

Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



Dinamično programiranje

osnove

Andrej Brodnik



Pregled

- **rekurzija** (*recursion*)
- pomnenje ali memoizacija (memoization)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve





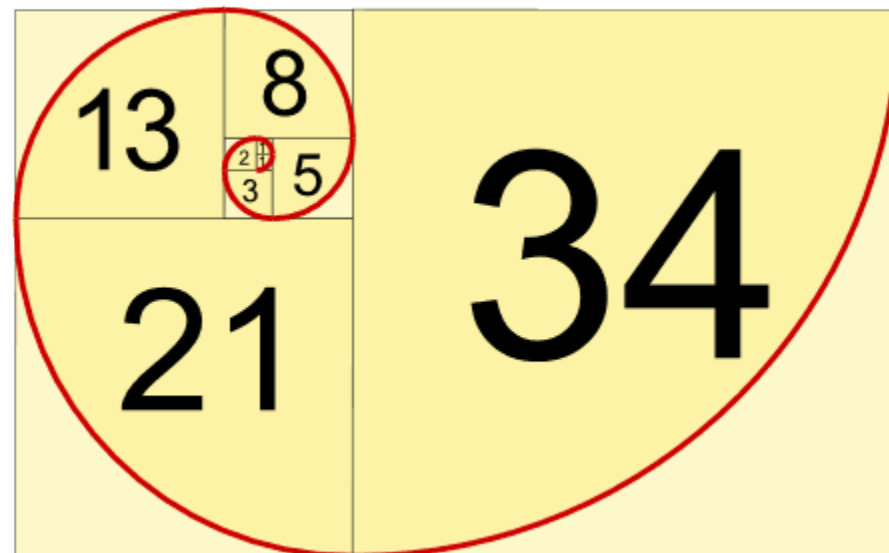
Fibonaccijeva števila

$F(n) = 1$, če $n=1$ ali 2

$F(n-1)+F(n-2)$, sicer

$F(n) = \{1, 1, 2, 3, 5, \dots\}$

In koliko je $F(64)$?





Fibonaccijsva števil

$F(n) = 1,$ če $n=1$ ali 2

$F(n-1)+F(n-2),$ sicer

```
int Fib (int n) {  
    if ((n == 1) || (n == 2)) return 1;  
    return Fib(n-1) + Fib(n-2);  
}
```



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1**: množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2**: Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve



Fibonaccijsva števila – pomnenje

```
fib_stevila[*] = nedefinirano;
```

```
int Fib (int n) {
```

```
    if fib_stevila[n] == nedefinirano {
```

```
        if ((n == 1) || (n == 2)) result = 1;
```

```
        result = Fib(n-1) + Fib(n-2);
```

```
        fib_stevila[n] = result;
```

```
    }
```

```
    return fib_stevila[n];
```

```
}
```



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve



Množenje matrik

Recimo, da moramo množiti matriki A in B, kjer je A dimenzije $a \times b$ in B dimenzije $b \times c$; rezultat je v matriki R ($a \times c$):

```
int Produkt (A, B) {  
    for i= 1 ... a  
        for j= 1 ... c  
            R[i,j]= 0;  
            for k= 1 ... b R[i,j]+= A[i,k]*B[k,j]  
    return R;  
}
```



```
int Produkt (A, B) {  
    for i= 1 ... a  
        for j= 1 ... c  
            C[i,j]= 0;  
            for k= 1 ... b C[i,j]+= A[i,k]*B[k,j]  
    return C;  
}
```

Množenje matrik

- Če natančno preštejemo, potrebujemo $a*b*c$ množenj.
- Recimo: A: 2×6 , B: 6×4 in C: 4×5
 - za $A \times B \Rightarrow 48$ množenj
 - za $B \times C \Rightarrow 120$ množenj
 - kaj pa $A \times B \times C$?



Množenje matrik

- Recimo: $A: 2 \times 6$, $B: 6 \times 4$ in $C: 4 \times 5$ in koliko množenj za $A \times B \times C$?
 - 1) $(A \times B) \times C \Rightarrow 48 + 40 = 88$ množenj
 - 2) $A \times (B \times C) \Rightarrow 120 + 60 = 180$ množenj
- Kaj pa v splošnem?



