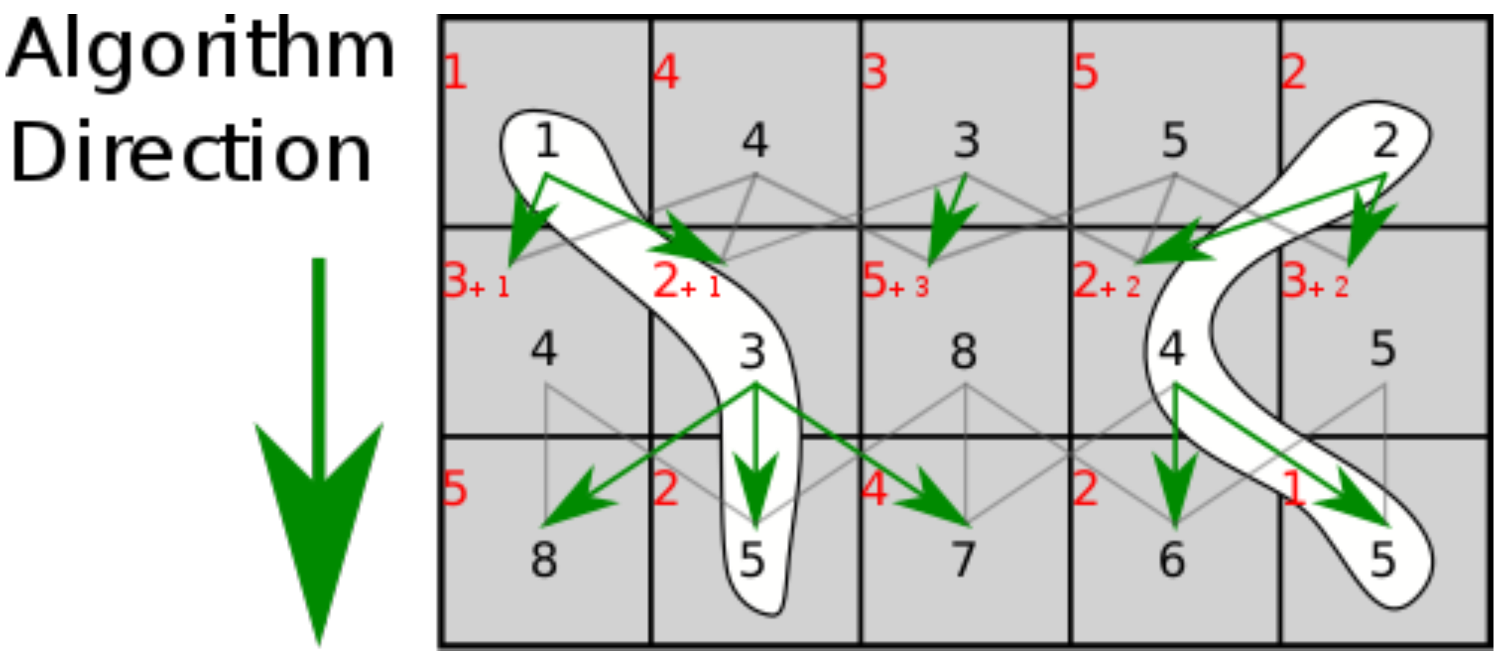
Dynamic Programming III: Knapsack Problem



Admin

- Assignment 5 is due a day early
  - Office hours: M 2.00-3.30pm, T 3-5 pm
  - TA hours: today 3.30-5, 9-10 pm
  - TA hours: tomorrow 5-10 pm
  - Late work may not be graded in time
- **Midterm** this Friday **(April 2);** no class
- Exam be released 10.00 am Friday; can be taken in any 24 hour period between **10.00 am Friday to 10.00 am Sunday**

