Iterativni i rekurzivni postupci

Opšti rekurentni nizovi

- Često je *r* mali broj, pa se tada neke petlje mogu zameniti nizom ekvivalentnih naredbi
- Takođe, izračunavanje narednog elementa često je jednostavno, pa se metod `f` može izostaviti
- Kao primer opšteg rekurentnog niza navodimo Fibonačijeve brojeve
- Svaki naredni Fibonačijev broj je zbir prethodna dva Fibonačijeva broja

Fibonačijevi brojevi

- Rekurentni niz Fibonačijevih brojeva definisan je sa:

$$f_n = f_{n-1} + f_{n-2}, n > 1$$
$$f_0 = 0, f_1 = 1$$

- Primenićemo opšti postupak za izračunavanje rekurentnih nizova od *r* promenljivih, uz nekoliko pojednostavljenja:
  - Nećemo definisati metod `f` zbog jednostavnosti izračunavanja novih elemenata
  - Nećemo koristiti niz početnih vrednosti, nego ćemo početne vrednosti odmah staviti u niz `f`
  - Izbacićemo neke nepotrebne dodele promenljivoj `ok`, jer ćemo povratnom vrednošću `-1` signalizirati grešku
  - Elementi niza biće tipa `int`
- Dobija se sledeći metod

Fibonačijevi brojevi

```
static int fibonacci(int n) {
  final int GRANICA = 50;
  int[] f = new int[3];
  boolean ok = true;
  if (0 <= n && n <= GRANICA) {
    f[0] = 0;
    f[1] = 1;
    f[2] = f[1] + f[0];
    if (n <= 2)
      return f[n];
    else {
      int i = 2;
      do {
        f[0] = f[1];
        f[1] = f[2];
        ok = Integer.MAX_VALUE - f[1] > f[0];
        if (ok) f[2] = f[1] + f[0];
        i++;
      } while (i < n && ok);
      if (ok) return f[2];
    }
  }
  return -1;
}
```

Fibonačijevi brojevi

- U slučaju Fibonačijevih brojeva izračunavanje može da se organizuje malo efikasnije koristeći činjenicu da je veza među njima jednostavna
- Naime, ako se sabiranje vrši i u `f[0]` i u `f[1]`, tako da u `f[0]` budu Fibonačijevi brojevi sa parnim indeksima, a u `f[1]` sa neparnim, tada se korak u petlji može povećavati za dva, što će skratiti broj prolazaka kroz petlju na pola
- Takođe, ne mora se koristiti niz, nego obične promenljive
- Radi jednostavnosti, eliminisaćemo i proveru gornje granice

Fibonačijevi brojevi

```java
static int fibonacci1(int n) {
  int f0, f1;
  boolean ok = true;
  if (0 <= n) {
    f0 = 0;
    f1 = 1;
    int i = 1;
    while (i < n && ok) {
      ok = Integer.MAX_VALUE - f1 - f1 > f0;
      // dva puta oduzimamo f1 jer cemo ga dva puta dodati
      if (ok) {
        f0 = f0 + f1;
        f1 = f0 + f1;
        i += 2;
      }
    }
    if (ok)
      if (n % 2 == 1) return f1;
      else return f0;
  }
  return -1;
}
```

Fibonačijevi brojevi

- Fibonačijevi brojevi mogu da se izračunavaju i rekurzivno, direktnim korišćenjem definicije:

```
static int fibonacci2(int n) {
  if (0 <= n)
    return fib(n);
  else
    return -1;
}
static int fib(int n) {
  if (n <= 1)
    return n;
  else
    return fib(n-1) + fib(n-2);
}
```

Fibonačijevi brojevi

- Prethodni način izračunavanja je vrlo neefikasan, jer se rekurzivnim pozivima metoda `fib` prethodni elementi rekurentnog niza nepotrebno izračunavaju više puta
- Npr. za izračunavanje $f_{25} = 75025$ izvršava se 242785 poziva metoda `fib`, dok se iterativnim postupkom u metodi `fibonacci1` koristi samo 26 sabiranja (ne računajući proveru prekoračenja opsega i povećanje brojača)
- Korišćenjem tehnike akumulirajućih parametara moguće je rekurzivno rešenje po efikasnosti, a i po prirodi, približiti iterativnom
  - Za računanje f_n biće potrebno samo n+1 poziva metoda
  - Metodom `fib` ćemo u stvari simulirati `while` petlju, jer će u svakoj "iteraciji" (rekurzivnom pozivu) parametar `f1` dobiti vrednost `f0 + f1`, parametar `f0` će dobiti vrednost `f1`, a poslednji parametar će igrati ulogu brojača

Fibonačijevi brojevi

```
static int fibonacci3(int n) {
  if (0 <= n)
    return fib(1, 0, n);
  else
    return -1;
}

static int fib(int f1, int f0, int n) {
  if (n == 0)
    return f0;
  else
    return fib(f0+f1, f1, n-1);
}
```

