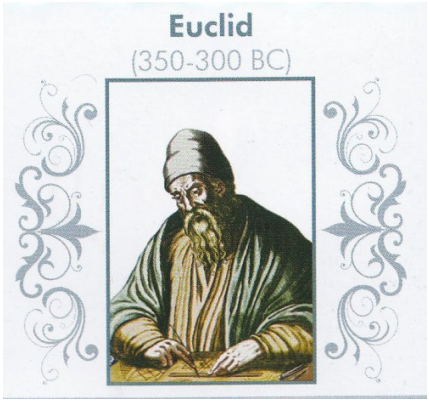
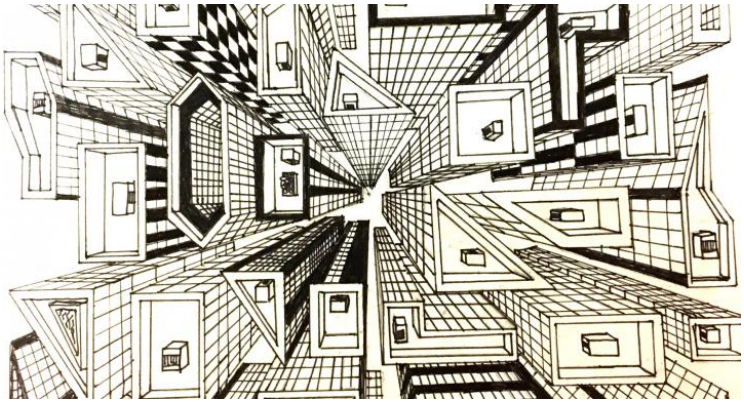
Задаци за размишљање – алгебарски алгоритми и редукција





1. На Кошутњаку је током првомајског уранка одржано такмичење за најбољу роштиљску кобасицу. Месар Сремац је направио ***N*** једнаких кобасица и потребно је да их (ножем) равномерно подели на ***K*** делова, тако да сваки од ***K*** чланова жирија добије једнаку количину кобасица за оцењивање. Да би подела била што квалитетнија, број резова кобасица мора бити што мањи. На пример, ако ***N***=2, ***K***=6, довољно је да месар сваку кобасицу са два реза подели на три једнака дела, што је укупно четири реза. На пример, ако ***N***=3, ***K***=4, месар може од сваке кобасице одрезати три четвртине. Три члана жирија оцениће те делове, а преостала три мања дела (од по једне четвртине) припашће четвртом члану жирија. Конструисати алгоритам сложености ***O (log (N+K))*** који рачуна најмањи укупан број резова потребан да се изврши тражена подела.

УЛАЗ	**ИЗЛАЗ**
20 20	0
49 7	0
35 36	35
90 54	36
56 98	84
56 48	40
51 34	17

1.
Укратко, наше решење је *rez* = **K**-NZD(**N**, **K**).

OPŠIRNIJI ODGOVOR
Замислимо да је месар преспојио свих N роштиљских кобасица једну за другом и тако добио велику кобасицу.
Праведна подела из формулације задатка (*тако да сваки од **K** чланова жирија добије једнаку количину кобасица за оцењивање*) изискује да сваки члан комисије добије један од **K** једнаких делова велике кобасице. Дакле, прережимо велику кобасицу са **K** – 1 резова.
Али, неки од тих **K** – 1 резова се не морају учинити ножем, јер представљају крајеве малих N роштиљских кобасица које смо спојили.
На пример, ако N=2, K=4, онда међу 3 реза само ће 2 бити учињена ножем (1. и 3. рез који ће пререзати 1. и 2. кобасицу на половине, док 2.рез ће пролазити између две мале кобасице).

Дакле, морамо пребројати колико има крајева малих роштиљских кобасица који представљају *већ изрезане крајеве*. Ако је **i**-ти рез по реду тзв. *већ изрезани крај*, онда важи да међу првих **i** делова (од **K** делова велике кобасице) можемо наћи неколико целих кобасица, нпр. **M** кобасица (**M** је цео број).

Дакле, (**i** / **K**) од **N** кобасица је једнако **M** кобасица тј. **M** = (**i** * **N**) / **K**.

Дакле, да би **M** био цео број, мора да важи да је **i*** **N** дељиво са **K**.

Зато креирамо бројачки циклус по сваком могућем **i** (из сегмента од 1 до **K** - 1) којим преварамо да ли смо наишли на *већ изрезане крајеве*.

Укратко, наше решење је *rez* = **K**-NZD(**N**, **K**).