Knapsack Problem

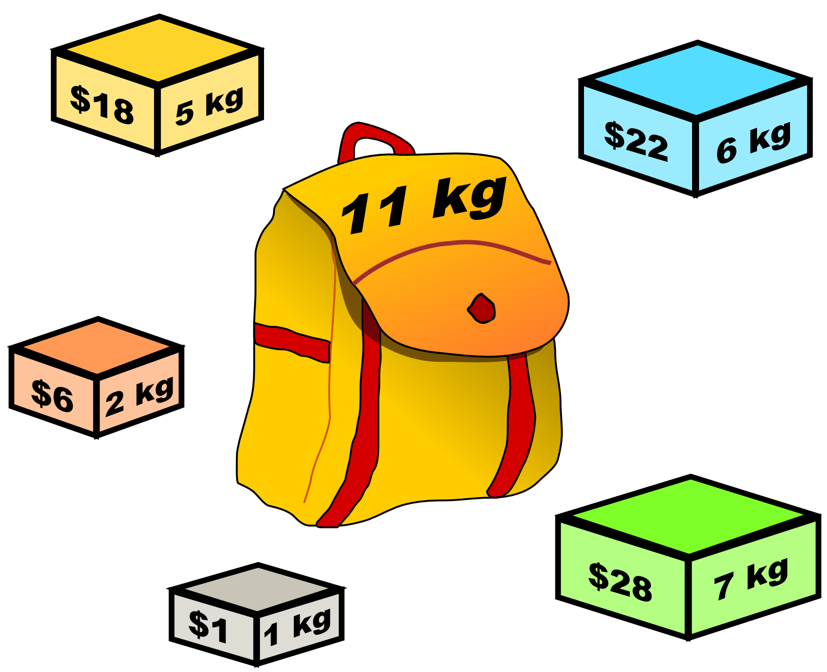
Reading: Chapter 6.4, KT

Knapsack Problem

- **Problem**. Pack a knapsack to maximize total value
- There are n items, each with weight w_i and value v_i
- Knapsack has total capacity C
- For any set of items T they fit in the Knapsack iff
    - **Capacity constraint**: $\sum_{i \in T} w_i \leq C$
- **Goal**: Find subset S of items that fit in the knapsack (satisfy the capacity constraint) **and maximize** the total value $\sum_{i \in S} v_i$
- **Assumption**. All weights and values are non-negative integers

Knapsack Problem

- Does greedily picking the highest value item work?
- Does greedily picking the lowest weight item work?

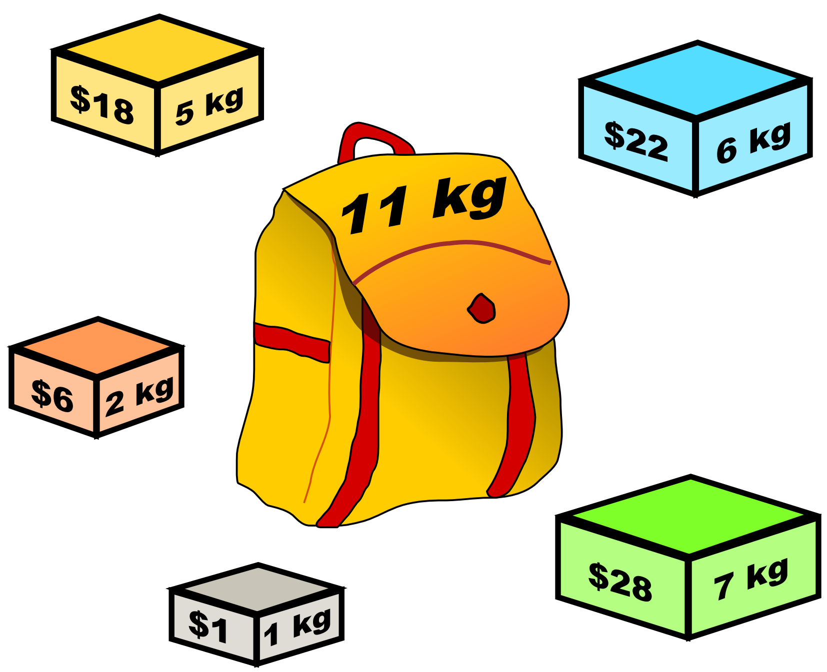


i	v_i	w_i
1	\$1	1 kg
2	\$6	2 kg
3	\$18	5 kg
4	\$22	6 kg
5	\$28	7 kg

Knapsack instance
(weight limit C = 11 kg)

Knapsack Problem

- Example (Knapsack capacity C = 11)
  - Optimal: {3, 4} has value \$40 (and weight 11)

