CS16 Section 9

Monday, April 13th - Wednesday, April 15th
Agenda

1. Icebreakers
2. Mini Assignment
3. MST: Prim-Jarnik and Kruskal
4. PageRank
5. Radix Code Review
Icebreakers (pick one!)

- Rose, Bud, and Thorn (something good, something you’re looking forward to, something bad)
- Meal recommendation on Thayer
- Ideal spring weekend artist?
Mini Assignment
A spanning tree and a minimum spanning tree are both a subset of edges in a graph that span every vertex (forming a tree). A minimum spanning tree is the spanning tree with the least possible total edge weight for a given graph.

A spanning forest is the collection of STs for unconnected graphs, while a MSF is the collection of MSTs.
Problem 2 Solution: MST
MST: Prim-Jarnik and Kruskal
Subsection 1.1 -- Prim Jarnik’s Algorithm

- Similar to Dijkstra’s
- Picking the lowest cost edge at each iteration . . . what kind of algorithm does it sound like?
- Cool animation here: https://visualgo.net/en/mst (Show the Prim’s animation)
Subsection 1.1 -- Prim Jarnik’s Algorithm

- Split up into partners and take out a sheet of paper
- Hand simulate Prim Jarnik’s on the graph shown in the next slide
- We will give you 5 to 10 minutes to hand simulate this graph, and then we will go over the answer together.
Random node set to cost 0

$PQ = [(0, A), (\infty, B), (\infty, C), (\infty, D), (\infty, E), (\infty, F)]$
Dequeue from PQ and update neighbors

\[ PQ = [ (4, B), (5, D), (\infty, C), (\infty, E), (\infty, F) ] \]
Deque from PQ and update neighbors

\[ PQ = [(4, C), (4, D), (6, E), (8, F)] \]
null

\[ PQ = [(2, E), (4, D), (8, F)] \]
Dequeue from PQ and update neighbors

\[ PQ = [(4, D), (4, F)] \]
null

Dequeue from PQ and update neighbors

PQ = [(3, F)]
PQ = [ ]

Dequeue from PQ and update neighbors
Subsection 1.2 -- Kruskal’s Algorithm

- Union Find
- Intuition -- ‘clouds’, used to make sure that the edges that are added do not create cycles
- Path compression -- how does it work?
- Cool animation here: https://visualgo.net/en/mst (Show the Kruskal’s animation)
Path Compression

- Instead of traversing up tree every time D's cloud is asked for
  - We only search for D's root once
  - As we follow chain of parents to A we set parents of D & C to A

\[ \text{Amortized } O(1) \]
Subsection 1.2 -- Kruskal’s Algorithm

- Split up into partners and take out a sheet of paper
- Hand simulate Kruskal’s on the graph shown in the next slide
- We will give you 5 to 10 minutes to hand simulate this graph, and then we will go over the answer together.
edges = [(C,E), (D,F), (B,C), (E,F), (B,D), (A,B), (A,D), (B,E), (B,F)]
edges = [(C, E), (D, F), (B, C), (E, F), (B, D), (A, B), (A, D), (B, E), (B, F)]
edges = [(D, F), (B, C), (E, F), (B, D), (A, B), (A, D), (B, E), (B, F)]
edges = [(B, C), (E, F), (B, D), (A, B), (A, D), (B, E), (B, F)]
edges = [(E,F), (B,D), (A,B), (A,D), (B,E), (B,F)]
edges = [(B,D), (A,B), (A,D), (B,E), (B,F)]
BD cannot be added because it would lead to a cycle.

edges = [(A, B), (A, D), (B, E), (B, F)]
edges = [(A, D), (B, E), (B, F)]
AD cannot be added because it would lead to a cycle

edges = \{ (B, E), (B, F) \}
BE cannot be added because it would lead to a cycle

edges = [(B, F)]
BF cannot be added because it would lead to a cycle

edges = [ ]
PageRank
At every round:

- Each vertex splits its PR evenly among its outgoing edges
- Each Vertex receives PR from all its incoming edges
- This is done using an update rule which is run on every vertex:

\[
PR(v) = \sum_{u \in \text{in}(v)} \frac{PR(u)}{|\text{out}(u)|}
\]
PageRank - Pseudocode

BasicPageRank(G, k):
   for v in V:
      v.rank = 1/|V|
   for i from 1 to k:
      for v in V:
         v.prevrank = v.rank
      for v in V:
         v.rank = 0
      for u in v.incoming:
         v.rank = v.rank + u.prevrank/|u.outgoing|

- \(O(|V| + |E|)\)
- Ranked according to the number of links it has to other existing web pages
- The more web pages it’s linked to, the higher the rank
PageRank

- What is a rank trap?
  - Nodes that are sinks that accumulate pagerank because they only receive PR through the incoming edges and do not distribute PR because there are no outgoing edges.

- How is this problem solved?
  - One solution is to distribute a fraction of each node’s PR to all other nodes.

\[
PR(v) = \frac{1 - d}{|V|} + d \cdot \left( \sum_{u \in \text{in}(v)} \frac{PR(u)}{|\text{out}(u)|} + \sum_{u \in \text{sinks}(G)} \frac{PR(u)}{|V|} \right)
\]
Radix Code Review
Radix Code Review

- You will be receiving a copy of a Radix Sort code that belongs to one of your section classmates! Try to look at the code and consider the style choices made, and give constructive feedback!
- At the end, we will all receive our original pieces of code and look at our feedback, commenting on what we learned.
Questions?

- Questions on any of the material?