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

int main () {
  int n, k; cin >> n; cin >> k;
  int rezani_kraj = 0;
  for (int i = 1; i < k; ++i)    rezani_kraj += (i * n % k == 0);
  cout << k - 1 - rezani_kraj << endl;
  return 0;
}
```

2. Пчелица Маја и њен друг Павао играју се на делу ливаде ограђене са N цветова поређаних у круг. Сматрајмо да је сваки цвет нумерисан редом бројевима од 1 до N. На почетку игре Маја се налази на цвету A, а Павао је на цвету B. Сваке секунде Маја прелети C светова редом и слети на цвет у кругу, а Павао прелети D цветова редом и слети на цвет у кругу. Игра траје све док се Маја и Павао не нађу на истом цвету. Конструисати алгоритам сложености O(log N) који ће израчунати број прелета које ће Маја и Павао учинити пре него што се нађу на истом цвету или одредити да ли се то никада неће догодити.

3. Мали Васа је добио задатак да изреже лист папира димензије **n** x **m** на квадрате максималне површине. Васа најпре исече највећи могућ квадрат тако што сече лист папира по најдужој страници (на пример за лист димензије **3** x **7**, највећи могућ квадрат је димензије **3** x **3).** Потом Васа склони квадрат и над преосталим правоугаоником понови исецање квадрата највеће површине. Кад исече највећи квадрат, Васа наставља исту операцију све док преостали правоугаоник не постане квадрат.

Конструисати алгоритам сложености ***O (log (n+m))*** који за дате вредности **n** и **m**, израчунава број квадрата који ће Васа добити након исецања на горе описан начин. У јединој линији стандардног улаза дати су цели бројеви **n** и **m** међусобно раздвојени бланко карактером ($0<\mathbf{n}<=10^{18}$,$0<\mathbf{m}<=10^{18}$). На стандардном излазу исписати број квадрата.

ПРИМЕР

УЛАЗ	**ИЗЛАЗ**
3 7	5
9999 9999	1

4. **Najkraća dopuna do palindroma**

vremensko ograničenje	memorijsko ograničenje
0.1 s	64 MB

Niska abaca nije palindrom (ne čita se isto sleva i sdesna), ali ako joj na početak dopišemo karaktere ac, dobijamo nisku acabaca koja jeste palindrom (čita se isto i sleva i sdesna). Napiši program koji za datu nisku određuje dužinu najkraćeg palindroma koji se može dobiti dopisivanjem karaktera s leve strane date niske.

Ulaz Sa standardnog ulaza se unosi niska sastavljena samo od N ($1 \leq N \leq 50000$ malih slova engleske abecede.

Izlaz Na standardni izlaz se ispisuje tražena dužina najkraće proširene niske koja je palindrom.

Primer 1

Ulaz

abaca

Izlaz

7

Primer 2

Ulaz

abcdefg

Izlaz

13

Rešenje je gfedcbabcdefg.

Primer 3

Ulaz

anavolimilovana

Izlaz

15

Reč je već palindrom, pa se ne dopisuje ni jedan karakter i rešenje je anavolimilovana.

Primer 4

Ulaz

anavolimilovanakapak

Izlaz

25

Rešenje je kapakanavolimilovanakapak.

Rešenje

Sporije rešenje (pokriva vremensko ograničenje za neke test primere)

Neka je data ulazna niska s="abbac"

Dovoljno je da postavimo poslednje slovo c ispred sadržaja niske s i dobićemo nisku "cabbac", koje jeste palindrom s minimalnom dužinom, i zadovoljava uslove zadatka.

Sličan postupak se može primeniti i u opštem slučaju, kada ulazna niska *s* sadrži podniz na početku niske koji je palindrom.

Predstavimo ulaznu nisku u obliku s = s1 + s2, gde s1 je najduži prefix od s, koji je palindrom.
Tada možemo da sastavimo niz t = s3 + s1 + s2, gde niska s3 je dobijena obrtanjem niske s2 (pročitane kao u ogledalu) .

Kako niska s1 je najduži palindrom, onda niska t je palindrom, napravljen po uslovima zadatka i to najkraći moguć.

Dužina niske t je:

L = n2+n1+n2=2*n2 + n1,

gde n1 je dužina za s1, n2 je dužina za s2 (naravno i za s3). Polazna dužina niske s je n tj. n = n1 + n2. Dakle, L = 2*(n-n1) + n1 = 2*n – n1 i program mora da ispiše L.

Ako želino da odredimo n1, onda redom proveravamo sve prefikse niske s i tražimo najduži palindrom među njima. Njegova dužina je n1.

```
#include <iostream>
#include <string>
```

```
using namespace std;

bool jePalindrom(const string& s, int n) {
for (int i = 0, j = n - 1; i < j; i++, j--)
if (s[i] != s[j])
return false;
return true;
}

int duzinaNajduzegPalindromskogPrefiksa(const string& s) {
for (int n = s.size(); n >= 1; n--)
if (jePalindrom(s, n))
return n;
}

int main() {
string s;
cin >> s;
// s razlazemo na prefiks + sufiks tako da je prefiks sto duzi
// palindrom. Tada je trazeni palindrom (palindrom koji se dobija sa
// sto manje dopisivanja slova na pocetak niske s) jednak:
// obrnut_sufiks + prefiks + sufiks
int duzinaPrefiksa = duzinaNajduzegPalindromskogPrefiksa(s);
int duzinaSufiksa = s.size() - duzinaPrefiksa;
int duzinaNajkracegPalindroma = duzinaSufiksa + duzinaPrefiksa + duzinaSufiksa;
cout << duzinaNajkracegPalindroma << endl;
return 0;
}
```

Algoritam složenosti O(n)

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

int duzinaNajduzegPalindromskogPrefiksa(const string& s) {
string sObratno(s.rbegin(), s.rend());
string str = s + '.' + sObratno;
vector<int> kmp(str.size() + 1, 0);
kmp[0] = -1;
for (int i = 0; i < str.size(); i++) {
int k = i;
while (k > 0 && str[i] != str[kmp[k]])
k = kmp[k];
kmp[i + 1] = kmp[k] + 1;
}
return min(kmp[kmp.size() - 1], (int)s.size());
}
```

```
int main() {
string s;
cin >> s;
// s razlazemo na prefiks + sufiks tako da je prefiks sto duzi
// palindrom. Tada je trazeni palindrom dobijen sa sto manje
// dopisivanja slova na pocetak jednak:
// obrni(sufiks) + prefiks + sufiks
int duzinaPrefiksa = duzinaNajduzegPalindromskogPrefiksa(s);
int duzinaSufiksa = s.size() - duzinaPrefiksa;
int duzinaNajkracegPalindroma = duzinaSufiksa + duzinaPrefiksa + duzinaSufiksa;
cout << duzinaNajkracegPalindroma << endl;
return 0;
}
```