Divide & Conquer, Greedy and Dynamic Programming Algorithms

Dr Muhammad Atif Tahir
FAST University, Karachi Campus

Divide & Conquer

- **Divide and conquer** (**D&C**) is an algorithm design paradigm based on multi-branched recursion
- A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems
- Until these subproblems become simple enough to be solved directly

Divide & Conquer

- The solutions to the sub-problems are then combined to give a solution to the original problem
- For example
  - Merge sort
  - Quick sort
  - **Karatsuba algorithm** (Fast multiplication algorithm)

Greedy Algorithm

- A greedy algorithm is a mathematical process that looks for
  - simple
  - easy-to-implement solutions
- to complex, multi-step problems by deciding which next step will provide the most obvious benefit.

Greedy Algorithm

- Prim’s Algorithm (Minimum Spanning Trees)

- Kruskal’s Algorithm (Minimum Spanning Trees)

- Dijsktra Algorithm (Single Source Shortest Path)

Dynamic Programming

- Dynamic Programming is a technique for algorithm design
- It is a tabular method in which we break
  - down the problem into subproblems
  - and place the solution to the subproblems in a matrix

Dynamic Programming

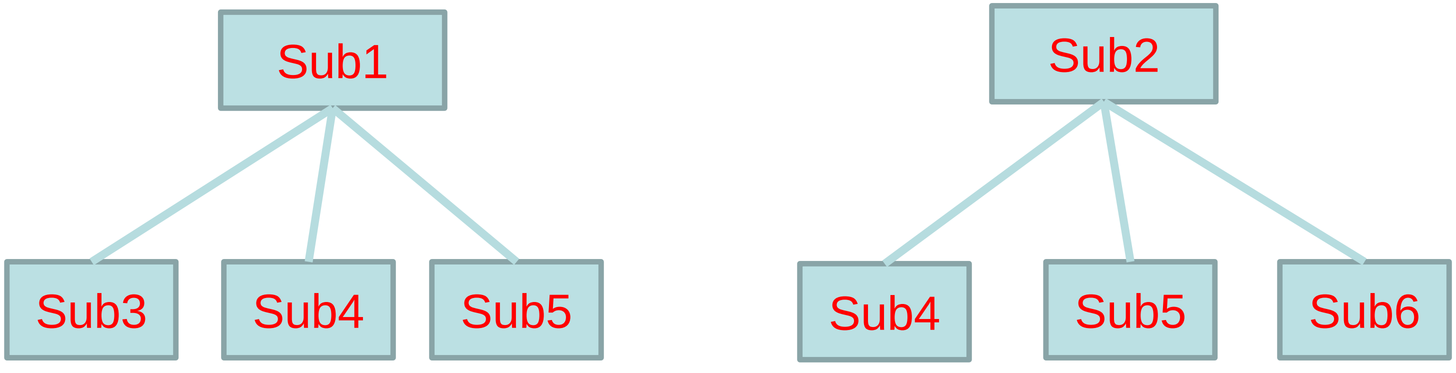
- The matrix elements can be computed:
  – iteratively, in a bottom-up fashion
  – recursively, using memoization
- Dynamic Programming is often used to solve optimization problems
- In these cases, the solution corresponds to an objective function whose value needs to be optimal (e.g. maximal or minimal)

Dynamic Programming

- Usually it is sufficient to produce one optimal solution, even though there may be many optimal solutions for a given problem

What is dynamic programming?

- Subproblems overlap
  - Subproblems share sub-subproblems



Dynamic Programming

- Dynamic programming algorithms
  - Solve each subproblem only **once**
  - If solve problems in a top-down manner, record sub problem solutions in a table (named: **top-down with memoization**)

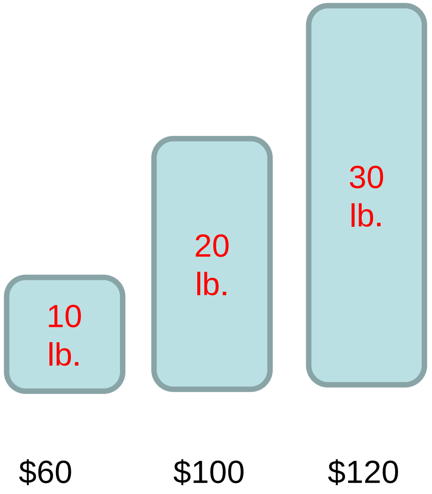
    In computing, **memoization** is an optimization technique used primarily to speed up computer programs by having function calls avoid repeating the calculation of results for previously processed inputs.
  - If solve the problems in a bottom-up manner, solve the smaller problem first (named: **bottom-up method**)

