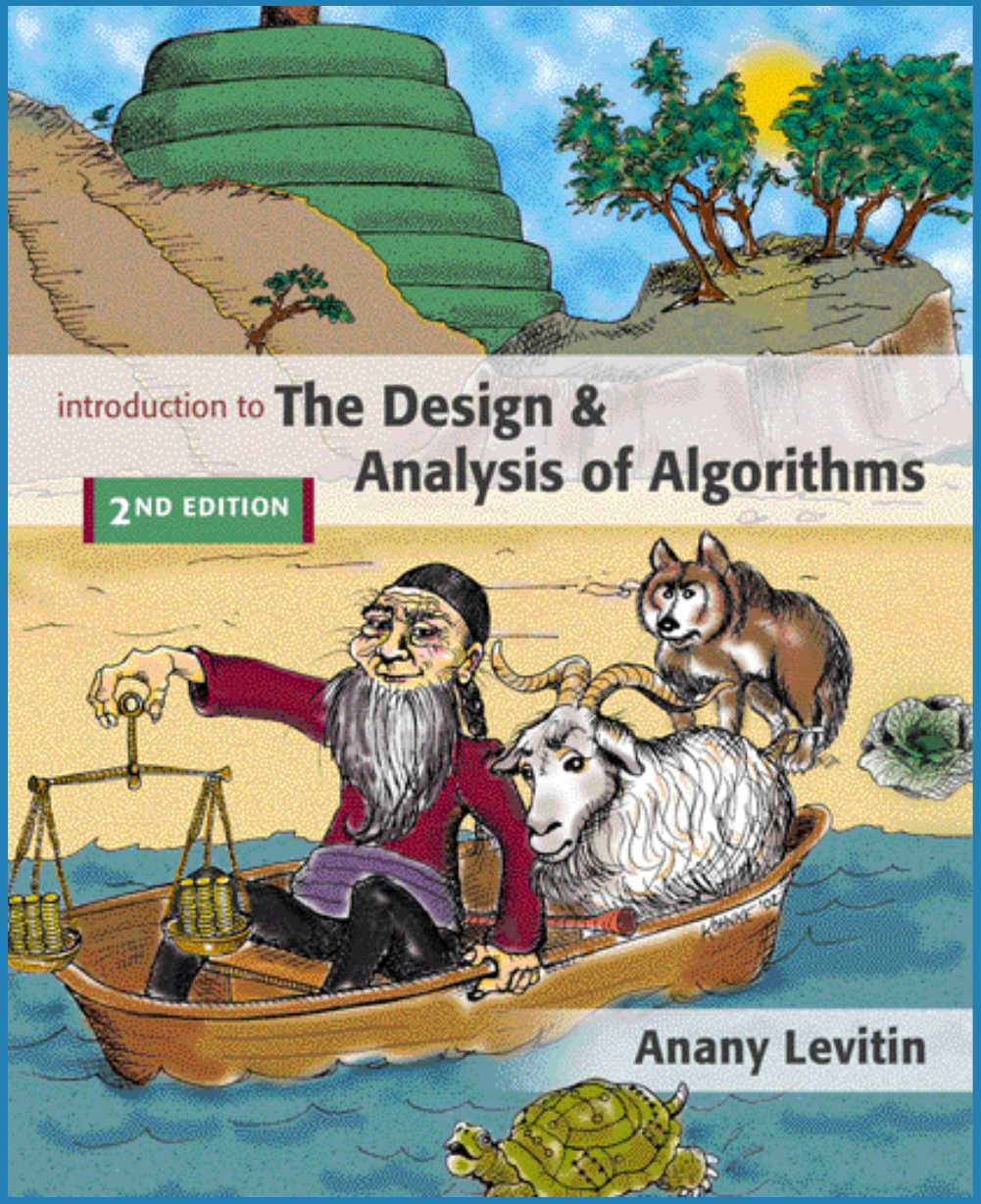
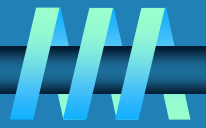
Chapter 9

Greedy Technique



Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Greedy Technique



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

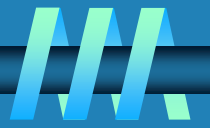
Defined by an objective function and a set of constraints

- ***feasible, i.e. satisfying the constraints***
- ***locally optimal (with respect to some neighborhood definition)***
- ***greedy (in terms of some measure), and irrevocable***