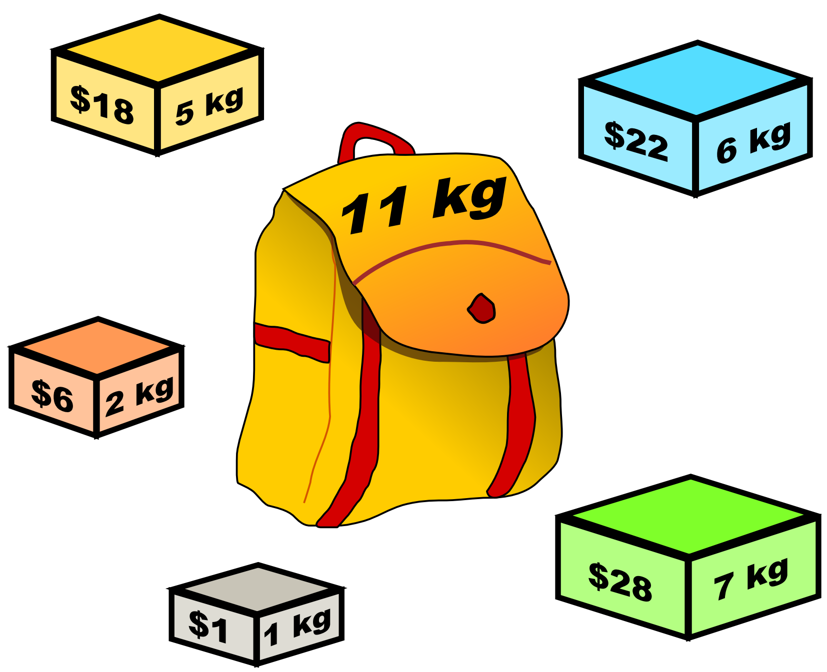


i	v_i	w_i
1	\$1	1 kg
2	\$6	2 kg
3	\$18	5 kg
4	\$22	6 kg
5	\$28	7 kg

knapsack instance
(weight limit W = 11)

Exponential Possibilities

- Given S items, how many subsets of items are there total?
  - 2^S: exponential possibilities
- Dynamic programming trades of space for time, and through memoization gives an efficient solution



i	v_i	w_i
1	\$1	1 kg
2	\$6	2 kg
3	\$18	5 kg
4	\$22	6 kg
5	\$28	7 kg

knapsack instance
(weight limit W = 11)

Towards a Subproblem

- Idea 1: Keep track of capacity
  - **Subproblem**. Let $T[c]$ denote the value of the optimal solution that uses capacity $\leq c$.
- Optimal solution: $T[C]$
- How do come up with a recurrence?
- Not obvious with just capacities

Subproblems and Optimality

- When items are selected we need to fill the remaining capacity optimally
- Subproblem associated with a given remaining capacity can be solved in different ways

Partial Selection #1

Partial Selection #2

- In both cases, remaining capacity: 10 kg but items left are different

Subproblem:
Optimal Substructure

Subproblem

- **Subproblem**

$\boxed{\text{OPT}(j, c)\text{: value of optimal solution using items } \{1,2,\ldots,j\} \text{ with total capacity } \leq c\text{, for } 1 \leq j \leq n,\ \ 0 \leq c \leq C}$

- **Final answer**

$\boxed{\text{OPT}(n, C)}$

Base Cases

- Let us think about which rows/columns can we fill initially
- What about the first row corresponding to item 1?
- OPT$(1,\ c)$: Value of optimal solution that uses item 1 and has total capacity at most c
- For $c = 1,2,\ldots,C$ we can fill out the first row as:

$$\boxed{\begin{array}{l} \text{OPT}(1,\ c) = v_1 \text{ if } c \geq w_1 \\ \text{OPT}(1,\ c) = 0 \text{ if } c < w_1 \end{array}}$$

Base Cases

- Let us think about which rows/columns can we fill initially
- What about the first row corresponding to item 1?
- OPT$(i, 0)$: Value of optimal solution that uses first i items and has total capacity at most 0
- For $i = 1,2,\ldots,n$ we can fill out the first column as:

$$\boxed{\text{OPT}(i, 0) = 0}$$

Optimal Substructure

- OPT(i, c): Let us try to construct the optimal solution that uses items $\{1,2,\ldots,i\}$ and capacity at most c
- What are the possibilities for the last ith item:
  - Either it is in the optimal solution or not, we consider both cases
- **Case 1.** Suppose it is **not** in the optimal solution, what is the optimal way to solve the remaining problem?
  - OPT(i, c) $=$ OPT$(i-1, c)$

Optimal Substructure

- OPT(i, c): Let us try to construct the optimal solution that uses items $\{1,2,\ldots, i\}$ and capacity at most c
- What are the possibilities for the last ith item:
  - Either it is in the optimal solution or not, we consider both cases
- **Case 2.** Suppose it **is** in the optimal solution, what is the recurrence of the optimal solution?
  - OPT(i, c) $= v_i +$ OPT$(i-1,\ c - w_i)$
  - This case only possible if $c \geq w_i$

Final Recurrence

- For $1 \leq i \leq n$ and $1 \leq c \leq C$, we have:

$$\boxed{\begin{aligned}&\text{OPT}(i, c) = \\ &\max\{\text{OPT}(i-1,\ c),\ v_i + \text{OPT}(i-1,\ c - w_i)\}\end{aligned}}$$

