# CS61B Spring 2015 Final Section (Amanda, Brendan, Japheth)

## Counting Sort and Radix Sort

**(1) Radix Sort Practice.** Use [least significant digit] radix sort to sort the following numbers: `32 75 17 55 52 57 2 10 51 30`. (Use a single digit for each pass of the sort.)

```
Sort on 1's:    10 30 51 32 52  2 75 55 17 57
Sort on 10's:    2 10 17 30 32 51 52 55 57 75
```

**(2) Choosing a Radix.** Suppose you are using radix sort to sort a list of Java `ints`, and it is important to sort them as quickly as possible. Explain why you would **never** choose to use exactly 512 buckets for this sort. (Source: Shewchuk Sp'05 Final Exam)

If we used 512 buckets for radix sort, then radix sort will inspect $\log_2 512 = 9$ bits per pass. However, Java `ints` are 32 bits. Using 512 buckets would require $ceil(32/9) = 4$ passes, but we could achieve the exact same result if we only inspected 8 bits per pass, using $2^8 = 256$ buckets.

**(3) Analysis.** What is the runtime and memory complexity of the following sorting algorithms? Let *n* be the number of items to sort, and *q* be the number of items in the alphabet.

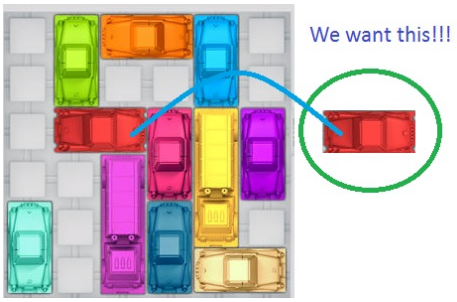
| | **Runtime Complexity** | **Memory Complexity** |
|---|---|---|
| **Counting Sort** | $\Theta(n+q)$<br><br>We must initialize all of the buckets (which takes $\Theta(q)$ time) and then insert each item (which takes $\Theta(n)$ time). | $\Theta(n+q)$<br><br>We keep an array of counts ($\Theta(q)$) and an array for the result ($\Theta(n)$). |
| **Radix Sort (LSD)** | $\Theta(ceil(\frac{b}{\log_2 q})(n+q))$<br><br>Let $b$ be the number of bits of the largest key. The number of passes is thus given by $ceil(\frac{b}{\log_2 q})$. Each pass of radix sort is like counting sort, so the time for each pass is $\Theta(n+q)$. | $\Theta(n+q)$<br><br>Effectively we are running counting sort multiple times. |

## Graph Traversals

**(4) Family Tree Traversal.** You are looking at your family tree, and and you want to traverse it with either DFS or BFS to find one particular person who is still alive. Which would you use?

DFS. It is safe to assume that someone who is alive is near the bottom levels of the tree, and DFS would get there faster than would BFS, except for a few edge cases at the bottom right of the tree.

**(5) Rush Hour.** Say we are trying to write a program that solves rush hour in the least number of moves. We wish to move the red car out of the grid using some series of moves. Assume we already have data structures for our Grid (including a `move(Car, Move)` method) and an algorithm implemented that outputs the next possible moves: `ArrayList<Move> getNextMoves(Grid)`.



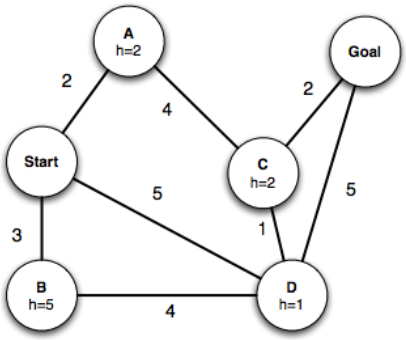

(a) How do we figure out which car we will end up moving next?

BFS and `getNextMoves()`. Since we want the least number of moves, we traverse with BFS. Even if there are 100 ways to get the red car, BFS ensures that it gets one of them with the least possible # of moves.

(b) What data structure will we need so that we do not have infinite loops of game state?

HashSet of seen states of the Grid.

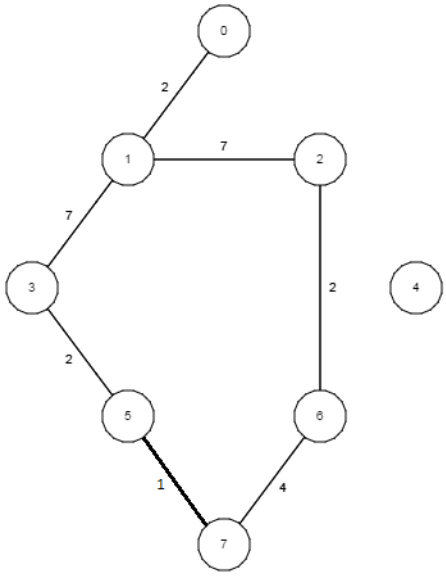
**(6) A* Search.** Perform A* search on the given graph, with the heuristic *h* and the edge weights to find the shortest path from the START to GOAL vertices. List the nodes in the order which we would visit them, as well as the path that A* search would return.




Nodes expanded: Start, A, D, C, Goal.

Path: Start, A, C, Goal.

**(7) Dijkstra's Algorithm.** Using Dijkstra's on the graph to the right, compute the cost of going from vertex 0 to any vertex.




Dijkstra's (a good visualization: https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html)

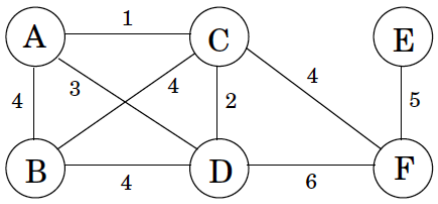
[Distances from 0-7]: 0, 2, 9, 9, ∞, 11, 11, 12

## Minimum Spanning Trees and Tries

**(8) Minimum Spanning Trees.** Under what circumstances could a graph have multiple valid minimum spanning trees? One valid minimum spanning tree?

We know a graph would have one valid minimum spanning tree if all its edge weights are unique. If a graph has multiple edges with the same weight, there may be more than one optimal solution. (Next problem)

**(9) Prim's and Kruskal's Algorithm.** List the edges of the minimum spanning tree of the graph below, in the order which they are added to the MST. Do so for Prim's Algorithm, then for Kruskal's Algorithm. (For Prim's algorithm, start at node A. For both, use an arbitrary tiebreaking scheme.) What is the total weight of this MST?




Kruskal's: AC, CD, AB, CF, EF
Prim's: AC, CD, AB, CF, EF

Total weight: 16

**(10) More MSTs.** Suppose you have a weighted graph where all edge weights are positive and distinct. Is it possible for a tree of shortest paths from some vertex s and the minimum spanning tree of the graph to not share any edges? If so, give an example. If not, give a reason why.

The lowest weight edge attached to s (let's say (s, u)) must be in the MST because of the cut property. It must also be in the shortest path tree because it is the shortest path from s to u.

**(11) Tries.** Prefix-free encoding is one where no code word is a prefix of another code word. Say we took a large array of words, compressed them using prefix-free encoding, then inserted them into a trie. What special property would this trie have? (Hint: What assumption can we now make about the nodes in this trie?)

A node marks the end of a full word if and only if it is a leaf node.