Množenje matrik – problem

Imamo:

- n matrik: $M_1, M_2, M_3, \dots, M_n$
- dimenzija matrike M_i je $d_{i-1} \times d_i$

Problem:

- kakšno naj bo zaporedje množenja matrik, da bomo opravili najmanj operacij?

Množenje matrik – formalneje

Imamo:

- n matrik: $M_1, M_2, M_3, \dots, M_n$
- dimenzija matrike M_i je $d_{i-1} \times d_i$

Opomba:

- definirajmo $c(i, j)$, kot ceno množenja j matrik od i . naprej (vključno)
- zanima nas najti čim manjši $c(1, n)$



Množenje matrik – premislek

Rešitev:

- recimo, postavimo oklepaje: $(M_1 \times M_2 \times \dots \times M_i) \times (M_{i+1}, \dots, M_n)$
- potem cena: $c(1, n) = c(1, i) + c(i+1, n-i) + d_0 * d_i * d_n$
- seveda: $c(i, 1) = 0$ za vsak $i = 1 \dots n$
- na koncu nas zanima, pri katerem i bo $c(1, n)$ najmanjši?



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve

Množenje matrik – opomba

- rešitev problema $P(1, i)$ je povsem neodvisna od rešitve problema $P(i, n-i)$

- $$c(1, n) = \begin{cases} 0 & \text{if } n=0 \\ \min_{i=0..n-1} c(1, i) + c(i+1, n-i) + d_1 d_{i+1} d_n & \text{sicer} \end{cases}$$

- Kako najdemo najboljši i ?
 - Poskusimo vse možnosti.



Množenje matrik

```
int Mnozenje (int prva, stevilo) {  
    if (stevilo == 1) return 0;  
    najcenejse= oo;  
    for (i= 1; i <= stevilo; i++) {  
        tmp= Mnozenje(prva, i) +  
            Mnozenje(prva+i, stevilo-i) +  
            d[prva-1] * d[prva+i-1] * d[prva+stevilo-1];  
        if tmp < najcenejse najcenejse= tmp;  
    }  
}
```


Množenje matrik – čas

- za vsak $i = 1 \dots n$ izračunamo $c(1, i)$, $c(i+1, n-i)$ in 2 množenji

- Skupaj:
$$\begin{aligned} T(n) &= \sum_{i=1}^n (T(i) + T(n-i) + 2) \\ &= \sum_{i=1}^n T(i) + \sum_{i=1}^n T(n-i) + \sum_{i=1}^n 2 \\ &= \sum_{i=1}^n T(i) + \sum_{j=n}^1 T(j) + 2n \\ &= 2 \sum_{i=1}^n T(i) + 2n = O(2^n) \end{aligned}$$

- kaj je narobe?



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve

Množenje matrik – čas

- recimo, da množimo matrike $M_1 \times M_2 \times M_3 \times M_4$
- potem štejemo število operacij za naslednja množenja:
 - $M_1 \times (M_2 \times M_3 \times M_4) \Rightarrow$
 $M_1 \times ((M_2 \times M_3) \times M_4), M_1 \times (M_2 \times (M_3 \times M_4))$
 - $(M_1 \times M_2 \times M_3) \times M_4 \Rightarrow$
 $((M_1 \times M_2) \times M_3) \times M_4, (M_1 \times (M_2 \times M_3)) \times M_4$
- večkrat naračunavamo (optimalno) ceno množenja istih podmatrik



Množenje matrik

```
int Mnozenje (int prva, stevilo) {  
    if (stevilo == 1) return 0;  
    najcenejse= oo;  
    for (i= 1; i <= stevilo; i++) {  
        tmp= Mnozenje(prva, i) +  
            Mnozenje(prva+i, stevilo-i) +  
            d[prva-1] * d[prva+i-1] * d[prva+stevilo-1];  
        if tmp < najcenejse najcenejse= tmp;  
    }  
}
```