For some problems, it yields a globally optimal solution for every instance. For most, does not but can be useful for fast approximations. We are mostly interested in the former case in this class.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Applications of the Greedy Strategy

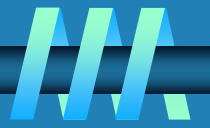


- **Optimal solutions:**
  - **change making for "normal" coin denominations**
  - **minimum spanning tree (MST)**
  - **single-source shortest paths**
  - **simple scheduling problems**
  - **Huffman codes**

- **Approximations/heuristics:**
  - **traveling salesman problem (TSP)**
  - **knapsack problem**
  - **other combinatorial optimization problems**

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

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

Q: What are the objective function and constraints?

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

Greedy solution: <1, 2, 0, 3>

Greedy solution is

- **optimal for any amount and "normal" set of denominations**

Ex: Prove the greedy algorithm is optimal for the above denominations.

- **may not be optimal for arbitrary coin denominations**

For example, $d_1 = 25c$, $d_2 = 10c$, $d_3 = 1c$, and $n = 30c$

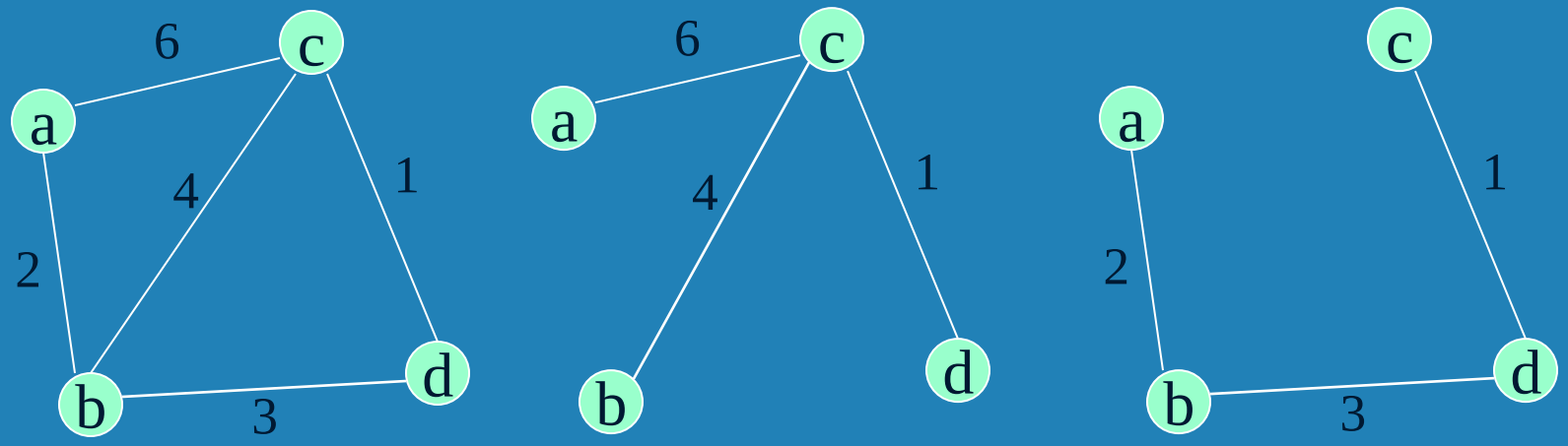
Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Minimum Spanning Tree (MST)

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

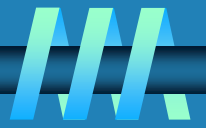
- ***Minimum spanning tree*** of a weighted, connected graph *G*: a spanning tree of *G* of the minimum total weight

Example:



Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

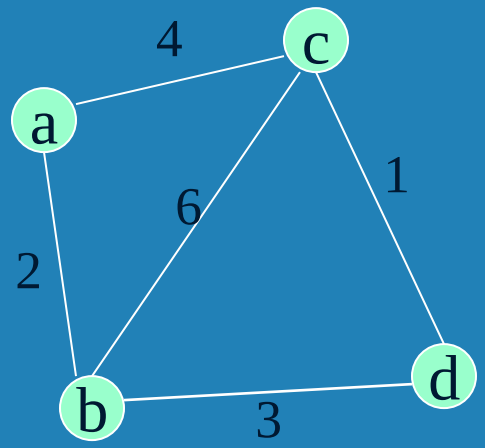
Prim's MST algorithm

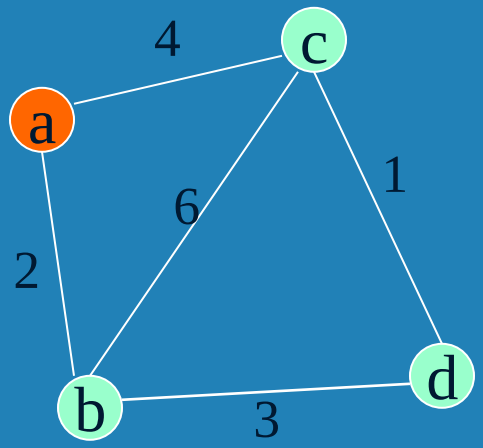


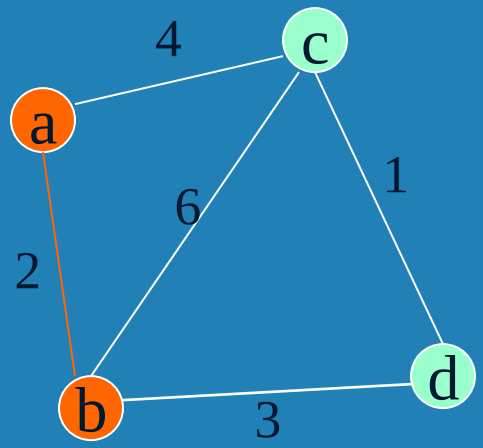
- Start with tree T_1 consisting of one (any) vertex and "grow" tree one vertex at a time to produce MST through a series of expanding subtrees $T_1, T_2, \ldots, T_n$
- On each iteration, construct T_{i+1} from T_i by adding vertex not in T_i that is closest to those already in T_i (this is a "greedy" step!)
- Stop when all vertices are included

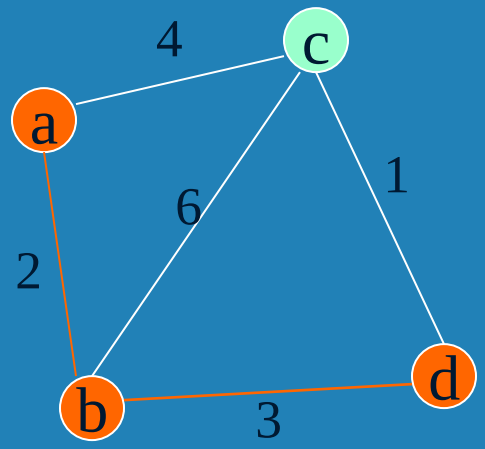
Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

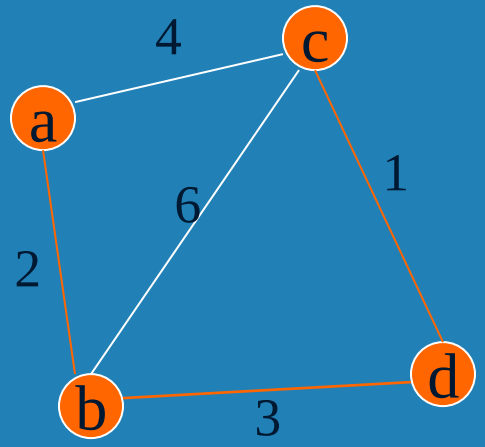
Example





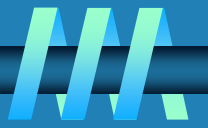






Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Notes about Prim's algorithm



- Proof by induction that this construction actually yields an MST (CLRS, Ch. 23.1). Main property is given in the next page.

