Nešto izmedju
broj 01, godina 2011 / 2012

Zadatak 1 (NI73 Zajednički faktor) *Dat je niz a prirodnih brojeva dužine n. Broj k je faktor broja m ukoliko je m deljivo sa k. Za dati niz naći broj, različit od* 1*, koji je faktor najvećem broju elemenata niz. Ukoliko postoji više ovakvih vrednosti, štampati najveću.*

U prvom redu ulaza nalazi se prirodni broj n ($1 \leq n \leq 1000$*) koji predstavlja broj elemenata niza. Naredni red sadrži n prirodnih brojeva koji čine elemente niza.*

U prvom i jedinom redu izlaza štamapti traženu vrednost.

Ulaz	*Izlaz*
4 2 3 4 6	2

Rešenje: Kao prvo, da napomenemo da smo, greškom, iz postavke zadatka izostavili da navedemo ograničenje za vrednosti elemenata niza $2 \leqslant a_i \leqslant 10^6$. Sa ovakvim ograničenjem, trivijalno rešenje u kom se za svaki broj od 2 do M (M je maksimalna vrednost elemenata niza) proverava koliko elemenata niza deli, nije efikasno. Medjutim, modifikacijom ove ideje se može dobiti optimalnija složenost.

Definišimo niz $(num[i])_{i=1}^{M}$ gde $num[i]$ predstavlja broj elemenata niza deljivih sa i. Krajnje rešenje će predstavljati indeks maksimalnog elementa niza num (ukoliko ih ima više uzimamo najveći). Sada možemo da obradjujemo redom elemente niza. Za konkretni element $a[k]$ treba povećati vrednosti $num[d]$ za svaki delilac d broj $a[k]$. Ukoliko bismo delioce tražili medju brojevima iz segmenta $[2, a[k]]$, dobili bismo istu složenost kao i kod trivijalnog algoritma. Medjutim, delioce možemo naći i brže. Pretpostavimo da je d delilac broj $a[k]$. Tada je to i broj $a[k]/d = d_1$. Kako je $d \cdot d_1 = a[k]$, barem jedan od brojeva d i d_1 je manji ili jednak od $\sqrt{a[k]}$. Dakle, dovoljno je da za potencijalne delioce broja $a[k]$ ispitujemo elemente iz segmenta $[2, \sqrt{a[k]}]$. Ukoliko je neki od njih delilac, povećavamo ne samo element $num[d]$ već i vrednost $num[a[k]/d]$. Ovim se dobija algoritam čija je složenost $O(n \cdot \sqrt{M})$.

Postoji i efikasniji algoritam koji rešava ovaj problem. Pristup je komplikovaniji, tako da ovde navodimo samo ideju. Naime, mi smo ovde tražili delioce svakog broja zasebno i nismo iskoristili činjenicu da njih ima mnogo. Kako je ograničenje M relativno malo, možemo uz pomoć Eratostenovog sita inicijalizovati sve proste brojeve manje od M i pomoću njih tražiti faktorizaciju elemenata niza.

```
#include<stdio.h>
#include<math.h>
#define MAX_N 1005
#define MAX_M 1000005

int a [MAX_M], num [MAX_M], n, m, max, sol;

    int main()
    {
        scanf("%d", &n);
        m = 0;
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &a [i]);
            if (a [i] > m)
                m = a [i];
        }
        for (int i = 0; i <= m; i++)
            num [i] = 0;

        for (int i = 0; i < n; i++)
            for (int d = 2; d <= sqrt((double) a [i]); d++)
                if (a [i] % d == 0)
                {
```

```
                num [d]++;
                num [a [i] / d]++;
            }
    max = sol = 0;
    for (int d = 2; d <= m; d++)
        if (num [d] >= max)
        {
            max = num [d];
            sol = d;
        }

    printf("%d\n", sol);
    return 0;
}
```

Zadatak 2 (NI74 Lepa matrica) *Data je kvadratna matrica prirodnih brojeva dimenzije $n \times n$. Za matricu kažemo da je lepa ukoliko pri odabiru bilo kojih n elemenata matrice, pri čemu smo odabrali po tačno jedan element iz svake vrste i svakog reda, dobijamo isti zbir.*

Za datu matricu ispitati da li je lepa.

Prva linija ulaza sadrži prirodn broj n, ($1 \leq n \leq 20$) koji predstavlja veličinu matrice. Narednih n linija sadrže po n prirodnih brojeva koji čine elemente matrice.

U prvi i jedini red izlaza ispisati `'DA'` *ukoliko je ulazna matrica lepa, inače štampati reč* `'NE'`.

