Zadatak br.5

Grupa 4

Novembar 2024

1 Rekurente relacije

1.1 Uvod

U prethodnim zadacima uveli smo različite načine prebrojavanja elemenata. Govorili smo o permutacijama, neuređenim izborima elemenata, uređenim izborima elemenata kao i o principu uključenja i isključenja. Ali šta se dešava u slučaju kada na već poznate načine ne možemo da prebrojimo elemente? U slučaju kada imamo niz gde je dato prvih nekoliko članova, a ostali članovi se mogu dobiti primenom relacije koja koristi prvih nekoliko datih članova, onda govorimo o nizu koji je definisan pomoću rekuretne relacije. U nastavku ćemo dati striktnu matematičku definiciju rekurente relacije kao i primere nizova koji su definisani pomoću rekurente relacije.

1.2 Definicija

Definicija: Neka je a_n, $n \in \mathbb{N}_0 = (a_0,\ a_1,\ ...)$ niz kompleksnih brojeva. Ako postoji funkcija F: $C^n \to C$ sa osobinom da za svako n $\geq$ k važi a_n = F(a_{n-1}, . . . , a_{n-k}) onda kažemo da je (1.1) rekurentna relacija reda k koja opisuje niz a_n, $n \in \mathbb{N}_0$.

Ako su dati prvih k-1 članova niza a_0, . . . , a_{k-1} , onda se na osnovu rekurente relacije na jedinstven načinmogu odrediti ostali članovi niza a_k, a_{k+1}, . . .

1.3 Primer 1

Napisati rekuretnu relacija koja opisuje aritmetički niz čiji je prvi član a, a razlika je d.

Proof. Za članove aritmetičkog niza važi: $a_0 = a$ $a_n = a_{n-1}$ + d, n $\geq$ 1 □

1.4 Primer 2

Napisati rekuretnu relacija koja opisuje geometrijski niz čiji je prvi član a, a količnik je 1.

Proof. Za članove geometrijskog niza važi: $a_0 = b$ $a_n = q \cdot a_{n-1}$, n ≥ 1 □

1.5 Primer 3

Napisati rekuretnu relacija koja opisuje niz faktorijela.

Proof. Za članove aritmetičkog niza važi: $a_0 = 1$ $a_n = n \cdot a_{n-1}$, n ≥ 1 □

1.6 Primer 4

Napisati rekuretnu relacija koja opisuje Fibonačijev niz. Fibonačijev niz je niz gde je prvi član 0, drugi član 1, a svaki sledeći član jednak je zbiru prethodna dva.

Proof. Ako član na poziciji $n \in \mathbb{N}_0$ označimo sa f_n onda važi: $f_0 = 0$ $f_1 = 1$ $f_n = f_{n-1} + f_{n-2}$, n ≥ 2 □

2 Hanojska Kula

2.1 Uvod

Hanojska kula se sastoji od tri štapa i određenog broja diskova različitih veličina, koji mogu da se pomeraju sa jednog štapa na drugi. Igra počinje sa diskovima uredno naslaganim na prvom štapu, tako da je najveći disk na dnu, a najmanji na vrhu.

Cilj je premestiti ceo toranj diskova na drugi štap, pridržavajući se sledećih pravila:

- Može se pomerati samo jedan disk odjednom.
- Svaki potez uključuje uzimanje gornjeg diska sa jednog štapa i njegovo postavljanje na drugi štap.
- Ni jedan disk se ne sme postaviti na manji disk.

2.2 Kako algoritam funkcioniše

1. Ako postoji samo jedan disk ($n = 1$), premesti ga direktno sa štapa *izvor* na štap *cilj*.
2. Inače:
   - Rekurzivno premesti $n - 1$ diskova sa štapa *izvor* na štap *pomoćni*, koristeći *cilj* kao pomoć.
   - Premesti najveći disk sa *izvor* na *cilj*.
   - Rekurzivno premesti $n - 1$ diskova sa štapa *pomoćni* na *cilj*, koristeći *izvor* kao pomoć.