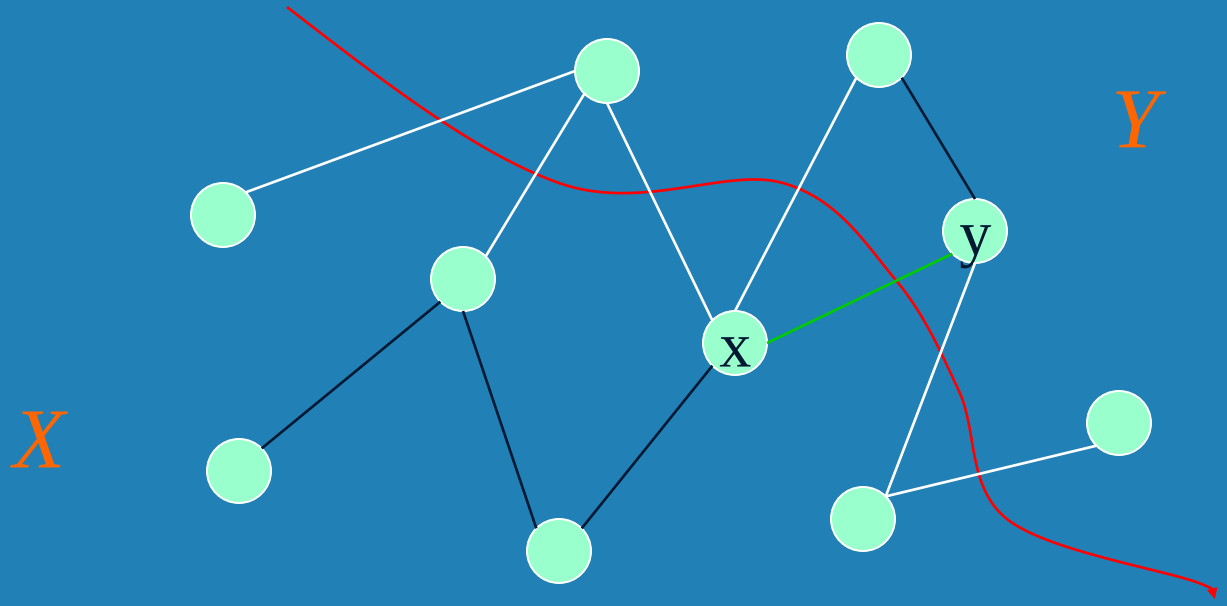
- Needs priority queue for locating closest fringe vertex. The Detailed algorithm can be found in Levitin, P. 310.

- Efficiency
  - $O(n^2)$ for weight matrix representation of graph and array implementation of priority queue
  - $O(m \log n)$ for adjacency lists representation of graph with n vertices and m edges and min-heap implementation of the priority queue

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

The Crucial Property behind Prim's Algorithm

Claim: Let $G = (V,E)$ be a weighted graph and (X,Y) be a partition of V (called a *cut*). Suppose $e = (x,y)$ is an edge of E across the cut, where x is in X and y is in Y, and e has the minimum weight among all such crossing edges (called a **light** edge). Then there is an MST containing e.



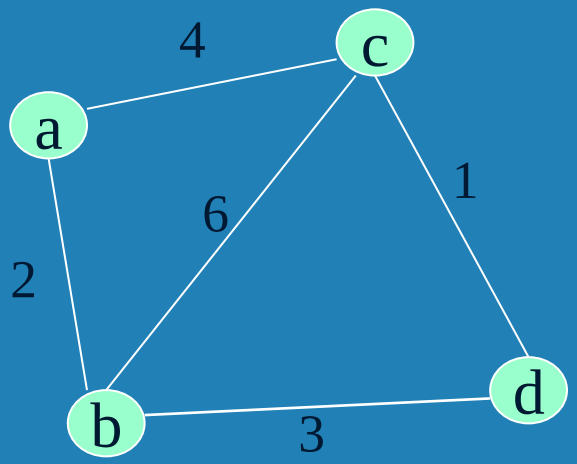
Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

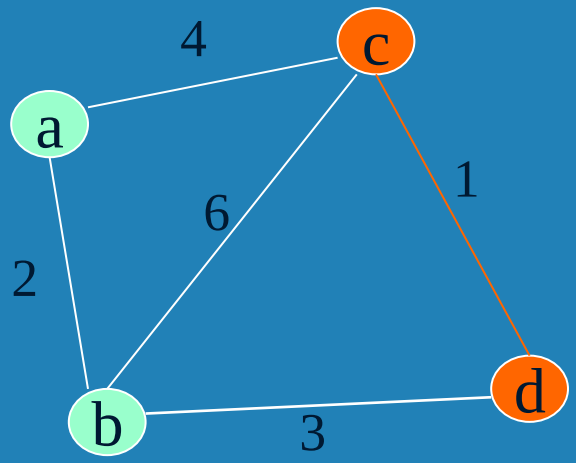
Another greedy algorithm for MST: Kruskal's

