Lecture 24: Final Review

CSE 373 – Data Structures and Algorithms

Administrivia

Please fill out:
- Course evaluations (for both section and lecture) so the TAs and I can improve
- Our custom feedback form https://docs.google.com/forms/d/e/1FAIpQLSfeTAECzu98RSgPfiAIYL8zlCcE1_mIFFp6gTJLlctMxoehFA/viewform so we can improve the course, and you can get extra credit!

There's a topics list:
- https://courses.cs.washington.edu/courses/cse373/19su/files/exams/topics_19su.pdf (It's a pdf linked on the exams page instead of just being on the exams page)

Go to your officially registered section tomorrow for part 1!
- Remember no note sheet tomorrow
- But you will have the same identities sheet as the midterm.

Part 2 will be here on Friday
- Note sheet with same rules as midterm (one 8.5" x 11" sheet of paper)
- And we'll give you the same identities sheet again.

5 minutes

Disneyland Scenario #3

You're a Disneyland employee, working the front of the Splash Mountain line. Suddenly, the crowd-control gates fall over and the line degrades into an unordered mass of people.

Sometimes you can tell who was in line before who; for other groups you aren't quite sure. You need to restore the line, while ensuring if you **knew** A came before B before the incident, they will still be in the right order afterward.

What are the vertices?

What are the edges?

What are we looking for?

What do we run?

Disneyland Scenario #3

You're a Disneyland employee, working the front of the Splash Mountain line. Suddenly, the crowd-control gates fall over and the line degrades into an unordered mass of people.

Sometimes you can tell who was in line before who; for other groups you aren't quite sure. You need to restore the line, while ensuring if you **knew** A came before B before the incident, they will still be in the right order afterward.

What are the vertices?

People

What are the edges?

Edges are directed, have an edge from X to Y if you know X came before Y.

What are we looking for?

- A total ordering (i.e. a line) consistent with all the ordering we do know.

What do we run?

- Topological Sort!

7 minutes

More Graph Modeling

You have three jars, one holds 3 cups, another 5 cups, and the last 9 cups.

You can:

- Fill a jar with water (to the top)
- Dump all the water from a jar (leaving none inside)
- Pour the contents of one jar to another. Stopping when either the first jar runs out or the second jar becomes full.

Your goal is to get exactly 7 cups of water into the jar that holds up to 9 cups, and no water in both of the other jars.

Describe an algorithm to find a set of movements to do (or determine that none exists)

Hint: this is a graph modeling problem!!

More Graph Modeling

You have three jars, one holds 3 cups, another 5 cups, and the last 9 cups.

What are the vertices?

- Have a vertex for each possible "state" we could be in
- e.g. "jar 1 contains 2 cups of water, jar 2 contains 4 cups of water, jar 3 is empty" Call this state (2,4,0)

What are the edges?

- Have an edge from state A to state B if a single operation will take you from A to B
- For example, we have an edge from: (2,4,0) to (1,5,1) because we can pour from jar 1 into jar 2 until it's full and enter the second state.
- Edges are directed (e.g. we can't get from (1,5,1) back to (2,4,0)

What are we looking for?

- A set of steps to get from (0,0,0) to (0,0,7)

Algorithm?

- Just need a path (not the shortest one). To make simpler, add a "dummy target" accessible from any good final state. Now run [B/D]FS from (0,0,0) and see if you can get to (0,0,7)

5 minutes

Even More Graph Modeling

You and your friends are going to get dinner on the Ave after the final. You'll leave straight from the exam, but you don't know which of the restaurants on the Ave you'll go to yet. You'd like to know in advance how to get from the exam to any of the restaurants as quickly as possible.

What are the vertices?

What are the edges?

What are we looking for?

Algorithm?

Even More Graph Modeling

You and your friends are going to get dinner on the Ave after the final. You'll leave straight from the exam, but you don't know which of the restaurants on the Ave you'll go to yet. You'd like to know in advance how to get from the exam to any of the restaurants as quickly as possible.