Pseudokod rekurzivnog algoritma:

```
hanoi(n, izvor, cilj, pomoćni):
    ako je n = 1:
        premesti disk sa "izvor" na "cilj"
    inače:
        hanoi(n - 1, izvor, pomoćni, cilj)
        premesti disk n sa "izvor" na "cilj"
        hanoi(n - 1, pomoćni, cilj, izvor)
```

2.3 Algoritam u jeziku Java

- Primer Java algoritma :

```java
import java.io.*;
import java.math.*;
import java.util.*;
class HanoiTower {
    static void hanoi(int n, char sa,
                          char na, char temp)
    {
        if (n == 0) {
            return;
        }
        hanoi(n - 1, sa, temp, na);
        System.out.println("Prebaci disk " + n + " sa stapa "
                        + sa + " na stap "
                        + na);
        hanoi(n - 1, temp, na, sa);
    }

    public static void main(String args[])
    {
        int N = 3;

        hanoi(N, 'A', 'C', 'B');
    }
}
```

Rešenje Hanojske kule za 3 diska

- Prebaci disk 1 sa štapa A na štap C
- Prebaci disk 2 sa štapa A na štap B
- Prebaci disk 1 sa štapa C na štap B
- Prebaci disk 3 sa štapa A na štap C

- Prebaci disk 1 sa štapa B na štap A
- Prebaci disk 2 sa štapa B na štap C
- Prebaci disk 1 sa štapa A na štap C

3 Rešenje rekurentne relacije

Cilj rešavanja rekurentne relacije je da se pronađe eksplicitna formula, poznata kao *opšte rešenje*, koja omogućava izračunavanje bilo kog člana niza bez potrebe za računanje prethodnih vrednosti. Ovo opšte rešenje često se izražava u zatvorenom obliku, što omogućava bržu i jednostavniju evaluaciju niza za bilo koji prirodan broj n.

Rešavanje rekurentne relacije uključuje izražavanje opšteg člana niza a_n kao funkciju koja zavisi od n,, tj. odrediti funkciju $a : \mathbb{N}_0 \to \mathbb{C}$ tako da za svako $n_2 \in \mathbb{N}_0$ važi

$$a_n = a(n)$$

Zadatak 1. *Neka je* $\{a_n\}_{n\in\mathbb{N}_0}$ *zadat na sledeći način:*

$$a_0 = 2, a_n = 5a_{n-1} + 2, n \geq 1$$

Odrediti rešenje rekurentne relacije metodom zamene unazad.

Rešenje. Korišćenjem metode zamene unazad, za $n \geq 1$, može se zaključiti da važi sledeće:

$$\begin{aligned} a_n &= 5a_{n-1} + 2 = 5(5a_{n-2} + 2) + 2 = 5^2 a_{n-2} + 5 \cdot 2 + 2 \\ &= 5^2(5a_{n-3} + 2) + 5 \cdot 2 + 2 = 5^3 a_{n-3} + 5^2 \cdot 2 + 5 \cdot 2 + 2 \\ &\vdots \\ &= 5^n a_0 + 2 \cdot (5^{n-1} + 5^{n-2} + \cdots + 5 + 1). \end{aligned}$$

S obzirom da je $5^{n-1} + 5^{n-2} + \cdots + 5 + 1$ geometrijska serija, može se ispisati kao:

$$a_n = 2 \cdot \left(\frac{5^{n+1} - 1}{5 - 1} \right).$$

[2] □

U prethodnom primeru smo došli do oblika rješenja rekurentne relacije. U sledećem će uz pomoć indukcije biti dokazano da to nije slučajnost.

Zadatak 2. *Neka je* $\{a_n\}_{n\in\mathbb{N}_0}$ *zadat na sledeći način:*

$$a_0 = 2, a_n = 5a_{n-1} + 2, n \geq 1$$

Dokazati da je $a_n = \frac{1}{2}(5^{n+1} - 1)$ *za svako* $n \in \mathbb{N}_0$.

Rešenje. Sada ćemo rešavati pomoću indukcije gdje je:

Baza $n = 0$:

$$a_0 = \frac{1}{2}(5^{0+1} - 1) = \frac{1}{2}(5 - 1) = 2.$$

Induktivna pretpostavka T_n:

$$a_n = \frac{1}{2}(5^{n+1} - 1).$$

Induktivni korak $T_n \to T_{n+1}$:
Trebamo pokazati da važi i za $n + 1$, odnosno da:

$$a_{n+1} = \frac{1}{2}(5^{n+2} - 1).$$

Po rekurentnoj relaciji imamo:

$$a_{n+1} = 5a_n + 2.$$

Zamenjujemo izraz za a_n iz pretpostavke:

$$a_{n+1} = 5 \cdot \frac{1}{2}(5^{n+1} - 1) + 2.$$

Dalje razvijamo:

$$= \frac{1}{2}(5^{n+2} - 5) + 2 = \frac{1}{2}(5^{n+2} - 1).$$

[2] □

Time smo dokazali da induktivna hipoteza važi i za $n + 1$, što završava induktivni korak.

