Final Review Document Solution

CS 61B Spring 2018

Antares Chen + Kevin Lin

Introduction

Wow this semester has gone by really fast. But before you guys can finish up this class and beat the game, there is one last boss - the final! The final is going to be cumulative across all the topics we've covered this semester. Anything that was mentioned in class is fair game.

Now, this packet will provide supplementary problems on topics covered after midterm 2. Like all previous packets, these questions are ranked by difficulty, but in no way is this packet comprehensive! Most notably, it lacks coding challenges since it's targeted to test your conceptual understanding of the topics. Further, it lacks a few hard questions since the questions I've thought of would probably detract from your studying.

For this final, I want you all to be smart, use your best judgement, and do the work! You have a little time left to really nail down this studying thing and with that I believe you guys have a great chance to really knock this test out of the ballpark. You may have faltered at times during this semester and things might have been hard. But like the great fisherman standing in the frozen sea once said, "**Never give up!**"

Think of all the people cheering you on.

Go out, read the textbook, find practice midterms and really work through them with your friends. Talk to the TA's, go to office hours, and use all the resources around you. Needless to say it is quite useless to continue banging your head against the books if the material doesn't stick.

Finally, if you find things becoming too stressful, simply step back, take a deep breath and go out for a long walk. Come back, remember that you're awesome and get back to work. Believe in yourself! And if you don't then believe in us who believe in you.

Introduction

Graphs
- Easy Mode
- Medium Mode
- Hard Mode

Dynamic Programming
- Easy Mode
- Medium Mode

Disjoint Sets and MST
- Easy Mode

- Medium Mode
- Hard Mode

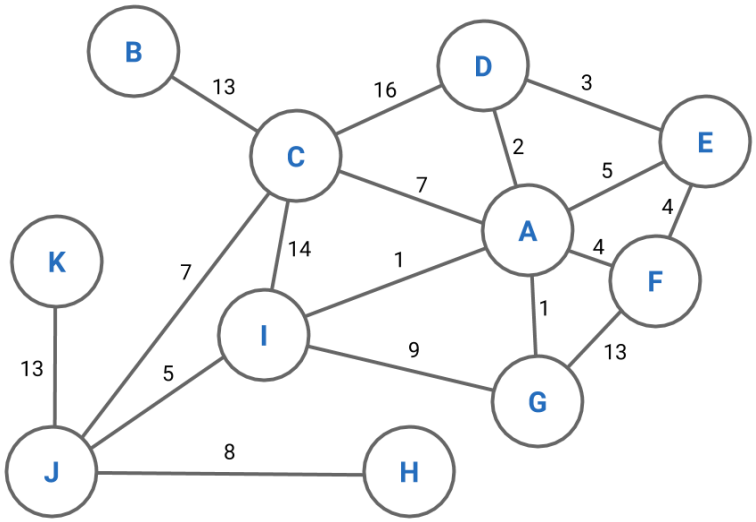
Out of Sorts
- Easy Mode
- Medium Mode
- Hard Mode

“Yesterday you said tomorrow; so just do it.” - Abraham Lincoln

Graphs

Easy Mode

Traversal Time Write out the listed traversals starting at node A. Break ties alphabetically.



1) Breadth First Search

   BFS: A C D E F G I B J H K

2) Depth First Search

   DFS: A C B D E F G I J H K

3) Dijkstra's

   Dijkstra's: A G I D F E J C H K B

Runtime Funtime (More Fun Times) For each algorithm, list the worst case runtime.

BFS	DFS	Dijkstra's	A*	Topological Sort
Theta(V + E)	Theta(V + E)	Theta((V + E) log V)	Theta((V + E) log V)	Theta(V + E)

And for each representation fill in the runtimes.

	Storage size	Add vertex	Add edge	Remove vertex	Remove edge	Query if edge exists
Adjacency List	Theta(V + E)	Theta(1)	Theta(1)	Theta(V + E)	Theta(E)	Theta(E)
Adjacency Matrix	Theta(V^2)	Theta(V^2)	Theta(1)	Theta(V^2)	Theta(1)	Theta(1)

Last Question When would you use an adjacency matrix over an adjacency list?

When the graph is very dense, or where the number of edges, |E|, approaches |V^2|.

Medium Mode

Graph Questions Answer if each problem is always true, sometimes true, or never true.

1) Given a graph with V vertices and V-1 edges, adding an edge would introduce a cycle

   Always true by the definition of a tree.

2) A DFS traversal starting at vertex u ending at v always finds the shortest path in an unweighted graph.

