Dinamičko programiranje

Zadaci

Princip indukcije koristi se da se ulaz za problem svede na jedan ili više manjih. Ako se svođenje može uvek izvesti, a bazni slučaj se može rešiti, onda je algoritam definisan indukcijom, odnosno dobijen je rekurzivni algoritam. ***Osnovna ideja je da se pažnja skoncentriše na smanjivanje problema, a ne na njegovo direktno rešavanje****.* Miodrag Živković *ALGORITMI*

1. Zadato je $n + 1$ prirodnih brojeva $a_1, a_2, \ldots, a_n$ i K. Naći algoritam za ispitivanje da li postoji podniz od p indeksa (1<=p<=n) $1 <= i_1 < i_2 < \ldots < i_p <= n$ takvih da je $a_{i1} + a_{i2} + \ldots + a_{ip} = K$. Vremenska, odnosno prostorna složenost algoritma treba da budu redom $O(nK)$, odnosno $O(K)$.

Ideja algoritma je da se redom za svaki podskup
$\{a[1], a[2], \ldots, a[i]\}$ $(1 <= i <=\cdot n)$ skupa $\{a[1], a[2], \ldots, a[n]\}$ odredi da li u njemu postoji podskup takav da je zbir vrednosti njegovih elemenata jednak k (za sve $k \in \{0, \ldots, K\}$).
Te informacije čuvamo u vektoru P od $K + 1$ elemenata (P[0..K]).
Koristimo činjenicu da u skupu $\{a[1], a[2], \ldots, a[i]\}$ postoji podskup takav da je zbir vrednosti njegovih elemenata jednak k akko u skupu $\{a[1], a[2], \ldots, a[i-1]\}$ postoji podskup takav da je zbir vrednosti njegovih elemenata jednak k ili $k - a[i]$.
Dakle, pojačana induktivna hipoteza je da umemo da rešimo problem ne samo za sumu elemenata koja je jednaka K, već i za elemente čija je suma manja od K.

Zato u i-tom prolazu kroz petlju čuvamo podatke o skupu $\{a[1], a[2], \ldots, a[i-1]\}$ u vektoru Prev od $K + 1$ elemenata (Prev[0..K]).

Algoritam daje odgovor ”da” (postavlja promenjivu Postoji na true) ako nakon izlaska iz petlje P[K] ima vrednost true, tj. u skupu $\{a[1], a[2], \ldots, a[n]\}$ postoji podskup takav da je zbir vrednosti njegovih elemenata K. Inace daje odgovor ”ne” (postavlja promenljivu Postoji na false).

```
Algoritam  Ranac;  //C-like pseudo kôd
Ulaz: {a[1],...,a[n]}, K
Izlaz: Postoji (bool)
main ()
{
Prev[0] = true;
for (k=1; k<= K; k++) Prev[k] = false;
for (i=1; i<=n; i++)
{
  for (k=0; k<=K; k++)
   {
     P[k] = false;
      if (Prev[k] == true) P[k] = true;
       else
          if (k>=a[i])
          if ((Prev[k-a[i]]==true) { P[k] = true;  Prev[k] == true;}
  }
   // for (j=1; j<=K; j++) Prev[j] = P[j];
}
```

if (P[K] == true) Postoji = true;
else Postoji = false;

}
Primer tabele formirane pri rešavanju problema ranca.
Ulaz su $n = 4$ predmeta veličina 8*;* 5*;* 4 i 3, a veličina ranca je $K = 11$.
Simbol "*I*" označava da postoji rešenje koje obuhvata predmet iz odgovarajuće vrste; simbol "*O*" označava da postoji rešenje, ali samo bez tog predmeta; simbol "-" označava da ne postoji rešenje.

	0	1	2	3	4	5	6	7	8	9	10	11
$k_1 = 8$	O	-	-	-	-	-	-	-	I	-	-	-
$k_2 = 5$	O	-	-	-	-	I	-	-	O	-	-	-
$k_3 = 4$	O	-	-	-	I	O	-	-	O	I	-	-
$k_4 = 3$	O	-	-	I	O	O	-	I	I	O	-	I

Prev[0]=true =>P[0]=true
i=1: a[1]=8 => Prev[0]=true => P[8]=true => Prev[8]=true;
i=2: a[2]=5 => Prev[0]=true => P[5]=true => Prev[5]=true;
i=3: a[3]=4 => Prev[0]=true => P[4]=true => Prev[4]=true; => Prev[9]=true => P[9]=true
i=4: a[4]=3 => Prev[0]=true => P[3]=true => Prev[3]=true; =>Prev[7]=true; => Prev[11]=true => P[11]=true

U algoritmu se izvršava:
jedna operacija provere uslova, dve operacije dodele,

jedna *K*-tostruka petlja sa jednom operacijom dodele,

jedna *nK*-tostruka slozena petlja (tj. dvostruka petlja po *n* i po *K*) sa najviše
tri operacije dodele i dve operacije provere uslova u svakom prolazu kroz unutrasnju petlju, i jednom $K+1$-strukom petljom sa jednom operacijom dodele pri svakom prolazu kroz spoljasnju petlju.