Fibonačijevi brojevi

- Jedan od najefikasnijih načina za izračunavanje Fibonačijevih brojeva je da se iskoristi činjenica da se Fibonačijevi brojevi dobijaju kao elementi stepena matrice *F*:

$$F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, F_n = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix}$$

  što se lako dokazuje matematičkom indukcijom

- Za $k = 1$ tvrđenje je trivijalno tačno. Neka je tačno za $k = n$, tada je:

$$F_{n+1} = F_n F = \begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} f_n & f_{n-1} + f_n \\ f_{n+1} & f_n + f_{n+1} \end{bmatrix}$$

  pa je

$$F_{n+1} = \begin{bmatrix} f_n & f_{n+1} \\ f_{n+1} & f_{n+2} \end{bmatrix}$$

Fibonačijevi brojevi

- Posmatrajmo matricu $G = F^2$, i raspišimo rekurentni niz sa elementima $G^k = (F^2)^k$. Dobija se:

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, G = \begin{bmatrix} f_1 & f_2 \\ f_2 & f_3 \end{bmatrix}, G^2 = \begin{bmatrix} f_3 & f_4 \\ f_4 & f_5 \end{bmatrix}, G^3 = \begin{bmatrix} f_5 & f_6 \\ f_6 & f_7 \end{bmatrix}, \ldots$$

  gde je matrica G definisana kao:

$$G = F^2 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

- Uočimo da su elementi donje vrste matrice G^k jednaki f_{2k} i f_{2k+1}
- Za izračunavanje stepena matrice G nije potrebno pamtiti sva 4 elementa, dovoljna su dva, npr. elementi donje vrste $g_1 = g_{21}$ i $g_2 = g_{22}$ jer:
  - $g_{12} = g_{21}$ (matrica je simetrična)
  - $g_{11} = g_{22} - g_{21} = g_2 - g_1$ (elementi matrice su Fibonačijevi brojevi)

Fibonačijevi brojevi

- Za efikasno stepenovanje matrice G koristićemo postupak uzastopnog kvadriranja, pa treba uzračunati vrednosti elemenata donje vrste kvadrata matrice G^k, odnosno matrice G^kG^k

- Pošto smo izrazili elemente matrice G^k preko g_1 i g_2, imamo:

$$\begin{bmatrix} g_2 - g_1 & g_1 \\ g_1 & g_2 \end{bmatrix}^2 = \begin{bmatrix} 2{g_1}^2 - 2g_1g_2 + {g_2}^2 & 2g_1g_2 - {g_1}^2 \\ 2g_1g_2 - {g_1}^2 & {g_1}^2 + {g_2}^2 \end{bmatrix}$$

- Ako se uvede pomoćna promenljiva `temp`, tada se nove vrednosti g_1 i g_2 izračunavaju iz starih sledećim programskim fragmentom:

```
temp = g1*g1;
g1 = 2*g1*g2 - temp;
g2 = g2*g2 + temp;
```

Fibonačijevi brojevi

- Analogan postupak se primenjuje na izračunavanje matrice *P*, u kojoj se akumulira proizvod
- Pamte se samo elementi donje vrste matrice *P*, označimo ih p_1 i p_2
- Proizvod matrica *P* i *G* izračunava se na uobičajen način (vrsta puta kolona), s tim da se elementi prve vrste matrice *P* ne izračunavaju (pa ih obeležavamo sa *)

$$\begin{bmatrix} * & * \\ p_1 & p_2 \end{bmatrix} \begin{bmatrix} g_2 - g_1 & g_1 \\ g_1 & g_2 \end{bmatrix} = \begin{bmatrix} * & * \\ p_1 g_2 - p_1 g_1 + p_2 g_1 & p_1 g_1 + p_2 g_2 \end{bmatrix}$$

- Koristeći pomoćnu promenljivu `temp`, u kojoj se čuva $p_1 g_1$, proizvod matrica *P* i *G* se izračunava sledećim programskim fragmentom:

```
temp = p1*g1;
p1 = p1*g2 + p2*g1 - temp;
p2 = p2*g2 + temp;
```

Fibonačijevi brojevi

```java
static int fibonacci4(int n) {
  if (0 <= n) {
    int p1 = 0, p2 = 1, g1 = 1, g2 = 2, temp;
    boolean izbor = (n % 2 == 1);
    n /= 2;
    while (n > 0) {
      if (n % 2 == 1) { // P = P * G
        temp = p1*g1;
        p1 = p1*g2 + p2*g1 - temp;
        p2 = p2*g2 + temp;
      }
      n /= 2;
      if (n > 0) { // G = G * G
        temp = g1*g1;
        g1 = 2*g1*g2 - temp;
        g2 = g2*g2 + temp;
      }
    }
    if (izbor) return p2; else return p1;
  }
  return -1;
}
```