Greedy Algorithms

Greedy Algorithms

- Optimization problems
  - Dynamic programming: Overkill sometimes.
  - Greedy algorithm:
    - Being greedy for local optimization with the hope it will lead to a global optimal solution, not always, but in many situations, it works.
    - Greedy algorithms tend to be easier to code

Greedy Algorithms

- A *greedy algorithm* always makes the choice that looks best at the moment
  - My everyday examples:
    - Walking to the Corner
    - Playing a bridge hand
  - The hope: a locally optimal choice will lead to a globally optimal solution
  - For some problems, it works

Elements of greedy strategy

- Determine the optimal substructure
- Develop the recursive solution
- Prove one of the optimal choices is the greedy choice yet safe
- Show that all but one of subproblems are empty after greedy choice
- Develop a recursive algorithm that implements the greedy strategy
- Convert the recursive algorithm to an iterative one.

Activity-Selection Problem

- Problem: get your money's worth out of a carnival
  - Buy a wristband that lets you onto any ride
  - Lots of rides, each starting and ending at different times
  - Your goal: ride as many rides as possible
    - Another, alternative goal that we don't solve here: maximize time spent on rides
- Welcome to the *activity selection problem*

Activity-Selection Problem

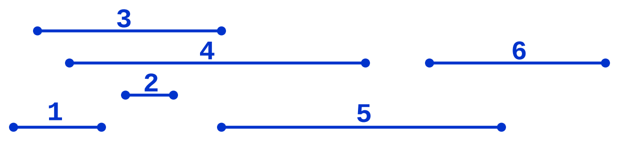
- Naïve strategy:
  - Ride the first ride, when get off, get on the very next ride possible, repeat until carnival ends
- *What is the sophisticated strategy?*

Activity-Selection

- Formally:
  - Given a set S of n activities

    s_i = start time of activity i

    f_i = finish time of activity i
  - Find max-size subset A of compatible activities



  - Assume activities sorted by finish time. That is, (without loss of generality) $f_1 \le f_2 \le \ldots \le f_n$

Review: Activity-Selection

- Formally:
  - Given a set S of n activities
    - s_i = start time of activity i f_i = finish time of activity i
  - Find max-size subset A of compatible activities
  - Assume activities sorted by finish time
- *What is optimal substructure for this problem?*

Review: Activity-Selection

- Formally:
  - Given a set *S* of *n* activities
    - s_i = start time of activity *i* f_i = finish time of activity *i*
  - Find max-size subset *A* of compatible activities
  - Assume activities sorted by finish time
- *What is optimal substructure for this problem?*
  - A: If *k* is the activity in *A* with the earliest finish time, then $A - \{k\}$ is an optimal solution to $S' = \{i \in S: s_i \geq f_k\}$

Activity Selection: Optimal Substructure

- Let *k* be the minimum activity in *A* (i.e., the one with the earliest finish time). Then $A - \{k\}$ is an optimal solution to $S' = \{i \in S: s_i \geq f_k\}$
  - In words: once activity #1 is selected, the problem reduces to finding an optimal solution for activity-selection over activities in *S* compatible with #1
  - Proof: if we could find optimal solution *B'* to *S'* with $|B| > |A - \{k\}|$,
    - Then $B \cup \{k\}$ is compatible
    - And $|B \cup \{k\}| > |A|$

Greedy Choice Property

- Dynamic programming? Memoize? Yes, but…
- Activity selection problem also exhibits the *greedy choice* property:
  - Locally optimal choice $\Rightarrow$ globally optimal sol'n
  - Theorem: If S is an activity selection problem sorted by finish time, then $\exists$ optimal solution $A \subseteq S$ such that $\{1\} \in A$
    - Sketch of proof: if $\exists$ optimal solution B that does not contain $\{1\}$, can always replace first activity in B with $\{1\}$ (*Why?*). Same number of activities, thus optimal.

Activity Selection: A Greedy Algorithm

- So actual algorithm is simple:
  - Sort the activities by finish time
  - Schedule the first activity
  - Then schedule the next activity in sorted list which starts after previous activity finishes
  - Repeat until no more activities
- Intuition is even more simple:
  - Always pick the shortest ride available at the time

Minimum Spanning Tree Revisited

- Recall: MST problem has optimal substructure
  - *Prove it*
- *Is Prim’s algorithm greedy? Why?*
- *Is Kruskal’s algorithm greedy? Why?*

The Knapsack Problem

- The famous *knapsack problem*:
  - A thief breaks into a museum. Fabulous paintings, sculptures, and jewels are everywhere. The thief has a good eye for the value of these objects, and knows that each will fetch hundreds or thousands of dollars on the clandestine art collector’s market. But, the thief has only brought a single knapsack to the scene of the robbery, and can take away only what he can carry. What items should the thief take to maximize the haul?

The Knapsack Problem

- More formally, the *0-1 knapsack problem*:
  - The thief must choose among n items, where the ith item worth v_i dollars and weighs w_i pounds
  - Carrying at most W pounds, maximize value
    - Note: assume v_i, w_i, and W are all integers
    - "0-1" b/c each item must be taken or left in entirety
- A variation, the *fractional knapsack problem*:
  - Thief can take fractions of items
  - Think of items in 0-1 problem as gold ingots, in fractional problem as buckets of gold dust

The Knapsack Problem

- *What greedy choice algorithm works for the fractional problem but not the 0-1 problem?*

The Knapsack Problem And Optimal Substructure

- Both variations exhibit optimal substructure
- To show this for the 0-1 problem, consider the most valuable load weighing at most W pounds
  - *If we remove item j from the load, what do we know about the remaining load?*
  - A: remainder must be the most valuable load weighing at most $W - w_j$ that thief could take from museum, excluding item j

Solving The Knapsack Problem

- The optimal solution to the fractional knapsack problem can be found with a greedy algorithm
  - *How?*
- The optimal solution to the 0-1 problem cannot be found with the same greedy strategy
  - Greedy strategy: take in order of dollars/pound
  - Example: 3 items weighing 10, 20, and 30 pounds, knapsack can hold 50 pounds
    - *Suppose item 2 is worth \$100. Assign values to the other items so that the greedy strategy will fail*

The Knapsack Problem: Greedy Vs. Dynamic

- The fractional problem can be solved greedily
- The 0-1 problem cannot be solved with a greedy approach
  - As you have seen, however, it can be solved with dynamic programming

Greedy vs. Dynamic Programming

- Knapsack problem
  - I1 (v1,w1), I2(v2,w2),…,In(vn,wn).
  - Given a weight W at most he can carry,
  - Find the items which maximize the values
- Fractional knapsack,
  - Greed algorithm, O(nlogn)
- 0/1 knapsack.
  - DP, O(nW).
  - Questions: 0/1 knapsack is an NP-complete problem, why O(nW) algorithm?

Typical tradition problem with greedy solutions

- Coin changes
  - 25, 10, 5, 1
  - How about 7, 5, 1
- Minimum Spanning Tree
  - Prim's algorithm
    - Begin from any node, each time add a new node which is closest to the existing subtree.
  - Kruskal's algorithm
    - Sorting the edges by their weights
    - Each time, add the next edge which will not create cycle after added.
- Single source shortest pathes
  - Dijkstra's algorithm
- Huffman coding
- Optimal merge

The End