REKURZIJE U INFORMATICI

Ana Crnković, XV. gimnazija, Zagreb

U ovom članku govorit ću o dinamičkom programiranju i navesti primjere gdje se ono koristi. Objasnit ću što je rekurzivno zadan niz, stanje niza i koliko stanja ima neka rekurzija. Pojmovi *skup, kartezijev produkt skupova, kardinalitet skupa, rekurzija* i *niz* objašnjeni su u članku „Uvod u rekurzije" u Matki 98.

U informatici, pri rješavanju nekog problema bitne su dvije stvari: koliko memorije koristi neki program i koliko je vremena potrebno da se taj program izvrši. Program je to bolji što je manje vremena potrebno da se izvrši i što manje memorije koristi. Naravno, ponekad je informatičarima bitnija memorija nego vrijeme izvršavanja programa i obratno. To ovisi o cilju programera.

Mi ćemo raspraviti vremenski aspekt u računanju rekurzija.

Recimo da imamo zadan Fibonaccijev niz:

$f(1) = 1, f(2) = 1,$

$f(n) = f(n-1) + f(n-2).$

Ovaj niz ima dvije početne vrijednosti, prvi član niza $f(1) = 1$ i drugi član niza $f(2) = 1$. Dana je relacija $f(n) = f(n-1) + f(n-2)$ koja je istovremeno i *rekurzija*. Tada kažemo da je ovaj niz *rekurzivno zadan*.

Problem $f(n)$ rješava se pomoću podproblema $f(n-1)$ i $f(n-2)$. Dakle, ako izračunamo $f(n-1)$ i $f(n-2)$, možemo izračunati i $f(n)$. Taj način rješavanja problema u informatici se naziva *dinamičko programiranje*.

Općenitije, *dinamičko programiranje* je rješavanje nekog problema na način da ga se podijeli na podprobleme čija rješenja onda koristimo da dobijemo rješenje glavnog problema.

Zapravo, čim imamo nekakvu *rekurziju*, kažemo da problem rješavamo *dinamičkim programiranjem*.

Kada promatramo koliko dugo treba da se problem izvrši, moramo imati na umu broj operacija zbrajanja i množenja potreban da se izračuna rješenje problema. Što više operacija imamo, to se duže program izvršava.

Iz *rekurzije* $f(n) = f(n-1) + f(n-2)$ znamo da je, ako želimo izračunati $f(n)$, potrebno izračunati $f(n-1)$ i $f(n-2)$. Da bismo izračunali $f(n-1)$ i $f(n-2)$, potrebno je izračunati opet $f(n-2)$ i još $f(n-3)$ i $f(n-4)$... Kako idemo dalje, doći ćemo do $f(3)$, $f(2)$ i $f(1)$.

Zapravo, da bismo izračunali $f(n)$, potrebno je izračunati $f(1)$, $f(2)$, $f(3)$..., $f(n-1)$, tj. trebamo izračunati f za sve vrijednosti od 1 do $n-1$. Tada kažemo da su 1, 2, 3..., n stanja koja može poprimiti funkcija f.

Kada bi bilo $n = 4$, tada bi skup svih stanja bio $S = \{1, 2, 3, 4\}$ i broj stanja $\text{card}(S) = 4$.

Rekurzija može ovisiti o više parametara. Npr. rekurzija

$f(n, k) = f(n-1, k-1) + f(n-1, k)$, $n \leq 3$ i $k \leq 4$.

U ovom slučaju *rekurzija* ovisi o dva parametra, n i k. Označimo li skup svih stanja sa S, tada je

$S = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4)\}$,

$S = \{1, 2, 3\} \times \{1, 2, 3, 4\}$.

Broj stanja je $\text{card}(S)$, a $\text{card}(S) = \text{card}(\{1, 2, 3\} \times \{1, 2, 3, 4\}) = 4 \cdot 3 = 12$.

U informatici nas zanima broj stanja jer što je više stanja, imamo više za računati, pa računalu duže treba da riješi problem. Osim samog broja stanja, bitno je da se generalno *f*(*stanje*) može brzo izračunati. Samih operacija zbrajanja i množenja obično ima malo u usporedbi s brojem stanja. Stanja može biti na tisuće i milijune. Iz tog razloga zanemarujemo sam broj operacija pri promatranju koliko će se dugo program izvršavati i promatramo samo broj stanja.

Vratimo se natrag na Fibonaccijev niz i promotrimo izračun njegovog petog člana:

$f(5) = f(4) + f(3)$

$f(5) = (f(3) + f(2)) + f(3)$

$f(5) = (f(2) + f(1)) + f(2) + (f(2) + f(1))$

$f(5) = 3f(2) + 2f(1)) = 3 + 2 = 5$

Primijetimo da je ovaj raspis poprilično dugačak. Da smo znali vrijednosti od $f(4)$ i $f(3)$, raspis bi bio nepotreban. Što dalje idemo s računanjem, recimo $f(6), f(7)..., f(n)$, to je raspis preko $f(1)$ i $f(2)$ duži, time nam i treba više vremena da ga raspišemo. Raspis je dugačak jer smo zapravo računali dva puta $f(3)$.

Da ne bismo morali raspisivati svaki član u nedogled, pamtit ćemo rezultate prijašnjih članova. Tako nakon što smo izračunali prijašnje elemente, recimo $f(3)$ i $f(4)$, sljedeći član $f(5)$ ne moramo raspisivati preko $f(1)$ i $f(2)$.

