F07 - Problemlösningsstrategier

5DV149 Datastrukturer och algoritmer

Niclas Börlin
niclas.borlin@cs.umu.se

2020-02-10 Mon

Design av algoritmer

- Problemlösningsstrategier:
  - Top-down.
  - Bottom-up.
- Typer av algoritmer (lösningstekniker)
  - *Brute force* (“råstyrka”).
  - Giriga algoritmer (*Greedy algorithms*).
  - Söndra och härska (*Divide-and-Conquer*).
  - Dynamisk programmering.
  - (Stokastiska (slumpbaserade).)
  - (*Branch-and-Bound*.)

Brute force

- Rättfram ansats: Utgå direkt från problemställningen med dess definitioner, begränsningar, etc.!
- Om problemet är kombinatoriskt: Gör en fullständig sökning!
  - Generera och enumerera alla tänkbara lösningar.
  - Kom ihåg den bästa lösningen.
- Egenskaper
  - Bra metod att starta med.
  - Garanterar en korrekt lösning om en sådan finns.
  - Garanterar inte effektivitet.
  - Ofta enkla, “naiva”, algoritmer.

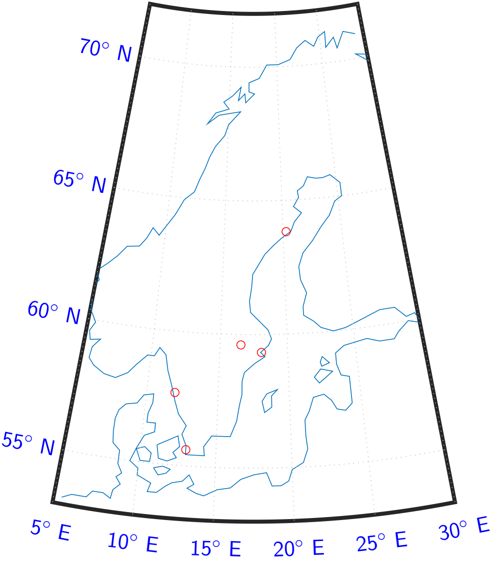
Brute force, exempel 1

- Linjär sökning: Finn det största talet talet i ett fält.
  - Gå igenom varje element. Kom ihåg det största.

```
Algorithm arrayMax(A,n)
   input: An array A storing n integers
   output: The maximum element of A
currentMax ← A[0]
for i ← 1 to n-1 do
   if currentMax < A[i] then
      currentMax ← A[i]
return currentMax
```

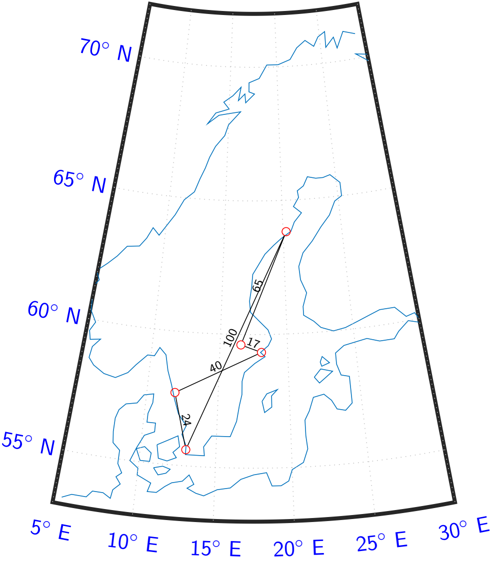
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:



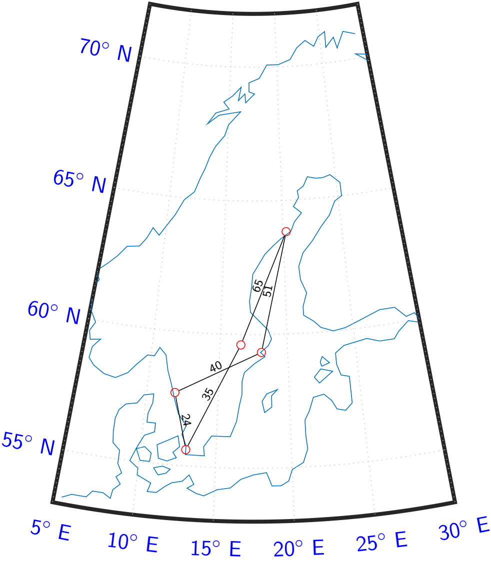
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil



