CS340 - Analysis of Algorithms

Midterm Review

Upcoming important dates:

Lab4 due thursday (9/25)

Project checkpoint 1 was due sunday

HW4 due today

Midterm next class

Midterm Details

1 hour and 20min exam

4 Problems

1 page of notes allowed (single sided)

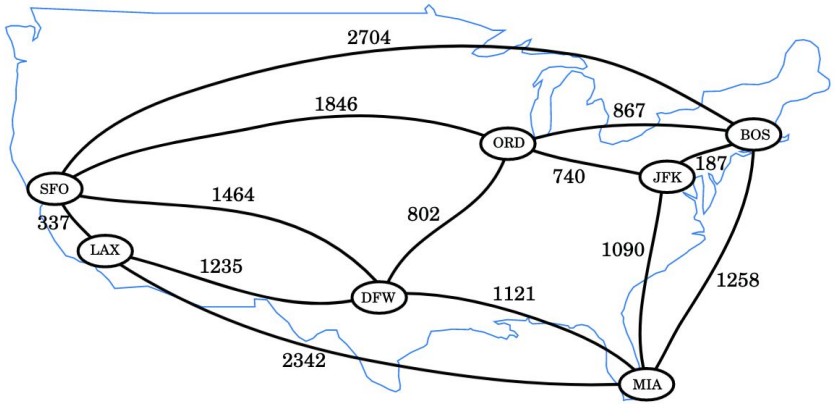
Topic List:

1. Ability to trace the following algorithms:
    a. Stable matching
    b. Greedy interval scheduling
    c. Dijkstra's
    d. Kruskals and Prim's
    e. Huffman's

"Ability to trace" =

- Clearly state what checks are performed
- Contents of the data structure at each iteration
- Current state of the solution

Q1: Trace Dijkstra's on the following graph



What does Dijkstras do?

Init:

- d[v] = inf forall v
- d[s] = 0
- Q = {s}

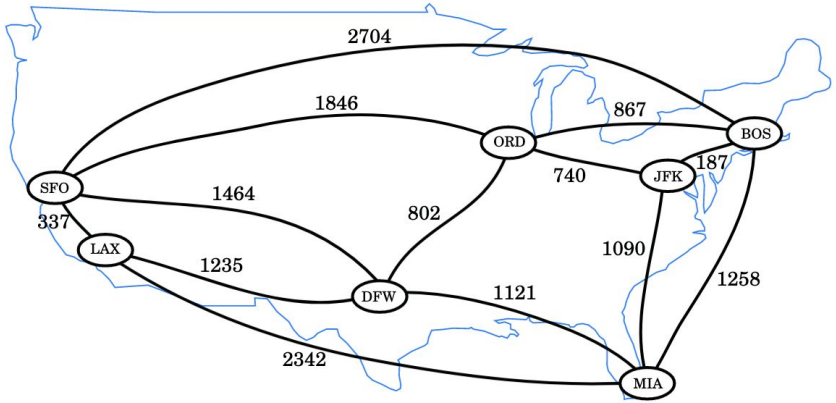
While Q is nonempty:

- u = poll min distance
- forall v adjacent to u
  - If d[u] + w(u, v) < d[v]
  - update d[v]
  - add v to the queue
  - pred[v] = u

Q1 Trace:

```
dijkstra(G, s){
  for each (u in V) { d[u] = infinity }
  d[s] = 0 pred[s] = null
  Q = priority queue of all vertices u keyed by d[u]
  while (Q is not empty) {
    u = extractMin from Q
    for each (v in Adj[u]) {
      if (d[u] + w(u, v) < d[v]) {
        d[v] = d[u] + w(u, v)
        decrease v's key value in Q to d[v]
        pred[v] = u //keeps track of the tree
      }
    }
  }
}
```

G =



s = SFO

- Clearly state what checks are performed
- Contents of the data structure at each iteration
- Current state of the solution

Q1b Trace:

Employers			Applicants		
Google(G)	Intel(I)	Apple(A)	Kate(K)	Clara(C)	Lisa(L)
K	K	L	A	A	G
C	L	K	G	I	I
L	C	C	I	G	A

1. Initially, everyone is unmatched
2. Employers pick their top choice and offer then a job
   a. If the applicant is unmatched, they accept
   b. If they already have a position, only switch if they prefer the employer to their current match

Q1b Trace:

