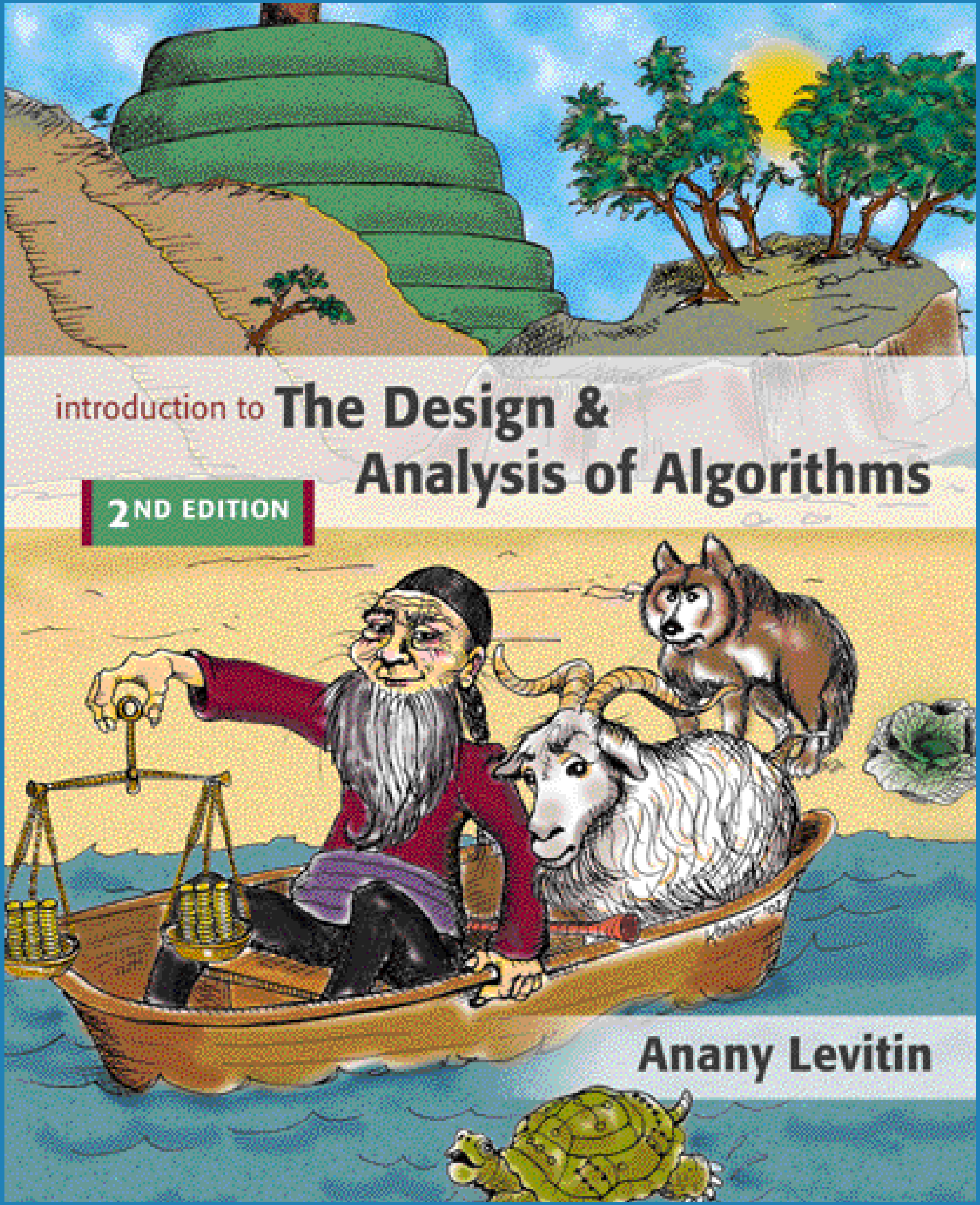
Chapter 9

Greedy Technique



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 $d_1 > \dots > d_m$, 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

