Algoritmer, datastrukturer och komplexitet

Övning 4

Anton Grensjö
grensjo@csc.kth.se

25 september 2015

Kursplanering

F9: Dynamisk programmering

Ö3: Dekomposition och DP

F10: Dynamisk programmering

F11: Korrekthet

F12: Grafer: MST och Dijkstra

Ö4: Dynamisk programmering

F13: Grafer: Maximalt flöde

Idag

- Mer om dynamisk programmering
- Inlämning och redovisning av labbteori 2

Dynamisk programmering

- Vanliga användningsområden:
  - Optimeringsproblem
  - Kombinatorik (räkna antalet av något)
- Kan användas för problem som:
  - Har optimala delstrukturer (dvs lösningen kan uttyckas rekursivt)
  - Har överlappande delproblem
- Generell strategi:
  1. Beskriv problemets lösning med en matematisk rekursion.
     - För optimeringsproblem: hitta strukturen hos en optimal lösning, och uttryck det i en rekursion.
  2. Konstruera basfall till rekursionen.
  3. Hitta en lämplig beräkningsordning.

Uppgift 1: Träskvandring

- Tina ska gå från vänster till höger genom ett träsk, i form av ett $n \times n$-rutmönster.
- Tina kan ta steg rakt åt höger, eller snett uppåt/nedåt åt höger.
- Det är olika jobbigt att gå på olika ställen. Att hamna på ruta (i, j) kostar arbetet $A[i][j] \in \mathbb{Z}^+$.

Beskriv en algoritm som hittar det minimala arbetet att ta sig igenom träsket, givet att:

a) Tina får starta var som helst på vänsterkanten och sluta var som helst på högerkanten.

Uppgift 1: Träskvandring

- Kan vi dela in problemet i delproblem?
  - Vad är den optimala vägen till ruta (i, j)?
  - Måste bero av den optimala vägen till $(i-1, j-1)$, $(i, j-1)$ och $(i+1, j-1)$.
  - Om vi känner till dessa, hur kan vi beräkna den optimala vägen till (i, j)?
- Varför är det lämpligt att använda DP?
  - Det finns en optimal väg till varje ruta.
  - Vägarna överlappar.
- Det finns inga cykler $\implies$ det är möjligt att hitta en lämplig beräkningsordning.

Uppgift 1: Träskvandring

- Indata: träskstorleken n och kostnaden $A[i,j]$ för att hamna på ruta (i,j).
- Låt $W[i,j]$ beteckna det minimala arbetet för att ta sig till ruta (i,j).

1. Basfall
   - $W[i,1] = A[i,1], \quad 1 \leq i \leq n$
2. Rekursion

$$W[i,j] = \begin{cases} \infty & \text{om } (i,j) \text{ ligger utanför träsket} \\ A[i,1] & \text{om } j = 1 \\ \min\left(W[i-1,j-1], W[i,j-1], W[i+1,j-1]\right) + A[i,j] & \\ & \text{annars} \end{cases}$$

3. Beräkningsordning? Kolumnvis! T.ex. uppifrån och ner.

- Givet W, hur beräknar vi svaret?

Uppgift 1: Träskvandring

Pseudokod för a)

for $j \leftarrow 1$ **to** n
    $W[1,j] = A[1,j]$

for $j \leftarrow 2$ **to** n
    for $i \leftarrow 1$ **to** n
        $fromabove = W[i-1,j-1]$ **if** $i > 1$, **else** ∞
        $fromsame = W[i,j-1]$
        $frombelow = W[i+1,j-1]$ **if** $i < n$, **else** ∞
        $W[i,j] = \min(fromabove, fromsame, frombelow) + A[i,j]$

$opt \leftarrow \infty$
for $i \leftarrow 1$ **to** n
    if $W[i,n] < opt$ **then**
        $opt \leftarrow W[i,n]$
return opt

Uppgift 1: Träskvandring

Beskriv en algoritm som hittar det minimala arbetet att ta sig igenom träsket, givet att:

b) Tina startar alltid i rutan $(n/2, 1)$ (men får sluta var som helst på högerkanten).