- **Memoization structure**: We store OPT$[i, c]$ values in a 2-D array or table using space $O(nC)$
- **Evaluation order**: In what order should we fill in the table?
  - Row-major order (row-by-row)

Running Time

- Takes $O(1)$ to fill out a cell, $O(nC)$ total cells
- Is this polynomial? By which I mean polynomial in the *size of the input*
- Input: Store n items, plus need to store C
    - $O(n)$ size of n items
    - How much space for C ? $\log C$ bits to be more precise
- Is $O(nC)$ polynomial?
    - Not polynomial in C, but polynomial in n
    - “Pseudopolynomial” - polynomial in the *value* of the input
- To think about: does this work if the weights are not integers?

Recipe for a Dynamic Program

- **Formulate the right subproblem.** The subproblem must have an optimal substructure
- **Formulate the recurrence.** Identify how the result of the smaller subproblems can lead to that of a larger subproblem
- **State the base case(s)**. The subproblem thats so small we know the answer to it!
- **State the final answer.** (In terms of the subproblem)
- **Choose a memoization data structure.** Where are you going to store already computed results? (Usually a table)
- **Identify evaluation order.** Identify the dependencies: which subproblems depend on which ones? Using these dependencies, identify an evaluation order
- **Analyze space and running time.** As always!

Partitioning Books

Reading: Linked on GLOW

Partitioning Work

- Suppose we have to scan through a shelf of books, and each book has a different size
- We want to divide the shelf into k region of books, and each region is assigned one of the workers
- Order of books fixed by cataloging system: cannot reorder/rearrange the books
- **Goal**: divide the work is a fair way among the workers

Linear Partition Problem

- **Input.** A input arrangement S of nonnegative integers $\{s_1, \ldots, s_n\}$ and an integer k
- **Problem.** Partition S into k ranges such that the **maximum sum** over all the ranges is **minimized**
- Example.
  - Consider the following arrangement

    $$100\ 200\ 300\ 400\ 500\ 600\ 700\ 800\ 900$$

  - If $k = 3$, a partition that minimizes the maximum sum:

    $$100\ 200\ 300\ 400\ 500\ |\ 600\ 700\ |\ 800\ 900$$

Optimal Substructure

- What should be our subproblem
  - Need a subproblem with optimal substructure (that we can recurse over)
- What are the things we need to keep track of?
  - What elements have we already partitioned out of $1,2,\ldots n$
  - Number of partitions we have used out of k
- Any ideas for what the subproblem should be?

Subproblem

- **Subproblem**

$M(i, j)$ be the optimal cost of partitioning elements $s_1, s_2, \ldots, s_i$ using j partitions, where $1 \leq i \leq n$, $1 \leq j \leq k$

- **Final answer**

$M(n, k)$

Base Cases

- Let us think about which rows/columns can we fill initially
- What about the first row corresponding to item 1?
- Remember that optimal cost is max sum over all partitions
- $M(1, j)$: optimal cost of partitioning s_1 across j partitions
- For $j = 1,2,\ldots,k$ we can fill out the first column as:

$$\boxed{M(1, j) = s_1}$$

Base Cases

- Let us think about which rows/columns can we fill initially
- What about the first row corresponding to item 1?
- Remember that optimal cost is max sum over all partitions
- $M(i,\ 1)$: optimal cost of partitioning $s_1, s_2, \ldots, s_i$ using only 1 partition
- For $i = 1,2,\ldots,n$ we can fill out the first column as:

$$\boxed{M(i,\ 1) = \sum_{\ell=1}^{i} s_\ell}$$

Base Cases Summary

- For $j = 1,2,\ldots,k$ we can fill out the first column as:

$$\boxed{M(1, j) = s_1}$$

- For $i = 1,2,\ldots,n$ we can fill out the first column as:

$$\boxed{M(i, 1) = \sum_{\ell=1}^{i} s_\ell}$$

Towards a Recurrence

- Want a recurrence for $M(i, j)$
- Notice that the jth partition starts after we place the $(j - 1)$st "divider"
- Where can we place the $j - 1st$ divider?
  - Suppose between t th and $(t + 1)$st element, where $1 \leq t \leq i$
  - What is the cost of placing the last divider here?
  - Max of the cost of
    - the last partition (the sum of all elements in it)
    - the optimal way to partition the elements to the "left"

Will be Continued in Next Lecture

Acknowledgments

- Some of the material in these slides are taken from
    - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
    - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)