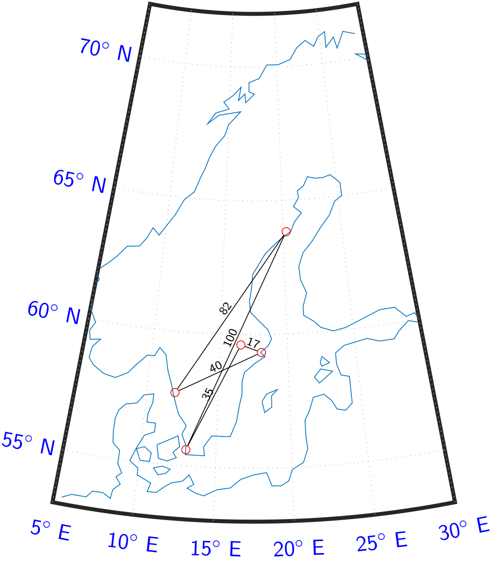
Brute force, exempel 2 — Handelsresande-problemet *(Travelling Salesman Problem — TSP)*

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil



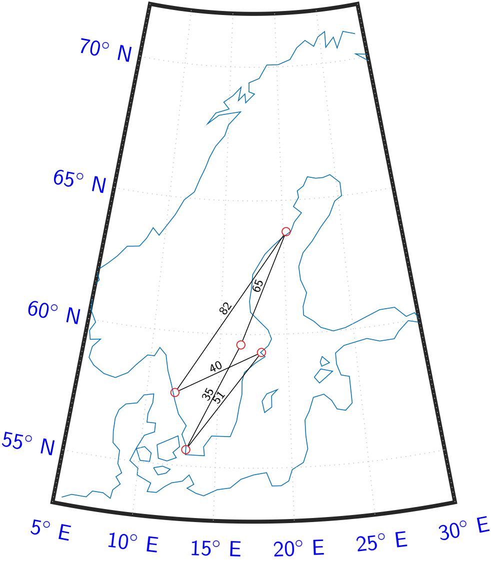
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil



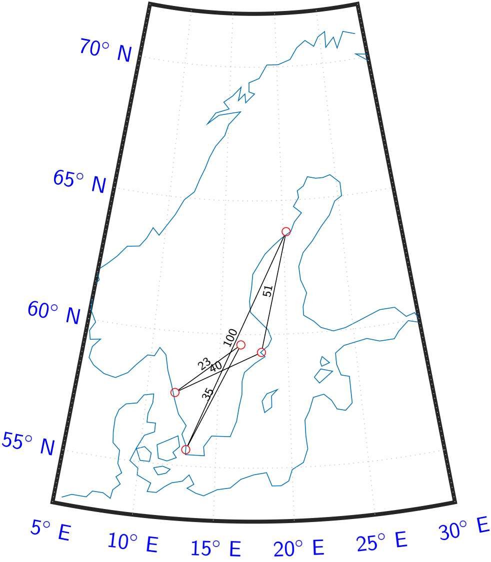
Brute force, exempel 2 — Handelsresande-problemet *(Travelling Salesman Problem — TSP)*

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil



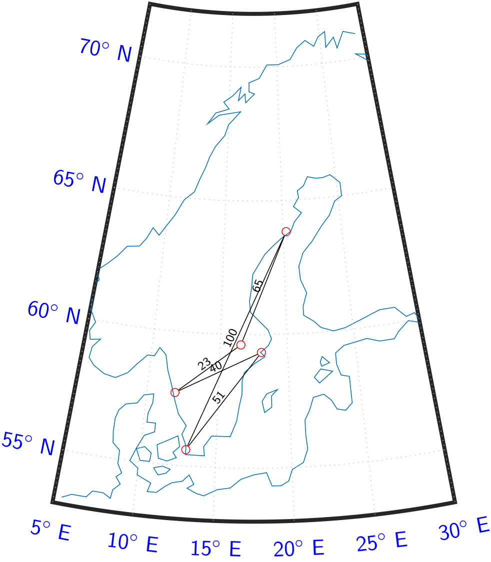
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil



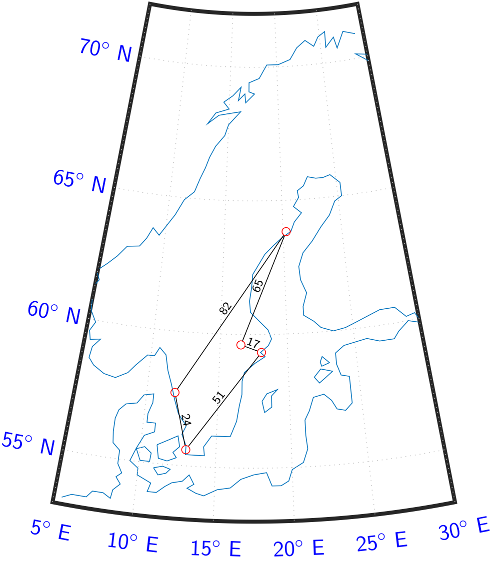
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil



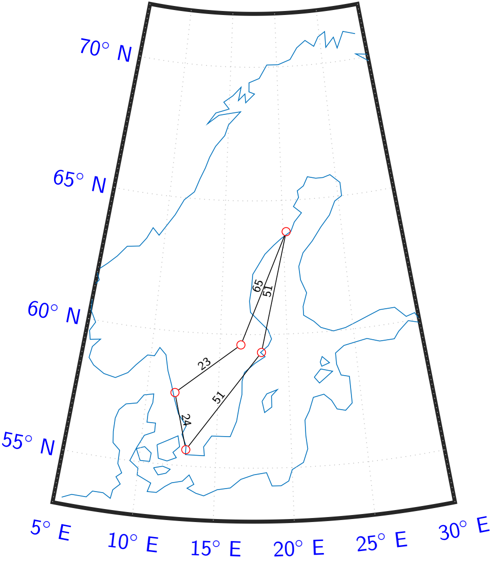
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil
  7. 1-3-2-4-5-1: 240 mil



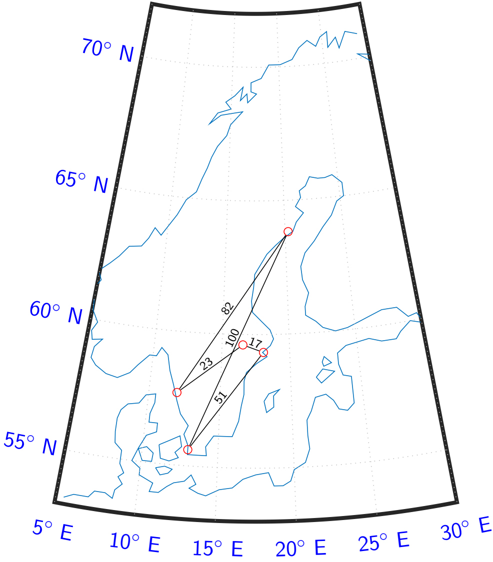
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil
  7. 1-3-2-4-5-1: 240 mil
  8. 1-3-2-5-4-1: 215 mil



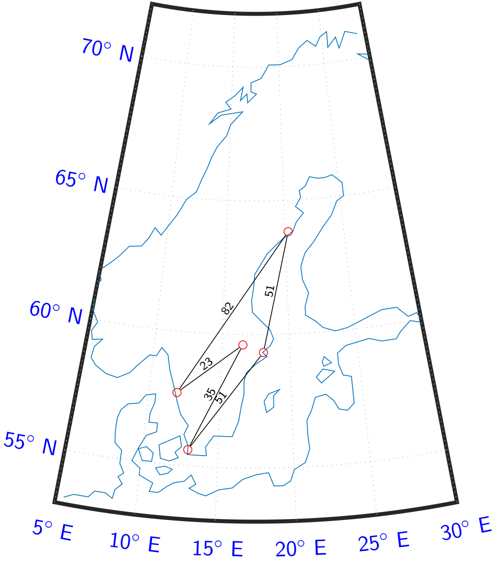
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil
  7. 1-3-2-4-5-1: 240 mil
  8. 1-3-2-5-4-1: 215 mil
  9. 1-3-4-2-5-1: 274 mil