- Vad behöver ändras?
- Ändra basfallen!
- Låt $W[n/2, 1] = A[n/2, 1]$, men $W[i, 1] = \infty$ för $i \neq n/2$.

Uppgift 1: Träskvandring

Beskriv en algoritm som hittar det minimala arbetet att ta sig igenom träsket, givet att:

c) Tina startar alltid i rutan $(n/2, 1)$ och slutar alltid i rutan $(n/2, n)$.

- Vad behöver ändras?
- Tidigare beräknade vi svaret som det minsta av alla värden på högerkanten i W.
- Nu bryr vi oss bara om $W[n/2, n]$. Returnera denna.

Uppgift 1: Träskvandring

d) Skriv ut den minst jobbiga väg Tina kan ta i c).

- Backtracking!
- Finns i detta fall två olika sätt.
- Den mest generella:
  - Skapa en till matris med “bakåtpekare”, som för varje ruta säger vilken ruta man kom dit ifrån.
  - När vi fyller i $W[i, j]$ så kollar vi vilken ruta det är bäst att komma från och väljer den. Spara då samtidigt information om vilken vi valde.
  - För att hitta vägen, börja från slutpositionen och följ pekarna tillbaka till startpositionen!
- Notera att det kan finnas flera optimala vägar, men det räcker att vi hittar en.

Uppgift 1: Träskvandring

d) Skriv ut den minst jobbiga väg Tina kan ta i c).

- Backtracking!
- Finns i detta fall två olika metoder.
- Alternativt, i detta fall:
  - Denna metod går också ut på att baklänges spåra vilken väg vi valde, men utan explicit lagrade pekare.
  - Antag att vi är i ruta (i,j). Om vi kom från ruta $(x,j-1)$ så måste $W[x,j-1] = W[i,j] - A[i,j]$, pga hur vår DP fungerar.
  - Titta på de tre möjliga rutorna vi kan ha kommit från och kolla vilken som stämmer.
  - Gå till den rutan och upprepa. Gör i övrigt samma som i den första metoden.
  - Vi slipper spara explicita bakåtpekare eftersom vi enkelt kan räkna ut vilken ruta vi måste ha kommit från.

Uppgift 2: Kombinera mynt och sedlar

Uppgift:

- Givet: en uppsättning mynt- och sedelslag med valörerna $v_1, \ldots, v_k$ kronor, samt ett maximalt belopp n (allt positiva heltal).
- Formulera en rekursion som för varje heltal b, $0 \leq p \leq n$ svarar på frågan
  "På hur många sätt kan man bilda summan b kr med hjälp av denna mynt- och sedeluppsättning?"

Lösning:

- Vad ser ni för struktur? Delproblem? Vad ska vår dynprogmatris säga?
- $N[b, j]$ - antalet sätt att bilda beloppet b med $v_1, \ldots, v_j$.

Uppgift 2: Kombinera mynt och sedlar

Lösning

- $N[b, j]$ - antalet sätt att bilda beloppet b med $v_1, \ldots, v_j$.
- Kan intuitivt se delproblem på två sätt:
  - Vad händer för lägre summor (dvs lägre värden på b)?
  - Vad händer om vi har färre valörer (dvs lägre värde på j)?
- Kan ni hitta en rekursiv tanke?
- Kom ihåg additionsprincipen från kombinatoriken.
- Antalet sätt att bilda summan b med de j första valörerna måste vara lika med summan av:
  - Antalet sätt att bilda den med de $j - 1$ första valörerna: $N[b, j - 1]$
  - Antalet sätt att bilda den med med de $j - 1$ första valörerna och v_j:

$$\sum_{i=0}^{\lfloor b/v_j \rfloor} N[b - i \cdot v_j, \;\; j - 1]$$

Uppgift 2: Kombinera mynt och sedlar

Lösning 1

- Basfall:
  - Vi kan bilda summan $b = 0$ på precis 1 sätt: genom att inte använda några sedlar/mynt alls! $N[0, j] = 1$.
  - Om vi bara har en valör ($j = 1$) så kan summan b bildas om och endast om $b \mod v_1 = 0$.