4 Homogene linearne rekurentne relacije sa konstantnim koeficijentima

Jedan od načina za rešavanje rekurentnih relacija jeste postupak zamene unazad, ali taj metod može biti nepraktičan za kompleksnije rekurentne nizove. Stoga ćemo se ovde fokusirati na specijalnu klasu rekurentnih relacija koje se mogu rešiti upotrebom karakteristične jednačine.

Rekurentne relacije su veoma važne u matematici i informatici jer omogućavaju da se odnosi izmedju članova niza izraze pomoću prethodnih članova tog istog niza. One se javljaju u različitim oblastima, uključujući teoriju brojeva, algoritme i mnoge druge grane nauke. Konkretno, linearne rekurentne relacije sa konstantnim koeficijentima su često jednostavne za rešavanje, a njihove metode rešavanja podsećaju na postupke korišćene za linearne diferencijalne jednačine.

U ovom radu ćemo se fokusirati na homogene linearne rekurentne relacije sa konstantnim koeficijentima. Uvodimo osnovne definicije i teoreme koje omogućavaju pronalaženje rešenja ovih relacija.

Rekurentna relacija niza $\{a_n\}_{n\in\mathbb{N}}$ je homogena linearna rekurentna relacija reda k sa konstantnim koeficijentima ako se može zapisati u obliku:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

gde su $c_1, \ldots, c_k$ konstante, $k \geq 1$, a $c_k \neq 0$.

Ova vrsta rekurentne relacije uvek ima trivijalno rešenje $a_n = 0$ za sve $n \geq 0$.

4.1 Karakteristična jednačina

Neka je niz $\{a_n\}_{n\in\mathbb{N}}$ odredjen homogenom linearnom rekurentnom relacijom sa konstantnim koeficijentima:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}.$$

Pretpostavljajući da postoji broj $r \neq 0$ takav da rešenje posmatrane rekurentne relacije bude oblika $a_n = r^n$ za svako $n \geq 0$, dolazimo do tzv. karakteristične jednačine:

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_k = 0.$$

Ako karakteristična jednačina rekurentne relacije

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

ima k različitih korena $x_1, x_2, \ldots, x_k$, tada važi:

(i) Za sve konstante $a_1, a_2, \ldots, a_k$, funkcija $a(n) = a_1 x_1^n + a_2 x_2^n + \cdots + a_k x_k^n$ je rešenje posmatrane rekurentne relacije.

(ii) Konstante $a_1, \ldots, a_k$ su jedinstveno odredjene početnim uslovima $a(0) = a_0, \ldots, a(k-1) = a_{k-1}$.

Dokaz. Zamenom izraza $a(n) = a_1 x_1^n + a_2 x_2^n + \cdots + a_k x_k^n$ u rekurentnu relaciju pokazujemo da su leve i desne strane jednake nuli, što potvrdjuje da je ovo rešenje.

Jedinstvenost rešenja može se dokazati upotrebom **Vandermondove determinante**. Naime, za različite korene $x_1, x_2, \ldots, x_k$, kvadratna matrica sa kolonama u formi $[x_i^j]_{i,j=0}^{k-1}$ ima nenultu determinantu, što znači da se sistem može jedinstveno rešiti. Ovaj rezultat omogućava da se konstante $a_1, \ldots, a_k$ jednoznačno odrede početnim uslovima.

Ako karakteristična jednačina

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_k = 0$$

ima korene $x_1, \ldots, x_p$, redom višestrukosti $k_1, \ldots, k_p$, tada je

$$a(n) = (a_{1,0} + a_{1,1} n + \cdots + a_{1,k_1-1} n^{k_1-1}) x_1^n + \cdots + (a_{p,0} + a_{p,1} n + \cdots + a_{p,k_p-1} n^{k_p-1}) x_p^n,$$

gde su konstante $a_{i,j}$ jedinstveno odredjene početnim uslovima.

Dokaz. Slično kao u prethodnom slučaju, dokaz se vrši zamenom predloženog oblika rešenja u rekurentnu relaciju i koristeći činjenicu da su koreni karakteristične jednačine višestruki. Kada postoje višestruki koreni, rešenje se proširuje dodatnim članovima koji sadrže n-ove potencije, kako bi se obuhvatila sva rešenja.