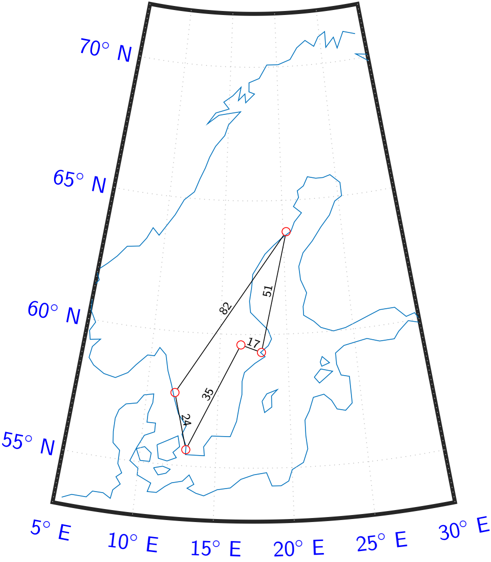
Brute force, exempel 2 — Handelsresande-problemet *(Travelling Salesman Problem — TSP)*

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil
  7. 1-3-2-4-5-1: 240 mil
  8. 1-3-2-5-4-1: 215 mil
  9. 1-3-4-2-5-1: 274 mil
  10. 1-3-5-2-4-1: 242 mil



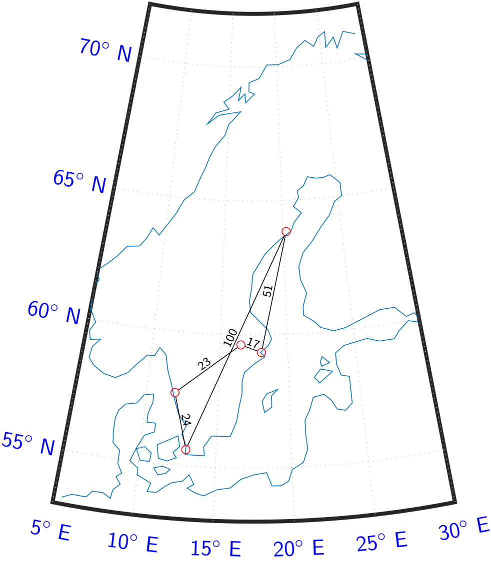
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil
  7. 1-3-2-4-5-1: 240 mil
  8. 1-3-2-5-4-1: 215 mil
  9. 1-3-4-2-5-1: 274 mil
  10. 1-3-5-2-4-1: 242 mil
  11. 1-4-2-3-5-1: 210 mil



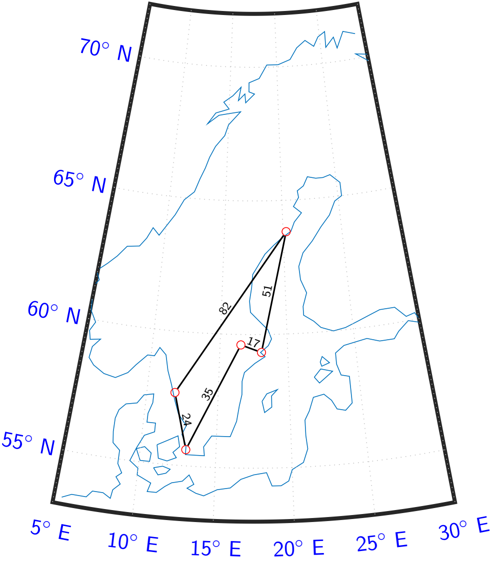
Brute force, exempel 2 — Handelsresande-problemet *(Travelling Salesman Problem — TSP)*

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil
  7. 1-3-2-4-5-1: 240 mil
  8. 1-3-2-5-4-1: 215 mil
  9. 1-3-4-2-5-1: 274 mil
  10. 1-3-5-2-4-1: 242 mil
  11. 1-4-2-3-5-1: 210 mil
  12. 1-4-3-2-5-1: 216 mil