   Sometimes true. If there is only a single path, then DFS will find that path (which happens to also be the shortest path). If there are multiple paths, then the returned path may not necessarily be the shortest.

3) Dijkstra's finds the shortest path for a graph with negative edge weights.

   Sometimes true. Try to construct several examples with three or four nodes with one negative edge weight.

4) A DFS on a directed acyclic graph produces a topological sort.

   Sometimes true. It works on a linear graph but not in general.

Dijkstra's Algorithm The runtime for Dijkstra's algorithm is O((V + E) log (V)); however this is specific only to binary heaps. Let's provide a more general runtime bound. Suppose we have a priority queue backed by some arbitrary heap implementation. Given that this unknown heap contains N elements, suppose the runtime of remove-min is f(N) and the runtime of change-priority is g(N).

1) What is the runtime of Dijkstra's in terms of f(V) and g(V)?

   In the worst case, we check every edge and decrease or increase the key of a node in the heap. In addition, we also always need to remove all vertices from the heap. The overall runtime is in O(Eg(V) + Vf(V))

2) Turns out the optimal version of Dijkstra's algorithm uses something called a fibonacci heap. The fibonacci heap has amortized constant time change-priority, and log time remove-min. What is the runtime of Dijkstra's algorithm?

   O(E + V log V)

Hard Mode

Algorithm Design Provide an algorithm matching the given time bound for each problem.

1) Suppose you are terribly lost in downtown SF (basically a maze), but happen to have an infinite amount of pennies (you made a killing investing in Kelp, a hot new Yelp-for-seafood startup that recently IPO'd). Discuss how you would get out of SF by solving the following problem.

   Let G be an undirected connected graph. Give an O(V + E) time algorithm to compute a path in G that traverses each edge in E exactly once in each direction.

   Modify depth first search. First, mark each edge with directions taken. If both directions exist, then remove the edge. To ensure all unexplored edges are explored, check unexplored edges before exploring singly traversed edges. Finally, when the algorithm gets stuck, backtrack to a predecessor node and try again.

2) A directed graph G is singly connected if for all vertices u, v, u connected to v implies that there is at most one simple path from u to v. Provide an efficient algorithm to determine whether or not a directed graph is singly connected.

```
for all unvisited vertices:
    visit vertex:
        run DFS from the vertex
        if vertex is visited twice:
            return False
return True
```

Dynamic Programming

Easy Mode

Count... Stairways? Rewrite this exponential-time procedure as a dynamic program.

```
public static int slowlyCountStairWays(int n) {
    if (n == 1) {
        return 1;
    } elif (n == 2) {
        return 2;
    } else {
        return slowlyCountStairWays(n - 1) + slowlyCountStairWays(n - 2);
    }
}

public static int quicklyCountStairWays(int n) {
    if (n == 1) {
        return 1;
    }
    int[] ways = new int[n + 1];
    ways[1] = 1;
    ways[2] = 2;
    for (int i = 3; i <= n; i++) {
        ways[i] = ways[i - 1] + ways[i - 2];
    }
    return ways[n];
}
```

Medium Mode

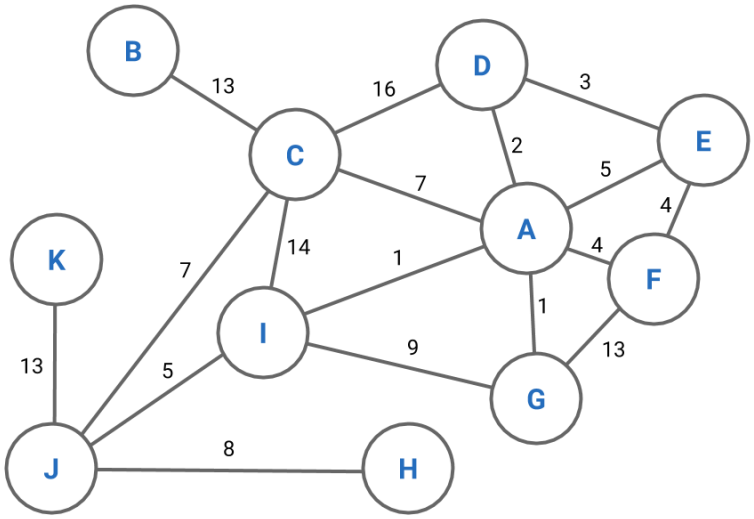
Tree Recursion <3, Never Change Once the machines take over, the denomination of every coin will be a power of two: 1-cent, 2-cent, 4-cent, 8-cent, 16-cent, etc. There will be no limit to how much a coin can be worth. Write a bottom-up dynamic program, `countChange`, that takes a positive integer `n` and returns the number of ways to make change for `n` using these coins of the future. Compute bottom-up; do not use memoization.

```
public static int countChange(int n) {
    int[] ways = new int[n + 1];
    ways[0] = 1;
    for (int coin = 1; coin <= n; coin *= 2) {
        for (int i = coin; i <= n; i++) {
            ways[i] += ways[i - coin];
        }
    }
    return ways[n];
}
```