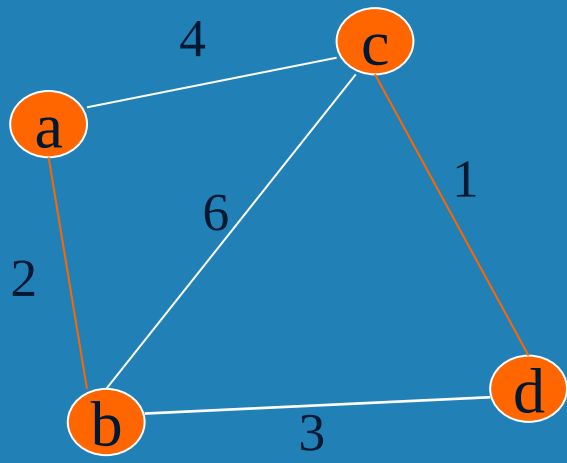
- Sort the edges in nondecreasing order of lengths
- "Grow" tree one edge at a time to produce MST through a series of expanding forests $F_1, F_2, \ldots, F_{n-1}$
- On each iteration, add the next edge on the sorted list unless this would create a cycle. (If it would, skip the edge.)

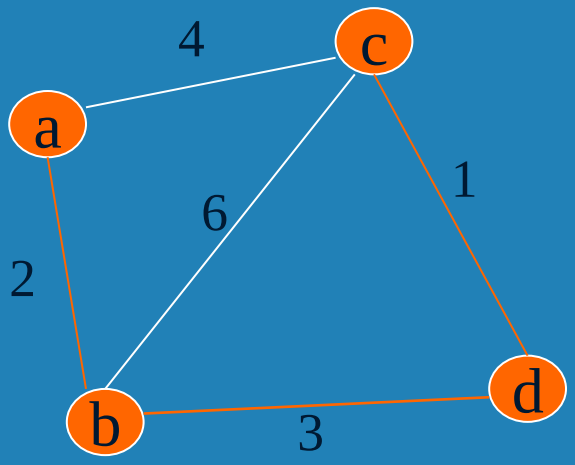
Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

