Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dinamično programiranje

osnove

7. 7. 2015

Andrej Brodnik

Pregled

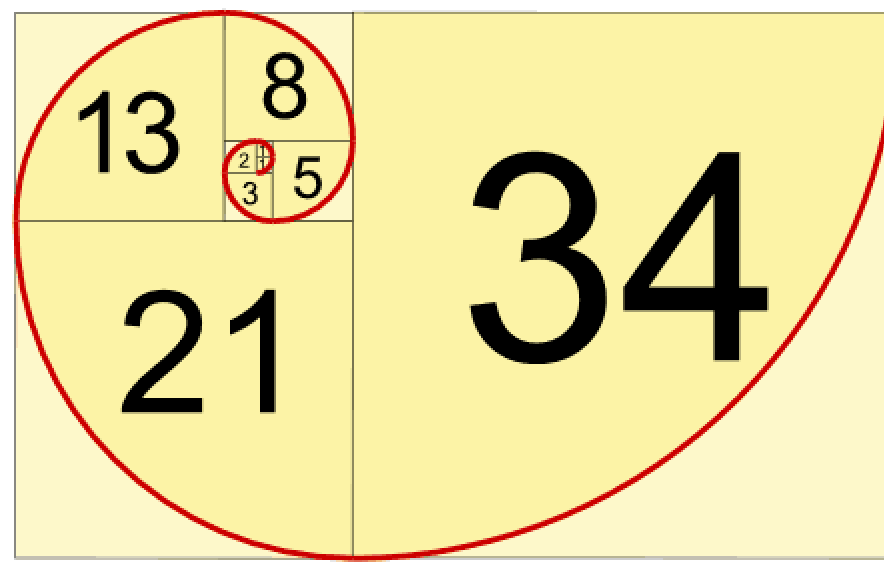
- **rekurzija** (*recursion*)
- pomnenje ali memoizacija (memoization)
- **dinamično programiranje – primer 1**: množenje matrik
  - definicija problema; osnovni program; pomnenje; kaj in kako se izračunava; program malo drugače; rekonstrukcija rešitve
- **dinamično programiranje – primer 2**: Needleman–Wunsch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

Fibonaccijeva števila

$$F(n) = \begin{cases} 1, & \text{če } n=1 \text{ ali } 2 \\ F(n-1)+F(n-2), & \text{sicer} \end{cases}$$

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

In koliko je $F(64)$?



Fibonaccijeva števila

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

Fibonaccijeva 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 x b in B dimenzije b x c; rezultat je v matriki R (a x 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;
}
```

Množenje matrik

```
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;
}
```

- Če natančno preštejemo, potrebujemo a*b*c množenj.
- Recimo: A: 2 x 6, B: 6 x 4 in C: 4 x 5
  - za A x B => 48 množenj
  - za B x C => 120 množenj
  - kaj pa A x B x C?

Množenje matrik

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

Množenje matrik – problem

Imamo:

- n matrik: $M_1, M_2, M_3, \ldots, M_n$
- dimenzija matrike M_i je d_{i-1} x 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, …, M_n
- dimenzija matrike M_i je d_{i-1} x 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: $(M_1 x\ M_2\ x \ldots x\ M_i)\ x\ (M_{i+1}, \ldots, M_n)$
- potem cena: $c(1, n) = c(1, i) + c(i+1, n\text{-}i) + d_0 * d_i * d_n$
- seveda: $c(i, 1) = 0$ za vsak $i = 1 \ldots n$
- 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)
-

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

- 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;
  }
}
```

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Množenje matrik – čas

- za vsak i= 1 … n izračunamo c(1, i), c(i+1, n-i) in 2 množenji
- Skupaj: $$T(n)=\sum_{i=1}^{n}\left(T(i)+T(n-i)+2\right)$$
$$=\sum_{i=1}^{n}T(i)+\sum_{i=1}^{n}T(n-i)+\sum_{i=1}^{n}2$$
$$=\sum_{i=1}^{n}T(i)+\sum_{j=n}^{1}T(j)+2\mathrm{n}$$
$$=2\sum_{i=1}^{n}T(i)+2\mathrm{n}=O\left(2^{n}\right)$$
- 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 x M_2 x M_3 x M_4
- potem štejemo število operacij za naslednja množenja:
  - M_1 x (M_2 x M_3 x M_4) =>

    M_1 x ((M_2 x M_3) x M_4), M_1 x (M_2 x (M_3 x M_4))
  - (M_1 x M_2 x M_3) x M_4 =>

    ((M_1 x M_2) x M_3) x M_4, (M_1 x (M_2 x M_3)) x 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 pomnenjem

```
c[prva,stevilo]= oo; // začetna nastavitev
int Mnozenje (int prva, stevilo) {
  if c[prva, stevilo]= oo {
    if (stevilo == 1) c[prva, stevilo]= 0;
    else {
      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 < c[prva, stevilo] c[prva, stevilo]= tmp;
    }
  }
  return c[prva, stevilo];
}
```

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 x M_2 x M_3 x M_4, velikosti: d[0...4]= (3, 6, 2, 4, 5)
- naračunali bomo matriko c[i,j], kjer i= 1 … 4, in j= 1 … 4-i

prva->	1	2	3	4
1				
2				
3				
4				

Množenje matrik – primer

- matrike M_1 x M_2 x M_3 x M_4, velikosti: d[0...4]= (3, 6, 2, 4, 5)
  - M_1 x (M_2 x M_3 x M_4) => M_1 x ((M_2 x M_3) x M_4), M1 x (M2 x (M3 x M4))
  - (M_1 x M_2 x M_3) x M_4 => ((M_1 x M_2) x M_3) x M_4, (M_1 x (M_2 x M_3)) x 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 x M_2 x M_3 x M_4, velikosti: d[0...4]= (3, 6, 2, 4, 5)
  - M_1 x (M_2 x M_3 x M_4) => M_1 x ((M_2 x M_3) x M_4), M_1 x (M_2 x (M_3 x M_4))
  - (M_1 x M_2 x M_3) x M_4 => ((M_1 x M_2) x M_3) x M_4, (M_1 x (M_2 x M_3)) x 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[prva,1] = 0

za prva= 1 … n

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

Računamo c[i,j] drugače – 2. korak

- Poleg tega:

  c[prva,2] = c[prva,1] + c[prva+1,1] + d[prva-1]*d[prva]*d[prva+1]

  za prva= 1 … 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[prva,p]:

  min(c[prva, i] + c(prva+i, p-i) + d[prva-1]*d[prva+i-1]*d[prva+p])

  za vse i= 1 … 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[prva,r]:

  min(c[prva, i] + c(prva+i, r-i) + d[prva-1]*d[prva+i-1]*d[prva+r])

  za vse i= 1 … r, si zapomnimo, za kateri i je bil optimum dosežen.
- Potrebujemo dodatno polje p[1,dolzina].
- 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
  - *Levenshtein*ova 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 (CTAATCTTAAGCCGGGAAGGCGC)?

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[]

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

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
- Primeri v učilnici
  - eden ne zahteva dinamičnega programiranja
  - drugi so različno zahtevni
  - dva sta v sorodu: 10684 in 507
  - eden zahteva pazljivost pri računanju (računska napaka): 1193