$\pi\mathrm{l}\alpha y\sqrt{\mathbf{mat}\chi}$

Dinamičko programiranje 2

Ivo Sluganović

U 7. broju *PlayMath*-a mogli ste pročitati članak[1] o dinamičkom programiranju. Budući da je ova metoda rješavanja iznimno složena i primjenjiva na velik broj zadataka, u prošlom sam broju uspio objasniti samo *najjednostavnije* primjere, a nisam spomenuo jedan, možda i jednostavniji, pristup rješavanju problema koji imaju svojstva potrebna da bi ih mogli rješavati uz pomoć dinamičkog programiranja. Ideja je da zadatak rješavamo **rekurzijom**, ali da na neki način izbjegnemo bespotrebno računanje istih potproblema više puta. To ćemo učiniti tako što ćemo, kada jednom izračunamo rješenje nekog potproblema, to rješenje zapamtiti i kada nam sljedeći puta zatreba, jednostavno ga pročitati s mjesta gdje smo ga zapamtili. Ovakav način rješavanja naziva se **memoizacija**, a njime uz vrlo male promjene u rekurzivnom rješenju složenost algoritma smanjujemo sa *eksponencijalne* na *polinomijalnu*. Pogledajmo zadatak koji smo već riješili u prošlom članku:

Primjer 1. Za zadane brojeve N i X ($N \leq 10000$ i $X \leq 100$), izračunajte ostatak cjelobrojnog dijeljenja N-tog Fibonaccijevog broja brojem X.

Rješenje. Neka je D polje u koje ćemo pamtiti vrijednosti koje smo već izračunali. Na početku na svako mjesto u polju upišimo broj -1 (budući da ni jedan Fibonaccijev broj nije jednak -1, na taj ćemo način točno znati jesmo li već izračunali vrijednost za neki Fibonaccijev broj). Sada još jedino trebamo dodati provjeru u kojoj gledamo jesmo li već izračunali neki Fibonaccijev broj i jednom izračunati broj zapisati u polje kako ga ne bismo morali ponovo računati.

funkcija $F(N)$
početak
  ako *(*$D[N] \neq -1$*)*
    vrati $D[N]$;
  ako *(*$N = 1$ *ili* $N = 2$*)*
    $D[N] = 1$;
  u suprotnom
    $D[N] = (F(N-1) + F(N-2)) \bmod X$;
  vrati $D[N]$;
kraj

✓

Ako promotrimo vremensku složenost ovog programa, vidimo da će se svaki Fibonaccijev broj računati samo jednom, tako da je složenost $O(N)$, što je veliko poboljšanje u odnosu na $O(N^{1.618})$, kolika je približna složenost obične rekurzije.

Kada riješimo neki zadatak rekurzijom, vrlo je lako pretvoriti rekurzivno rješenje u memoizaciju. To radimo koristeći jedno dodatno polje u koje zapisujemo jednom izračunate vrijednosti. Budući da je često lakše smisliti rekurzivno rješenje, najjednostavnije je prvo smisliti rješenje u obliku rekurzije, a onda ga kasnije *pretvoriti* u memoizaciju.

Za vježbu, pokušajte memoizacijom riješiti primjere i zadatke iz prošlog članka.

Krenimo s malo složenijim primjerima:

Primjer 2. Zadan je niz brojeva A veličine N i broj K. Figurica kreće sa bilo kojeg polja i može skočiti najviše K polja ulijevo ili udesno. Sa nekog broja figurica smije skočiti samo na strogo veći broj. Potrebno je ispisati koliki je najveći mogući zbroj brojeva koje figurica može posjetiti. ($N \leq 10000, K \leq 100$)

[1] Vidi: Dinamičko programiranje, *PlayMath* br. 7 (2006.)

$\pi \mathrm{l} \alpha y \sqrt[\chi]{\mathbf{mat}}$

Rješenje. Zadatak ćemo prvo riješiti memoizacijom, a zatim dinamičkim programiranjem. Kako bi izgledala rekurzija? To bila funkcija koja bi za neko polje vraćala zbroj brojeva koji možemo postići ako krenemo od tog polja. Unutar rekurzije pokušavali bismo svih $2K$ skokova (K ulijevo i K udesno) i među onima kojima skačemo na veći broj odabrali onaj sa najvećim zbrojem. Potrebno je još dodati memoizaciju. Polje D na početku ćemo ispuniti sa -1. Kada izračunamo koliki je najveći zbroj koji možemo postići sa nekog polja, zapisat ćemo ga na odgovarajuće mjesto u polju kako ga ne bismo ponovo morali računati.

funkcija $REK(X)$
početak
  ako $(D[X] \neq -1)$
    vrati $D[X]$;
  $D[X] = A[X]$;
  za $I = 1$ *do* K **radi**
    ako $(X + K \leq N$ *&* $A[X] < A[X + K]$ *&* $REK(N + K) + A[X] > D[X])$
      $D[X] = REK(N + K) + A[X]$;
    ako $(X > K$ *&* $A[X] < A[X - K]$ *&* $REK(N - K) + A[X] > D[X])$
      $D[X] = REK(N - K) + A[X]$;
  vrati $D[N]$;
kraj