Fractional knapsack and 0-1 knapsack

- After you break into a house, how to choose the items you put into your knapsack, to maximize your income.
- Each item has a weight and a value
- Your knapsack has a maximum weight
- 0-1 knapsack
  - You can only take an item or leave it there
- Fractional knapsack
  - You can take part of an item

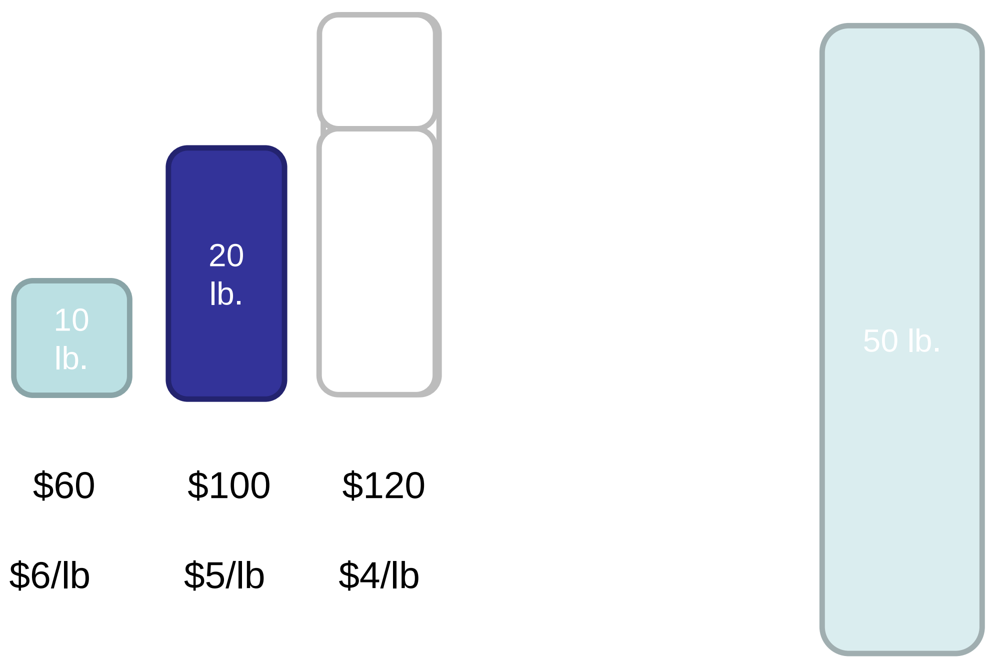
Fractional knapsack

- The **fractional knapsack problem** is a classic problem that can be solved by greedy algorithms
- E.g.
  - your knapsack can contain 50 lp. Stuff;
  - the items are as in the figure
  - What is your algorithm?



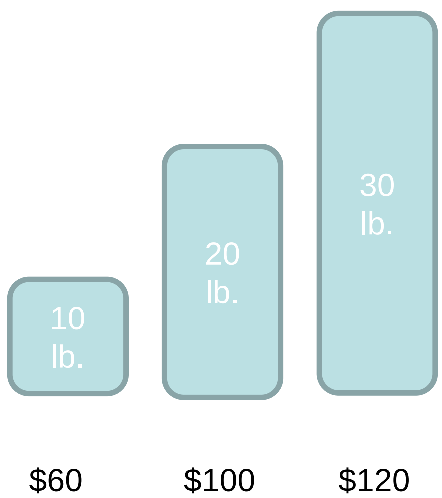
Fractional knapsack

- A greedy algorithm for **fractional knapsack problem**
- Greedy choice: choose the maximum value/lb. item



0-1 knapsack

- The **0-1 knapsack problem** is a classic problem that can **not** be solved by greedy algorithms
- Can you design an algorithm of this problem?
  - As part of next Lecture



References

- Lecture Slides of
  - Haidong Xue at GSU

- http://en.wikipedia.org/wiki/