Heaps and Priority Queues

Instructors: Sam McCauley and Dan Barowy

May 9, 2022

Admin

- Final review Friday (no quiz!)
- If you have an "exam hardship" let me know as soon as possible
- Lab 6 graded; back any minute now. Lab 7 waiting on a backend issue; probably later today
    - Labs 8 and 9, and last quiz, should be soon...
- Please bring your computers (or something) on Wednesday for course evals at end of class
- Any questions?

Adjacency List vs Adjacency Matrix

- Adjacency List is (often) much faster for listing neighbors of a vertex:
  - Adjacency Matrix gives time proportional to the total number of vertices, Adjacency List gives time proportional to the degree.
- Adjacency Matrix is much faster for looking up if there is an edge bettween two vertices
- Adjacency List (on a graph with n vertices and m edges) is much more space efficient if $m < n^2$

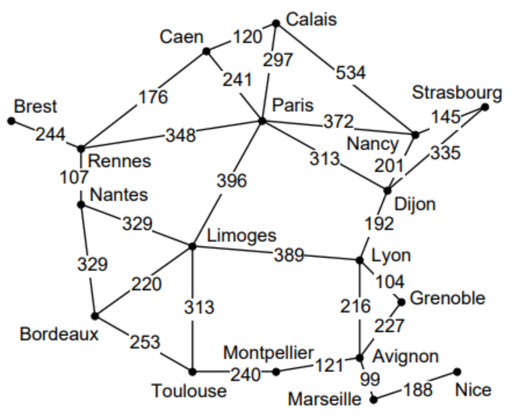
Shortest Path in Graph

Shortest path in a graph

- Breadth-first short finds the path with the smallest length in a graph

- Let’s look at a visualization of this

Shortest path: what do we really want?



- Not all edges are the same!
- It takes a different amount of time to travel down different roads, or take different flights
- What if we have numerical labels, and the length of the path is the sum of the labels?

Trying out BFS

- Does BFS work if we want to take path labels into account?

- Can we come up with an example where it doesn’t?

What do we want?

- Want to explore the paths *in order of length*
- So: something like BFS, but want to explore the shortest path next
- Called Dijkstra’s algorithm. We’ll discuss in detail on Wednesday
- Let’s look at how it runs
- For now: what do we need to implement this?

New operation on a data structure

- Need to know: what is the shortest remaining path to explore
- In BFS, we kept all vertices we wanted to explore in a queue
- Now, we don't want first in first out. We want the vertex with smallest path length out.
- We want to keep a collection of vertices in a data structure, with the ability to remove the smallest
- This is called a *priority queue*

Priority Queues (Slide Deck Change)

Benefits of heaps?

- Balanced binary search tree: time for removeMin? add?
  - Both $O(\log n)$
- Heap?
  - Both $O(\log n)$
  - But *much* better constants, much simpler
- Creating a balanced binary tree of size n using an unsorted `Vector`?
  - $O(n \log n)$
- Creating a heap of size n using an unsorted `Vector`?
  - $O(n)$

Heap vs priority queue

- Priority queue is the interface
- Heap is the specific implementation
- Like Map vs Hashtable. There are other ways to implement a Map; similarly, there are other ways to implement a priority queue

Summary

- In short: heaps are much simpler and have much better constants
- Extremely common in practice!
- HeapSort is one of the most common sorting methods, especially if you want $O(n \log n)$ guaranteed worst-case running time
- We saw min heaps. Can get a "max heap" by flipping the requirement: the root element must be largest in the heap. Then can get good `removeMax` performance