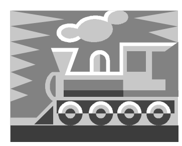
Dynamické programování 2 - další příklady

I.Kolingerová

Obsah:

1. Rekapitulace DP



- Trochu jako D&C, ale jde zdola nahoru, každý podproblém řeší jen 1x
- V tabulce uložena nejlepší dílčí řešení, kombinují se pro získání větších a větších řešení
- Nevíme, které podproblémy vyřešit, tak vyřešíme všechny
- Minimalizace úsilí důsledným využíváním dílčích výsledků

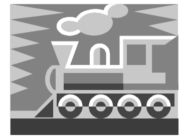
2

- Úskalí:
  - ne vždy lze zkombinovat řešení menších problémů na řešení většího podproblému
  - počet podproblémů může být příliš velký
- Charakteristické rysy problému pro DP:
  - nejlepší řešení obsahuje optimální řešení podproblémů
  - podproblémy se prolínají
- Charakteristické rysy problému pro D&C:
  - generuje nové a nové (disjunktní) podproblémy

Použitelnost DP

- Problém vhodný pro DP: princip optimality - podstatný je stav, ne způsob, jakým se do něj došlo
- Důležitá je cena operací, ne samotné operace
- Většina kombinatorických problémů
- Největší omezení: počet dílčích řešení, která je nutno sledovat - potřebujeme "zastavovací místa", implicitní pořadí prvků - pokud není splněno, exponenciální počet možných dílčích řešení - nedostupné množství paměti

Typický problém pro DP



- Dekomponovatelný do posloupnosti rozhodnutí, přijatých v různých etapách
- Každá etapa má určitý počet možných stavů
- Rozhodnutí nás převádí ze stavu v jedné etapě do stavu v další etapě
- Nejlepší posloupnost rozhodnutí v určité etapě je nezávislá na rozhodnutích v dřívějších etapách
- Cena přechodu mezi stavy v různých etapách je jasně definovaná
- Existuje rekurentní vztah pro výběr nejlepšího rozhodnutí

5

Př.1: Nejdelší společný podřetězec

Dány 2 znakové řetězce A,B délky *m* a *n*, např. A=HELLO, B=ALOHA, nejděte nejdelší spojitou sekvenci vyskytující se v obou

Řešení 1:

- Pro každý z *m* počátečních bodů A kontroluj nejdelší společný podřetězec začínající na každém z *n* počátečních bodů B
- Kontroly v průměru v $O(m)$ čase, celkem $O(m^2 n)$ čas

6

Řešení 2 užívající DP:

- Nechť $L_{i,j}$ = max.délka společných řetězců končících v A_i a B_j

- $A_i = B_j$ -> $L_{i,j} = L_{i-1,j-1} + 1$
- $A_i <> B_j$ -> $L_{i,j} = 0$

LongestCommonSubstring (A,m,B,n)

```
for i := 0 to m do L[i,0] := 0;
for j := 0 to n  do L[0,j] := 0;
len := 0; answer := <0,0>; // Délka a konec nejdelšího podřet.
for i := 1 to m do
     for j := 1 to n do begin
          if A[i]<>B[j] then L[i,j] := 0
          else
           begin
              L[i,j] := L[i-1,j-1] + 1;
              if L[i,j] > len then
                 begin len := L[i,j] ; answer := <i,j> end
           end
```

O(mn)

Příklad:

	A	L	O	H	A
H	0	0	0	1	0
E	0	0	0	0	0
L	0	1	0	0	0
L	0	1	0	0	0
O	0	0	2	0	0

```
for i := 0 to m do L[i,0] := 0;
for j := 0 to n  do L[0,j] := 0;
len := 0; answer := <0,0>; // Délka a konec nejdel. podřet.
for i := 1 to m do
    for j := 1 to n do
        begin
          if A[i]<>B[j] then L[i,j] := 0
          else
            begin
             L[i,j] := L[i-1,j-1] + 1;
             if L[i,j] > len then
                begin len := L[i,j] ; answer := <i,j> end
            end
```

U této úlohy by šlo matici ušetřit

answer = 5,3, len=2

9

Př.2: Nejdelší společná podposloupnost

Řetězec C je podposloupnost řetězce A, jestliže C lze získat rušením 0 či více symbolů z A

Př.: houseboat a ousbo

Řetězec C je společná podposloupnost řetězců A a B, jestliže C je podposloupností A i B

