Rubrika: Nešto izmedju [rešenja problema]

časopis Tangenta, broj 02, godina 2010 / 2011

Problem: NI67 Brojevi

Zadatak se ispostavio malo teži od očekivanog. Naime većina učenika je rešila problem kvadratnim algoritmom: povećavamo broj n sve dok ne dobijemo broj sastavljen od datih cifara. Zadatak se može rešiti mnogo jednostavnije ukoliko simuliramo slično rezonovanje kada dati decimalni broj povećavamo za 1. U tom slučaju ukoliko je cifra jedinice različita od 9 uvećaćemo je za 1 i to će biti rešenje. U suprotnom cifru jedinice postavljamo na 0 i analogno razmatranje prenosimo na cifru desetice. Jedini specijalni slučaj je kada imamo broj oblika $99\ldots99$ gde je krajnje rešenje $100\ldots00$.

U našem slučaju jedino što treba promeniti je da kada cifru hoćemo da uvećamo za 1, ustvari predjemo na prvu sledeću veću cifru koju smemo da koristimo. Takodje, umesto da neku cifru prebacimo na vrednost 0 mi je postavljamo na najmanju moguću datu cifru.

Problem: NI68 Permutacija

Razmotrimo prvo slučaj kada nemamo fiksiranih elemenata permutacije. Posmatraćemo pozicije jedinica počev od najmanje. Označimo ovu poziciju sa p_1. Kako prvih p_1 elemenata čine permutaciju, a ni jedan drugi manji prefiks ne, najlakši primer koji ovo zadovoljava bi bio upravo obrunuta permutacija: $\{p_1, p_1 - 1, \ldots, 2, 1\}$. Sada posmatramo sledeću jedinicu na poziciji p_2. Deo izmedju indeksa p_1 i p_2 popunjavamo slično samo počev od $p_1 + 1$ čime dobijamo da niz ima oblika: $\{p_1, \ldots, 2, 1, p_2, p_2 - 1, \ldots, p_1 + 1\}$. Sličan postupak nastavljamo sve do kraja. Ovde je naravno uslov da $d[n]$ mora biti jednako 1. Primera radi za niz $d = \{0, 0, 0, 1, 0, 0, 1, 0, 1\}$, gore opisanim algoritmom bi dobili $\{4, 3, 2, 1, 7, 6, 5, 9, 8\}$.

Kako rešavamo slučaj kada imamo fiksirane elemente? Gornji pristup možemo posmatrati i malo drugačije: kada popunjavamo deo izmedju indeksa p_k i p_{k+1} redjaćemo redom elemente počev od p_{k+1} pa sve do $p_k + 1$ - ukoliko on nije negde već fiksiran gde naravno preskačemo popunjene elemente. Ovim algoritmom sigurno dobijamo rešenje ukoliko ono postoji. U slučaju da u nekom koraku dobijemo protivurečnost, ispisaćemo da rešenje ne postoji.

Problem: NI69 Malo igranja

Ovaj problem predstavlja standardni problem iz teorije igara koji rešavamo dinamičkim programiranjem. Posmatrajmo prvo malo uopšteniji problem. Označimo trenutnu konfiguraciju igre za dva igrača sa G. Igrač koji je na potezu može iz ove konfiguracije preći u konfiguracije: $G_1, \ldots, G_k$. Nakon odigranog poteza igrači menjaju poredak: onaj koji je bio prvi na potezu sada je drugi, a onaj koji je povlačio drugi potez sada igra prvi. Kada možemo reći da igrač koji je na potezu ima pobedničku strategiju za poziciju G? Kako bi on imao pobedničku strategiju on mora preći u konfiguraciju u koju drugi igrač ima pobedničku strategiju - iz razloga što tada on postaje drugi igrač. Dakle sada smo sveli problem ispitavanja konfiguracije G na konfiguracije $G_1, \ldots, G_k$.

