Nešto izmedju
broj 02, godina 2011 / 2012

Zadatak 1 (NI76 Usvajanje zakona) *U jednoj državi ima n regiona. U k−tom regionu živi a[k] političara. Da bi se novi zakon usvojio u k−tom regionu, potrebno je da za njega glasa strogo više od a[k]/2 političara. Da bi se zakon usvojio na nivou čitave države, potrebno je da bude usvojen u strogo više od pola regiona. Koji je najmanji broj političara koji su saglasni sa zakonom potreban da bi se zakon usvojio na nivou države?*

U prvom redu ulaza nalazi se prirodni broj n ($1 \leqslant n \leqslant 100000$*) koji predstavlja broj regiona u državi. Naredni red sadrži n prirodnih brojeva koji predstavljaju broj političara po regionima. Broj političara u jednom regionu neće biti veći od* 1000*. Elementi niza a su parni brojevi.*

U prvom i jedinom redu izlaza štampati traženi broj političara.

Ulaz	*Izlaz*
5 2 4 6 2 4	7

Rešenje: Da bi zakon bio usvojen u državi, potrebno je da se izglasa u $m = \lfloor n/2 \rfloor + 1$ regiona. Kako je potrebno naći najmanji broj saglasnih političara, najoptimalnije je da se zakon usvoji u tačno m regiona sa što manje političara. Zaista, ukoliko bi se zakon usvoji u više od m regiona, mogli bi izbaciti jedan od njih čime bi dobili rešenje sa manjim brojem saglasnih političara. Sa druge strane, za svaki region nam je potrebno $a[k]/2 + 1$ glasova ($a[k]$ su parni brojevi), tako da je optimalnije uzeti region sa manjim brojem političara.

Ovim smo dobili generalnu ideju: sortiramo niz $(a[k])_{k=1}^{n}$ i kao konačno rešenje vratimo vrednost

$$sol = \frac{a[1] + a[2] + \ldots a[m]}{2} + m$$

Ukoliko bi koristili algoritam *quick sort* za sortiranje, dobili bi složenost $O(n \log n)$ koja zadovoljava ograničenja problema. Medjutim, alogortam možemo implementirati i u boljoj složenosti: linearnoj, $O(n)$. Kako je 1.000 ograničenje za elemente niza a, niz možemo sortirati na implicitni način. Definišimo niz $num[k]$, $k \in [1, 1000]$, kao broj elemenata niza a koji imaju vrednost k. Ovaj niz možemo inicijalizovati jednim prolaskom kroz niz. Nakon toga sortirani niz dobijamo kao: $num[1]$ puta broj 1, $num[2]$ puta broj 2, ..., $num[n]$ puta broj n (ovaj algoritam sortiranja je poznat pod nazivom sortiranje prebrojavanjem). Ovde iznosimo implementaciju ovog algoritma koju je poslao učenik Aleksandar Ivanović.

```
#include <cstdio>
#include <algorithm>
#include <cstring>
const int MaxNumber = 1000 + 5;

int N, Number, Cnt[MaxNumber];

int main ( void )
{
    scanf ( "%d", &N );
    memset ( Cnt, 0, sizeof ( Cnt ) );
    for ( int i = 0; i < N; i++ )
    {
```

```
        scanf ( "%d", &Number );
        Cnt[Number]++;
    }

    int Sol = N / 2 + 1, Need = N / 2 + 1;
    for ( int i = 2; ; i += 2 )
    {
        int Have = min ( Cnt[i], Need );
        Sol += Have * i / 2;
        Need -= Have;
        if ( ! Need ) break;
    }

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

Zadatak 2 (NI77 Heširanje) *Dat je niz a različitih prirodnih brojeva dužine n. Jedna mogućnost heširanja brojeva jeste posmatranje nekog podskupa njegovih cifara iz binarnog zapisa. Na primer, ako za heširanje uzmemo prvu, treću i četvrtu binarnu cifru tada od broja* $197 = (11000101)_2$ *dobijamo* 011*. Kako dužine binarnih zapisa brojeva mogu imati različite dužine, pretpostavlja se da svaki od njih ima proizvoljan broj vodećih nula.*

Napisati program koji za dati niz a odredjuje velicinu najmanjeg skupa pozicija, tako da su heširane vrednosti datih brojeva takoddje različite.

U prvom redu ulaza nalazi se prirodni broj n ($1 \leq n \leq 1000$*) koji predstavlja dužinu niza a. Naredni red sadrži n različitih prirodnih brojeva koji predstavljaju elemente niza. Vrednosti elemenata niza su iz segmenta* $[0, 1000]$*.*

U prvom i jedinom redu izlaza štampati traženi broj binarnih pozicija.

Ulaz	*Izlaz*
4 4 7 15 10	2

U navedenom primeru, elemente možemo predstaviti u binarnom zapisu kao niz $(0100, 0111, 1111, 1010)$*. Ukoliko izabremo binarne pozicije* 1 *i* 4*, dobijamo niz:* $(00, 01, 11, 10)$*. Ovo je dobar izbor jer su dobijene sve različine vrednosti. Odabirom samo jedne binarne pozicije se ne dobijaju različite vrednosti. Dakle, rešenje primera je* 2*.*

Rešenje: U ovom problemu se ideja backtrack-a prosto nameće. Naime, nama je potreban podskup cifara binarnog zapisa. Kako je ograničenje za elemente niza 1000, svaki element niza možemo predstaviti sa najviše 10 binarnih cifara. Ovim dobijamo da je broj podskupova koje treba ispitati jednak 2^{10}, tako da možemo priuštiti tu konformnost ispitivanja svakog podskupa posebno.

Postavlja se pitanje kako generisati sve podskupove nekog skupa? Podskup skupa sa n elemenata možemo jedoznačno preslikati u binarni niz dužine n, gde je k-ti elemenet jednak jedinici ako i samo ako k-ti element skupa ulazi u niz (ovde možemo koristiti pojam skupa i pojam niza kao isti jer su elementi različiti). Ovo možemo generisati običnom rekurzivnom funkcijom. Medjutim, postoji jednostavnije rešenje: opisani binarni niz posmatrati kao binarni zapis nekog broja. Na taj način svaki broj $mask \in [0, 2^n - 1]$ jednoznačno definiše podskup: k-ti element ulazi ako i samo ako je k-ta binarna cifra u broju $mask$ jedinica.

Za svaki podksup, heširamo sve vrednosti niza i proveravamo da li se neke od njih poklapaju. Kako heširana vrednost manja od početne vrednosti, za ovo ispitivanje možemo koristiti boolean niza koji nam koristi za ispitivanje da li se neka od vrednost pojavila ili ne. Ukoliko su sve vrednosti različite,

dobili smo potencijalno rešenje i ispitujemo da li je broj elemenata tog podskupa manji od trenutnog najboljeg rešenja. Ukupna složenost ovog algoritma je $O(2^{10} \cdot n)$, gde dodajemo množilac n jer za svaki podskup imamo prolazak kroz niz a kada računamo heš vrednosti.

Izloženi kod predstavlja optimalnu implementaciju sa korišćenjem gore opisane binarne maske. Čitaocu preporučujemo da implementira i rekurzivnu varijantu rešenje (odnosno generaciju podskupova).