Dakle, vremenska slozenost algoritma je:
$T(n) \cdot K + c1 + n(K + c2(K + 1)) = O(nK)$

Prostorna složenost algoritma je $O(K)$, jer koristi dva pomoćna vektora od $K + 1$ elemenata (P i Prev) i lokalne promenljive *i* i *k*.

2. ***Isti zadatak, ali drukčije tabelarno rešenje koje održava 0/1 prirodu formulacije problema***
Dat je prirodan broj `K` i `n` predmeta različitih veličina, tako da `i`-ti predmet ima veličinu a[i], `1 ≤ i ≤ n`.
Pronaći podskup predmeta čija je suma veličina jednaka tačno `K` ili utvrditi da takav podskup ne postoji.

Baza indukcije:

```
P(1, k) - za k = 0 - uvek postoji rešenje od nula sabiraka
        - za k ≠ 0 - resenje postoji ako je k = k₁
```

Induktivna hipoteza - umemo da rešimo P(n - 1, k) za sve k iz { 0, ..., K }

Problem P(n, k) se svodi na P(n - 1, k) i P(n - 1, k - a[i] koje znamo da rešimo po induktivnoj hipotezi.

Ovo nas vodi složenosti $O(2^n)$. Međutim, ukupan broj različitih problema je nK.

```
// Ulaz: a - niz koji sadrži veličine predmeta, dužine n
//       K - veličina ranca
// Izlaz: P - matrica takva da je P[i,k].postoji == true
//       akko postoji rešenje problema ranca sa prvih i
//       predmeta za veličinu ranca k;
//       P[i,k].pripada = true akko i-ti predmet pripada
//       tom rešenju
ranac(a, K) {
    // ranac veličine 0 se uvek može popuniti
    P[0, 0].postoji = true;

    // Ako imamo 0 predmeta, ne možemo popuniti ranac veličine K ≠ 0
    for (k: {1, … , K})   // isto kao for (k=1;k<=K;k=k+1)
        P[0, k].postoji = false;

    for (i: {1, …, n}) {
        for (k: {0, …, K}) {
            P[i, k].postoji = false;

            if (P[i - 1, k].postoji) {
                // ako postoji rešenje bez i-tog predmeta
                P[i, k].postoji = true;
                P[i, k].pripada = false;

            } else if (k - a[i] ≥ 0) {
                if (P[i - 1, k - a[i]].postoji) {
                    // Ako postoji rešenje sa i-tim predmetom
                    P[i, k].postoji = true;
                    P[i, k].pripada = true;
                }
            }
        }
    }
}
```

Vremenska i prostorna složenost algoritma su O(nK). Složenost rekonstrukcije rešenja je O(n).

3. Demonstrirati proces rešavanja problema ranca za veličinu ranca K = 12 i veličine predmeta a1= 9, a2 = 7, a3 = 4, a4 = 10, a5 = 3.

Matrica Postoji/Pripada:

	0	1	2	3	4	5	6	7	8	9	10	11	12
	1												
a1 = 9	1/0									1/1			
a2 = 7	1/0							1/1		1/0			
a3 = 4	1/0				1/1			1/0		1/0		1/1	
a4 = 10	1/0				1/0			1/0		1/0	1/1	1/0	
a5 = 3	1/0			1/1	1/0			1/0		1/0	1/0	1/0	1/1

Zaključak: postoji popunjavanje ranca predmetima a1 i a5.

4. ***Varijanta problema ranca*** - dati su brojevi a_1, a_2 , …, a_n i K, potrebno je predstaviti K u obliku zbira nekih brojeva a_i, i = 1, …, n pri čemu se sabirci mogu ponavljati proizvoljan broj puta.

```
ranacSaPonavljanjem(a, K) {
    // ranac veličine 0 se uvek može popuniti
    P[0, 0].postoji = true;

    // Ako imamo 0 predmeta, ne možemo popuniti ranac veličine K ≠ 0
    for (k: {1, … , K})
        P[0, k].postoji = false;

    for (i: {1, …, n}) {
        for (k: {0, …, K}) {
            P[i, k].postoji = false;

            if (P[i - 1, k].postoji) {
                // ako postoji rešenje bez i-tog predmeta
                P[i, k].postoji = true;
                P[i, k].pripada = false;

            } else if (k - a[i] ≥ 0) {
                if (P[i, k - a[i]].postoji) {   // << razlika
                    // Ako postoji rešenje sa i-tim predmetom
                    P[i, k].postoji = true;
                    P[i, k].pripada = true;
                }
            }
        }
    }
}
```

5. Demonstrirati proces rešavanja problema ranca za veličinu ranca K = 12 i veličine predmeta a1= 9, a2 = 7, a3 = 4, a4 = 10.

	0	1	2	3	4	5	6	7	8	9	10	11	12
	1												
a1 = 9	1/0									1/1			
a2 = 7	1/0							1/1		1/0			
a3 = 4	1/0				1/1			1/0	1/1	1/0		1/1	1/1
a4 = 10	1/0				1/0			1/0	1/0	1/0	1/1	1/0	1/0

matrica Postoji/Pripada:

Postoji popunjavanje pomoću tri predmeta a3.

