin).

И докато правим всички тези гимнастики не забравяме при всяко събуждане да помним времето, когато е станало. Тази променлива накрая ще е отговорът на задачата!

Има много други начини да решим задачата, но за тях ще говорим в някои от следващите кръгове на Урановата дивизия.

Кодът на програмата, написан на езика C++

```
#include<iostream>
using namespace std;
int main(){
    long long subujdane[49], tekmin, tekchas,brap,brmin,posledenspqst;
    int subuden_normalno[49], subujdane_sled;

    for(brap=0;brap<49;brap=brap+1){
        cin>>tekchas>>tekmin;
        subujdane[brap]=tekchas*60+tekmin;
        subuden_normalno[brap]=0;
    }

    for(brmin=0;brmin<24*60+14*7;brmin=brmin+1){
        for(brap=0;brap<49;brap=brap+1){
            if(subujdane[brap]==brmin){

                if (subuden_normalno[brap]<0) continue;

                if (subuden_normalno[brap]==0) subujdane_sled = 0;
                else subujdane_sled = 7;

                posledenspqst=brmin;

                //ako ima dolen apartament i toi bi spal sled subujdane_sled
                //minuti - subudi go
                if(brap-7>=0 && subujdane[brap-7]>brmin+subujdane_sled){
                    subujdane[brap-7]=brmin+subujdane_sled;
                    subuden_normalno[brap-7]=1;
                }

                //ako ima goren apart....
                if(brap+7<=49 && subujdane[brap+7]>brmin+subujdane_sled){
                    subujdane[brap+7]=brmin+subujdane_sled;
                    subuden_normalno[brap+7]=1;
                }

                //ako ima lqv...
                if(brap%7!=0 && subujdane[brap-1]>brmin+subujdane_sled){
```

```
                    subujdane[brap-1]=brmin+subujdane_sled;
                    subuden_normalno[brap-1]=1;
                }

                //ako ima desen..
                if(brap%7!=6 && subujdane[brap+1]>brmin+subujdane_sled){
                    subujdane[brap+1]=brmin+subujdane_sled;
                    subuden_normalno[brap+1]=1;
                }

                subuden_normalno[brap] = -1;

                if (subujdane_sled==0) {
                    brmin--;
                    break;
                }
            }
        }
    }
    cout<<posledenspqst/60<<":"<<posledenspqst%60<<"\n";
    return 0;
}
```