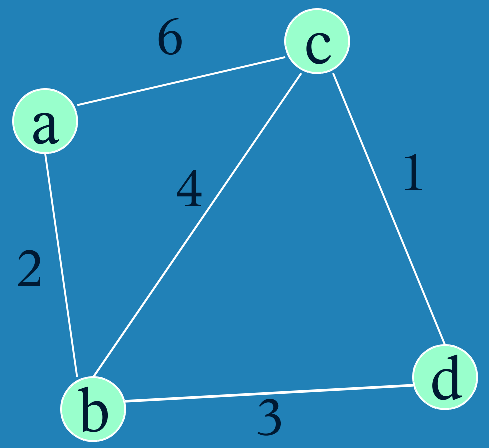
- optimal for any amount and "normal'' set of denominations
- may not be optimal for arbitrary coin denominations

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:



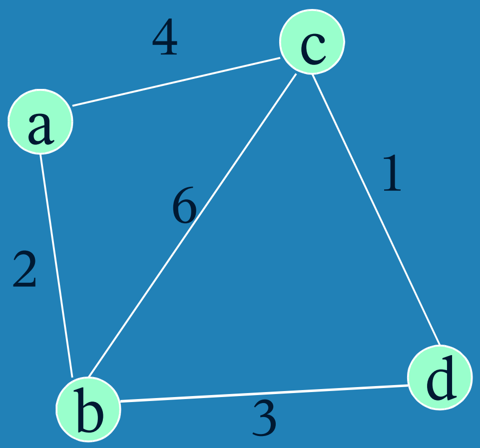
Prim's MST algorithm

- 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

Example



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

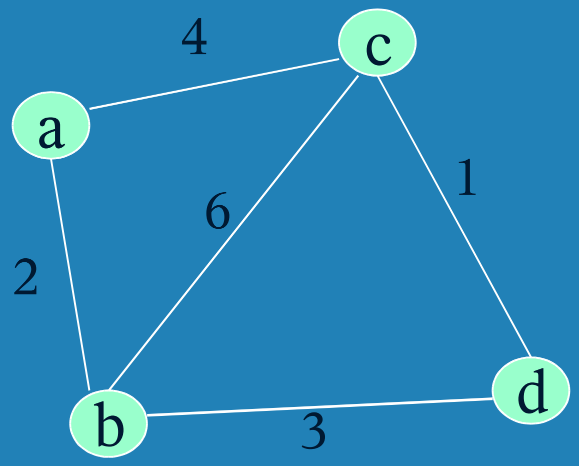
Another greedy algorithm for MST: Kruskal's

- 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, …, 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.)

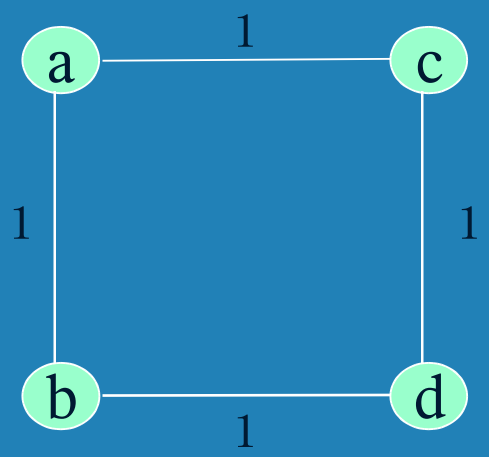
Example



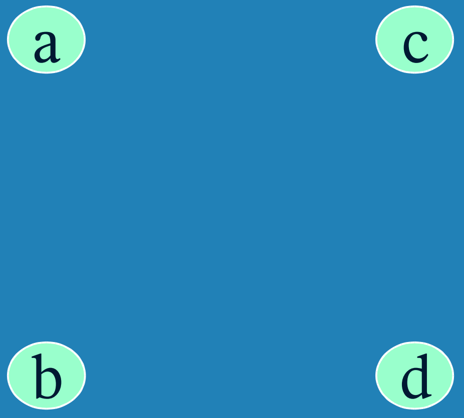
Notes about Kruskal's algorithm

- **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**
- ***Union-find* algorithms – see section 9.2**

Minimum spanning tree vs. Steiner tree



vs



Shortest paths – Dijkstra's algorithm

Single Source Shortest Paths Problem: Given a weighted connected graph G, find shortest paths from source vertex *s* to each of the other vertices

