Greedy Technique

Constructs a solution to an optimization problem piece by piece through a sequence of choices that are:

- feasible
- locally optimal
- irrevocable

For some problems, yields an optimal solution for every instance.
For most, does not but can be useful for fast approximations.
Applications of the Greedy Strategy

- **Optimal solutions:**
  - change making for “normal” coin denominations
  - minimum spanning tree (MST)
  - single-source shortest paths
  - simple scheduling problems
  - Huffman codes

- **Approximations:**
  - traveling salesman problem (TSP)
  - knapsack problem
  - other combinatorial optimization problems
Change-Making Problem

Given unlimited amounts of coins of denominations $d_1 > ... > d_m$, give change for amount $n$ with the least number of coins.

Example: $d_1 = 25c$, $d_2 = 10c$, $d_3 = 5c$, $d_4 = 1c$ and $n = 48c$

Greedy solution:
Change-Making Problem

Given unlimited amounts of coins of denominations $d_1 > \ldots > d_m$, give change for amount $n$ with the least number of coins

Example: $d_1 = 25c$, $d_2 = 10c$, $d_3 = 5c$, $d_4 = 1c$ and $n = 48c$

Greedy solution:

Greedy solution is

- optimal for any amount and “normal” set of denominations
- may not be optimal for arbitrary coin denominations
Minimum Spanning Tree (MST)

- **Spanning tree** of a connected graph $G$: a connected acyclic subgraph of $G$ that includes all of $G$’s vertices

- **Minimum spanning tree** of a weighted, connected graph $G$: a spanning tree of $G$ of minimum total weight

Example:
Prim’s MST algorithm

- Start with tree $T_1$ consisting of one (any) vertex and “grow” tree one vertex at a time to produce MST through a series of expanding subtrees $T_1, T_2, \ldots, T_n$

- On each iteration, construct $T_{i+1}$ from $T_i$ by adding vertex not in $T_i$ that is closest to those already in $T_i$ (this is a “greedy” step!)

- Stop when all vertices are included
Notes about Prim’s algorithm

- Proof by induction that this construction actually yields MST
- Needs priority queue for locating closest fringe vertex

Efficiency

- $O(n^2)$ for weight matrix representation of graph and array implementation of priority queue
- $O(m \log n)$ for adjacency list representation of graph with $n$ vertices and $m$ edges and min-heap implementation of priority queue
Another greedy algorithm for MST: Kruskal’s

- Sort the edges in nondecreasing order of lengths

- “Grow” tree one edge at a time to produce MST through a series of expanding forests $F_1, F_2, \ldots, F_{n-1}$

- On each iteration, add the next edge on the sorted list unless this would create a cycle. (If it would, skip the edge.)
Example
Notes about Kruskal’s algorithm

- Algorithm looks easier than Prim’s but is harder to implement (checking for cycles!)

- Cycle checking: a cycle is created iff added edge connects vertices in the same connected component