

Föreläsning 1

Giriga algoritmer och dynamisk programmering

Douglas Wikström
KTH Stockholm
`popup-help@csc.kth.se`

Dagens citat

Using the wrong algorithm to solve a problem is like trying to cut a steak with a screwdriver: you may eventually get a digestable result, but you will expend considerably more effort than necessary, and the result is unlikely to be aesthetically pleasing.

– Cormen, Leiserson and Rivest

Divide and Conquer

1. Dela upp huvudproblemet i två eller fler mindre delar.
2. Lös varje delproblem rekursivt.
3. Kombinera lösningarna för delproblemen till en lösning för huvudproblemet.

Merge Sort

Input: Lista $L = [u_1, u_2, \dots, u_n]$.

Output: Elementen i L i sorterad ordning.

MERGESORT(L)

(1) **if** $|L| = 1$

(2) **return** L

(3) **else**

(4) $L_l \leftarrow \text{MERGESORT}([u_1, \dots, u_{\lfloor n/2 \rfloor}])$

(5) $L_r \leftarrow \text{MERGESORT}([u_{\lfloor n/2 \rfloor + 1}, \dots, u_n])$

(6) **return** MERGE(L_l, L_r)

Där MERGE kombinerar två sorterade listor till en.

Disjunkta intervall (1/3)

Aktiviteter skall utföras i en föreläsningssal. Varje aktivitet har en start- och slut-tid. Schemalägg så många aktiviteter som möjligt.

Formellt: Givet är tidsintervall $[s_1, f_1), \dots, [s_n, f_n)$, dvs aktiviteter. Välj maximal delmängd av parvis disjunkta intervall.

Disjunkta intervall (2/3)

Input: Intervall $[s_1, f_1), \dots, [s_n, f_n)$.

Output: Maximal delmängd parvis disjunkta intervall.

DISJUNCTINTERVAL($[s_1, f_1), \dots, [s_n, f_n)$)

- (1) Sortera intervallen så att $f_1 \leq f_2 \leq \dots \leq f_n$.
- (2) $S \leftarrow \{1\}$, $j \leftarrow 1$
- (3) **for** $i \leftarrow 2$ **to** n
- (4) **if** $s_i \geq f_j$
- (5) $S \leftarrow S \cup \{i\}$
- (6) $j \leftarrow i$
- (7) **return** S

Disjunkta intervall (3/3)

- ▶ **Greedy Choice.** Intervallet $I_{\min}$ med den minsta slutpunkten är med i någon optimal lösning.
- ▶ **Optimal Substructure.** Optimal lösning till $\{I_1, \dots, I_n\} - I_{\min}$ kombinerad med $I_{\min}$ är optimal lösning för $\{I_1, \dots, I_n\}$.

Intervalltäckning (1/3)

I en gammal biltunnel finns ett antal lampor monterade med ojämna mellanrum. Lamporna är olika smutsiga och ger därför olika stora ljuskäglor. Dvs, varje lampa lyser upp ett bestämt avsnitt av vägen. Den miljöorienterade nyanställde funderar över vilka lampor som skall tändas så att hela tunneln är belyst med minimal elförbrukning.

Formellt: Givet är ett intervall $[s, f]$, dvs tunneln, och intervall $[s_1, f_1], \dots, [s_n, f_n]$, dvs ljuskäglornas utsträckning. Välj minimal delmängd $S \subset \{1, \dots, n\}$ så att $[s, f] \subset \bigcup_{i \in S} [s_i, f_i]$, dvs belys hela tunneln.

Intervalltäckning (2/3)

Input: Målintervall $[s, f]$ och intervall $[s_1, f_1], \dots, [s_n, f_n]$ så att $[s, f] \subset \bigcup [s_i, f_i]$.

Output: En minimal delmängd $S \subset \{1, \dots, n\}$ så att $[s, f] \subset \bigcup_{i \in S} [s_i, f_i]$.