Function `findStableMatching`(E, A)
- All $e \in E$ and $a \in A$ are unpaired
- **while** *there is an e unpaired that hasn't made an offer to every a* **do**
  - choose such an e
  - let a be the highest-ranked applicant in e's preference list who e has not made an offer to yet
  - **if** *a is unpaired* **then**
    - pair a and e
  - **end**
  - **else**
    - a is currently paired with e'
    - **if** *a prefers e' to e* **then**
      - e remains unpaired
    - **end**
    - **else**
      - a is paired with e and e' becomes unpaired
    - **end**
  - **end**
- **end**
- **return** the set of pairs

Q1c Trace - Huffmans Code

A	.33
B	.07
C	.4
D	.2

1. Put all characters in a min heap
2. Build the tree bottom up by iteratively selecting the two minimum prob values (u, v) and making them siblings in the bottom depth
3. Re-add z (the parent of u and v with prob p(u) + p(v) to the heap)

Q1c Trace - Huffmans Code

```
// C is a list of chars with associated probabilities
huffman(Node[] C){
  for each (x in C){
    add x to Q sorted by x.prob
  }
  n = size of C
  for (i = 1 to n-1) {
    z = new Node
    z.left = x = extractMin from Q
    z.right = y = extractMin from Q
    z.prob = x.prob + y.prob
    insert z into Q
   }
   return last element in Q as root
}
```

A	.33
B	.07
C	.4
D	.2

Topic List:

2. Understanding of Runtime analysis:

a. Stable matching
a. Greedy interval scheduling
b. Dijkstra’s
c. Kruskals and Prim’s
d. Huffman’s

Q2 Runtime Analysis:

Suppose that you are given a data structure that can perform the following operations in the following time over a data set of size n:

Check if two elements are in the same set: O(1)

Merge the set containing element a and the set containing element b: $O(2^{\log n})$

Create a new set and add an element node to it: $O(\sqrt{\log n})$

Give a time analysis of Kruskal's algorithm using this data structure on a graph of n nodes and m edges where m >> n

What does Kruskal's do?

Adds edges increasing order of weight that do not add cycles

1. Sort edges
2. Place each node in its own set
3. For each edge (u,v):
    a. Check if adding it would create a cycle
    b. If not, add the edge and merge the sets containing u and v for further cycle checks

Q2 Runtime Analysis:

Suppose that you are given a data structure that can perform the following operations in the following time over a data set of size n:

Check if two elements are in the same set: O(logn)

Merge the set containing element a and the set containing element b:

Create a new set and add an element node to it:

Give a time analysis of Kruskal's algorithm using this data structure on a graph of n nodes and m edges where m >> n

```
kMST(G=(V, E)){
  T = {}
  place each vertex in a set by itself
  sort E in increasing order by weight
  for each ((u, v) in E in sorted order){
    // u and v in different sets
    if (find(u) != find(v)) {
      add (u, v) to T
      union(u, v)
    }
  }
}
```

Topic List:

3. Ability to synthesize, analyze, and prove your own greedy algorithm

Problems to study:

- Solved exercises 4.1 and 4.2
- HW3: 4.3
- HW4: Q2

Q3 Greedy Algorithms:

Scholarship Allocation: You have scholarships of different amounts: Sc = {sc_1, sc_2, ..., sc_n}.

and students: St = {st_1, st_2, ..., st_n}. who each have different minimum financial needs: F = {f_1, f_2, ..., f_n}.

A student can only attend if they get at least their need.

Goal: Maximize the number of students funded.

1. Come up with a small example!
2. Derive psuedocode
3. Run it on the example
4. Analyze runtime
5. Proof of correctness
    a. Don’t forget termination!

Topic List

4. Ability to express problems as problems on graphs

Problems to Study:

- HW 3 Q3
- Lecture 9 warmup

Q4: Reduction to Graphs

Given a starting word, an ending word, and a dictionary of valid words, find the shortest sequence of words from the start to the end. Each word in the sequence must be in the dictionary and differ from the previous word by only one letter.

Explain what the different parts of the graph represent and how a solution to the problem would be represented on the graph.

Summary

Midterm next class

You should know the following algorithms:

1. Stable matching (Gale-Shapley)
2. Greedy interval scheduling
3. Dijkstra's
4. Kruskals and Prim's
5. Huffman's

You should be able to:

1. Trace the above algorithms
2. Understand and derive their runtime complexity with different data structures
3. Propose and analyze your own greedy algorithm to a problem
4. Reformualte problems as problems over a graph