Disjoint Sets and MST

Easy Mode

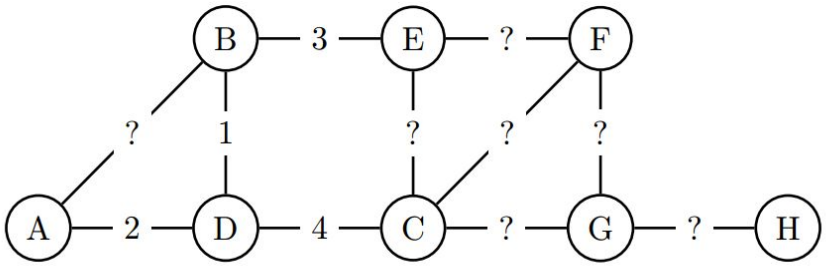
Run Forest Run For the following graph, draw the MST.



Now run Kruskal's algorithm using a disjoint set structure on the graph above. After 5 iterations, what does the disjoint set data structure look like?

Medium Mode

Hidden Weight In this graph, some of the edge weights are known, while the rest are unknown.



$$cost(A, D) = 2, cost(B, D) = 1, cost(C, D) = 4, cost(B, E) = 3$$

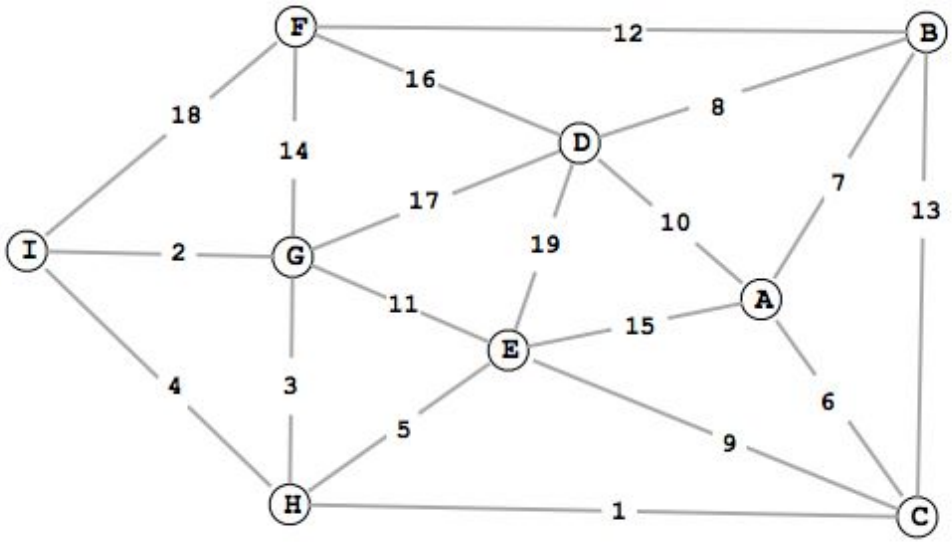
List all edges that must belong to a minimum spanning tree, regardless of what the unknown edge weights turn out to be. Justify your answers.

(G, H): This is the only edge that can connect H to any other vertex, so it must be included in any MST. Remember that any MST must span all of the vertices.

(B, D): The cut property. This is the minimum weight edge between D and the rest of the vertices in the graph.

(B, E): Also by cut property. This is the minimum weight edge between the A-B-D tree and the rest of the vertices in the graph.

Princeton Schminceton Answer the following questions for the graph below. Note that the edge weights are distinct with values ranging from 1 to 1.



1) Complete the sequence of edges in the MST in the order that Kruskal's algorithm includes them.

