lecture 14

- Subset sum
- Knapsack

Resources for this lecture

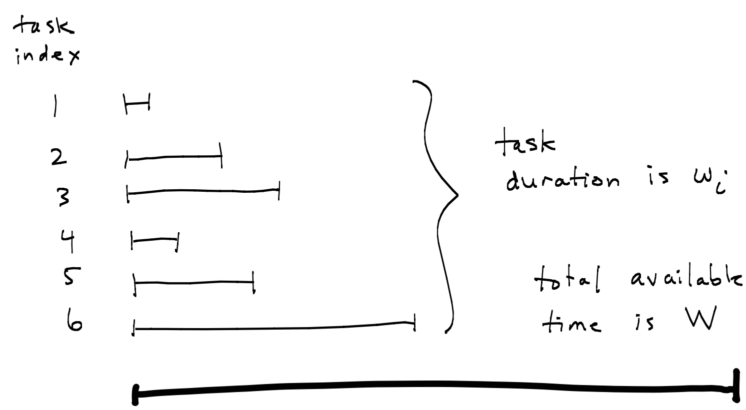
- I used Kleinberg & Tardos ch. 6.4
- Also see Roughgarden week 3

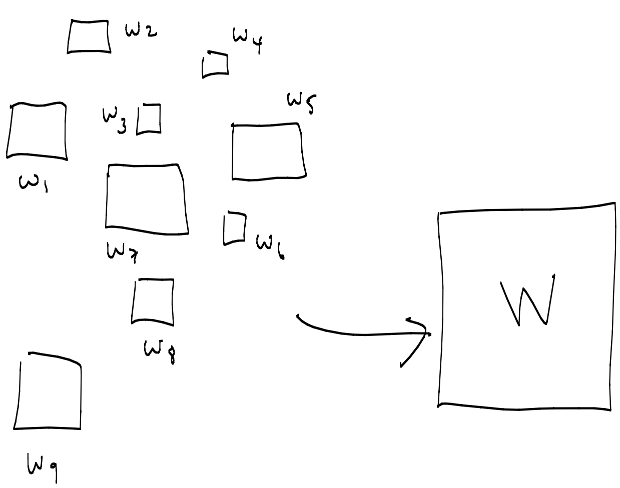
https://class.coursera.org/algo2-2012-001/lecture

We have a machine (resource) that can do only one task at a time. Task i takes time w_i. Which tasks should we do?

Problem 1: Maximize the number of tasks that can be completed in time W.

This is similar to interval scheduling but now we only have durations, not start and finish times.





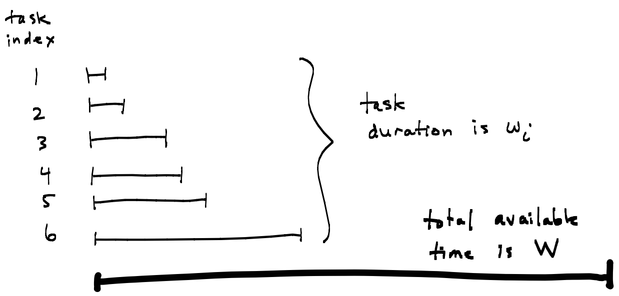
Problem 1 (restated)

Given a set of N items with weights $w_i \geq 0$ and given a bound W, find the largest subset $S \subseteq \{1, 2, \ldots N\}$ of items such that $\sum_{i \in S} w_i \leq W$.

Greedy approach

- order the intervals by increasing w_i
- Find the largest k such that

$$\sum_{i=1}^{k} w_i \le W .$$



Q: Why does greedy work?

A:

Intuitively, choosing the smallest w_i leaves the most remainder.

How to prove it mathematically?

Proof that greedy finds optimal solution:

By contradiction:

Let greedy choose items $\{1, 2, \ldots k\}$.

Assume there exists a subset with $k+1$ items, $S = \{i_1, i_2, i_3, \ldots, i_k, i_{k+1}\}$ (increasing sequence)

such that $\sum_{j=1}^{k+1} w_{i_j} \le W$.