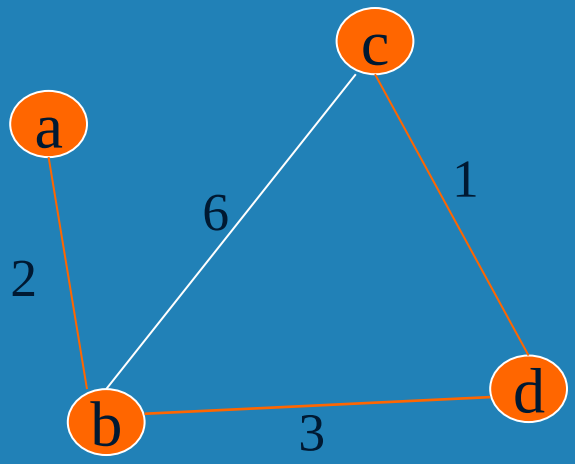
Example

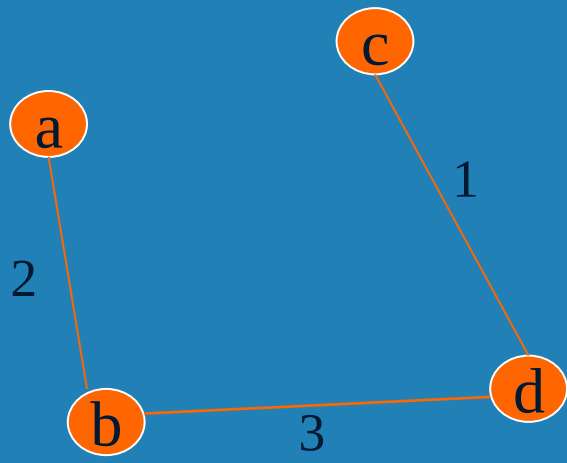






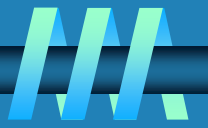






Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

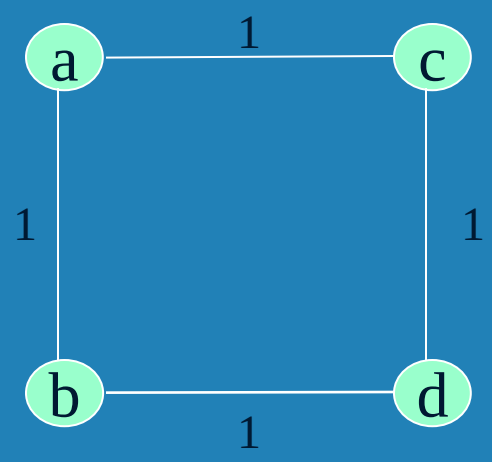
Notes about Kruskal's algorithm



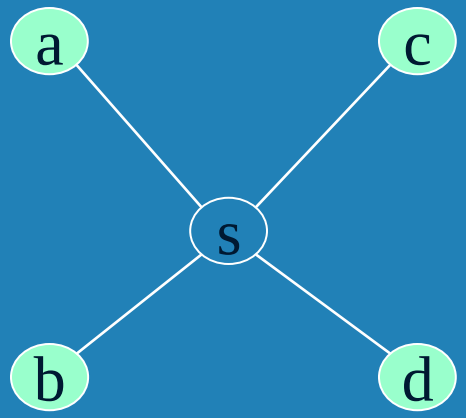
- Algorithm looks easier than Prim's but is harder to implement (checking for cycles!)
- Cycle checking: a cycle is created iff added edge connects vertices in the same connected component
- *Union-find* algorithms – see section 9.2
- Runs in $O(m \log m)$ time, with $m = |E|$. The time is mostly spent on sorting.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Minimum spanning tree vs. Steiner tree



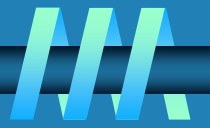
vs



In general, a Steiner minimal tree (SMT) can be much shorter than a minimum spanning tree (MST), but SMTs are hard to compute.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Shortest paths – Dijkstra's algorithm



***Single Source Shortest Paths Problem*: Given a weighted connected (directed) 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**

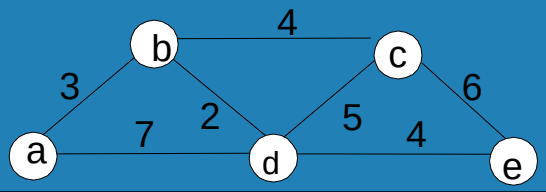
$$d_v + w(v,u)$$

where

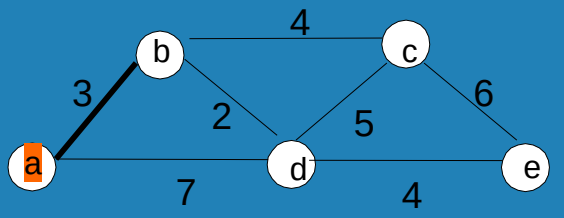
- ***v*** **is a vertex for which shortest path has been already found on preceding iterations (such vertices form a tree rooted at *s*)**
- d_v **is the length of the shortest path from source *s* to *v***
- $w(v,u)$ **is the length (weight) of edge from *v* to *u***

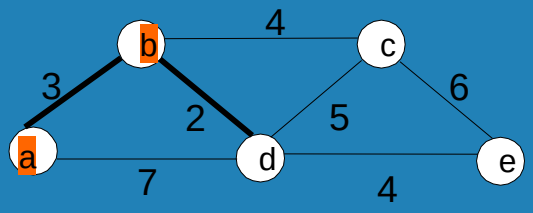
Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

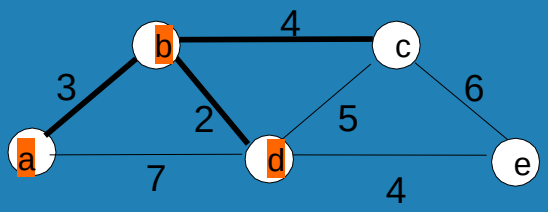
Example

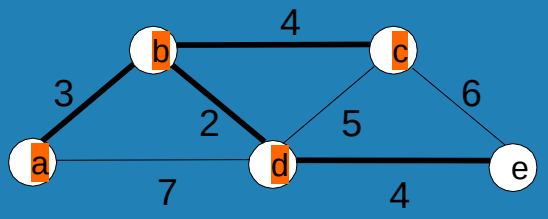


Tree vertices	Remaining vertices	
a(-,0)	b(a,3)	c(-,∞) d(a,7) e(-,∞)
b(a,3)	c(b,3+4)	d(b,3+2) e(-,∞)
d(b,5)	c(b,7)	e(d,5+4)
c(b,7)	e(d,9)	
e(d,9)		









Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Notes on Dijkstra's algorithm

- **Correctness can be proven by induction on the number of vertices.**

We prove the invariants: (i) when a vertex is added to the tree, its correct distance is calculated and (ii) the distance is at least those of the previously added vertices.