What are the vertices?

- Vertex for PAA, every restaurant on the Ave, and probably a bunch of "locations" in-between

What are the edges?

- Sidewalks or any other direct connections between our vertices.
- Weighted? – yes! Time it takes to walk along the sidewalk
- Directed? – Maybe. Does it take longer to walk one direction than the other? If not undirected is fine.

What are we looking for?

- Shortest path from PAA to every restaurant

Algorithm?

- Dijkstra's, with PAA as source. Automatically find distances to all possible targets. Can reconstruct paths using predecessor pointers for any destination we need.

Code Modeling

What is the in-practice running time of this code?

3 minutes

```
//counts the number of vertices reachable from source.
public void countElements(Graph g, Vertex source){
      ChainedHashSet<Vertex> seen = new ChainedHashSet<>();
      Queue<Vertex> toProcess = new Queue<>();
      toProcess.enquque(source);
      seen.put(source);
      int numReachable = 1; //count v itself.
      while(!v.isEmpty()){
            Vertex curr = toProcess.dequeue();
            for(Vertex v : curr.outNeighbors()){
                  if( !seen.contains(v) )
                        seen.put(v);
                        count++;
            }
      }
      return count;

}
```

Code Modeling

What is the in-practice running time of this code?

```
//counts the number of vertices reachable from source.
public void countElements(Graph g, Vertex source){
        ChainedHashSet<Vertex> seen = new ChainedHashSet<>();
        Queue<Vertex> toProcess = new Queue<>();
        toProcess.enquque(source);
        seen.put(source);
        int numReachable = 1; //count v itself.
        while(!v.isEmpty()){
            Vertex curr = toProcess.dequeue();
            for(Vertex v : curr.outNeighbors()){
                    if( !seen.contains(v) )
                            seen.put(v);
                            count++;
                    }
        }
        return count;

}
```

Happens at most n times each call is $O(1)$ time.

Happens at most m times
In-practice, each call is $O(1)$ time.

Total in-practice time: $O(m+n)$

Code Modeling

What is the worst-case running time of this code?

2 minutes

```
//counts the number of vertices reachable from source.
public void countElements(Graph g, Vertex source){
      ChainedHashSet<Vertex> seen = new ChainedHashSet<>();
      Queue<Vertex> toProcess = new Queue<>();
      toProcess.enquque(source);
      seen.put(source);
      int numReachable = 1; //count v itself.
      while(!v.isEmpty()){
            Vertex curr = toProcess.dequeue();
            for(Vertex v : curr.outNeighbors()){
                  if( !seen.contains(v) )
                        seen.put(v);
                        count++;
            }
      }
      return count;

}
```

Code Modeling

What is the worst-case running time of this code?

```
//counts the number of vertices reachable from source.
public void countElements(Graph g, Vertex source){
        ChainedHashSet<Vertex> seen = new ChainedHashSet<>();
        Queue<Vertex> toProcess = new Queue<>();
        toProcess.enquque(source);
        seen.put(source);
        int numReachable = 1; //count v itself.
        while(!v.isEmpty()){
            Vertex curr = toProcess.dequeue();
            for(Vertex v : curr.outNeighbors()){
                    if( !seen.contains(v) )
                            seen.put(v);
                            count++;
                    }
        }
        return count;

}
```

Happens at most n times each call is $O(1)$ time.

Happens at most m times
Worst-case, most contains calls take $\Theta(n)$ time (everything collides)

Total in-practice time: $O\big(m(m+n)\big) = O(m^2 + mn)$

3 minutes

Sorting Design Decision

You and your friends can never agree on which of the n restaurants on the Ave to eat dinner at. You made a dataset of all the restaurants and their important properties.

You want to be able to sort the dataset, under the following circumstances:

1. A friend will yell out what they care about (e.g. show me them ordered by average meal price), and you want to ensure that when someone new yells a new requirement you break ties with the previous ordering.