Dijkstra's algorithm: Similar to Prim's MST algorithm, with a different way of computing numerical labels: Among vertices not already in the tree, it finds vertex *u* with the smallest sum

$$d_v + w(v,u)$$

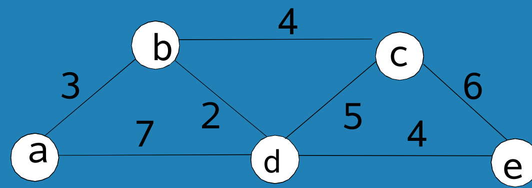
where

v is a vertex for which shortest path has been already found on preceding iterations (such vertices form a tree)

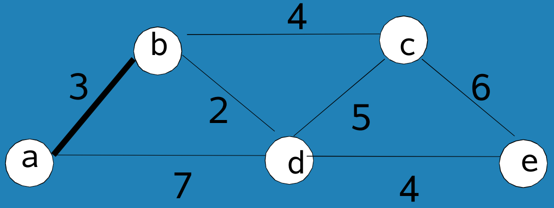
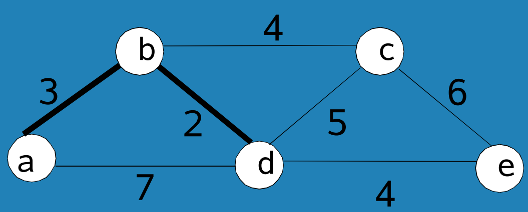
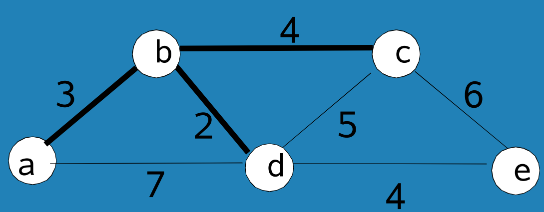
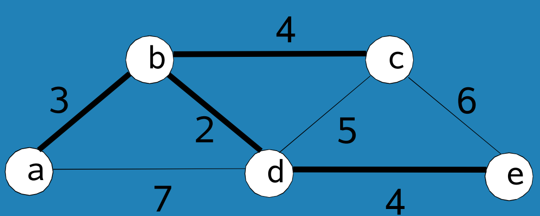
d_v is the length of the shortest path form source to v

$w(v,u)$ is the length (weight) of edge from v to u

Example



Tree vertices	Remaining vertices
a(-,0)	b(a,3) c(-,∞) d(a,7) e(-,∞)
b(a,3)	c(b,3+4) d(b,3+2) e(-,∞)
d(b,5)	c(b,7) e(d,5+4)
c(b,7)	e(d,9)
e(d,9)	

Notes on Dijkstra's algorithm

- Doesn't work for graphs with negative weights
- Applicable to both undirected and directed graphs
- Efficiency
  - $O(|V|^2)$ for graphs represented by weight matrix and array implementation of priority queue
  - $O(|E|\log|V|)$ for graphs represented by adj. lists and min-heap implementation of priority queue
- Don't mix up Dijkstra's algorithm with Prim's algorithm!

Coding Problem

Coding: assignment of bit strings to alphabet characters

Codewords: bit strings assigned for characters of alphabet

Two types of codes:

- ***fixed-length encoding*** (e.g., ASCII)
- ***variable-length encoding*** (e,g., Morse code)

Prefix-free codes: no codeword is a prefix of another codeword

Problem: If frequencies of the character occurrences are known, what is the best binary prefix-free code?

Huffman codes

- Any binary tree with edges labeled with 0's and 1's yields a prefix-free code of characters assigned to its leaves

- Optimal binary tree minimizing the expected (weighted average) length of a codeword can be constructed as follows

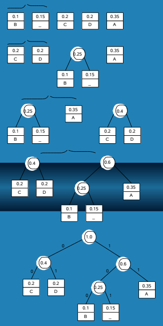
Huffman's algorithm

Initialize n one-node trees with alphabet characters and the tree weights with their frequencies.

Repeat the following step n-1 times: join two binary trees with smallest weights into one (as left and right subtrees) and make its weight equal the sum of the weights of the two trees.

Mark edges leading to left and right subtrees with 0's and 1's, respectively.

Example



character	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

average bits per character: 2.25

for fixed-length encoding: 3

compression ratio: (3-2.25)/3*100% = 25%