5 Nehomogene rekuretne relacije sa konstantnim koeficijentima

To su rekurentne relacije koje su oblika:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + f(n),$$

gde važi da su $c_1, c_2, c_3, \ldots, c_k$ konstante, $k \geq 1$, $c_k \neq 0$, a f funkcija skupa N_0.

Kada umesto $f(n)$ uzmemo 0, tada dobijamo odgovarajuću homogenu rekurentnu relaciju koju koristimo u daljem rešavanju. Na sličan način se rešavaju i diferencijalne jednačine sa konstantnim koeficijentima.

Teorema 46 Ako je $a_n^{(p_1)}$ jedno partikularno rešenje nehomogenog dela linearne rekurentne relacije sa konstantnim koeficijentima, tada za svako a_n postoji rešenje $a_n^{(h)}$ odgovarajuće homogene rekurentne relacije gde važi

$$a_n = a_n^{(h)} + a_n^{(p_1)}, \quad n \geq 0.$$

Dokaz. Neka je $a_n^{(p_1)}$ dato partikularno rešenje. Tada za svako $n \in N_0$ važi

$$a_n^{(p_1)} = c_1 a_{n-1}^{(p_1)} + c_2 a_{n-2}^{(p_1)} + \ldots + c_k a_{n-k}^{(p_1)} + f(n).$$

Posmatrajmo sada proizvoljno rešenje $a_n^{(p)}$. Za njega važi

$$a_n^{(p)} = c_1 a_{n-1}^{(p)} + c_2 a_{n-2}^{(p)} + \ldots + c_k a_{n-k}^{(p)} + f(n), \quad n \in N_0.$$

Ako od druge relacije oduzmemo prvu, dobijamo

$$a_n^{(p)} - a_n^{(p_1)} = c_1 \left(a_{n-1}^{(p)} - a_{n-1}^{(p_1)}\right) + c_2 \left(a_{n-2}^{(p)} - a_{n-2}^{(p_1)}\right) + \ldots + c_k \left(a_{n-k}^{(p)} - a_{n-k}^{(p_1)}\right).$$

Vidimo da $f(n) - f(n) = 0$, što znači da je $a_n^{(p)} - a_n^{(p_1)}$ rešenje homogene rekurentne relacije tj. za svako rešenje nehomogene rekurentne relacije a_n postoji rešenje $a_n^{(h)}$ homogene jednadžine tako da se $a_n^{(p)}$ može izraziti pomoću $a_n^{(h)}$ i $a_n^{(p_1)}$ na sledeći način:

$$a_n = a_n^{(h)} + a_n^{(p_1)}.$$

5.1 Teorema 1

Neka je

$$f(n) = \left(b_m n^m + b_{m-1} n^{m-1} + \cdots + b_1 n + b_0\right) s^n, b_0, \ldots, b_m \in \mathbb{R}$$

Ako je s koren karakteristicne jednacine visestrukosti l (ako nije koren tada je l = 0) onda postoji partikularno resenje oblika

$$a_n^{(p)} = n^l \cdot \left(c_m n^m + c_{m-1} n^{m-1} + \cdots + c_1 n + c_0\right) s^n$$

5.2 Teorema 2

Ako su $a_n^{(p_1)}$ i $a_n^{(p_2)}$ redom resenja nehomogenih rekurentnih relacija

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + f_1(n)$$
$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + f_2(n),$$

onda je $a_n^{(p_1)} + a_n^{(p_2)}$ resenje nehomogene rekurentne relacije

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + f_1(n) + f_2(n)$$

Dokaz. Ako su $a_n^{(p_1)}$ i $a_n^{(p_2)}$ redom resenja nehomogenih rekurentnih relacija onda vazi

$$a_n^{(p_1)} = c_1 a_{n-1}^{(p_1)} + c_2 a_{n-2}^{(p_1)} + \cdots + c_k a_{n-k}^{(p_1)} + f_1(n)$$
$$a_n^{(p_2)} = c_1 a_{n-1}^{(p_2)} + c_2 a_{n-2}^{(p_2)} + \cdots + c_k a_{n-k}^{(p_2)} + f_2(n)$$

Sabiranjem prethodne dve doibijamo

$$a_n^{(p_1)} + a_n^{(p_2)} = c_1 \left(a_{n-1}^{(p_1)} + a_{n-1}^{(p_2)}\right) + c_2 \left(a_{n-2}^{(p_1)} + a_{n-2}^{(p_2)}\right) + \cdots + c_k \left(a_{n-k}^{(p_1)} + a_{n-k}^{(p_2)}\right) + f_1(n) + f_2(n)$$