Př.: houseboat a computer - společná podposloupnost out (nebo oue)

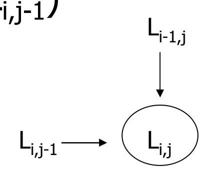
Řetězec C je nejdelší společná podposloupnost (LCS) řetězců A a B, jestliže C je společnou podposloupností A i B a neexistuje podposloupnost delší

10

Řešení užívající DP:

- Nechť $L_{i,j}$ = délka LCS v A[1..i] a B[1..j]
- $A_i = B_j$ -> $L_{i,j} = L_{i-1,j-1} + 1$
- $A_i <> B_j$ -> $L_{i,j} = \max(L_{i-1,j}, L_{i,j-1})$

znak se neshoduje,
tak zachovám nejlepší předchozí řešení



LongestCommonSubsequence (A,m,B,n)

```
for i := 0 to m do L[i,0] := 0;
for j := 0 to n  do L[0,j] := 0;
for i := 1 to m do
     for j := 1 to n do begin
         if A[i]=B[j] then
           L[i,j] := L[i-1,j-1] + 1
         else
            L[i,j] := max (L[i-1,j], L[i,j-1]);
length := L[m,n]
```

O(mn)

Příklad:

	C	O	M	P	U	T	E	R
H	0	0	0	0	0	0	0	0
O	0	1	1	1	1	1	1	1
U	0	1	1	1	2	2	2	2
S	0	1	1	1	2	2	2	2
E	0	1	1	1	2	2	3	3
B	0	1	1	1	2	2	3	3
O	0	1	1	1	2	2	3	3
A	0	1	1	1	2	2	3	3
T	0	1	1	1	2	3	3	3

```
for i := 0 to m do L[i,0] := 0;
for j := 0 to n  do L[0,j] := 0;
for i := 1 to m do
    for j := 1 to n do begin
                if A[i]=B[j] then
        L[i,j] := L[i-1,j-1] + 1
                else
        L[i,j] := max (L[i-1,j], L[i,j-1]);
length := L[m,n]
```

Max = 3, ale nevíme, které znaky -
- nutno ukládat, kde shoda

⇒OUE (nebo OUT,
pokud schovávány poslední indexy,
ne první)

Př.3: Přibližná shoda řetězců

DP pro řetězce

- Nalézt všechny výskyty slova v textu, v textu mohou být malé překlepy.
- Jak najít řetězec nejbližší k danému vzoru?
- P - vzor. řetězec, T-řetězec ve stejné abecedě
- Editační vzdálenost mezi P a T - nejmenší počet změn pro transformaci podřetězce T na P
- Povolené změny:
  1. Substituce - 2 odpovídající znaky se liší: KAT -> CAT
  2. Vložení - vložení znaku, kt. je v P, do T: CT -> CAT
  3. Rušení - zrušit v T znak, kt. není v P: CAAT -> CAT

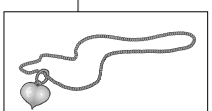
Př.: P=abcdefghijkl, T=bcdeffghixkl, P odpovídá T po 3 změnách

DP pro řetězce

- Vypadá jako obtížný problém
- Rozložit na posl. znak + zbytek, posl. znak buď odpovídá nebo jedna ze 3 změn
- Přesněji: D[i,j] - min. počet rozdílů mezi P1, P2,..Pi a částí T končící v j - cena editace prvních j znaků T na prvních i znaků P

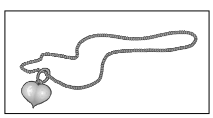
D[i,j] = min ze 3 možných způsobů rozšíření kratšího řetězce

1. if Pi=Tj then D[i-1,j-1] else D[i-1,j-1] + 1 (shoda nebo substituce)
2. D[i-1,j] + 1 (znak navíc v P, neposouváme pointer v T a platíme cenu za vkládání)
2. D[i,j-1] + 1 (znak navíc v T, neposouváme pointer v P a platíme cenu za rušení)



15

DP pro řetězce



- D[0,j] - cena shody prvních j znaků textu "se žádným znakem" vzoru
- D[0,j] = j (cena rušení, pokud shoda celého vzoru vůči celému textu),
- D[0,j] = 0 (pokud se vzor vyskytne v dlouhém textu, mělo by stát stejně, ať je blízko začátku nebo konce)
- D[i,0] = i (rušení prvních i znaků vzoru není zadarmo)
- Program tvoří D[n,m] (n = délka vzoru, m - délka textu)