$$N[b,j] = \begin{cases} 1 & \text{om } b = 0 \\ 1 & \text{om } j = 1 \text{ och } b \mod v_1 = 0 \\ 0 & \text{om } j = 1 \text{ och } b \mod v_1 \neq 0 \\ \sum_{i=0}^{\lfloor b/v_j \rfloor} N[b - i \cdot v_j,\ \ j-1] & \text{om } 0 < b \leq n \end{cases}$$

- Beräkningsordning? Radvis eller kolumnvis.
- Behöver vi spara hela matrisen?

Uppgift 2: Kombinera mynt och sedlar

Lösning 2

- På hur många sätt kan vi bilda beloppet b med hjälp av de j första valörerna? Dela in i två fall!
  - Antingen använder vi inte v_j, då är antalet sätt $N[b, j-1]$...
  - ...eller så använder vi v_j, då är antalet sätt $N[b-v_j, j]$.

$$N[b,j] = \begin{cases} 1 & \text{om } b = 0 \\ 0 & \text{om } j = 0 \text{ och } b > 0 \\ N[b,\ \ j-1] & \text{om } 0 < b < v_j \text{ och } j > 0 \\ N[b-v_j,\ \ j] + N[b,j-1] & \text{om } v_j \leq b \leq n \text{ och } j > 0 \end{cases}$$

- Beräkningsordning? Kolumnvis.
- Behöver vi spara hela matrisen?

Uppgift 3: Längsta gemensamma delsträng

Exempel

Strängarna `ALGORITM` och `PLÅGORIS` har den gemensamma delsträngen `GORI`. Den längsta gemensamma delsträngen hos dessa strängar har alltså längd 4.

Notera!

I en delsträng måste tecknen ligga i en sammanhängande följd.

Uppgift:

- Konstruera en effektiv algoritm som givet två strängar $a_1 a_2 \cdots a_m$ och $b_1 b_2 \cdots b_n$ beräknar den längden hos den längsta gemensamma delsträngen.
- Algoritmen ska använda DP och gå i tid $\mathcal{O}(nm)$.

Uppgift 3: Längsta gemensamma delsträng

- Vad kan vi hitta för delproblemstruktur? Vad ska vår dynprogmatris säga?
- Låt $M[i,j]$ vara den längsta gemensamma delsträngen som slutar på position i och j i sträng a respektive b.
  - Med andra ord: låt $M[i,j]$ vara antalet bokstaver till vänster om (och inklusive) a_i som överensstämmer med lika många bokstäver till vänster om (och inklusive) b_j.
- Den längsta gemensamma delsträngen måste då vara det största talet i matrisen M.
- Basfall? Hur bör rekursionen se ut?

$$M[i,j] = \begin{cases} 0 & \text{om } i = 0 \text{ eller } j = 0, \\ M[i-1,\ j-1] + 1 & \text{om } a_i = b_j, \\ 0 & \text{annars} \end{cases}$$

Uppgift 3: Längsta gemensamma delsträng

$$M[i,j] = \begin{cases} 0 & \text{om } i = 0 \text{ eller } j = 0, \\ M[i-1,\ j-1] + 1 & \text{om } a_i = b_j, \\ 0 & \text{annars} \end{cases}$$

- Känns detta igen?
- Vi löste detta problem under förra övningen!
- Enda skillnaden är att vi nu vet att vi söker det största talet i M.