- **Doesn't work for graphs with negative weights (whereas Floyd's algorithm does, as long as there is no negative cycle). Can you find a counterexample for Dijkstra's algorithm?**

- **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! More details of the algorithm are in the text and ref books.**

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Coding Problem

***Coding*: assignment of bit strings to alphabet characters**

E.g. We can code {a,b,c,d} as {00,01,10,11} or {0,10,110,111} or {0,01,10,101}.

***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)**

E.g. if P(a) = 0.4, P(b) = 0.3, P(c) = 0.2, P(d) = 0.1, then the average length of code #2 is 0.4 + 2*0.3 + 3*0.2 + 3*0.1 = 1.9 bits

***Prefix-free codes (or prefix-codes)*: no codeword is a prefix of another codeword**

It allows for efficient (online) decoding!

E.g. consider the encoded string (msg) 10010110…

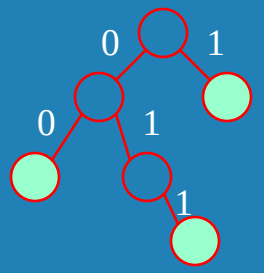
Problem: If frequencies of the character occurrences are known, what is the best binary prefix-free code?

The one with the shortest average code length. The average code length represents on the average how many bits are required to transmit or store a character.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

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



represents {00, 011, 1}

- Optimal binary tree minimizing the 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.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Example

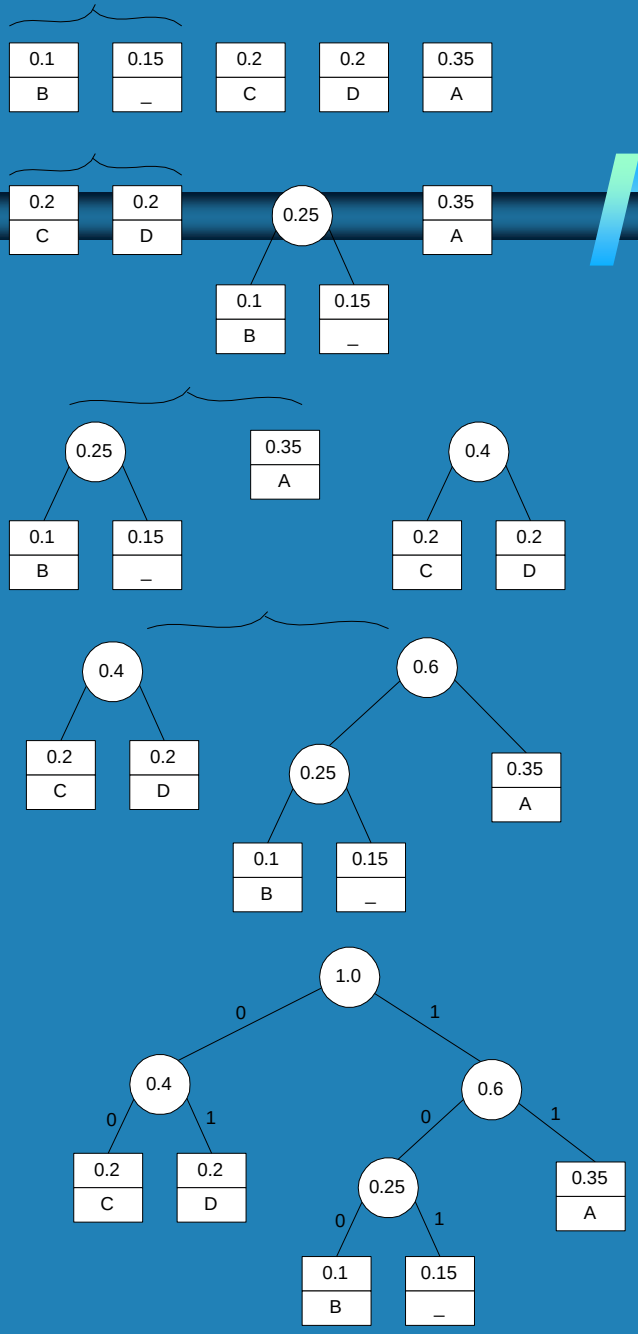
character	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

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%



Copyright © 2007 Pearson Addison-Wesley. All rights reserved.