Množenje matrik – s pomnjenjem

```
c[prva,stevil0]= oo; // začetna nastavitvev
int Mnozenje (int prva, stevil0) {
    if c[prva, stevil0]= oo {
        if (stevil0 == 1) c[prva, stevil0]= 0;
        else {
            for (i= 1; i <= stevil0; i++) {
                tmp= Mnozenje(prva, i) +
                    Mnozenje(prva+i, stevil0-i) +
                    d[prva-1] * d[prva+i-1] * d[prva+stevil0-1];
                if tmp < c[prva, stevil0] c[prva, stevil0]= tmp;
            }
        }
    }
    return c[prva, stevil0];
}
```



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve



Množenje matrik – primer

- recimo, da množimo matrike $M_1 \times M_2 \times M_3 \times M_4$, velikosti: $d[0...4] = (3, 6, 2, 4, 5)$
- naračunali bomo matriko $c[i,j]$, kjer $i = 1 \dots 4$, in $j = 1 \dots 4-i$

prva->	1	2	3	4
1				
2				
3				
4				

Množenje matrik – primer

- matrike $M_1 \times M_2 \times M_3 \times M_4$, velikosti: $d[0...4] = (3, 6, 2, 4, 5)$
 - $M_1 \times (M_2 \times M_3 \times M_4) \Rightarrow M_1 \times ((M_2 \times M_3) \times M_4)$, $M_1 \times (M_2 \times (M_3 \times M_4))$
 - $(M_1 \times M_2 \times M_3) \times M_4 \Rightarrow ((M_1 \times M_2) \times M_3) \times M_4$, $(M_1 \times (M_2 \times M_3)) \times M_4$

prva->	1	2	3	4
1	0	0	0	0
2		0+0+48		
3		48+0+120		
4				

Množenje matrik – primer

- matrike $M_1 \times M_2 \times M_3 \times M_4$, velikosti: $d[0...4] = (3, 6, 2, 4, 5)$
 - $M_1 \times (M_2 \times M_3 \times M_4) \Rightarrow M_1 \times ((M_2 \times M_3) \times M_4)$, $M_1 \times (\underline{M_2 \times M_3} \times \underline{M_4})$
 - $(M_1 \times M_2 \times M_3) \times M_4 \Rightarrow ((M_1 \times M_2) \times M_3) \times M_4$, $(M_1 \times (M_2 \times M_3)) \times M_4$

prva->	1	2	3	4
1	0	0	0	0
2		0+0+48	0+0+40	
3		48+0+120		
4				

0+40+60



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve



Računamo $c[i,j]$ drugače – 1. korak

- Zagotovo:

$$c[\text{prva}, 1] = 0$$

za $\text{prva} = 1 \dots n$

prva->	1	2	3	4
1	0	0	0	0
2				
3				
4				

Računamo $c[i,j]$ drugače – 2. korak

- Poleg tega:

$$c[\text{prva}, 2] = c[\text{prva}, 1] + c[\text{prva}+1, 1] + d[\text{prva}-1] * d[\text{prva}] * d[\text{prva}+1]$$

za $\text{prva} = 1 \dots n$

prva->	1	2	3	4
1	0	0	0	0
2	X	X	X	
3				
4				

Računamo $c[i,j]$ drugače – p. korak

- Sedaj pa $c[\text{prva}, p]$:

$$\min(c[\text{prva}, i] + c[\text{prva}+i, p-i] + d[\text{prva}-1]*d[\text{prva}+i-1]*d[\text{prva}+p])$$

za vse $i = 1 \dots p$

prva->	1	2	3	4
1	0	0	0	0
2	X	X	X	
3	X	X		
4	X			



Računamo $c[i,j]$ drugače

```
int Mnozenje (int stevilo) {  
    for (prva= 1; prva <= stevilo; prva++) c[prva,1]= 0;  
    for (dolzina= 2; dolzina <= stevilo; dolzina++) {  
        for (prva= 1; prva+dolzina <= stevilo; prva++) {  
            c[prva, dolzina]= najcenejša  
        }  
    }  
}
```