```
#include<stdio.h>
#define MAX_N 1001

int n, a [MAX_N], sol;
bool mark [MAX_N];

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

        sol = 10;
        for (int mask = 0; mask < (1 << 10); mask++)
        {
            bool ok = true;
            for (int i = 0; i < n; i++)
                mark [i] = false;

            int currentSol = 0;
            for (int index = 0; index < 10; index++)
                if ((mask & (1 << index)) != 0)
                    currentSol++;

            for (int i = 0; i < n; i++)
            {
                int hes = 0, currentPos = 0;
                for (int index = 0; index < 10; index++)
                    if ((mask & (1 << index)) != 0)
                    {
                        if ((a [i] & (1 << index)) != 0)
                            hes = hes + (1 << currentPos);
                        currentPos++;
                    }
                if (mark [hes])
                    ok = false;
                mark [hes] = true;
            }

            if ((ok) && (sol > currentSol))
                sol = currentSol;
        }

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

Zadatak 3 (NI78 Knjige) *Na stolu se nalazi gomila od n knjiga, poredjanih jedna na drugu. Tri tipa operacija se mogu vršiti nad ovom gomilom:*

- *add a - dodajemo novu knjigu a na vrh gomile*
- *rotate - rotiramo prvih k knjiga sa vrh (prva postaje k-ta, a k-ta postaje prva)*
- *query - upit koji treba vratiti trenutnu knjigu sa vrha*

Broj k iz druge operacije je dat na početku ulaza i biće fiksiran za sve rotacije. Knjige su označene različitim prirodnim brojevima. Napisati program koji za svaki od upita iz ulaza, vraća knjigu sa vrha.

U prvom redu ulaza nalaze se tri prirodna broja n, k i m ($1 \leq k \leq n \leq 1000$, $1 \leq m \leq 100.000$) koji predstavljaju broj knjiga sa gomile na početku, broj knjiga koji se rotira i broj operacija koje treba simulirati, redom. Naredna linija sadrži n prirodnih brojeva koji označavaju početno stanje gomile. Prvi broj predstavlja knjigu sa vrha. Narednih m linija opisuju operacije. Forme operacija su opisane u tekstu problema.