Cime je pokazano resenje rekuretne relacije

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + f_1(n) + f_2(n)$$

□

Primer 1:
Data je rekurentna relacija:

$$a_{n+1} - 2a_n = 2^n, \quad n \geq 0, \quad a_0 = 1$$

$$a_n = a_n^h + a_n^p$$

gde je a_n^h rešenje homogene jednačine, a a_n^p posebno rešenje.
Homogena jednačina je

$$a_{n+1}^h - 2a_n^h = 0$$

Pretpostavimo rešenje oblika $a_n^h = \alpha \cdot 2^n$. Uvodeći ovu pretpostavku u homogenu jednačinu, dobijamo:

$$\alpha r^{n+1} - 2\alpha r^n = 0$$

Deljenjem sa αr^n (pod pretpostavkom da $\alpha \neq 0$ i $r \neq 0$), dobijamo

$$r - 2 = 0 \Rightarrow r = 2$$

Dakle, opšte rešenje homogene jednačine je

$$a_n^h = \alpha \cdot 2^n$$

Pretpostavimo da je partikularno rešenje oblika

$$a_n^p = A \cdot n \cdot 2^n$$

a_{n+1}^p:

$$a_{n+1}^p = A \cdot (n+1) \cdot 2^{n+1} = 2A \cdot (n+1) \cdot 2^n$$

Sada ubacujemo a_{n+1}^p i a_n^p u početnu rekurentnu relaciju

$$a_{n+1}^p - 2a_n^p = 2A \cdot (n+1) \cdot 2^n - 2 \cdot (A \cdot n \cdot 2^n)$$

Pojednostavljivanjem dobijamo

$$= 2A \cdot 2^n$$

Pošto želimo da ovaj izraz bude jednak 2^n, dobijamo

$$2A = 1 \Rightarrow A = \frac{1}{2}$$

Sabiranjem homogenog i partikularnog rešenja, dobijamo opšte rešenje

$$a_n = a_n^h + a_n^p = \alpha \cdot 2^n + \frac{1}{2} \cdot n \cdot 2^n$$

Možemo faktorisati 2^n:

$$a_n = \left(\alpha + \frac{n}{2}\right) \cdot 2^n$$

Koristeći početni uslov $a_0 = 1$

$$a_0 = \left(\alpha + \frac{0}{2}\right) \cdot 2^0 = \alpha$$

Dakle, $\alpha = 1$.

Rešenje rekurentne relacije je

$$a_n = (2 + n) \cdot 2^{n-1}$$

Primer 2:
Data je rekurentna relacija

$$a_{n+2} + 3a_{n+1} + 2a_n = 3^n, \quad n \geq 0, \quad a_0 = 0, \quad a_1 = 1$$

Opet posmatramo prvo homogenu relaciju

$$a_{n+2}^h + 3a_{n+1}^h + 2a_n^h = 0$$

Pretpostavimo da je $a_n^h = r^n$, karakteristična jednačina je

$$r^2 + 3r + 2 = 0$$

što faktorišemo kao

$$(r+2)(r+1) = 0$$

Rešenja su $r = -2$ i $r = -1$, pa je homogeno rešenje

$$a_n^h = \alpha(-2)^n + \beta(-1)^n$$

Za partikularno rešenje pretpostavimo oblik

$$a_n^p = A \cdot 3^n$$

$$a_{n+1}^p = 3A \cdot 3^n, \quad a_{n+2}^p = 9A \cdot 3^n$$

Ubacimo u rekurentnu relaciju

$$9A \cdot 3^n + 3 \cdot (3A \cdot 3^n) + 2 \cdot (A \cdot 3^n) = 3^n$$

što daje

$$20A \cdot 3^n = 3^n \Rightarrow A = \frac{1}{20}$$

Tada je partikularno rešenje

$$a_n^p = \frac{1}{20} \cdot 3^n$$

Pa je opšte rešenje

$$a_n = a_n^h + a_n^p = \alpha(-2)^n + \beta(-1)^n + \frac{1}{20} \cdot 3^n$$

Koristimo početne uslove za $a_0 = 0$ i $a_1 = 1$
Za $n = 0$:

$$\alpha + \beta + \frac{1}{20} = 0 \Rightarrow \alpha + \beta = -\frac{1}{20}$$

Za $n = 1$:

$$-2\alpha - \beta + \frac{3}{20} = 1 \Rightarrow -2\alpha - \beta = \frac{17}{20}$$

Rešavanjem sistema dobijamo

$$\alpha = -\frac{4}{5}, \quad \beta = \frac{3}{4}$$

Konačno rešenje je

$$a_n = -\frac{4}{5} \cdot (-2)^n + \frac{3}{4} \cdot (-1)^n + \frac{1}{20} \cdot 3^n$$

Primer 3: Odrediti broj n-cifrenih sekvenci sa ciframa 0, 1, 2, i 3 u kojima se cifra 3 nikada ne pojavljuje desno od cifre 0.