Čas in prostor

- **Prostor:** v obeh primerih enak in je odvisen od velikosti
 - $d[]$ - n vrednosti ter
 - $c[]$ - $n(n+1)/2 = O(n^2)$ vrednosti
- **Čas:** v obeh primerih enak in odvisen od tega kdaj napolnimo $c[1,n]$
 - za vsako polje $c[i,j]$ iščemo najboljšo vrednost, kar traja $O(n)$ korakov
 - ker je $O(n^2)$ polj v $c[i,j]$, potrebujemo $O(n^3)$ časa



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; **rekonstrukcija rešitve**
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve

Rekonstrukcija rešitve

- Doslej smo računali **koliko** stane najcenejše množenje in ne **katero** je to.
- Ko računamo $c[\text{prva}, r]$:
$$\min(c[\text{prva}, i] + c(\text{prva}+i, r-i) + d[\text{prva}-1]*d[\text{prva}+i-1]*d[\text{prva}+r])$$

za vse $i = 1 \dots r$, si zapomnimo, za kateri i je bil optimum dosežen.
- Potrebujemo dodatno polje $p[1, \text{dolžina}]$.
- MMG, *cost* in *parent*



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve

Needleman–Wunsch

- V biologiji osebke določa DNK, ki je zaporedje nukleotidov A, C, G in T
 - iz DNK se gradijo aminokisline (preko RNK), katerih število je tudi omejeno (24): 3 nukleotidi (*kodon*) določajo eno aminokislino
 - *Levenshteinova* razdalja
- Dva osebka sta si bolj v sorodu, če imata bolj podobno DNK
 - edino enojajčni dvojčki imajo enako DNK
- Koliko sta si v sorodu osebka Iztok (AACGCGG) in Ljubinica (CTAATCTTAAGCCGGGGAAGGCGC)?

Needleman–Wunsch (wikipedia)

```
for i=0 to length(A)
  F[i,0] = d*i
for j=0 to length(B)
  F[0,j] = d*j
for i=1 to length(A)
  for j=1 to length(B) {
    Match = F[i-1,j-1] + S(A[i], B[j])
    Delete = F[i-1, j] + d
    Insert = F[i, j-1] + d
    F[i,j] = max(Match, Insert, Delete)
  }
```

Čas in prostor

- **Prostor:** v obeh primerih enak in je odvisen od velikosti
 - $A[]$ in $B[]$ - n in m vrednosti ter
 - $F[]$ - $n m = O(nm)$ vrednosti
- **Čas:** v obeh primerih enak in odvisen od tega kdaj napolnimo $F[n,m]$
 - za vsako polje $F[i,j]$ iščemo najboljšo vrednost, kar traja **3 korake**
 - ker je $O(nm)$ polj v $c[i,j]$, potrebujemo **$O(nm)$** časa



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve



Needleman–Wunsch

```
for i=0 to length(A)
  F[i,0] = d*i
for j=0 to length(B)
  F[0,j] = d*j
for i=1 to length(A)
  for j=1 to length(B) {
    Match = F[i-1,j-1] + S(A[i], B[j])
    Delete = F[i-1, j] + d
    Insert = F[i, j-1] + d
    F[i,j] = max(Match, Insert, Delete)
  }
```

Čas in prostor – boljše?

- **Prostor:** v obeh primerih enak in je odvisen od velikosti
 - $A[]$ in $B[]$ - n in m vrednosti ter
 - $F[0,n]$ in $F[m,0]$ – **$O(n+m)$** vrednosti
 - pri računanju optimuma potrebujemo samo 3 vrednosti, kar pomeni, da lahko nekatere $F[i,j]$ pozabimo – pozabimo jih lahko večino, saj jih potrebujemo samo **$O(n+m)$**
- **Čas:** ostaja **$O(nm)$** , ker še vedno naračunavamo vse $F[]$



Pregled

- **rekurzija** (*recursion*)
- **pomnenje ali memoizacija** (*memoization*)
- **dinamično programiranje – primer 1:** množenje matrik
 - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2:** Needleman–Wunsch
 - osnovni prostor je $O(nm)$
 - zmanjšanje prostora na $O(n+m)$ a brez rekonstrukcije rešitve



Izzivi

- Rekonstrukcija rešitve za Needleman-Wunsch
 - koliko prostora potrebujemo
- Sprogramirajte kar smo pregledali