CS lecture #29 Shortest Paths

Warm-up: When is DFS better than BFS? Do they work for all graphs?
- BFS gives shortest path, as well as what's reachable.
- Memory usages
- Spindly graphs = DFS worse (think of Edges in Space).

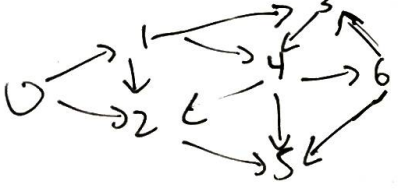
For something like google maps, is BFS better?
- No. Our metric is not vertices. Our metric now is shortest road amt.
- So how Do we add length of roads?

Dijkstra's Algorithm

We now augment our graph so each edge has a weight.
- solution always doesn't have a cycle.

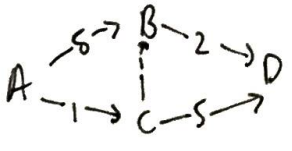
To find the shortest path from source vertex to every other:
- Solution will be a tree!
- Why?
  1) Can't include cycles.
  2) Every node has only 1 parent.
     ↳ we only use the shortest path. Even if they're the same, just use one of them.

Question: If G is a connected edge-weighted graph w/ V vertices & E edges, how many edges are in the SPT (Shortest Paths Tree) of G?



Always V-1. For each vertex, there is exactly one input edge.
(Assuming every vertex is reachable)

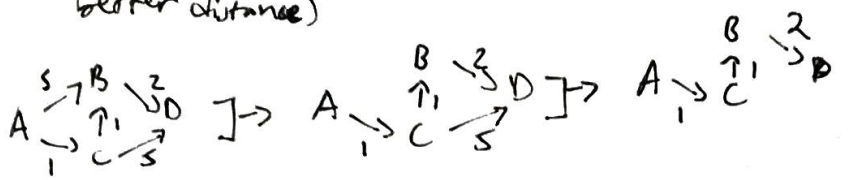
Question: What is the shortest paths tree?



Finding a Shortest Paths Tree Algorithm:

Dijkstra's Algorithm

- Visit vertices in order of best-known distance from source, relaxing each edge from the visited vertex. (Relaxing an edge = Add to SPT if better distance)



Bigger example @ 28:10 in lecture.

↳ shows fringe representation.

take something out of fringe, process all of its edges.

- Why is Dijkstra's correct for non-negative edges? Path to X is optimal after X has been dequeued. Inductive argument.
  - Suppose path to just-dequeued vertex V is optimal. Then, after relaxation of V's edges, path to vertex X at top of PQ will be optimal.

Runtime:

PQ operation count, assuming binary heap pq:

- Insertion: V, each costs $O(\log V)$ time
- Deletemin: V, each costing $O(\log V)$ time
- Decrease priority: E, each costing $O(\log V$ time$]$.

Overall runtime: $O(V^{*} \log(V) + V^{*} \log(V) + E^{*} \log V)$

- assuming $E > V$, this is $O(E \log V)$

A* → good for finding the single target. *d(source, v) + h(v)

How do we improve over Dijkstra's?

Look at a ~~location~~ location v if:

- We know already the fastest way to get to V.
- V is also the fastest way to reach the goal, taking in account the time to get to V.

Demo @ 46:10 in lecture.

How do we get our estimate?

- It's an arbitrary heuristic h(v)
- heuristic: "using experience to learn and improve"
- Does not need to be perfect.
- h(v) does not change as alg runs.

```
h(v) {
    return computeLineDistance(v.latLong, NYC.latLong);
}
```

Impact of heuristic:

if we just use h=0, end up with Dijkstra's.
if we just use h=1000, end up with Dijkstra's.

heuristic that overestimates is inadmissible

A* yields the shortest path as long as heuristic is admissible.

- if h(v) never overestimates distance to goal, you'll get the right answer.

Note* we did A* in HW4!

- PQ contained nodes in order of miles so far + estimated Distance.
- Graph was stored implicitly in neighbors().
- All distances were 1.