CSE 2320 Notes 6: Greedy Algorithms

(Last updated 9/12/18 3:28 PM)

CLRS 16.1-16.3

6.A. CONCEPTS

Commitments are based on *local* decisions:

NO backtracking (will see in stack rat-in-a-maze - Notes 10)

NO exhaustive search (will observe in dynamic programming - Notes 7)

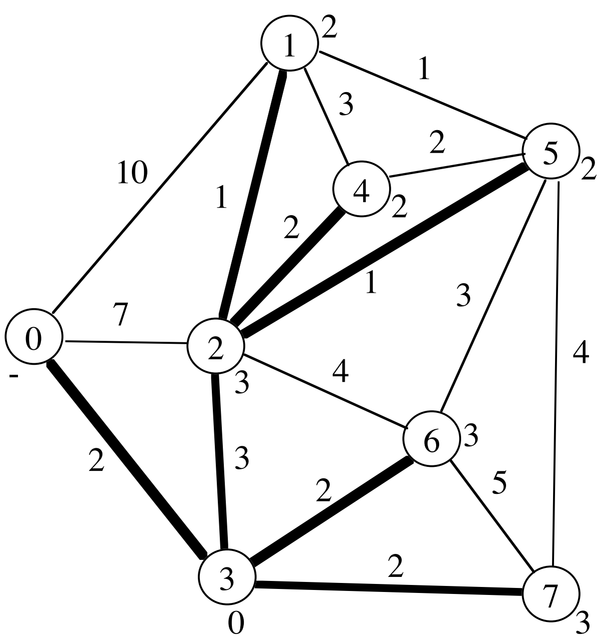
Approaches:

1. Sort all items, then make decisions on items based on ordering.
2. Items are placed in heap and then processed by loop with delete and priority changes.

MAIN ISSUE: NOT efficiency . . . Quality of Solution instead

Special situations - exact solution (these three path problems are asides for now . . .)

Prim's Minimum Spanning Tree (Notes 15, min-heap)



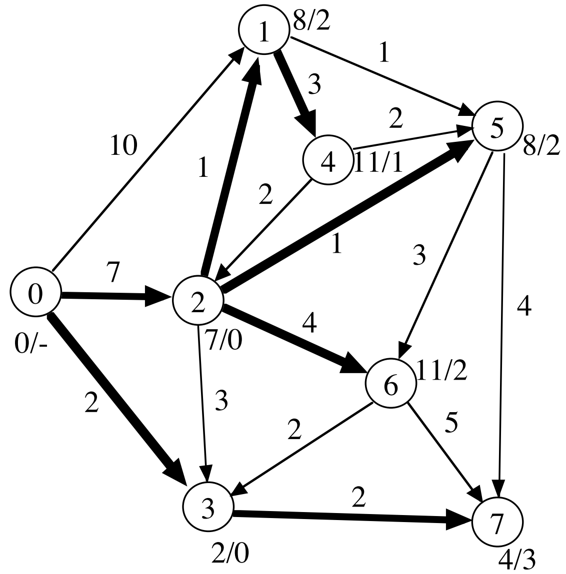
n vertices - choose *n* - 1 edges to give tree with minimum sum of (undirected) edge weights. Path for each vertex is one that minimizes the maximum weight appearing on the path.

Each vertex is labeled with its predecessor on path back to the source (vertex 0).

Each round augments the tree with the minimum weight edge.

So, vertices are finalized in ascending "min of maxes" order (0, 3, 6, 7, 2, 1, 5, 4).

Dijkstra's (`http://www.cs.utexas.edu/~EWD`) Shortest Path (Notes 16, min-heap)



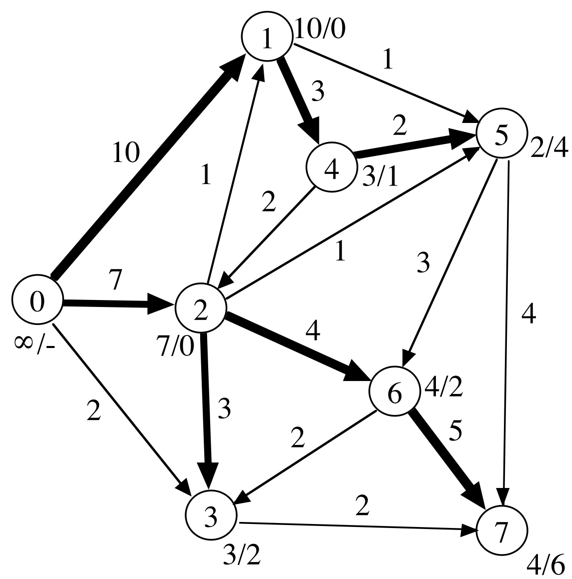
n vertices - choose *n* - 1 edges to give tree with a path from source to each vertex that minimizes the sum of (directed) edge weights on the path.