Vratimo se sada našem problemu. Označimo početnu matricu sa a, sa $a[i, j]$ elemente matrice a sa $A[i, j]$ podmatricu sa temenima $([1, 1], [1, j], [i, j], [i, 1])$. Kako se potez sastoji od uklanjanja poslednje vrste ili kolone, moguće konfiguracije su upravo matrice $A[i, j]$. Dakle iz konfiguracije $A[i, j]$ igrač koji je na potezu može preću u konfiguraciju $A[i - 1, j]$ (ako je $i > 1$ i ako je zbir u vrsti paran) i

$A[i, j-1]$ (ako je $j > 1$ i ako je zbir u koloni paran). Iz gore opisanog pristupa dobijamo da prvi igrač ima pobedničku startegiju za konfiguraciju $A[i, j]$ akko drugi igrač ima pobedničku strategiju za konfiguraciju $A[i-1, j]$ ili $A[i, j-1]$ (naravno ukoliko su one moguće).

Definišimo matricu $d[i, j]$ kao 1 ukoliko prvi igrač ima pobedničku strategiju za konfiguraciju $A[i, j]$, odnosno 2 ako drugi igrač pobedjuje. Sada možemo definisati rekurentnu vezu izmedju elemenata matrice d kao:

$$d[i,j] = \begin{cases} 2, & \text{ako je } i \cdot j = 0 \text{ jer prvi nema potez} \\ 1, & \text{ako je } d[i-1,j] = 2 \text{ i zbir u poslednjoj vrsti je paran} \\ & \text{ili } d[i,j-1] = 2 \text{ i zbir u poslednjoj koloni je paran} \\ 2, & \text{inače} \end{cases}$$

Ostaje još da razmotrimo kada je potez koji vodi iz konfiguracije $A[i, j]$ u $A[i-1, j]$ moguć (analogno i za $A[i, j-1]$). Kako ne bi stalno računali sumu brojeva $a[i, 1], \ldots, a[i, j]$ definisaćemo matricu *sumaLevo* baš na taj način:

$$sumaLevo[i,j] = a[i,1] + \ldots + a[i,j]$$

koju možemo popuniti preko

$$sumaLevo[i,j] = \begin{cases} a[i,1], & \text{ako je } j = 1 \\ sumaLevo[i,j-1] + a[i,j], & \text{inače} \end{cases}$$

Analogno definišemo i matricu *sumaGore* za ispitivanje kolona. Elemente ovih matrica možemo pamtiti po modulu 2, pošto je jedino parnost bitna. Složenost ovog algoritma je $O(n \cdot m)$, jer nakon inicijalizacije matrica *sumaLevo* i *sumaGore* računanje elementa matrice d je konstantno. Vrednost elementa $d[n, m]$ predstavlja krajnje rešenje.

Kod ovog algoritma nećemo izneti ovde već ostavljamo čitaocu za vežbu. Članak o Dinamičkom programiranju iz ovog broja će vam pomoći u implementaciji.

Programerski trikovi

Ovaj prelep problem poznatiji je kao *"A Linear Time Majority Vote Algorithm"*. Postoji više linearnih rešenja, a ovde ćemo izložiti dva:

- Problem možemo rešiti samo uz pomoć dve dodatne promenjive i jednim prolaskom kroz niz. Pretpostavimo da je prvi element niza traženi broj i postavimo frekvencu na 1. Označimo sa *sol* trenutno rešenje a sa *num* broj pojavljivanja. Kada predjemo na sledeći element niza, ukoliko je on jednak našem trenutnom rešenju povećavamo *num*. U suprotnom, smanjujemo broj pojavljivanja i ako dobijamo da je $num = 0$, proglašavamo trenutni element za rešenje.

```
public int mostCommon(int[] a)
{
    int sol = 0, num = -1;
    foreach (int number in a)
    {
        if (number == sol)
            num++;
        else
            num--;

        if (num <= 0)
```

```
                sol = number;
        }
        return sol;
    }
```

- Drugi pristup je preko bitova. Definisaćemo niz *bit* kao:

$$bit[k] = \text{broj elemenata niza koji imaju 1 na bit poziciji } k$$

  Nakon inicijalizacije imamo da one *bit* pozicije sa vrednostima većim od $n/2$ čine traženi broj. Važno je uočiti da bilo koji bit koji ne učestvuje u traženom broju ima frekvencu manju ili jednaku od $n/2$ u *bit* nizu.