$\{1, 2, 3, \ldots k\}$

$\{i_1, i_2, i_3, \ldots, i_k, i_{k+1}\}$

Since $w_1 \le w_2 \le w_3 \le \ldots \le w_N$

it follows that $w_j \le w_{i_j}$

and so $\underbrace{\sum_{j=1}^{k} w_j}_{\text{greedy}} \le \underbrace{\sum_{j=1}^{k} w_{i_j}}_{\text{first } k \text{ in the allegedly better solution}}$

But then the greedy solution would not have stopped after k, since

$$\sum_{j=1}^{k} w_j + w_{i_{k+1}} \le \underbrace{\sum_{j=1}^{k+1} w_{i_j}}_{\text{assumed}} \le W .$$

Thus, the assumed sequence $\{i_j\}$ cannot exist.

(contradiction)

Problem 2 ("subset sum")

Find the subset $S \subseteq \{1, 2, \ldots N\}$ that maximizes $\sum_{i \in S} w_i$ such that

$$\sum_{i \in S} w_i \le W .$$

Example:

$w_1 = 1, \quad w_2 = 1, \quad w_3 = 9, \quad w_4 = 9$

$W = 18$

Problem 1 $\Rightarrow$ $S = \{w_1, w_2, w_3\}$

Problem 2 $\Rightarrow$ $S = \{w_3, w_4\}$

Question: how many subsets of $\{1, 2, 3, 4, \dots, N\}$ are there?

Answer:

$$2^N = 2 \times 2 \times \cdots 2$$

i.e. each element is either in or out.

To solve Problem 2 efficiently, we must avoid the exponential number of subsets

Define:

$$Opt(N, W) \equiv \max_S \left\{ \sum_{i \in S} w_i : \sum_{i \in S} w_i \le W \right\}$$

Dynamic programming: how to break this problem into smaller problems?

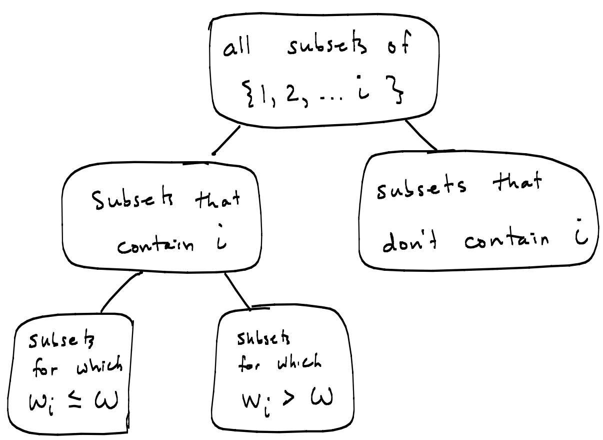
"Smaller"? $\Rightarrow$ reduce N or W

$Opt(i, \omega)$

$\equiv$ maximum $\sum_{j \in S} w_j$ such that

- $S \subseteq \{1, 2, \dots i\}$
- $\sum_{j \in S} w_j \le \omega \le W$

To find $Opt(i, w)$, which subsets of $\{1, 2, \dots i\}$ do we consider?



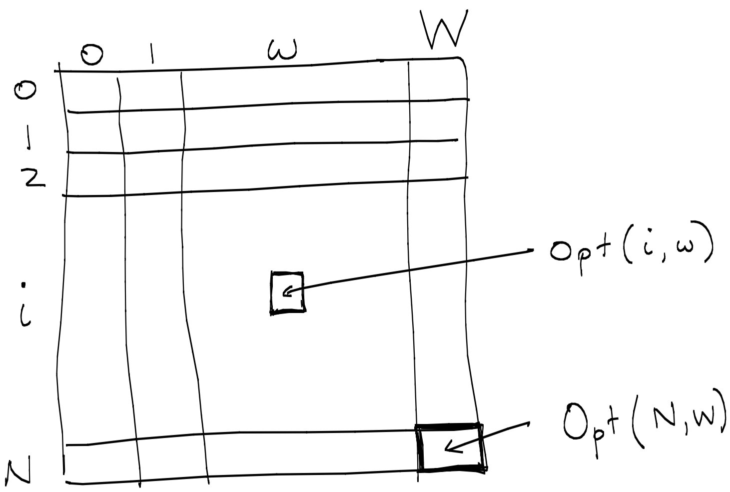
if $w_i > \omega$ // then we can't use i

$$Opt(i, \omega) = Opt(i-1, \omega)$$

else

$$Opt(i, \omega) = \max\{ \underbrace{Opt(i-1, \omega)}_{i \notin S}, \underbrace{w_i + Opt(i-1, \omega - w_i)}_{i \in S} \}$$