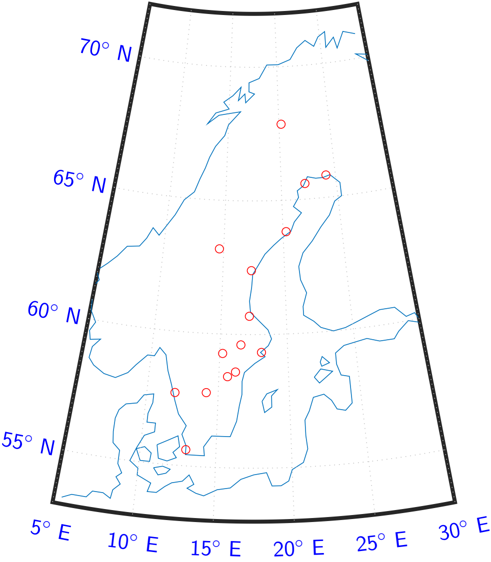
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 5$, 12 alternativ:
  1. 1-2-3-4-5-1: 247 mil
  2. 1-2-3-5-4-1: 216 mil
  3. 1-2-4-3-5-1: 274 mil
  4. 1-2-4-5-3-1: 273 mil
  5. 1-2-5-3-4-1: 249 mil
  6. 1-2-5-4-3-1: 280 mil
  7. 1-3-2-4-5-1: 240 mil
  8. 1-3-2-5-4-1: 215 mil
  9. 1-3-4-2-5-1: 274 mil
  10. 1-3-5-2-4-1: 242 mil
  11. 1-4-2-3-5-1: 210 mil
  12. 1-4-3-2-5-1: 216 mil



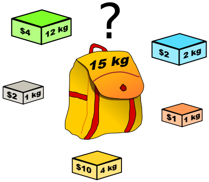
Brute force, exempel 2 — Handelsresande-problemet (*Travelling Salesman Problem — TSP*)

- Given n städer, finn den kortaste rutten som besöker varje stad exakt en gång.
- Komplexitet: $(n-1)!/2$.
- För $n = 15$: $4.4 \cdot 10^{10}$ alternativ.



Brute force, exempel 3 — *The 0-1 Knapsack Problem*

- Givet en mängd med n element där element i har värde $v_i > 0$ och en vikt $w_i > 0$:
  - *Välj element med maximalt värde utan att den totala vikten blir mer än W.*
- Låt $x_i \in \{0, 1\}$:
  - Om $x_i = 1$ så är elementet med.
  - Om $x_i = 0$ så låter vi elementet vara.

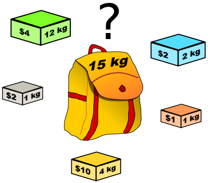


Brute force, exempel 3 — *The 0-1 Knapsack Problem*

- Matematisk formulering:

$$\max_{x_i \in \{0,1\}} \sum_{i=1}^{n} x_i v_i \text{ med begränsningen } \sum_{i=1}^{n} x_i w_i \leq W.$$

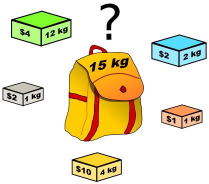
- Kombinatoriskt problem, komplexitet 2^n.



Brute force, exempel 3 — *The 0-1 Knapsack Problem*

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

Maxvikt: $W = 15$



i	x_1	x_2	x_3	x_4	x_5	$\sum x_i w_i$	$\sum x_i v_i$
0	0	0	0	0	0	0	0
1	0	0	0	0	1	4	10
2	0	0	0	1	0	1	1
3	0	0	0	1	1	5	11
4	0	0	1	0	0	1	2
5	0	0	1	0	1	5	12
6	0	0	1	1	0	2	3
7	0	0	1	1	1	6	13
8	0	1	0	0	0	2	2
9	0	1	0	0	1	6	12
10	0	1	0	1	0	3	3
11	0	1	0	1	1	7	13
12	0	1	1	0	0	3	4
13	0	1	1	0	1	7	14
14	0	1	1	1	0	4	5
15	0	1	1	1	1	8	15
16	1	0	0	0	0	12	4
17	1	0	0	0	1	16	14
18	1	0	0	1	0	13	5
19	1	0	0	1	1	17	15
20	1	0	1	0	0	13	6
21	1	0	1	0	1	17	16
22	1	0	1	1	0	14	7
23	1	0	1	1	1	18	17
24	1	1	0	0	0	14	6
25	1	1	0	0	1	18	16
26	1	1	0	1	0	15	7
27	1	1	0	1	1	19	17
28	1	1	1	0	0	15	8
29	1	1	1	0	1	19	18
30	1	1	1	1	0	16	9
31	1	1	1	1	1	20	19

Brute force, summering

- Många problem vet man inte av någon bättre lösning.
- Ger ofta hög tidskomplexitet.
- Går ofta att effektivisera de naiva algoritmerna.
  - Avbryta så fort man inser att vägen inte leder till en lösning.
  - Testa alternativen i någon speciell ordning.
  - Avslutar när vi funnit *en* lösning.
  - Avslutar när vi funnit en lösning som är nästan optimal.
- Eller så relaxerar vi problemet (släpper på någon begränsning).