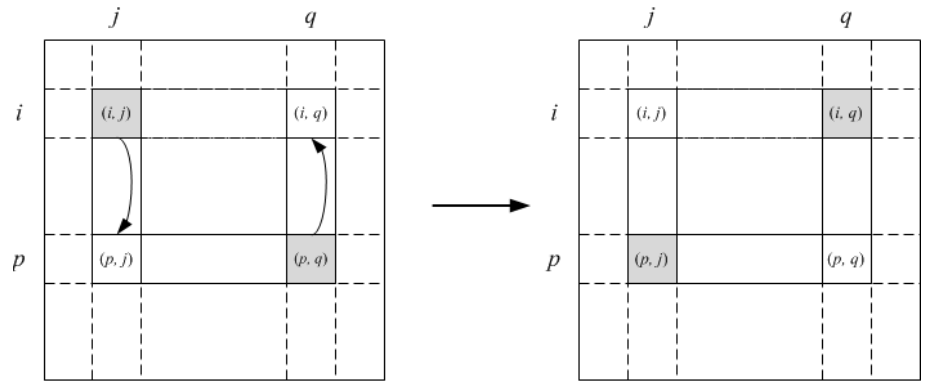
Ulaz	*Izlaz*
3	NE
1 2 3	
4 5 6	
9 8 7	

Matrica iz primera nije lepa jer imamo da je $2 + 4 + 7 \neq 1 + 6 + 8$.

Rešenje: Iako nije neophodno imajući u vidu postavku zadatka kojom se uzimaju u obzir samo matrice malih dimenzija, dajemo rešenje koje se može primeniti za velike matrice i koje predstavlja algoritam kvadratne složenosti. U jeziku teorije grafova problem se može definisati i kao: ispitati da li za dati težinski bipartitni graf preko matrice povezanosti, svako potpuno uparivanje ima konstantnu sumu. Označimo datu matricu sa A. Pretpostavimo da za neki par elemenata sa ideksima (i, j) i (p, q) važi da je $A[i, j]+A[p, q] \neq A[i, q]+A[p, j]$. U ovom slučaju bi bilo koji izbor koji uključuje elemente na pozicijama (i, j) i (p, q) mogao da se promeni u izbor sa različitom sumom ako umesto ova dva elementa uzmemo elemente sa pozicija (i, q) i (p, j) (za dodatno objašnjenje pogledati sliku). Ovim dobijamo da je za svaki par (i, j) i (p, q) potrebno da bude zadovoljen uslov $A[i, j] + A[p, q] = A[i, q] + A[p, j]$. Ispostavlja se da je ovo ujedno i dovoljan uslov. Formalni dokaz nije komplikovan ali ga ovde nećemo izneti do kraja. Ideja je da se od proizvoljnog izbora elemenata, a preko gornje smene medju parovima, može doći do izbora koji sadrži samo elemente sa glavne dijagonale. Time dobijamo da je suma svakog izbora jednaka sumi elemenata na glavnoj dijagonali, drugim rečima da je konsantna. Gornji pristup se može dodatno ubrzati. Naime, dovoljno je proveriti da li je $A[1, 1] + A[p, q] = A[1, q] + A[p, 1]$ za svaki par (p, q) (zašto?).

Složenost prvog pristupa je $O(n^4)$, dok poboljšana verzija ima kvadratnu složenost. Implementacija ovog algoritma je jednostavna i ovde je nećemo izložiti.

Zadatak 3 (NI75 Palindrom) *Palindrom je reč koja se jednako piše i sa leve i sa desne strane. Ukoliko reč nije palindrom, ona se može podeliti na podreči koje su palindromi. Podreč predstavlja deo reči sastavljen od uzastopnih karaktera. Napisati program koji za datu reč nalazi najmanji broj delova*



Slika 1: Promena izbora dva elementa matrice A

na koje je treba podeliti, tako da svaki dobijeni deo bude palindrom.

Prva i jedina linija ulaza sadrži string sastavljen od malih slova engleskog alfabeta.
U prvi i jedini red izlaza ispisati minimalni broj delova na koje treba podeliti ulazni string.