Each vertex is labeled with its shortest path distance from source and its predecessor.

Each round augments the tree with the last edge for the shortest (uncommitted) path.

So, vertices are finalized in ascending shortest-path distance order (0, 3, 7, 2, 1, 5, 4, 6).

Maximum Capacity Path for Network Flow (Notes 17, max-heap)



n vertices - choose *n* - 1 edges to give tree with path from source (0) to each vertex that maximizes the minimum capacity of the (directed) edge weights on the path.

Each vertex is labeled with its maximum capacity from source and its predecessor.

Each round augments the tree with the last edge for the maximum capacity (uncommitted) path.

So, vertices are finalized in descending maximum capacity order (0, 1, 2, 6, 7, 3, 4, 5).

More frequently - heuristic (approximation)

6.B. EXAMPLE – activity scheduling (unweighted interval scheduling)

n actitivites

Start time (activity starts *exactly* at time)

Finish time (activity finishes *before* this time)

One room

Goal: Maximize *number* of activities. (Unlike weighted interval scheduling in Notes 7)

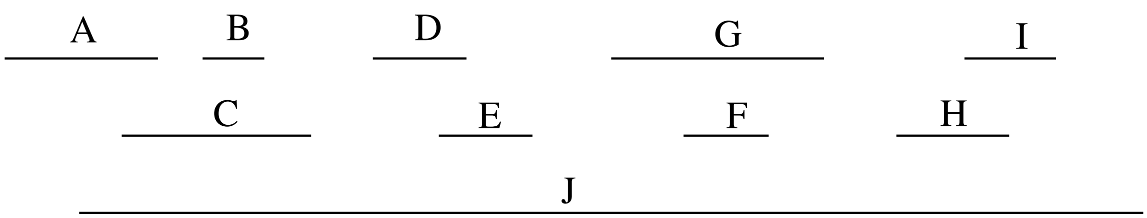
Greedy Solution:

1. Sort activities in ascending finish time order.

2. Consider each activity according to sorted order:

   Include activity in schedule only if it does not overlap with other activities already in schedule

Optimal or heuristic?



Optimality Proof:

1. Suppose there is an alternate (optimal) schedule with a different first activity:

   $s_? \ldots f_? <$ rest of schedule $>$

   But $s_1 \ldots f_1$ can replace $s_? \ldots f_?$ since $f_1 \leq f_?$

2. Same argument applies to replacing other activities in the schedule

Problems that can be solved optimally by a greedy method have a simpler structure than problems requiring dynamic programming.

6.C. Knapsack Problem

Can carry *k* pounds (to sell) in your knapsack.

Wish to maximize the amount of revenue.

Greedy approach: Choose according to descending order of $$$/lb.

Fractional (divisible) version:

$$$/lb for each divisible item.

Example:

k = 10 lbs

Perfume:	$500/lb, 1 lb available
Chocolate:	$30/lb, 5 lbs available
Beans:	$2/lb, 5 lbs available
Rice:	$1/lb, 5 lbs available

Optimal or heuristic?

0/1 (indivisible) version:

Example:

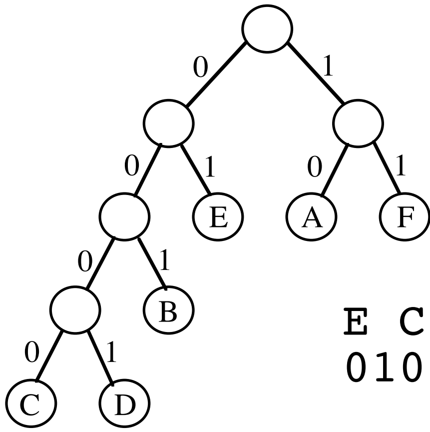
k = 10 lbs

Bottle of wine:	5 lbs, $100 ($20/lb)
Rare book:	6 lbs, $102 ($17/lb)
Sword:	4 lbs, $76 ($19/lb)
Lobster	2 lbs, $42 ($21/lb)

Greedy says to choose _______________, but optimal is ___________________.

6.D. HUFFMAN CODES - elementary data compression for a *static* distribution of symbols in an *alphabet*.