Uppgift 3: Längsta gemensamma delsträng

Pseudokod

Förra veckan:

for $j \leftarrow 0$ **to** n
    $M[0,j] \leftarrow 0$
for $i \leftarrow 1$ **to** m
    $M[i,0] \leftarrow 0$
    for $j \leftarrow 1$ **to** n
        if $a_i = b_j$ **then**
            $M[i,j] \leftarrow M[i-1,j-1]+1$
        else $M[i,j] \leftarrow 0$
return M

Uppgift 3: Längsta gemensamma delsträng

Pseudokod

Returnera endast det största talet:

$max \leftarrow 0$
for $j \leftarrow 0$ **to** n
    $M[0,j] \leftarrow 0$
for $i \leftarrow 1$ **to** m
    $M[i,0] \leftarrow 0$
    for $j \leftarrow 1$ **to** n
        if $a_i = b_j$ **then**
            $M[i,j] \leftarrow M[i-1,j-1]+1$
            if $M[i,j] > max$ **then** $max \leftarrow M[i,j]$
        else $M[i,j] \leftarrow 0$
return max

- Tidskomplexitet: $\Theta(nm)$.
- Testkör algoritmen på $a = \texttt{HACKER}, b = \texttt{ATTACK}$.

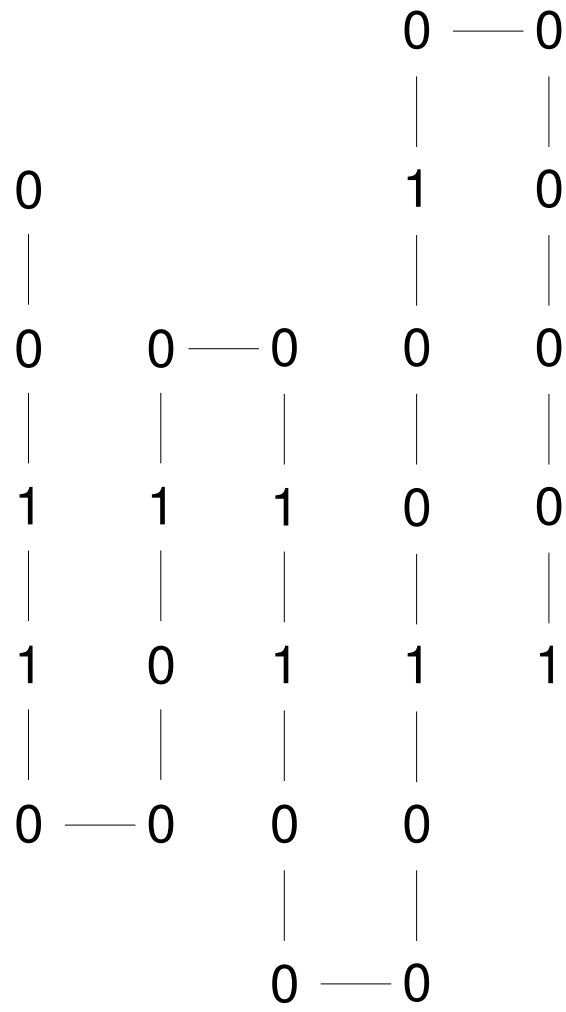
Uppgift 4: Proteinvikning

Tillämpning inom bioinformatik.

- Ett protein är en lång kedja av aminosyror.
- Enligt vår förenklade modell kan en aminosyra vara antingen hydrofil eller hydrofob.
- Säg att proteinet ligger i någon vattenbaserad lösning. De hydrofoba aminosyrorna kommer då att klumpa ihop sig, för att få så lite kontakt med vattnet som möjligt.
- Vi vill räkna ut hur proteinet kommer vika sig.
- Indata: Ett protein i form av en binär sträng. Ettor motsvarar hydrofoba aminosyror, nollor motsvara hydrofila aminosyror.

