CS 341: ALGORITHMS
Lecture 14: graph algorithms V — single source shortest path
Readings: see website
Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

DIJKSTRA’S ALGORITHM
Single-source shortest path in a graph with non-negative edge weights

PROBLEM: SINGLE SOURCE SHORTEST PATHS (SSSP)
• Input: graph $G = (V,E)$ and a non-negative weight function $w(e)$ defined for every edge $e$
• Problem: for every node $v \neq s$, output a path $s \rightarrow v$ with the smallest total weight (among all paths $s \rightarrow v$)
  • i.e. each path $P$ should minimize $w(P) = \sum_{e \in P} w(e)$

APPLICATION: DRIVING DISTANCE TO MANY POSSIBLE DESTINATIONS
• Single source: from where you are
• Shortest path to all destinations
• Display a subset of destinations
• Include the optimal distances computed using SSSP algorithm
• Other heuristics... traffic? Lights?
• Weights can combine many factors

GAME AI: path finding with waypoints
Divide game world into linear paths. Then send game characters in straight lines between waypoints.
If some linear paths are much faster/slower, use weighted SSSP.

[video clip]
**Proof**

- **Theorem**: At the end of the algorithm, for all \( u \), \( \text{dist}(u) \) is exactly the total weight of the shortest \( s \to u \) path.
- We prove this in two parts.
  - \( \text{dist}(u) \leq \) the total weight of the shortest \( s \to u \) path (case \( \leq \))
  - \( \text{dist}(u) \geq \) the total weight of the shortest \( s \to u \) path (case \( \geq \))

**CORRECTNESS: INTUITION**

- Dijkstra's algorithm iteratively constructs a set \( \text{OPT} \) of nodes for which we know the shortest path from \( s \) (initially \( \text{OPT} = \{ s \} \)).
- After each relaxation step, we grow \( \text{OPT} \) by adding the node in \( V \setminus \text{OPT} \) with the smallest \( \text{dist} \).

**Case \( \leq \)**

- Let \( P \) be any arbitrary \( s \to u \) path \( v_0 \to v_1 \to \cdots \to v_r \) where \( v_0 = s \) and \( v_r = u \).
- For any index \( j \) let \( l_j \) denote \( w(v_j \to v_{j+1}) \).
- We prove by induction: \( \text{dist}(v_j) \leq l_j \) for all \( j \).

**Case \( \geq \)**

- Let \( P' \) be the path \( s \to \cdots \to \text{pred}(\text{pred}(u)) \to \text{pred}(u) \to u \).
- I.e., the reverse of following \( \text{pred} \) pointers from \( u \) back to \( s \).
- We show \( \text{dist}(u) \) is as long as this path (and hence as long as the shortest path).
- Denote the nodes in \( P' \) by \( v_0, v_1, \ldots, v_t \) where \( v_0 = s \) and \( v_t = u \).
- Let \( l_t = w(v_t \to v_{t-1} \to \cdots \to v_0) \).
- Prove by induction: \( \forall j \geq 0 \), \( \text{dist}(v_j) = l_j \).
- Base case: \( \text{dist}(v_0) = \text{dist}(s) = 0 = l_0 \).
CASE ≥

- \( P = v_0 \rightarrow \cdots \rightarrow v_t = s \rightarrow \cdots \rightarrow v_i \)
- \( L_j = w(v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_j) \)

**Inductive step:** suppose \( \forall j > 0 : dist[v_j-1] = L_j-1 \)
- When we set \( \text{pred}[v_j] = v_{j-1} \), we set \( \text{dist}[v_j] = \text{dist}[v_{j-1}] + w(v_{j-1}, v_j) \)

Recall:
- By I.H., \( \text{dist}[v_j] = L_j-1 + w(v_j-1, v_j) \)
- By definition \( L_j = L_{j-1} + w(v_{j-1}, v_j) \)
- So \( \text{dist}[v_j] = L_j \)

That means it’s equal to the length of the shortest path!

**OUTPUTTING ACTUAL SHORTEST PATH(S)?**

- To compute the actual shortest path \( s \rightarrow t \)
- inspect \( \text{pred}[t] \)
  - If it is NULL, there is no such path
  - Otherwise, follow \( \text{pred} \) pointers back to \( s \), and return the reverse of that path

**WEBSITE DEMONSTRATING DIJKSTRA’S ALG**

Shortest Paths and Negative Weight Cycles

Subsequent algorithms we will be studying will solve shortest path problems as long as there are no problems having negative weight.

If there is a negative weight cycle, then there is no shortest path (why?!)

There is still a shortest simple path, but there are apparently no known efficient algorithms to find the shortest simple paths in graphs containing negative weight cycles.

If there are no negative weight cycles, we can assume WLOG that shortest paths are simple paths (any path can be replaced by a simple path having the same weight).

Negative weight edges in an undirected graph are not allowed, as they would give rise to a negative weight cycle (consisting of two edges) in the associated directed graph.

BELLMAN- FORD

The Bellman-Ford algorithm solves the single source shortest path problem in any directed graph without negative weight cycles.

The algorithm is very simple to describe:

Repeat $n - 1$ times: relax every edge in the graph (where relax is the updating step in Dijkstra’s algorithm).

$O(nm)$ outer iterations

$O(1)$ inner iterations per outer iteration

Could be $O(n^3)$

Total $O(nm)$

Could be $O(n^3)$

BEST CASE EXECUTION

If technically suffices to do one iteration of the outer loop

WORST CASE EXECUTION

Need $n$ iterations of outer loop

WHY BELLMAN-FORD WORKS

• Not going to prove this (by induction), but the crucial lemma is:
• After $i$ iterations of the outer for-loop,
  • if $D[u] = \infty$, it is equal to the weight of some path $s \rightarrow u$; and
  • if there is a path $P = (s \rightarrow u)$ with at most $i$ edges, then $D[u] \leq w(P)$
• So, after $n - 1$ iterations, if a path $P$ with at most $n - 1$ edges, then $D[u] \leq w(P)$. [Note: any more edges would create a cycle.]
• So, if $u$ is reachable from $s$, then $D[u]$ is the length of the shortest simple path (no cycles) from $s$ to $u$

Of course every simple path has at most $n - 1$ edges

So what if we do another iteration and some $D[u]$ improves?

There is a negative cycle!

A MORE DETAILED IMPLEMENTATION

• With early stopping
• and checking for negative cycles

Dijkstra’s is similar, but consistently achieves good ordering using its priority queue

Since the longest possible path without a cycle can be $n - 1$ edges, the edges must be scanned $n - 1$ times to ensure the shortest path has been found for all nodes.
**BONUS SLIDE**

- Why can’t you just modify a graph with negative weights by finding the minimum edge weight \(W_{\text{min}}\), and adding that to each edge, so you no longer have negative edges and can run Dijkstra’s algorithm?

- **Exercise:** can you find a graph for which this will cause Dijkstra’s algorithm to return the wrong answer?

- **Solution:**
  - Consider a graph with 5 nodes: \(s, a, b, c, t\)
  - And edges \(s \rightarrow a\) with weight \(-10\), \(b \rightarrow t\) with weight \(10\)
  - \(s \rightarrow b\) weight \(-1\), \(b \rightarrow c\) weight \(-1\), \(c \rightarrow t\) weight \(-1\)
  - What happens if you modify this graph as proposed, then run Dijkstra’s to find the shortest path from \(s\) to \(t\)?