INTERVALCOVER($[s_1, f_1], \dots, [s_n, f_n]$)

- (1) $S \leftarrow \emptyset, t \leftarrow s$
- (2) **while** $t < f$
- (3) $f_i \leftarrow \max\{f_j \mid s_j \leq t \leq f_j\}$
- (4) $S \leftarrow S \cup \{i\}$
- (5) $t \leftarrow f_i$
- (6) **return** S

Intervalltäckning (3/3)

- ▶ **Greedy Choice.** Intervallet $I_{\max}$ som innehåller s och har maximal slutpunkt $f_{\max}$ är med i någon optimal lösning.
- ▶ **Optimal Substructure.** Optimal lösning till $[f_{\max}, f]$, $\{I_1, \dots, I_n\} - I_{\max}$ kombinerad med $I_{\max}$ är optimal lösning för $\{I_1, \dots, I_n\}$.

Kontinuerliga kappsäcksproblemet (1/2)

En tjuv gör inbrott i en gruva och finner separata högar med finkornigt metallspån av olika metaller. Han har med sig en kappsäck som rymmer en viss mängd och vill stjäla till ett så stort värde som möjligt. Hur mycket skall han ta av varje metall?

Formellt: Givet är kappsäckens kapacitet och värde/vikt-par $[(v_1, w_1), \dots, (v_n, w_n)]$. Välj $[a_1, \dots, a_n]$ så att $a_i \leq w_i$ och så att $\sum_{i=1}^n a_i \frac{v_i}{w_i}$ maximeras.

Kontinuerliga kappsäcksproblemet (2/2)

Input: Kapacitet W och värde/vikt-par $[(v_1, w_1), \dots, (v_n, w_n)]$.

Output: $[a_1, \dots, a_n]$, $a_i \leq w_i$, så att $\sum_{i=1}^n a_i \frac{v_i}{w_i}$ maximeras.

CONTKNAPSACK($W, [(v_1, w_1), \dots, (v_n, w_n)]$)

- (1) Sortera värde/vikt-paren så att $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \dots \geq \frac{v_n}{w_n}$.
- (2) **for** $i = 1$ **to** n
- (3) $a_i \leftarrow \min\{W, w_i\}$
- (4) $W \leftarrow W - a_i$
- (5) **return** $[a_1, \dots, a_n]$

Huffmankodning (1/2)

Givet en tabell över hur vanligt varje tecken är i ett alfabet, hitta den prefixfria binära kod som ger kortast förväntad kodlängd. En kod är prefixfri om inget kodord är ett prefix till något annat kodord.

Formellt: Givet är tecken/vikt-par $\{(a_1, f_1), \dots, (a_n, f_n)\}$ för ett alfabet $A = \{a_1, \dots, a_n\}$. Hitta prefixfri kod $\tau : A \mapsto \{0, 1\}^*$ som minimerar $\sum_{i=1}^n f_i \cdot |\tau(a_i)|$, där $|\tau(a_i)|$ är längden av kodordet $\tau(a_i)$.

Huffmankodning (2/2)

Vi identifierar en binär prefixfri kod τ med ett 0/1-kantmärkt träd. Koderna för ett löv ges då av vägen till lövet.

Input: Tecken/vikt-par $\{(a_1, f_1), \dots, (a_n, f_n)\}$.

Output: 0/1-kantmärkt träd sådant att $\sum_{i=1}^n f_i \cdot |\tau(a_i)|$ minimeras.

