Greedy Algorithms

- Levitin Ch 9

Overview

- **change-making problem**: give change for a specific amount n with the least number of coins of the denominations $d_1 > ... > d_m$
  - greedy approach: use largest denomination that is less than n, and reduce remainder. Repeat until the total of the change is equal to the change required.
- **greedy technique**: suggests constructing solution through sequence of steps, each expanding partial solution, until a complete solution is reached
  - for each step, the choice made must be:
    - **feasible**, i.e. satisfies problem constraints
    - **locally optimal**
    - **irrevocable**: cannot be changed on subsequent steps
- in some instances greedy technique does provide globally optimum solution
  - minimum spanning tree problem: **Prim's, Kruskal's** algorithms
- often greedy technique will not yield optimum solution, but may still be useful to produce approximate solution
- **Dijkstra's algorithm**: shortest-path in a weighted graph
- **Huffman trees**: data compression method
- greedy algorithms are typically intuitive and simple
- often difficult to prove they are optimal (if they are), approaches include:
  - mathematical induction, or
  - show that on each step greedy approach does at least as well as any other algorithm could
  - show optimal solution (rather than optimal efficiency)

Minimum Spanning Tree Problem

- **spanning tree**: connected acyclic subgraph (i.e. tree) containing all vertices of an undirected connected graph
- **minimum spanning tree**: spanning tree of smallest weight for a weighted graph
- **weight**: sum of weights on all edges

- given n points, connect them in the cheapest possible way, such that there is a path between every pair of points
- arises in all sorts of networks: communications, computer, transportation, electrical
  - cheapest way to achieve connectivity
  - identifies clusters in data sets
  - classification in archaeology, biology, …
  - helpful for constructing approximate solutions to more difficult problems e.g. travelling salesman problem
- exhaustive search approach
  - exponential growth with graph size (especially for dense graphs)
  - generating all spanning trees is more difficult than finding minimum spanning tree

Prim's Algorithm

- construct minimum spanning tree through sequence of expanding subtrees
- initial subtree consists of arbitrary vertex
- algorithm expands the tree greedily by attaching *nearest* neighbour not in the tree
  - *nearest*: vertex out of tree connected to vertex in tree by edge of smallest weight
- algorithm stops when all vertices are in the tree
- total iterations: $n - 1$

```
Prim(G)
# Prim's algorithm for constructing minimum spanning tree
# Input: weight graph G=<V,E>
# Output: E_T, set of edges composing minimum spanning tree of G
V_T = {v_0} # arbitrary vertex
E_T = empty_set
for i from 1 to |V|-1:
    find minimum weight edge e* = (v*, u*) among all edges (v, u), such
        that v is in V_T,
    and u is in V-V_T
    V_T = union(V_T, u*)
    E_T = union(E_T, e*)
return E_T
```

- implementation requires each vertex not in the current tree to have information about shortest edge connecting it to vertex: attach labels to vertex
  - name of nearest vertex
  - weight (length) edge

- if no adjacent vertices:
  - name: ∞
  - label: *null*