2. Your friends get impatient. You need to ensure no single execution of the sort takes too long.

Sorting Design Decision

You and your friends can never agree on which of the n restaurants on the Ave to eat dinner at. You made a dataset of all the restaurants and their important properties.

You want to be able to sort the dataset, under the following circumstances:

1. A friend will yell out what they care about (e.g. show me them ordered by average meal price), and you want to ensure that when someone new yells a new requirement you break ties with the previous ordering.

2. Your friends get impatient. You need to ensure no single execution of the sort takes too long.

Requirement 1 means a **stable sort** is required. Requirement 2 means you want worst-case $O(n \log n)$. **Merge sort** meets both of those requirements (none of our other sorts do).

4 minutes

Design Decision

You and your friends can never agree on which of the n restaurants on the Ave to eat dinner at. Tired of all the yelling and sorting, you decide on another solution.

All of you will rank the restaurants from 1 to n, and a value k will be chosen, then:

```
For(each friend f)
    For(i = 0; i < k; i++)
       if only one restaurant remains, go there.
       else remove f's least favorite remaining restaurant as a possibility
    End For
 End For
 If more than one restaurant remains, pick a remaining restaurant at random.
```

What ADT(s) would you want to use in this problem?

What data structures would you use to implement them?

How do you use these structures to execute this algorithm? What is the running time of the steps you have to do? (Don't try to find the overall time of the algorithm; it depends on way too many factors to cleanly analyze)

3 minutes

Design Decision

Many possibilities. Here are two:

Doubly-Linked list implementation of List ADT; List contains restaurants objects that know everyone's rankings

- Finding restaurant to delete involves iterating over list to find restaurant to delete ($O(n)$)
- And deleting it $O(1)$

Heap implementation of priority queue; queue contains restaurant objects that know everyone's rankings

- When we start processing a new person, buildHeap according to their preferences $O(n)$
- To remove least favorite restaurant just remove min $O(\log n)$

2 minutes

Design Decision

Here's another!

Have a set of remaining restaurants and a list for each person's preferences.

Use a hash set for the restaurants.

To choose:

- Iterate from the end of each person's preference, if the set still has the restaurant, remove it. Else move up one in the list until you've deleted k.

- Worst case lookup in the lists? $O(1)$ (either use an arraylist, or implement a custom "reverse order" iterator)
- In-practice time for each lookup and delete? $O(1)$
- Worst case time for each lookup and delete? $O(n)$

P/NP

We'll only ask multiple choice questions on P vs. NP.

P (stands for "Polynomial")

The set of all decision problems that have an algorithm that runs in time $O(n^k)$ for some constant k.

NP (stands for "nondeterministic polynomial")

The set of all decision problems such that if the answer is YES, there is a proof of that which can be verified in polynomial time.

NP-complete

Problem B is NP-complete if B is in NP and
for all problems A in NP, A reduces to B in polynomial time.

3 minutes

Sample questions:

Which of the following problems is NP-complete?
a) 3-SAT
b) Minimum Spanning Tree
c) 2-Coloring

Which of the following would solve the P vs. NP?
a) Finding a polynomial time algorithm for 3-SAT
b) Showing there is no polynomial time algorithm for 2-SAT
c) Reducing 2-SAT to 3-SAT

Which of the following is not a thing to try when you think the problem you want to solve might be NP-complete?
a) See if your problem is a special case that does have an efficient algorithm.
b) Try an approximation algorithm.
c) Implement a polynomial time algorithm for 3-SAT

Sample questions:

Which of the following problems is NP-complete?

a) 3-SAT

b) Minimum Spanning Tree

c) 2-Coloring

Which of the following would solve the P vs. NP?

a) Finding a polynomial time algorithm for 3-SAT

b) Showing there is no polynomial time algorithm for 2-SAT

c) Reducing 2-SAT to 3-SAT

Which of the following is not a thing to try when you think the problem you want to solve might be NP-complete?

a) See if your problem is a special case that does have an efficient algorithm.

b) Try an approximation algorithm.

c) Implement a polynomial time algorithm for 3-SAT