HUFFMAN($\{(a_1, f_1), \dots, (a_n, f_n)\}$)

- (1) $S \leftarrow \{(a_1, f_1, a_1), \dots, (a_n, f_n, a_n)\}$
- (2) **while** $|S| \geq 2$
- (3) Hitta $(b_i, f_i, t_i), (b_j, f_j, t_j) \in S$ med minimala f_i och f_j .
- (4) $S \leftarrow S \setminus \{(b_i, f_i, t_i), (b_j, f_j, t_j)\}$
- (5) $S \leftarrow S \cup \{(a_i \circ a_j, f_i + f_j, \text{TREENODE}(t_i, t_j))\}$
- (6) **return** S

Binomialtal och Pascals triangel (1/3)

$\binom{n}{k}$ är antalet sätt att välja en k -delmängd från en mängd med n element. Vi summerar antalet sätt att välja k element där första elementet är med respektive inte med.

$$\binom{n}{k} = \begin{cases} 1 & \text{om } n = k \text{ eller } k = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{annars} \end{cases}$$

Pascals triangel:

								k
				1				0
			1		1			1
		1		2		1		2
	1		3		3		1	3
	1	4		6		4	1	4
1	5	10		10		5	1	5

Binomialtal och Pascals triangel (2/3)

Dynamisk programmering med memoisering. $T[i, j]$ är en tabell där alla element från början är satta till $\perp$. Tabellen kan återanvändas mellan anrop.

Input: Heltal n och k , $0 \leq k \leq n$.

Output: $\binom{n}{k}$

$\text{BIN}(n, k)$

- (1) **if** $T[n, k] = \perp$
- (2) **if** $n = k \vee k = 0$
- (3) $T[n, k] = 1$
- (4) **else**
- (5) $T[n, k] \leftarrow \text{BIN}(n - 1, k - 1) + \text{BIN}(n - 1, k)$
- (6) **return** $T[n, k]$

Binomialtal och Pascals triangel (3/3)

Vanlig dynamisk programmering från botten och upp.

Input: Heltal n och k , $0 \leq k \leq n$.

Output: $\binom{n}{k}$

$\text{BIN}(n, k)$

```
(1)  for  $i = 0$  to  $n$ 
(2)       $T[i, 0] \leftarrow T[i, i] \leftarrow 1$ 
(3)  for  $i = 2$  to  $n$ 
(4)      for  $j = 1$  to  $\min\{i - 1, k\}$ 
(5)           $T[i, j] \leftarrow T[i - 1, j - 1] + T[i - 1, j]$ 
(6)  return  $T[n, k]$ 
```

Maximal delsumma (1/2)

Formellt: Givet heltal $x_1, \dots, x_n$, finn $\max_{s \leq f} \left\{ \sum_{i=s}^f x_i \right\}$.

Maximal delsumma (2/2)

Input: Heltal $x_1, \dots, x_n$.

Output: $\max_{s \leq f} \{ \sum_{i=s}^f x_i \}$.

MAXSUBARRAY($x_1, \dots, x_n$)

- (1) $S[0] \leftarrow 0$
- (2) **for** $i = 1$ **to** n
- (3) $S[i] \leftarrow \max\{0, S[i-1]\} + x_i$
- (4) **return** $\max_i \{S[i]\}$

Längsta växande delsekvens (1/2)

Två kretsar har n kontakter var på rad märkta med talen $1, 2, \dots, n$ i någon ordning. Vi vill parvis förbinda så många matchande kontakter som möjligt med sladdar utan att korsa några sladdar.

Formellt: Givet en n -permutation π , finn storleken på maximal delmängd $S \subset \{1, 2, \dots, n\}$ så att $\pi(i) < \pi(j)$ för alla $i < j$ i mängden S .

Längsta växande delsekvens (2/2)

Input: Permutation π .

Output: Storleken på maximal delmängd $S \subset \{1, 2, \dots, n\}$ så att $\pi(i) < \pi(j)$ för alla $i < j$ i mängden S .

LONGINCSUBSEQ(π)

- (1) $T[0] \leftarrow -\infty$
- (2) **for** $i = 1$ **to** n
- (3) $T[i] \leftarrow \infty$
- (4) **for** $i = 1$ **to** n
- (5) Finn j så att $T[j - 1] < \pi(i) < T[j]$.
- (6) $T[j] \leftarrow \pi(i)$
- (7) **return** $\max\{j : T[j] < \infty\}$