Uppgift 4: Proteinvikning

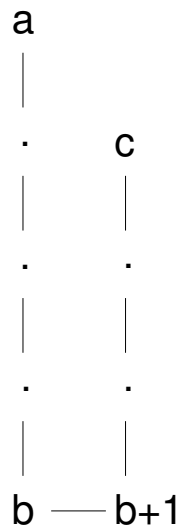
- Strängen (proteinet) ska vikas i ett tvådimensionellt gitter/rutnät.
- Mål: maximera antalet ettor som ligger intill varandra i gittret (lodrätt eller vågrätt), utan att vara närliggande i strängen.
- Problem: Konstruera en optimal dragspelsvikning av en proteinsträng av längd n.
- Dragspelsvikning: en vikning så att strängen först går en sträcka rakt uppåt, sedan rakt nedåt, osv.



Uppgift 4: Proteinvikning

- Givet: Antag att strängen lagras i `p[1..n]`. Följande algoritm beräknar vinstfunktionen för en delsträng `p[a..c]`, om den viks vid `p[b]`.

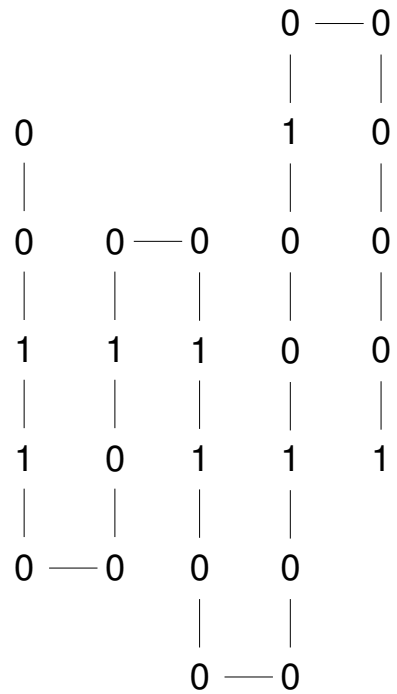
```
profit(a,b,c) =
  shortest←min(b-a,c-(b+1));
  s←0;
  for i←1 to shortest do
    if p[b-i]=1 and p[b+1+i]=1 then
      s←s+1;
  return s;
```



Uppgift 4: Proteinvikning

INMATNING: *En binär sträng med n tecken.*
PROBLEM: *Hitta den dragspelsvikning av indatasträngen som ger det största värdet på målfunktionen, alltså det största antalet par av ettor som ligger bredvid varandra men inte direkt efter varandra i strängen.*

- Hitta delstrukturer! Vad ska vi låta ett element i vår dynprogmatris vara?



Uppgift 4: Proteinvikning

- Låt $q_{a,b}$ vara det maximala värdet på målfunktionen man kan få för en vikning av suffixet `p[a..n]` av proteinet, givet att den första sträckan slutar i b.
- Rekursion:

$$q_{a,b} = \max_{b+1<c\leq n} (\texttt{profit}(a,b,c) + q_{b+1,c}).$$

- Basfall? Om vi inte viker någonstans måste viktfunktionen bli 0:

$$q_{a,n} = 0 \text{ för } 1 \leq a < n.$$

- Svaret:

$$\max_{1<b\leq n} q_{1,b}$$

- Beräkningsordning?

Uppgift 4: Pseudokod

```
for a←1 to n-1 do q[a,n]←0;
for b←n-1 downto 2 do
  for a←1 to b-1 do
    t←-1;
    for c←b+2 to n do
      v←profit(a,b,c)+q[b+1,c];
      if v>t then t←v;
    q[a,b]←t;
max←0;
for b←2 to n do
  if q[1,b]>max then max←q[1,b];
return max;
```

- Tre nästlade for-slingor.
- Anrop till profit tar $\mathcal{O}(n)$ tid.
- Tidskomplexitet: $\mathcal{O}(n^4)$.

Nästa vecka

- Grafalgoritmer
  - Minsta spännande träd
  - Maximalt flöde (viktigt inför labb 3)
  - Hitta Eulercykel
  - Bipartit matchning (också bra inför labb 3)
- Undre gränser

Mina slides finns numera att hitta under Kursinnehåll/Övningsanteckningar på kurshemsidan.