Za svaki upit iz ulaza, štampati knjigu sa vrha gomile (u redosledu operacija iz ulaza).

Ulaz	*Izlaz*
`3 2 4`	`1`
`1 2 3`	`1`
`query`	
`add 4`	
`rotate`	
`query`	

Rešenje: Iako na prvi pogled deluje da nam je potrebna neka složenija struktura podataka, uspostavlja se da problem možemo rešiti jednostavno uz pomoć dvostruko povezane liste. Ovu strukturu ovde nećemo opisivati, ali objašnjenje iste možete potražiti u svakoj knjizi o osnovama programiranja.

Prva stvar koju treba primetiti jeste da nam je u svakom koraku potrebno samo prvih k knjiga, jer sve knjige nakon k-te nikada neće uticati na traženi upit (rotacija ne utiče na njih). Dakle, u svakom koraku ćemo čuvati samo prvih k knjiga. Knjige ćemo ubacivati u dvostruko povezanu listu. U svakom trenutku ćemo u promenjivoj *direction* čuvati orijentaciju naše liste, odnosno šta je vrh naše gomile a šta dno. Ovim se pri upitu štampa prva knjiga u odnosu na smer *direction*. Ako je $direction = 1$, to znači da je prva knjiga sa gomile zapravo poslednji element liste, a ako je $direction = -1$ onda je prva knjiga sa gomile prvi element naše liste. Operacija rotacije se na ovan način implemenetira kao: $direction = -direction$ tj. samo menjamo smer liste. Dodavanje knjige zahteva dva koraka: dodvanje knjige na vrh liste (tj. novog elemenata u listi na njen vrh) i izbacivanje poslednje knjige (tj. izbacivanje poslednje elementa iz liste). Kao i pre, prvi odnosno poslednji element liste odredjujemo na ostavu smera.

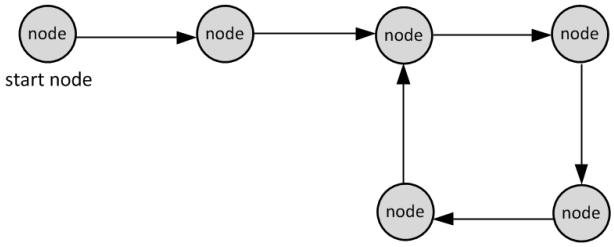
Složenost ovog algoritma je $O(m)$, jer svaku komandu i upit izvršavamo u konstantnom vremenu. Memorijska složenost je $O(n)$.

Zadatak 4 (Programerski trika PT4) *Data je jednostruko povezana listam, čiji je element predsavljen parom:*

- *data - podatak za taj čvor (možete pretpostaviti da je ovo tipa double)*
- *next - pokazivač na naredni element liste*

Ispitati da li data lista sadrži ciklus? Lista je data preko pokazivača na njen koren. Rešenje treba da je linearne složenosti u odnosu na broj elemenata liste.

Rešenje: Programerski trik iz ovog broja je jako interesantan. U literaturi je poznat pod nazivom Floidov algoritam za nalaženje ciklusa. Iako jednostavnog teksta, rešenje nije uopšte očigledno. Pre svega vidimo da rešenje ne treba da zavisi od tipa podatka koji se čuvaju u čvorovima (ili samih adresa), tako da pristupi sa memorizacijom ili heširanjem padaju u vodu.



Ideja je sledeća: u svakom trenutku ćemo pamtiti dva pokazivača p_1 i p_2, pri čemu će se pokazivač p_2 kretati duplo brže od pokazivača p_1. Konkretnije u svakoj iteraciji imamo sledeće promene:

$$p_1 = p_1 \to next \qquad p_2 = (p_2 \to next) \to next$$

Na početku oba pokazivača pokazuju na početak liste. Ukoliko lista sadrži ciklus, tada će se ova dva pokazivača poklopiti: p_2 će "prestići" pokazivač p_1 za ciklus. U zavisnosti da li je ciklus parne ili neparne dužine, može se pokazati da je potrebno najviše tri obilaska ciklusa od strane pokazivača p_2 da se susretnu.

Za ovaj problem postoji još jedno "rešenje" koje ispituje postojanje ciklusa ali na kraju ne ostavlja listu u početnom stanju. Naime, pri kretanju kroz listu, kada prelazimo sa čvora u na čvor $v = u \to next$ obrćemo ovu ivicu. Na taj način, ukoliko postoji ciklus u listi mi ćemo se vratiti na početni čvor liste. Ovim je nadjen ciklus, medjutim ne postoji način da se lista vrati u prvobitno stanje.