   1 2 3 5 6 7 8 12

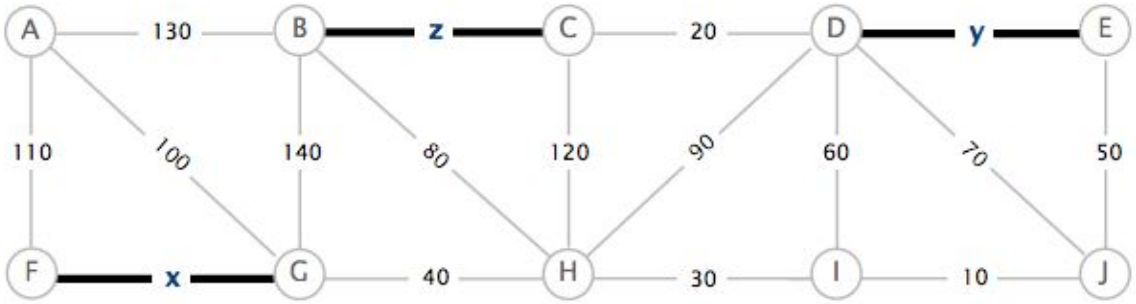
2) Suppose that the edge D-I of weight w is added to the graph. For which values of w is the edge D-I in a MST?

   w <= 8 as there already exists an edge on the cut between D and the rest of vertices in the graph of weight w = 8.

3) Given a minimum spanning tree T of a weighted graph G, describe an O(V) algorithm for determining whether or not T remains a MST after an edge (x, y) of weight w is added.

   By the tree property, consider the (one and only) path between x and y. For each edge on that path, if it's the case that w is less than the weight of any one edge on that path, then the MST will change. This is in O(V) as the number of edges in a tree is equal to the number of vertices less one.

Princeton Part 2 Suppose that a MST of the following edge-weighted graph contains the edges with weights x, y, and z.



1) List the weights of the other edges in the MST in ascending order of weight

   10 20 30 40 50 100

2) Circle which one or more of the following can be the value of x?

   5 15 25 35 45 55 65 75 85 95 105 115 125 135 145

3) Circle which one or more of the following can be the value of y?

   5 15 25 35 45 55 65 75 85 95 105 115 125 135 145

4) Circle which one or more of the following can be the value of z?

   5 15 25 35 45 55 65 75 85 95 105 115 125 135 145

Constant Kruskal's Suppose that for a given graph G, the weights of its edges are in the range 1 to V where V is the number of vertices. How fast can Kruskal's algorithm run?

Use a counting sort to speed up the sorting time which now takes O(E + V) time. Then, adding to the tree and managing the disjoint sets takes time in O(E * alpha(V)) where alpha is the inverse ackermann function.

What if the edge weights are integers in the range 1 to W where W is an asymptotic variable?

Like above, there are two expensive phases: sorting and then disjoint set management. The total runtime is in O(E + W + E * alpha(V)).

Hard Mode

Random MST Properties State if the following are true or false. If they are true, provide an informal argument else provide a counterexample.

1) Suppose G is a graph and T is a MST of G. If we decrease the weight of any edge in T, then T is still a MST.

   Yes. We know that if T is an MST of G with total weight W, then it must be the case that all other valid MSTs of the graph now have total weight W - weight(edge) just like T.

2) A graph has a unique minimum spanning tree if, for every cut of the graph (every way you can split a graph into two separate parts), there is a unique light edge crossing the cut.

   True. Suppose this is false. We can show that there exists two distinct MSTs and then show that there exists a cut with two light edges crossing it.

Update MST Suppose instead of decreasing the weight of an edge in the MST, we decrease the weight of any random edge not in the MST. Give an efficient algorithm for finding the MST in the modified graph.

Remember that MSTs are still trees! There is only one unique path between any two vertices.

Suppose an edge (u, v) chosen at random is decreased. First, find a path from u to v in the MST. Then, add the edge (u, v), creating a cycle. Traverse the cycle and remove the edge with maximum weight. This will break the cycle and result in the new MST.

Out of Sorts

Easy Mode

Classify the Sorts Answer the following questions assuming you have some input lists where N is the list's size.

1) List all the sorts that run worst case $O(N^2)$.

   Insertion sort, selection sort, quicksort.

2) List all the sorts that run worst case O(N log N).

   Heap sort, merge sort.

3) List all the stable sorts.

   Insertion sort, selection sort, merge sort, LSD radix sort.

Fill the Table Fill in the cells of the following table.

Algorithm	Best case runtime	Worst case runtime	Best-case example	Worst-case example
Heap Sort	Theta(N)	Theta(N log N)	A list with all the same element; no bubbling is necessary.	Any list where we need to bubble down on all removals.
Quicksort	Theta(N log N)	Theta(N^2)	Good pivot that is near the median of the list.	Bad pivot that is an extreme value of the list.
Merge Sort	Theta(N log N)	Theta(N log N)	Always the same order of growth for a list of length N.	Always the same order of growth for a list of length N.
Selection Sort	Theta(N^2)	Theta(N^2)	Always the same order of growth for a list of length N.	Always the same order of growth for a list of length N.
Insertion Sort	Theta(N)	Theta(N^2)	Sorted list, number of inversions K < N.	Reverse-sorted list, K = N^2.
Radix Sort	Theta(N)	Theta(N)	Runtime depends on alphabet size and key length.	Runtime depends on alphabet size and key length.