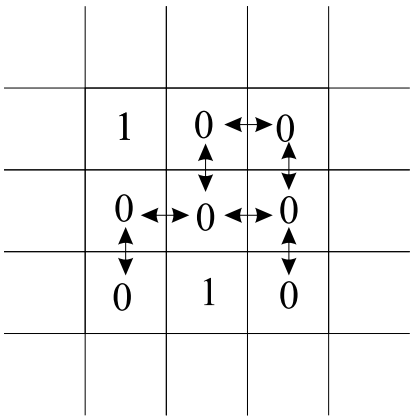
✓

Dinamičko rješenje nije toliko jednostavno jer nije odmah očito kojim redoslijednom rješavati potprobleme.

Da bismo ovaj zadatak riješili dinamičkim programiranjem, trebamo se sjetiti da je niz brojeva po kojima skačemo poredan uzlazno, tj. da je svaki broj na koji skočimo veći od svih prethodnih koje smo posjetili. Zato trebamo poredati brojeve uzlazno i tim redoslijedom ih rješavati. Napravimo novo polje P u koje na početku upišemo prvih N prirodnih brojeva. Poredajmo brojeve tako u usporedbi neka dva elementa niza P, $P[i]$ i $P[j]$, uspoređujemo $A[P[i]]$ sa $A[P[j]]$. Sada prolazimo po polju P i tim redoslijedom rješavamo potprobleme.

Ovo je primjer zadatka u kojem je jednostavnije rješavati memoizacijom nego dinamičkim programiranjem jer nije odmah očito kojim redoslijedom treba rješavati potprobleme. Kada pokušavamo neki zadatak riješiti memoizacijom, moramo paziti da se ne dogodi da je za rješenje nekog potproblema opet potrebno to isto rješenje, tj. da se ne zavrtimo u krug.

Primjer 3. Zadana je matrica veličine $N \times M$. U matrici su zapisani brojevi 0 i 1. Ako je dopušteno kretati se samo po brojevima 0: pomicati se u 4 glavna smjera, potrebno je izračunati koliko je najmanje koraka potrebno da bismo sa polja $(1, 1)$ došli do polja (N, M).



(Krivo) rješenje. Vrlo bismo lako mogli pomisliti da je ovaj zadatak rješiv dinamičkim programiranjem: Neka $rek(X, Y)$ prima koordinate polja na kojem se trenutno nalazimo i vraća koliko je koraka potrebno da bismo sa tog polja došli do polja (N, M). Sada je:

$$rek(X, Y) = \min\{rek(X-1, Y), rek(X, Y-1), rek(X+1, Y), rek(X, Y+1)\} + 1.$$

Treba uzeti u obzir da se ne možemo kretati po poljima na koja se nalazi broj jedan, pa ćemo za takva polja vraćati neki vrlo veliki broj kojima neće imati utjecaja na krajnji rezultat, a $rek(N, M) = 0$. *Zašto to nije točno rješenje?*

Pogledajmo što se događa kada na taj način računamo $rek(X, Y)$. Neka je prvi poziv rekurzije potreban da bismo izračunali $rek(X, Y)$ poziv $rek(X-1, Y)$. Da bismo izračunali $rek(X-1, Y)$, potrebno je 4 puta pozvati rekuziju:

$$rek(X-1-1, Y),\ rek(X-1, Y-1),\ rek(X-1+1, Y) \text{ i } rek(X-1+1, Y+1).$$

Problem se krije u trećem pozivu rekurzije. Da bismo izračunali $rek(X, Y)$, potrebno je izačunati $rek(X-1, Y)$, ali da bismo izračunali $rek(X-1, Y)$, potrebno je, između ostaloga, izračunati i $rek(X-1+1, Y)$, tj. $rek(X, Y)$, te tako dolazi do *ciklusa*. Zbog toga ovaj zadatak nije moguće riješiti dinamičkim programiranjem ni memoizacijom (ovakav tip zadatka rješava se metodom koja se naziva BFS – *Breadth First Search*[2], ali ona će biti obrađena u nekom drugom članku). ✓

Pogledajmo jedan primjer u kojem nije tako jasno na koji način možemo pamtiti neko stanje:

Primjer 4. Za niz znakova koji se jednako čita s obje strane kažemo da je palindrom. Neka je cijena nekog niza znakova najmanji broj dijelova na koji ga moramo rastaviti da bi svaki tako dobiveni dio bio palindrom. Potrebno je ispisati cijenu zadanog niza znakova.