Ulaz	*Izlaz*
`abaccbcb`	`3`

Rešenje: Ovaj zadatak je sa Hrvatskog Županjskog takmičenja 2001. godine. Problem se rešava dinamičkim programiranjem. Označimo string sa s, a njegovu dužinu sa n. Primetimo prvo da rešenje uvek postoji, jer svako slovo posebno predstavlja palindrom. Definisaćemo pomoćni niz $(d[i])_{i=1}^{n}$ kao

$$d[k] = \text{ minimalni broj palindroma na koje se može podeliti podstring } s[1] \dots s[k]$$

Pretpostavimo da smo inicijalizovali vrednosti $d[1], d[2], \dots, d[k]$ i pokušajmo da preko njih izračunamo vrednost $d[k+1]$. Pretpostavimo da se optimalna vrednost za prefiks dužine $k+1$ dobija kada je karakter $a[k+1]$ desni kraj palindroma dužine b. Tada prvih $k+1-b$ karaktera moraju takodje biti podeljeni na optimalan način. Kako je $k+1-b \leqslant k$ imamo da je traženi optimalni broj palindroma upravo $d[k+1-b]+1$ (jedan dodajemo za poslednji palindrom kome pripada karakter $a[k+1]$). Odavde dobijamo sledeću rekurentnu vezu:

$$d[k+1] = \min\{d[i] \mid i \leqslant k \ , \ a[i+1]a[i+2] \dots a[k+1] \text{ je palindrom}\} + 1$$

Algoritam: Pseudo kod za problem Palindrom

Input: string s koji ispitujemo i njegova dužina n
Output: minimalni broj podstringova na koji se s može podeliti

```
1  d [1] = 1;
2  for k ← 2 to n do
3  |  d [k] = k;
4  |  for i ← k − 1 to 1 do
5  |  |  if (s[i]s[i + 1] · · · s[k − 1] je palindrom) then
6  |  |  |  d[k] = min{d[k], d[i] + 1};
7  |  |  end
8  |  end
9  end
10 return d[n]
```

Direktna implementacija gornje relacije ima složenost $O(n^3)$ - za svaki element niza d proveravamo sve pre njega, pri čemu moramo proveriti i uslov palindroma. Onima koji imaju malo više

iskustva u dinamičkom programiranju, predlažemo da pokušaju da ovaj algoritam optimizuju. Naime, moguće je na početku izračunati koji podstringovi stringa s su palindromi. Ovo se može uraditi definisanjem gornje trougaone matrice *palindrom*, gde je vrednost $palindrom[i, j] = true$ ukoliko je $a[i]a[i+1]\ldots a[j]$ palindrom; $false$ inače. Druga opcija je koristiti heširanje.

Zadatak 4 (Programerski trika PT3) *Problem se sastoji iz dva dela:*

- *Dat je niz a prirodnih brojeva dužine* $2n+1$. *U nizu se nalaze n parova jednakih elemenata (vrednosti se razlikuju medju parovima) i jedan element koji je različit od svih ostalih. Napisati algoritam koji nalazi vrednost koja je različita od svih vrednosti u nizu u jednom prolasku. Primera radi za niz:* $(1, 3, 2, 2, 1)$ *rešenje je* 3.

- *Slučno gornjem problemu samo što je niz dužine* $2n + 2$ *i postoje dve vrednosti koje su različite od svih ostalih. Kao i prvom delu imamo n parova istih elemenata. Napisati algoritam koji u linearnom vremenu nalazi ove dve vrednosti. Primera radi za niz:* $(1, 3, 2, 2, 1, 4)$ *rešenje je* 3 *i* 4.

Rešenje: Za rešavanje oba dela ovog zadatka koristimo operaciju XOR i njene osobine da za svaki prirodni broj a važe jednakosti $a\ XOR\ a = 0$ i $0\ XOR\ a = a$. Kako je XOR i komutativna operacija, onda se prvi deo zadatka svodi na njenu primenu na sve elemente niza. Kako će svaki par jednakih elemenata dati 0, onda će se rezultat svesti na traženi broj. Dakle, traženi element koji nema svog para je

$$solution = a[1]\ XOR\ a[2]\ XOR\ \ldots\ XOR\ a[2 \cdot n + 1].$$

Postupak rešavanja drugog dela problema je nešto složeniji. Označimo tražene brojeve sa A i B. Primenom operacija XOR na sve elemente niza dobijamo broj različit od 0, i to vrednost $s = A\ XOR\ B$, jer su svi ostali elementi upareni sa sebi jednakima dali 0. Posmatrajmo sada bilo koju binarnu poziciju broja s koja je jednaka 1. Ona ima poziciju na kojoj se odgovarajuće bit pozicije brojeva A i B razlikuju po vrednosti. Podelimo sada početni niz a u dva podniza a_0 i a_1, tako da su u a_0 svi oni brojevi iz a koji na posmatranoj bit poziciji imaju 0, a a_1 oni koji imaju 1. Sada, kao i u rešenju prvog dela problema, broj A dobijamo primenom operacije XOR na sve elemente niza a_0, a broj B na elemente niza a_1.