Run the Sorts Listed below are partially sorted lists, that would occur in the middle of running a sort on the input list. For each partially sorted list, name the algorithm that may have sorted it. Choices include heapsort, quicksort (assuming the first element as the pivot), mergesort, insertion sort, LSD radix sort, MSD radix sort and selection sort.

1) 12, 7, 8, 4, 10, 2, 5, 34, 14
   7, 8, 4, 10, 2, 5, 12, 34, 14
   4, 2, 5, 7, 8, 10, 12, 14, 34

   Quicksort. Pivots in red.

2) 23, 45, 12, 4, 65, 34, 20, 43
   4, 12, 23, 45, 65, 34, 20, 43

   Insertion sort.

3) 12, 32, 14, 11, 17, 38, 23, 34
   12, 14, 11, 17, 23, 32, 38, 34

   MSD radix sort.

4) 45, 23, 1, 65, 34, 3, 76, 25
   23, 45, 1, 65, 3, 34, 25, 76
   1, 23, 45, 65, 3, 25, 34, 76

   Merge sort.

5) 23, 44, 12, 11, 54, 33, 1, 41
   54, 44, 33, 41, 23, 12, 1, 11
   44, 41, 33, 11, 23, 12, 1, 54

   Heap sort.

Average Case Runtimes What are the average case runtimes for quicksort and insertion sort?

Theta(N log N) for quicksort and Theta(N^2) for insertion sort. For quicksort, consider the probability of choosing worst-case pivots every time for large N. For insertion sort, we can make an argument for the expected number of inversions in any given dataset.

Medium Mode

Conceptual Questions

1) Give an example of when insertion sort is more efficient than mergesort.

   When the input is small, almost sorted, or has few inversions.

2) When would you use mergesort over quicksort?

   Merge sort is a stable sort.

3) When would you use radix sort over a comparison sort?

   When the alphabet is well-defined and each object can be individually compared on each radix.

Design Questions

1) You have an array of integers where the length = N and the maximum number of digits is given by K. Suppose that K >> N (that is K is much larger than N). What sort would you use?

   Quicksort since the input is an array of integers where stability is not a concern.

2) Assume for the previous question that you used LSD radix sort. What is the runtime?

   Because K >> N, the runtime will be in O(K(N + R)) which is in O(K) for this problem.

3) Suppose you are given an integer array with length = N such that there are N - sqrt(N) copies of the minimum element located in the front of the array. What is the worst case runtime of insertion sort on this list?

   The total number of inversions can be given by the arithmetic sum to sqrt(N) which is in Theta(N). Insertion sort on the remaining elements takes Theta(N) time to resolve each inversion. The total runtime is in Theta(N + N) or Theta(N).

4) For the same construction, what is the worst case runtime of selection sort?

   Still Theta(N^2) because we need to find the minimum element in each iteration.

Hard Mode

CS 61B Lane

The rent is too dang high in Berkeley and so you decide to buy a house located at the idyllic CS 61B Lane. CS 61B Lane is, of course, a friendly neighborhood and so everyone who lives on CS 61B Lane is automatically friends with her two closest neighbors.

For example, on CS 61B Lane we may have houses for: Aidan, Kevin, Kaylee, Ching, Alex, and Matt[1]. Here Kaylee is friends with Aidan, Kevin, Ching and Alex while Alex is friends with Kaylee, Ching, and Matt.

Let us represent CS 61B Lane as an array of House objects. Each House object has two instance variables: the owner's name and the address. Now given an array of N houses in *no particular order*, give an algorithm that returns a data structure such that when you query the data structure with a person's name, it will return to you a list of friends.

Suppose you are given an array of objects *House* in no particular order. Each House object has two fields: the owner's name and the address. Now suppose we are given the array for CS 61B Lane. However, CS 61B Lane is a friendly neighborhood, so every owner is friends with the two closest houses.

Provide a the best possible. runtime bound for construction of and querying from the data structure.

Create a HashMap<String, LinkedList<String>> and then sort the houses by address. For each house, create a list of the two closest houses to its left and right and put them into the HashMap.

Construction takes O(N log N) mainly for sorting the list. Querying is a constant time operation to index into the HashMap.

[1] Antares actually lives in a box off the side of Euclid...


Across The Vast
Expanse of Space
Memes Echo
Into Eternity