Definišemo a_n kao broj takvih n-cifrenih sekvenci koje zadovoljavaju uslov.

Analiziramo slučajeve za svaku cifru na poslednjem mestu:

1. Ako je poslednja cifra 1 ili 2: Broj sekvenci koje se završavaju sa 1 ili 2 jednak je broju $(n-1)$-cifrenih sekvenci koje zadovoljavaju uslov, što je a_{n-1}.

2. Ako je poslednja cifra 0: Sve cifre levo od nje moraju takodje zadovoljavati uslov, pa je broj sekvenci koje zavrsavaju sa 0 takodje a_{n-1}.

3. Ako je poslednja cifra 3: U ovom slučaju, cifra 3 ne sme biti desno od cifre 0, što znači da nema nula u sekvenci. Dakle, broj n-cifrenih sekvenci koje sadrže samo cifre 1, 2 i 3 iznosi 3^{n-1} (jer imamo 3 izbora za svaku poziciju: 1, 2 ili 3).

Dakle rekurentna relacija glasi

$$a_n = 3a_{n-1} + 3^{n-1}, \quad a_0 = 1, \quad a_1 = 4$$

6 Rekurzivni problemi i njihova resenja

6.1 Problem 1

6.1.1 Problem

Racunanje faktorijela broja n gde je $n! = n * (n-1) * (n-2) * \ldots * 1$

6.1.2 Rekurzivni pristup

$factorial(n) = n * factorial(n-1)$ gde je bazni slucaj $factorial(1) = 1$

6.1.3 Algoritam u jeziku Java

- Primer Java algoritma :

```
//Java program koji racuna faktorijel broja n
public class Factorial {
 public static int factorial(int n) {
    // Bazni slucaj: ako je n 0 ili 1, faktorijel je 1
    if (n <= 1) {
        return 1;
    }
    // Rekurzivni slucaj: n * factorial(n-1)
    return n * factorial(n - 1);
    }
 }
```

6.1.4 Objašnjenje algoritma

Funkciji factorial(int n) se prosledjuje celobrojna vrednost ciji faktorijel trazimo. Po ulasku u funkcija if blok pita da li je broj n manji ili jednak 1 i ako jeste vraca vrednost 1. Ako je broj veci od 1 funkcija vraca broj n pomnozen sa povratnom vrednosti funkcije factorial(n-1). Rekurzivnim pozivima se n umanjuje dok ne dodje do broja 1 i tada se vrednosti vracaju u pocetni poziv funkcije odakle se rezultat vraca.

6.2 Problem 2

6.2.1 Problem

Treba pronaci element u prethodno sortiranom nizu

6.2.2 Rekurzivni pristup

Poredimo vrednost trazenog elementa sa elementom na sredini niza, ako nije pronadjen rekurzivno trazimo element u levoj ili desnoj polovini niza

6.2.3 Algoritam u jeziku Java

- Primer Java algoritma :

```
//Java program koji pronalazi element u sortiranom nizu
    koriscenjem binarne pretrage
 public class BinarySearch {
 public static int binarySearch(int[] array, int target, int
     left, int right) {
    // Bazni slucaj: ako je levi indeks veci od desnog, element
        ne postoji u nizu
    if (left > right) {
        return -1; // Oznacava da nije pronadjen
```

```
    }

    // Racunanje indeksa srednjeg elementa
    int mid = left + (right - left) / 2;

    // Proveri da li je element na sredini trazeni
    if (array[mid] == target) {
        return mid;
    }

    // Rekurzivni slucajevi
    if (array[mid] > target) {
        // Ako je srednji element veci od trazenog pretrazujemo
            levu polovinu niza
        return binarySearch(array, target, left, mid - 1);
    } else {
        // Ako je suprotan slucaj pretrazujemo desnu polovinu
            niza
        return binarySearch(array, target, mid + 1, right);
        }
    }
}
```