Opt(i, w)



Iterative approach

for $i = 0$ to N
  $Opt[i][0] = 0$

for $w = 0$ to W
  $Opt[0][w] = 0$

for $i = 1$ to N
  for $w = 1$ to W
    $Opt(i, w) =$ see above recurrences

Exercise

$w_1 = w_2 = 2$, $w_3 = 3$, $W = 6$

Find $Opt[i][w]$

w_i	i	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
2	1	0	0	2	2	2	2	2
2	2	0	0	2	2	?		
3	3	0						

w_i	i	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
2	1	0	0	2	2	2	2	2
2	2	0	0	2	2	4	4	4
3	3	0	0	2	3	4	?	

Opt(i, w)

negative weights ?

w_i	i	0	1	2	3	4	5	6 = W
	0	0	0	0	0	0	0	0
2	1	0	0	2	2	2	2	2
2	2	0	0	2	2	4	4	4
3	3	0	0	2	3	4	5	5

← Opt(N, W)

Given the table Opt[ ][ ]
find $S \subseteq \{1, 2, \dots N\}$ such that

$$\sum_{j \in S} w_j = Opt[N][W]$$

w_i	i	0	1	2	3	4	5	6 = W
	0	0	0	0	0	0	0	0
2	1	0	0	2	2	2	2	2
2	2	0	0	2	2	4	4	4
3	3	0	0	2	3	4	5	5

$5 = 2 + w_3$, $5 \neq 4$ $\therefore w_3 \in S$

w_i	i	0	1	2	3	4	5	6 = W
	0	0	0	0	0	0	0	0
2	1	0	0	2	2	2	2	2
2	2	0	0	2	2	4	4	4
3	3	0	0	2	3	4	5	5

$2 = 0 + w_2$, $2 = 2$

Thus,

either solution works.

w_i	i	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
2	1	0	0	2	2	2	2	2
2	2	0	0	2	2	4	4	4
3	3	0	0	2	3	4	5	5

useless

Solutions $\{w_1, w_3\}$ $\{w_2, w_3\}$

<u>Claim:</u> running time and space required is $O(NW)$.

<u>Exercise:</u> what if we used a recursive approach instead?

lecture 14

- Subset sum
- Knapsack

Knapsack

Given a set of N items with weights w_i and values v_i, and given a bound W on the total weight (as before), find a subset S of the items such that $\sum_{i \in S} w_i \leq W$ (as before)

and $\sum_{i \in S} v_i$ is maximized.

e.g.

	w_i	v_i
bar of gold	large	large
brick	large	small
stack of \$10,000 bills	small	large
stack of Canadian Tire bills	small	small

~~Subset Sum~~ Knapsack

if $w_i > w$ // then we can't use i

$$Opt(i, w) = Opt(i-1, w)$$

else

$$Opt(i, w) = \max \{ Opt(i-1, w), \underset{v_i}{\cancel{w_i}} + Opt(i-1, w - w_i) \}$$

Algorithm for knapsack is identical to that of subset sum, except for that minor change in the recurrence.

Time and space are again

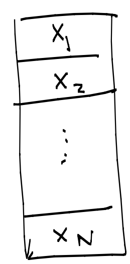
$$O(N \; W).$$

Subtlety:

- N is the number of elements in $\{ w_1, w_2, \ldots w_N \}$.
- W is a number (always one number).

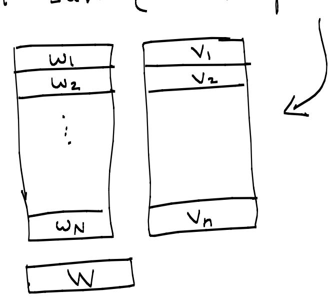
In theoretical computer science, one expresses the time or space used in a computation (algorithm) in terms of the "size" (memory used) of the input.

e.g. sorting N numbers



But for subset sum (or knapsack)

we have



The W in $O(NW)$ doesn't refer to the size of the input.

Exercise (Advanced): how to reconcile this?