16

Editační vzdálenost (P,T)

DP pro řetězce

Step 1: // Inicializace - pro plný pattern matching
        for i := 0 **to** n **do** D[i,0] = i
Step 2: **for** i := 0 **to** m **do** D[0,i] = i
Step 3: // Hlavní část - vyhodnocení rekurence
        for i := 1 **to** n **do**
            for j := 1 **to** m **do begin**
                D[i,j] := min (D[i-1,j-1] + matchcost (pi,tj),
                                D[i-1,j]+1,D[i,j-1]+1)

                end

O(mn)

17

Editační vzdálenost (P,T)

DP pro řetězce

		b	c	d	e	f	f	g	h	i	x	k	l
	0	1	2	3	4	5	6	7	8	9	10	11	12
a	1	**1**	2	3	4	5	6	7	8	9	10	11	12
b	2	**1**	2	3	4	5	6	7	8	9	10	11	12
c	3	2	**1**	2	3	4	5	6	7	8	9	10	11
d	4	3	2	**1**	2	3	4	5	6	7	8	9	10
e	5	4	3	2	**1**	2	3	4	5	6	7	8	9
f	6	5	4	3	2	**1**	**2**	3	4	5	6	7	8
g	7	6	5	4	3	2	2	**2**	3	4	5	6	7
h	8	7	6	5	4	3	3	3	**2**	3	4	5	6
i	9	8	7	6	5	4	4	4	3	**2**	3	4	5
j	10	9	8	7	6	5	5	5	4	3	**3**	4	5
k	11	10	9	8	7	6	6	6	5	4	4	**3**	4
l	12	11	10	9	8	7	7	7	6	5	5	4	**3**

Matice s cenou
pro výše uvedený
příklad,
optimum zvýrazněno

18

DP pro řetězce

- Pokud chceme najít nejlépe pasující podřetězec v textu: cena je min z posledního řádku (nejlevnější shoda vzoru končícího kdekoliv v textu)
- Rekonstrukce: z buňky s nejnižší cenou jdeme v matici nahoru a zpět, známe ceny sousedů a odpovídající znaky, takže lze rekonstruovat, kt. výběr jsme udělali pro dosažení výsledku v buňce
- Směr vlevo nebo nahoru nebo vlevo a nahoru - rušení, vkládání, substituce
- Shoda - na výběru nezáleží
- Cena rekonstrukce: *O(m+n), ale kvadr. paměť*
- Nebo: neskladovat celou matici, stačí 2 sloupce při průchodu zleva doprava, pak nejde rekonstrukce přiřazení

19

DP pro řetězce

- Alternativa: Hirschbergův rekurzivní algoritmus - D&C alg., kt. spočte přiřazení v *O(nm)* čase a *O(m)* paměti

- Aplikace přibližné shody řetězců: OCR, shoda genů, hledání plagiátů, ...
- Někdy změna znaku považována za dražší
- Pokud nepovolená, hledání nejdelší společné podsekvence

20

Př.4: Násobení matic

Najděte nejúčinnější způsob, jak vynásobit matice M_1, $M_2,\dots,M_n$, kde matice M_i má r_{i-1} řádek a r_i sloupců

Poznámky:

- Měnit pořadí nelze, není komutativní
- Sdružovat lze, je asociativní

$\Rightarrow$ vhodné pro DP

- Výsledné počty operací se mohou dramaticky lišit podle užitých závorek

21

Příklad

Násobení matic

Vynásobte $M=M_1xM_2xM_3xM_4$

pro

M_1 (10,20), M_2(20,500), M_3(50,1), M_4(1,100)

Násobení $M=M_1x(M_2xM_3xM_4)$ - 125 000 operací

$M=(M_1x(M_2xM_3))xM_4$ - 2200 operací

Exponenciální počet řazení, nutno napřed všechny n-1 násobení, pak n-2, až (n-1)! řazení

Lze pomocí DP v $O(n^3)$

22

Řešení užívající DP:

Nechť $m_{i,j}$ = min.cena výpočtu M_i x ... x M_j

$$m_{i,j} = \min_{i \le k < j} (m_{i,k} + m_{k+1,j} + r_{i-1} r_k r_j) \text{ if } i<j$$

rozdělit na násobení i..k a k+1..j,
tohle jsou ceny těchto násobení

tohle je cena násobení
těch dvou „kusů" dohromady
=
počet řádků matice i
x počet sloupců matice k
(a zároveň počet řádků k+1)
x počet sloupců matice j