6.2.4 Objašnjenje algoritma

Niz mora prethodno biti sortiran u rastucem poretku. Na pocetku svakog prolaska kroz funkciju if blok pita da li je levi indeks veci od desnog, ako jeste element ne postoji u datom nizu i funkcija vraca -1. Nakon toga racuna se indeks elementa na sredini i pita se da li je element sa tim indeksom jednak trazenom elementu, ako jeste vraca se srednji indeks. Ako nije funkcija dalje pita da li je element na sredini veci od trazenog, ako jeste funkcija se ponovo poziva i prosledjuju joj se indeksi leve granice niza i prvog elementa pre srednjeg, ako nije funkcija se poziva i kao levi indeks prosledjuje joj se indeks elementa posle srednjeg i desne granice tog niza. Niz je tako upola manji i potraga se nastavlja dok se element ne pronadje.

6.3 Problem 3

6.3.1 Problem

Treba proci kroz lavirint sa pravouglim coskovima

6.3.2 Rekurzivni pristup

Prolazimo od tacke do tacke i vracamo se korak unazad ako naletimo na zid kroz koji ne moze da se prodje

6.3.3 Algoritam u jeziku Java

- Primer Java algoritma :

```java
//Java program koji pronalazi izlaz iz lavirinta
public class MazePathfinding {
// Definisemo lavirint pomocu matrice, 1 oznacava put, 0
    oznacava zid, krecemo iz gornjeg levog coska i treba doci
    do donjeg desnog
static int[][] maze = {
    {1, 0, 1, 1, 1},
    {1, 1, 1, 0, 1},
    {0, 1, 0, 1, 1},
    {1, 1, 1, 1, 0},
    {1, 0, 1, 1, 1}
};

// Pratimo posecene tacke kako ne bi ponavljali cikluse
static boolean[][] visited = new
    boolean[maze.length][maze[0].length];

public static boolean isPath(int x, int y) {
    // Bazni slucaj: proveri da li x,y u granicama lavirinta,
        da li je put otvoren i da li je tacka neposecena
    if (x < 0 || y < 0 || x >= maze.length || y >=
        maze[0].length || maze[x][y] == 0 || visited[x][y]) {
        return false;
    }

    // Ako dodjemo do donjeg desnog coska put je pronadjen
    if (x == maze.length - 1 && y == maze[0].length - 1) {
        return true;
    }

    // Obelezimo tacku kao posecenu
    visited[x][y] = true;

    // Rekurzivni poziv da se pomerimo u sva 4 smera
    if (isPath(x + 1, y) || isPath(x - 1, y) || isPath(x, y +
        1) || isPath(x, y - 1)) {
        return true;
    }

    // Izbrisemo tacku iz mape pronadjenih (vracamo se korak
        unazad)
    visited[x][y] = false;
    return false;
}
}
```

6.3.4 Objašnjenje algoritma

Lavirint se definise putem matrice gde elementi sa vrednosti 1 oznacavaju put kojim se moze proci, a elementi sa vrednosti 0 zid kroz koji se ne moze proci. Pravi se jos jedna matrica koja je iste velicine kao lavirint i prati da li su tacke posecene. Na pocetku prolaska kroz funkciju proveravamo da li su x i y u granicama lavirinta, da li stojimo na tacki koja je put i da li je ona vec posecena. Ako bilo koji od uslova nije ispunjen funkcija vraca false. Nakon toga pitamo da li se trenutno nalazimo u donjem desnom cosku, funkcija vraca true ako je uslov ispunjen i svaki prethodni poziv ce takodje vratiti true cime oznacavamo da smo pronasli put. Ako uslov nije ispunjen funkcija prolazi dalje i obelezava tacku na kojoj stojimo kao posecenu. Nakon toga funkcija se rekurzivno poziva i pomeramo se u sva 4 smera. Ako ne mozemo da se pomerimo ni u jedan od 4 smera jer neki uslov iz prvog if bloka nije ispunjen to znaci da smo naleteli na corsokak i moramo se vratiti korak u nazad. Funkcija brise ovu tacku iz mape posecenih i vraca vrednost false. Rekurzivni pozivi idu sve dok se ne dodje do donjeg desnog coska.