Prefix Code Tree



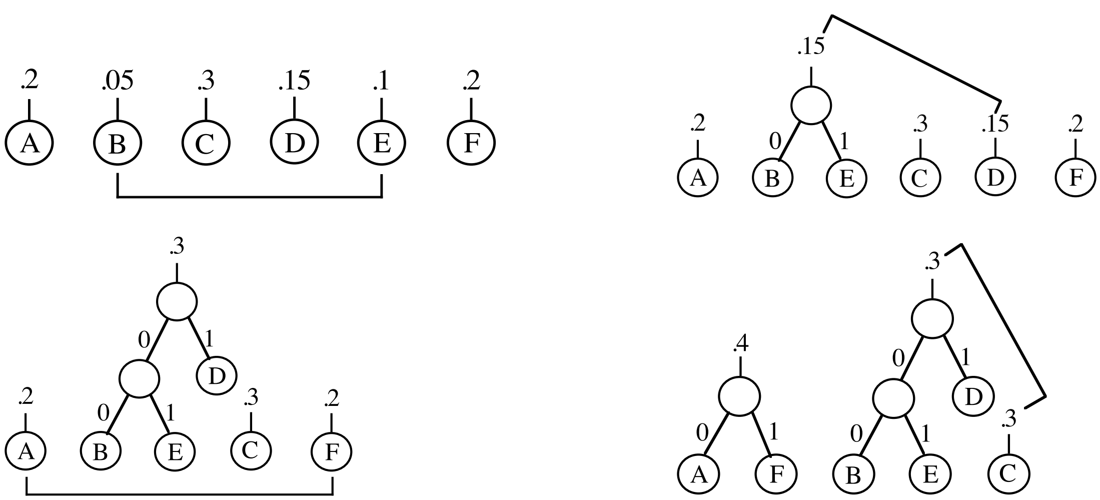

```
E C    D    F A E B
01000000011111001001
```

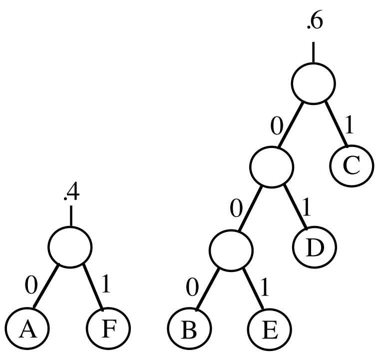
Concept: Letters that appear more often (higher probability) should be assigned shorter codes.

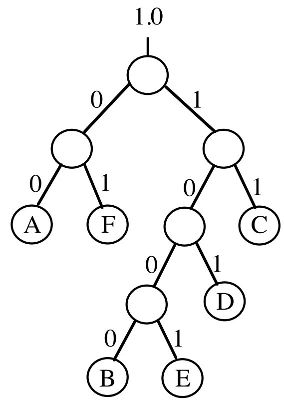
Evaluating a particular code tree (even if not optimal)

Symbol	Probability	Bits	Probability•Bits
A	.2	2	.4
B	.05	3	.15
C	.3	4	1.2
D	.15	4	.6
E	.1	2	.2
F	.2	2	.4
	===		===
	Σ=1.0		Σ=2.95= Expected bits per symbol

Algorithm: Build up subtrees by pairing trees with lowest probabilities (use min-heap).







Very easy to implement tree using table with $2n$ - 1 entries (`http://ranger.uta.edu/~weems/NOTES2320/huffman.c`):

i	probability	left	right
0	.2	-	-
1	.05	-	-
2	.3	-	-
3	.15	-	-
4	.1	-	-
5	.2	-	-
6	.15	1	4
7	.3	6	3
8	.4	0	5
9	.6	7	2
10	1.0	8	9

Symbol	Probability	Bits	Probability•Bits
A	.2	2	.4
B	.05	4	.2
C	.3	2	.6
D	.15	3	.45
E	.1	4	.4
F	.2	2	.4
	===		===
	Σ=1.0		Σ=2.45= Expected bits per symbol

Optimality: If two minimum-weight trees are *not* the ones combined, then the expected bits per symbol will be larger than would be computed by the algorithm.

Time: If there are n symbols, then there are n - 1 subtree combining steps to perform. Each step calls `heapExtractMin` twice and `minHeapInsert` once. $O(n \log n)$ overall.

(Aside, more in Notes 7) - Ordinary Huffman coding is not *order preserving*. The result of comparing two strings, before and after compression, may be different.

Using `strcmp()` on the strings:

```
X =  A    B    E    \0

Y =  A    B    F    \0
```

Using `memcmp()` on the bitstrings from the optimal Huffman code tree:

```
X =  00   1000 1001

Y =  00   1000 01
```

Under what condition will a Huffman code tree be order preserving?