Rješenje. Kada bi zadatak rješavali rekurzivno, rekurzija bi primala niz znakova i vraćala njegovu cijenu. Ako je niz znakova palindrom, cijena je jedan, inače ga pokušamo na sve moguće načine rastaviti na dva niza znakova i od svih rastava odabrati onaj u kojem je zbroj cijena dvaju nizova znakova na koje smo rastavili početni niz minimalna. To je tražena cijena početnog niza znakova.

Vremenska složenost ovakvog rješenja je eksponencijalna, ali moguće je njenu složenost smanjiti na polinomijalnu. Neka npr. računamo cijenu niza znakova $ABACA$. Niz znakova $ABACA$ pokušat ćemo rastaviti na sljedeće nizove znakova:

$$A, BACA \qquad AB, ACA \qquad ABA, CA \qquad ABAC, A.$$

Sada, i kada računamo cijenu niza znakova $BACA$, i kada računamo cijenu niza znakova ACA, računat ćemo cijenu niza znakova CA. Vidimo da smo isti potproblem računali više puta, tj. da možemo rješenje potproblema nekako zapamtiti i na taj način smanjiti vremensku složenost. Sljedeći je problem kako zapamtiti jednom izračunatu vrijednost jer indeks nekog polja ne može biti niz znakova. Potrebno je sjetiti se da npr. niz znakova BA možemo opisati kao podniz niza $ABACA$ od 2. do 3. znaka (uključujući). Sada svaki niz znakova za koji tražimo cijenu (a to će biti samo oni koji su podnizovi početnog niza znakova) možemo opisati sa 2 broja i tako možemo u dvodimenzionalno polje zapisati jednom izračunatu cijenu. Budući da ni jedna cijena ne može biti -1, možemo u dvodimenzionalno polje D na početku svuda upisati -1, što označava da nismo do sada riješili promatrani potproblem. Funkcija palindrom prima dva prirodna broja koji određuju neki podniz znakova i vraća je li taj podniz palindrom.

[2]Vidi *Filip Nikšić:* Traženje najkraćeg puta, *PlayMath* br. 3(2003.) str. 7 i ispravak algoritma u *PlayMath*-u br. 4(2004.) na str. 52

$\pi\mathrm{l}\alpha y\sqrt{\mathbf{mat}\chi}$

```
funkcija REK(A, B)
početak
    ako (D[A][B] ≠ −1)
        vrati D[A][B];
    ako ( PALINDROM(A, B) )
        D[A][B] = 1;
        vrati D[A][B];
    D[A][B] = REK(A, A) + REK(A + 1, B);
    za I = A + 1 do B − 1 radi
        ako ( REK(A, I) + REK(I + 1, B) < D[A][B])
            D[A][B] = REK(A, I) + REK(I + 1, B);
    vrati D[A][B];
kraj
```

✓

Osvrt

U ovom članku obrađeni su neki malo kompliciraniji primjeri ili su istaknute neke pogreške koje možemo napraviti kada zadatke rješavamo memoizacijom ili dinamičkim programiranjem. Preporučujem vam da pročitate i članak *Kombinatorni algoritmi* u ovom broju *PlayMath*-a jer se bavi vrlo sličnim problemima.

Zadatci

1. Zadana su dva niza znakova, A i B, duljine do 1000 znakova. Podniz nekog niza znakova je niz znakova koji se dobije izbacivanjem nekog broja (nula ili više) znakova iz početnog niza. Potrebno je ispisati duljinu najdužeg zajedničkog podniza znakova, tj. onog koji je podniz oba niza znakova.
   ULAZ: fibonacci, banana
   IZLAZ: 3

2. Zadan je niz prirodnih brojeva veličine N. Na koliko se načina može dobiti zadani broj X kao zbroj nekog broja brojeva iz zadanog niza? ($X \leq 1000$, $N \leq 1000$)
   ULAZ: 1 3 4 5 6, 10
   IZLAZ: 15

3. Zadan je niz sa manje od 1000 prirodnih brojeva. Potrebno je pronaći duljinu najdužeg podniza u kojem je svaki element djeljiv njegovim prethodnim (osim naravno prvog).
   ULAZ: 2 8 6 216 36 3
   IZLAZ: 4

4. Zadan je niz sa manje od 1000 prirodnih brojeva. Potrebno je pronaći duljinu najdužeg uzastopnog podniza u kojem je svaki element strogo manji od svog prethodnika.
   ULAZ: 10 4 7 6 2 3 3
   IZLAZ: 4

5. Na stolu se na početku nalazi $K \leq 1000$ kamenčića. U svakom potezu igrač može sa stola uzeti X kamenčića, pod uvjetom da se na stolu nalazi barem X kamenčića i da je X iz zadanog skupa prirodnih brojeva S. Pobjednik je onaj igrač koji zadnji sa stola uzme neki broj kamenčića. Za zadani skup S i prirodni broj K odredi pobjednika.
   ULAZ: 10, 1 4
   IZLAZ: Pobjednik je drugi igrač.