MatrixMultiplication (n,r,m)

tohle je těžké (i a j jsou od sebe vzdálenost length a tu měníme odspoda)

for i := 1 **to** n **do** m[i,i] := 0;
for length := 1 **to** n-1 **do**
  for i := 1 **to** n-length **do**
    begin
    j := i + length;
    m[i,j] := min $_{i \le k < j}$ (m [i,k] +m[k+1,j]
        + r[i-1]*r[k] *r[j])
    end

počet matic.násobení (tj.součiny kolika matic zkoumáme)

O(mn)

K našemu příkladu:

- Napřed zjistíme, co stojí násobení 2 matic M_1xM_2, M_2xM_3, M_3xM_4

0			
-	0		
-	-	0	
-	-	-	0

length	i	j	k
1	1	2	1
1	2	3	2
1	3	4	3

$m_{1,2} = \min(m_{1,1} + m_{2,2} + r_0r_1r_2)$

$m_{2,3} = \min(m_{2,2} + m_{3,3} + r_1r_2r_3)$

$m_{3,4} = \min(m_{3,3} + m_{4,4} + r_2r_3r_4)$

- Pak zjistíme, co stojí násobení 3 matic $M_1xM_2xM_3$, $M_2xM_3xM_4$
- $M_1xM_2xM_3$: je lepší cesta přes $M_1x(M_2xM_3)$ nebo $(M_1xM_2)xM_3$?

0	10^4		
-	0	10^3	
-	-	0	$5x10^3$
-	-	-	0

length	i	j	k
2	1	3	1,2
2	2	4	2,3

$$m_{1,3} = \min(m_{1,1} + m_{2,3} + r_0r_1r_3,\; m_{1,2} + m_{3,3} + r_0r_2r_3) = \min(1200,10500)=1200$$

Násobení matic

- Pak zjistíme, co stojí násobení 3 matic $M_1xM_2xM_3$, $M_2xM_3xM_4$
- $M_2xM_3xM_4$: je lepší cesta přes $M_2x(M_3xM_4)$ nebo $(M_2xM_3)xM_4)$?

0	10^4	1200	
-	0	10^3	
-	-	0	$5x10^3$
-	-	-	0

length	i	j	k
2	1	3	1,2
2	2	4	2,3

$m_{2,4}$ = min ($m_{2,2} + m_{3,4} + r_1r_2r_4$,
$m_{2,3} + m_{4,4} + r_1r_3r_4$)
$m_{2,4}$ = min ($5000+100x10^3$,
$10^3 + 2x10^3$) =
= $3x10^3$

27

Násobení matic

- Nakonec zjistíme cenu násobení 4 matic (už víme, jak nejlépe vynásobit dvojice a trojice)

0	10^4	1200	
-	0	10^3	$3x10^3$
-	-	0	$5x10^3$
-	-	-	0

length	i	j	k
3	1	4	1,2,3

$m_{1,4}$ = min ($m_{1,1} + m_{2,4} + r_0r_1r_4$,
$m_{1,2} + m_{3,4} + r_0r_2r_4$,
$m_{1,3} + m_{4,4} + r_0r_3r_4$)
$m_{1,4}$ = min ($3x10^3$+ $2x10^3$,
$10^4 + 5x10^3 + 50x10^4$,
1200+1000) = 2200

28

Násobení matic

Nejmenší cena 2200, pokud použiju $m_{1,3}$ a pro získání m_{13} použiju $m_{2,3}$

0	10^4	1200	**2200**
-	0	10^3	$3x10^3$
-	-	0	$5x10^3$
-	-	-	0

Nejvhodnější násobení tedy:

$M_1x(M_2xM_3)xM_4$

Kontrola:

M(10,20)xM(20,50)xM(50,1)x M(1,100)=M(10,20)xM(20,1)x M(1,100)=M(10,1)xM(1,100) =M(10,100)

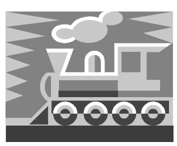
1000 operací

200 operací

1000 operací

29

∑ 2200 operací



Dynamické programování

- Závěr k DP: náhrada heuristiky globálním optimem obvykle přinese malé zlepšení za cenu většího času, nutno zvážit podle aplikace, zda se vyplatí
- Jsou problémy, kde DP nestačí (většina globálních extrémů ☹), nutno pak zůstat u heuristiky (často založené na greedy, simulovaném žíhání, genetických algoritmech ...)

30