7 Zadaci reseni u Java kodu

7.1 Zadatak 1

7.1.1 Postavka zadatka

Prvi element je $a_0 = 2$, a drugi je $a_1 = 1$.

$$a_n = 4a_{n-1} - 4a_{n-2}, n \geq 2$$

7.1.2 Pristup u Javi

Koristicemo iterativni pristup kako bi izbegli duboku rekurziju za velike n

7.1.3 Algoritam u jeziku Java

- Primer Java algoritma :

```
//Java program resava rekurentnu relaciju
 public class RecurrenceSolver {
 public static int solveRecurrence(int n) {
     // Bazni slucajevi
     if (n == 0) return 2;
     if (n == 1) return 1;

     // Promenljive koje pamte poslednje dve koriscene vrednosti
     int a_n_minus_2 = 2; // a_0
     int a_n_minus_1 = 1; // a_1
     int a_n = 0;
```

```java
        // Iterativno racunanje do n-tog elementa
        for (int i = 2; i <= n; i++) {
            a_n = 4 * a_n_minus_1 - 4 * a_n_minus_2;

            // Azuriranje promenljivih za sledecu iteraciju
            a_n_minus_2 = a_n_minus_1;
            a_n_minus_1 = a_n;
        }

        return a_n;
    }

    public static void main(String[] args) {
        int n = 5; // Racunanje 5. elementa kao primer
        System.out.println("The term a(" + n + ") = " +
             solveRecurrence(n));
    }
}
```

7.1.4 Objašnjenje algoritma

Program ispita bazne slucajeve, nakon cega belezi vrednosti a_0 i a_1. Ulazi u for petlju gde svakim prolaskom azurira promenljivu a_n i promenljive koje koristi u racunu. Nakon zavrsene for petlje vraca vrednost n-tog elementa.

7.2 Zadatak 2

7.2.1 Postavka zadatka

Prvi element je $a_0 = 1$, drugi je $a_1 = 2$, a treci je $a_2 = 3$.

$$a_n = 4a_{n-1} + a_{n-2} - 4a_{n-3}, n \geq 3$$

7.2.2 Pristup u Javi

Koristicemo iterativni pristup kako bi izbegli duboku rekurziju za velike n

7.2.3 Algoritam u jeziku Java

- Primer Java algoritma :

```java
 //Java program resava rekurentnu relaciju
  public class RecurrenceSolver {
  public static int solveRecurrence(int n) {
      // Bazni slucajevi
      if (n == 0) return 1;
      if (n == 1) return 2;
```

```java
        if (n == 2) return 3;

        // Promenljive koje pamte poslednje tri koriscene vrednosti
        int a_n_minus_3 = 1; // a_0
        int a_n_minus_2 = 2; // a_1
        int a_n_minus_1 = 3; // a_2
        int a_n = 0;

        // Iterativno racunanje do n-tog elementa
        for (int i = 3; i <= n; i++) {
            a_n = 4 * a_n_minus_1 + a_n_minus_2 - 4 * a_n_minus_3;

            // Azuriranje promenljivih za sledecu iteraciju
            a_n_minus_3 = a_n_minus_2;
            a_n_minus_2 = a_n_minus_1;
            a_n_minus_1 = a_n;
        }

        return a_n;
    }

    public static void main(String[] args) {
        int n = 7; // Racunanje 7. elementa kao primer
        System.out.println("The term a(" + n + ") = " +
             solveRecurrence(n));
    }
}
```

7.2.4 Objašnjenje algoritma

Program ispita bazne slucajeve, nakon cega belezi vrednosti a_0, a_1 i a_2. Ulazi u for petlju gde svakim prolaskom azurira promenljivu a_n i promenljive koje koristi u racunu. Nakon zavrsene for petlje vraca vrednost n-tog elementa.

Reference

[1] Kenneth H. Rosen, *Discrete mathemathics and its appliacations Seventh Edition*, McGraw-Hill, 2012

[2] Dragan Stevanović i Miroslav Ćirić i Slobodan Simić i Vladimir Baltić, *DISKRETNA MATEMATIKA OSNOVE KOMBINATORIKE I TEORIJE GRAFOVA*, Matematički institut u Beogradu, 2007

[3] Sussana S. Epp, *Discrete Mathematics with Applications*, Cengage Learning, Inc. 2019

[4] Richard A. Brualdi, *Introductory Combinatorics, Fifth Edition*, Pearson Education,2009

[5] Jiri Matoušek, Jaroslav Nešetril *Invitation to Discrete Mathematics, 2nd edition*, Oxford University Press, 2008

[6] Jovanka Pantovic, Skrita: Rekurentne relacije, Fakultet Tehnickih Nauka, Univerzitet u Novom Sadu