6. ***Varijanta problema ranca -*** Dati su brojevi S1, S2, …, Sn i K. **ALI** je sada predmetima tipa i pridružena vrednost v_i, i = 1, …, n. Konstruisati algoritam koji ranac puni do vrha i to predmetima sa najvećim mogućim zbirom vrednosti, tj. zadatak je odrediti nenegativne cele brojeve a1, a2, …, a_n takve da je:

$$\sum_{i=1}^{n} a_i S_i = K$$

i da suma $\Sigma a_i v_i$ ima maksimalnu vrednost.

Uputstvo:
Za svaki par (i, k) gde je $0 \le i \le n$ i $0 \le k \le K$,
neka je C[i, k]najveća moguća vrednost ranca sa sadržajem veličine (mase) k, ako se u njemu mogu naći samo prvih i predmeta i
neka je I(i, k) broj predmeta tipa i u tom maksimalno vrednom sadržaju ranca veličine k.

Podskup prvih i elemenata sa sumom veličine k takav da ima najveću vrednost može se dobiti na dva načina:

1. podskup ne sadrži i-ti element:

   tada važi $C[i, k] = C[i - 1, k]$ i važi $I[i, k] = 0$

2. podskup sadrži i-ti element:

   tada važi: $C[i, k] = C[i, k - S_i] + v_i$ i važi $I[i, k] = I[i, k - S_i] + 1$.

Prema tome, $C[i, k] = \max(C[i - 1, k], C[i, k - S_i] + v_i)$

Na ovaj način se pomoću O(nK) operacija izračunavaju matrice C i I.

```
    // Ulaz:
    //       S - vektor dužine n sa veličinama predmeta
    //       v - vektor dužine n sa vrednostima predmeta
    //       K - veličina ranca
    // Izlaz:
    //       C - matrica, tako da je C[i, k] najveća moguća vrednost ranca sa
    //           sadržajem veličine k, ako je moguće koristiti samo prvih i
//predmeta
    //       l - matrica, tako da je l[i, k] broj predmeta tipa i u tom
//maksimalno vrednom
    //           sadržaju ranca veličine k
    ranacSaVrednostima(S, K) {
        for (i: { 0, … , n }) {
            C[i, 0] = 0; //najveca vrednost ranca sa sadrzajem veličine 0
            l[i, 0] = 0;
        }

        for (k: { 0, … , K }) {
            C[0, k] = 0; //najveca vrednost ranca sa sadrzajem veličine k
//koji sadrzi prvih 0 elemenata
            l[0, k] = 0;
        }

        for (i: { 1, … , n }) {
            for (k: { 0, … , K }) {
                C[i, k] = C[i - 1, k];
                l[i, k] = 0;

                if (k ≥ S[i] &&  //tj. Ako predmet velicine S[i] moze da stane
                        (C[i, k - S[i]] > 0 || k - S[i] == 0) &&
                        C[i, k - S[i]] + v[i] > C[i, k]) {

                    C[i, k] = C[i, k - S[i]] + v[i]; //ubacujemo i-ti predmet
                    l[i, k] = l[i, k - S[i]] + 1;
```

```
            }
        }
    }
```

7. (Za razmišljanje -Svesti sledeći problem na neki od poznatih algoritama rađenih na vežbama.) Dat je skup predmeta i njihove mase u tonama (mase su izražene celim brojevima). Potrebno je izabrati one predmete kojim je moguće popuniti brod koji može da ponese K tona, tako da se maksimizuje broj predmeta na brodu. Navesti algoritam na koji svodite ovaj problem, i ukratko objasniti potrebne izmene algoritma ili potrebne izmene podataka koje prosleđujete algoritmu.

Diskusija

Loše ili nepotpune ideje

1.
"svodimo na ranac, samo maksimizujemo broj predmeta umesto vrednosti"

2. "svodimo na ranac", ali se ne navodi tačno na koji od problema ranca se zadati problem svodi (sa ponavljanjem ili bez, sa vrednostima ili bez);

3. sortiranje datih elemenata po težinama.
Mana: Sortiranje je problematično za složenost algoritma kad je broj predmeta velik;
Ili ako je sortiranje obrnuto od zeljenog poretka9 što bi dovelo do pogrešnog rezultata algoritma.

4. Neki su postavljali da je vrednost predmeta jednaka masi i tražili maksimum - svako rešenje će imati istu ukupnu vrednost, jer svi predmeti imaju istu ukupnu masu - K;

5. U nekoliko radova je spominjano da posle izvršavanja algoritma možemo da pregledamo sva rešenja koja imamo, i da izaberemo ono sa najviše predmeta. Algoritam za problem ranca (bilo koja varijanta koju smo radili) daje jedno rešenje na kraju.

Najjednostavnije rešenje je bilo svesti problem na ranac
a) kod koga nema ponavljanja predmeta,
b) kod koga maksimizujemo vrednost predmeta u rancu.
c) Vrednost svakog predmeta postavimo da bude 1.
Ukupna vrednost predmeta u rancu će biti baš ukupan broj predmeta.