Giriga (*Greedy*) algoritmer

- Metod:
  - I varje steg, titta på alla möjliga nästa steg och välj det som ger störst förbättring.
- För vissa problem kan en girig algoritm ge optimal lösning:
  - Om den optimala lösningen kan nås via stegvisa lokala förändringar av starten.
- Giriga algoritmer specialfall av heuristiska (tumregelsbaserade).
  - Tumregel: Ta så mycket så fort som möjligt!
- Bra alternativ till *brute force*-algoritmer.

Giriga algoritmer, exempel

- Problem: Lämna tillbaka växel med så få mynt som möjligt.
  - Heuristik:
    - Ta alltid det myntet med högst värde i varje iteration.
- Minimalt uppspännande träd:
  - Kruskals algoritm.
  - Prims algoritm.
- Kortaste vägen i en graf (Dijkstras algoritm).
- Huffman-kodning.

The 0-1 Knapsack Problem, girig algoritm, exempel

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

- Välj alltid det värdefullaste element som får plats!

The 0-1 Knapsack Problem, girig algoritm, exempel

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

- Välj alltid det värdefullaste element som får plats!
- Iteration 1: $x_5 = 1$, $\sum x_i v_i = 10$, $\sum x_i w_i = 4$.

The 0-1 Knapsack Problem, girig algoritm, exempel

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

- Välj alltid det värdefullaste element som får plats!
- Iteration 1: $x_5 = 1$, $\sum x_i v_i = 10$, $\sum x_i w_i = 4$.
- Iteration 2: $x_3 = 1$, $\sum x_i v_i = 12$, $\sum x_i w_i = 5$.

The 0-1 Knapsack Problem, girig algoritm, exempel

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

- Välj alltid det värdefullaste element som får plats!
- Iteration 1: $x_5 = 1$, $\sum x_i v_i = 10$, $\sum x_i w_i = 4$.
- Iteration 2: $x_3 = 1$, $\sum x_i v_i = 12$, $\sum x_i w_i = 5$.
- Iteration 3: $x_2 = 1$, $\sum x_i v_i = 14$, $\sum x_i w_i = 7$.

The 0-1 Knapsack Problem, girig algoritm, exempel

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

- Välj alltid det värdefullaste element som får plats!
- Iteration 1: $x_5 = 1$, $\sum x_i v_i = 10$, $\sum x_i w_i = 4$.
- Iteration 2: $x_3 = 1$, $\sum x_i v_i = 12$, $\sum x_i w_i = 5$.
- Iteration 3: $x_2 = 1$, $\sum x_i v_i = 14$, $\sum x_i w_i = 7$.
- Iteration 4: $x_4 = 1$, $\sum x_i v_i = 15$, $\sum x_i w_i = 8$.

The 0-1 Knapsack Problem, girig algoritm, exempel

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

- Välj alltid det värdefullaste element som får plats!
- Iteration 1: $x_5 = 1$, $\sum x_i v_i = 10$, $\sum x_i w_i = 4$.
- Iteration 2: $x_3 = 1$, $\sum x_i v_i = 12$, $\sum x_i w_i = 5$.
- Iteration 3: $x_2 = 1$, $\sum x_i v_i = 14$, $\sum x_i w_i = 7$.
- Iteration 4: $x_4 = 1$, $\sum x_i v_i = 15$, $\sum x_i w_i = 8$.
- Inget till element får plats: klara!

The 0-1 Knapsack Problem, girig algoritm, exempel

Element	Värde	Vikt
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

- Välj alltid det värdefullaste element som får plats!
- Iteration 1: $x_5 = 1$, $\sum x_i v_i = 10$, $\sum x_i w_i = 4$.
- Iteration 2: $x_3 = 1$, $\sum x_i v_i = 12$, $\sum x_i w_i = 5$.
- Iteration 3: $x_2 = 1$, $\sum x_i v_i = 14$, $\sum x_i w_i = 7$.
- Iteration 4: $x_4 = 1$, $\sum x_i v_i = 15$, $\sum x_i w_i = 8$.
- Inget till element får plats: klara!
- Optimal lösning på $O(n)$ tid! Eller?