$f(3) = f(1) + f(2) = 2$

$f(4) = f(3) + f(2) = 2 + 1 = 3$

$f(5) = f(4) + f(3) = 3 + 2 = 5$

U *dinamičkom programiranju* pamćenje rješenja podproblema naziva se još *memoizacija*.

Ono što *memoizacija* radi je sprječavanje da dva ili više puta računamo isti podproblem, a time se skraćuje raspis. Zato je korištenje *memoizacije* jako bitno i koristi se u informatici.

Pogledajmo par zadataka koji se mogu rješavati *dinamičkim programiranjem*. U primjeru s Fibonaccijevim nizom rekurzija je poznata. No, najčešće to nije slučaj, već je zadatak programera pronaći rekurziju koja opisuje zadani problem.

Problem 1. Na koliko je načina moguće imati n kuna ako postoje kovanice od 1, 2 i 5 kuna?

Rješenje: Ako imamo n kuna u rukama, tih n kuna mogli smo dobiti na tri načina:

1. Već smo imali $n - 5$ kuna i dodali smo još 5 kn.
2. Već smo imali $n - 2$ kune i dodali smo još 2 kn.
3. Već smo imali $n - 3$ kuna i dodali smo još 3 kn.

Neka je $g(n)$ broj načina na koji smo mogli dobiti n kuna. Tada iz prijašnje primjedbe znamo da je

$$g(n) = \begin{cases} g(n-5) + g(n-2) + g(n-1), & \text{za } n > 0 \\ 1, & \text{za } n = 0 \\ 0, & \text{za } n < 0 \end{cases}$$

Zapravo nam je $g(0)$ početna vrijednost, kao i g za negativne brojeve. Znamo da je $g(0) = 1$ jer je broj načina na koji možemo imati 0 kuna jednak 1. Također je g za negativne brojeve jednak nula jer ne možemo imati negativan broj kuna u rukama.

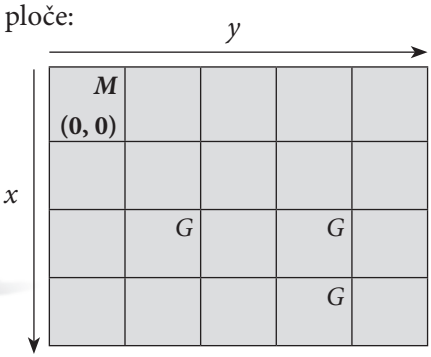
Broj stanja jednak je n.

Problem 2. Marko se nalazi na ploči. Ploča je podijeljena na m redaka i n stupaca. Marko je tek ušao na ploču i nalazi se na polju (0, 0). Marko je posebna osoba i odlučio se micati samo dolje ili samo desno. Dakle, s polja (x, y) Marko može otići samo na polje dolje $(x + 1, y)$ ili polje desno $(x, y + 1)$.

Na nekim se poljima nalaze gljive, na ploči označene s G. Ako se Marko nalazi na polju koje ima gljive, ubrat će jednu gljivu.

Pitanje: Koliko je najviše gljiva Marko ubrao ako se nalazi na polju (x, y)?

Primjer jedne ploče:



Pitamo se koliko je najviše gljiva Marko pokupio ako se nalazi na određenom polju. Da bismo lakše opisivali rješenja, reći ćemo da je rješenje na pitanje „koliko je najviše gljiva Marko pokupio ako se nalazi na polju (x, y)" zapravo $f(x, y)$. Koliko gljiva Marko može ubrati s polja (x, y) označimo s $gljiva(x, y)$.

U svakom slučaju znamo, budući da se Marko može micati samo dolje i samo desno, da je došao na polje (x, y) ili s polja odozgo, tj. s polja $(x - 1, y)$, ili s polja lijevo, tj. s polja $(x, y - 1)$.

Prije nego je došao na polje (x, y) Marko je najviše mogao imati $\max(f(x - 1, y), f(x, y - 1))$ gljiva. Kada je Marko došao na polje (x, y), ubrao još jednu gljivu, ako je gljiva bilo. Koliko je Marko ubrao gljiva s polja (x, y), jednom kada je bio na tom samom polju, jest $gljiva(x, y)$.

Pomoću tih zapažanja sada znamo da je

$$f(x,y)=\begin{cases} 0, & x<0 \text{ ili } y<0 \\ \max(f(x-1,y), f(x,y-1)) + gljiva(x,y) & x\geq 0 \text{ i } y\geq 0 \end{cases}$$

gdje je

$$gljiva(x,y)=\begin{cases} 1, & \text{ako se na polju } (x,\ y) \text{ nalazi gljiva} \\ 0, & \text{ako se na polju } (x,\ y) \text{ ne nalazi gljiva} \end{cases}$$

Mi promatramo koliko gljiva Marko ima od kretanja s polja (0, 0). Za nas, Marko nikad nije bio iznad niti lijevo od tog početnog polja, pa tamo nije ni pokupio nikakve gljive. Zato je $f(x, y) = 0$ kada je $x < 0$ ili $y < 0$, i to nam služi kao početne vrijednosti.

Zgodan redoslijed računanja $f(x, y)$ je red po red, s lijeva na desno.

Tako ćemo uvijek znati potrebne $f(x - 1, y)$ i $f(x, y - 1)$ za svaki $f(x, y)$, tj. imat ćemo već zapamćene potrebne vrijednosti za računanja sljedećeg elementa. Tako možemo lako izračunati $f(x, y)$ za cijelu matricu.

Stanja su svi mogući parovi (x, y) kojih ima $m \cdot n$, tako znamo da je broj stanja jednak $m \cdot n$.