Relaxering — *The Fractional Knapsack Problem*

- Samma som *The 0-1 Knapsack Problem*, men vi får ta en *del* av varje element.
- Låt $x_i \in [0, 1]$ vara *andelen* vi tar av element i.
- Matematisk formulering:

$$\max_{x_i \in [0,1]} \sum_{i=1}^{n} x_i v_i \text{ med begränsningen } \sum_{i=1}^{n} x_i w_i \leq W.$$

- I varje iteration, ta så mycket som möjligt av det element som har högst *värde per viktenhet* v_i / w_i.
- Kan lösas i $O(n \log n)$ tid (sortering).

The Fractional Knapsack Problem, exempel

Element	Värde	Vikt	Värde/vikt
5	10	4	2.5
3	2	1	2
2	2	2	1
4	1	1	1
1	4	12	0.33

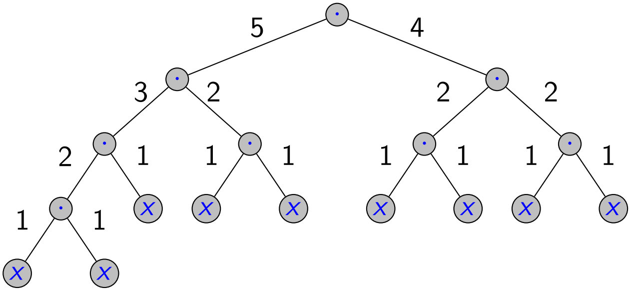
- Iteration 1: $x_5 = 1$, $\sum x_i w_i = 4$.
- Iteration 2: $x_3 = 1$, $\sum x_i w_i = 5$.
- Iteration 3: $x_2 = 1$, $\sum x_i w_i = 7$.
- Iteration 4: $x_4 = 1$, $\sum x_i w_i = 8$.
- Iteration 5: $x_1 = \frac{7}{12}$, $\sum x_i w_i = 15$.

Söndra och härska (*Divide-and-Conquer*)

- Metod:
  - Söndra: Dela upp problemet i två eller flera delar som löses rekursivt. Delarna bör vara ungefär lika stora.
  - Härska: Konstruera en slutlösning från dellösningarna.
- Leder till rekursiva algoritmer
  - Kan vara en bra lösning om det är svårt hitta iterativa lösningar.
  - Är ibland effektivare även om det finns iterativ lösning.
  - Ibland beräknas en dellösning många gånger (= ineffektivt).
- $O(n \log n)$ är vanligt.
- *Merge-sort* och *Quick-sort*

Söndra och härska, exempel: Beräkna x^n

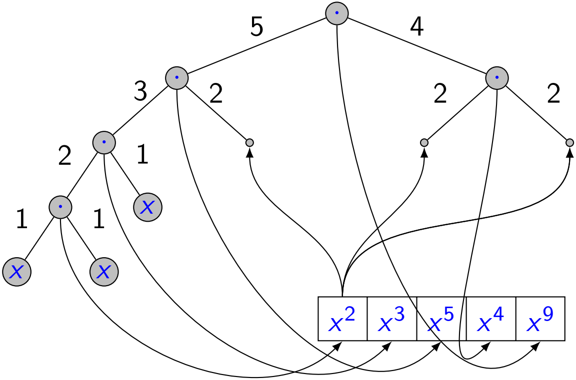
- Beräkna iterativt $f(x) = x \cdot x \cdots x$ ger en algoritm som är $O(n)$.
- *Divide-and-Conquer*: Vi kan bryta ner problemet och beräkna $x^{\lceil n/2 \rceil} \cdot x^{\lfloor n/2 \rfloor}$ rekursivt.
- Exempel: Beräkna x^9.



- Fast det ger inget!
- I de rekursiva anropen så beräknar vi i stort sett samma värden.
- Kan vi utnyttja detta och vinna något?

Dynamisk programmering

- Använder lite minne till att undvika att lösa samma delproblem flera gånger.
- Metod:
  - Ställ upp en tabell som håller reda på redan kända lösningar.
  - För varje nytt anrop kollar man om man redan löst det problemet.
  - Om inte löser man det och sätter in lösningen i tabellen.



Andra exempel

- En-dimensionellt: Fibonacci-sekvensen:

$$
\begin{aligned}
&F(n) = F(n-1) + F(n-2),\\
&F(0) = 1,\\
&F(1) = 1.
\end{aligned}
$$

- Multi-dimensionell dynamisk programmering:
  - Matrisbaserad shortest path (Floyd).
  - 0-1 Knapsack.