CIS 121—Data Structures and Algorithms—Fall 2020

DFS Edges & Single Source Shortest Path—Monday, November 2 / Tuesday, November 3

Single Source Shortest Path Algorithms

Definitions

Definition 1 (Greedy algorithm)**.** A greedy algorithm is one which always makes the choice that looks best at the moment—the *locally optimal* choice—in order to find the best *globally optimal* solution. Greedy algorithms do not always yield optimal solutions, but for many problems they do.

Definition 2 (Shortest path)**.** A shortest path from vertex s to vertex t is a directed path from s to t with the property that no other such path has a lower total edge weight.

Definition 3 (Negative Weight Cycle)**.** A negative weight cycle is a cycle with weights that sum to a negative number.

Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest path between two given vertices in a weighted graph, assuming that the graph's edge weights are non-negative. The running time of the algorithm is $O(E \log V + V \log V)$ when the graph is implemented using adjacency lists. The pseudo-code for the algorithm is given below.

Pseudocode

```
DIJKSTRA(G, s)
 1  for each vertex v ∈ V_G
 2      dist[v] = ∞
 3      parent[v] = NIL
 4  dist[s] = 0
 5  Q = V_G
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each vertex v ∈ G.Adj[u]
 9          if dist[v] > dist[u] + w(u, v)
10              dist[v] = dist[u] + w(u, v)
11              parent[v] = u
```

Runtime

The running time of Dijkstra's algorithm has two components, $E \log V$ and $V \log V$. Let us first consider the $V \log V$ term: this component derives from the maximum size (V) of the heap used to store vertices, and the running time of heap operations such as INSERT and REMOVEMIN is $O(\log V)$.

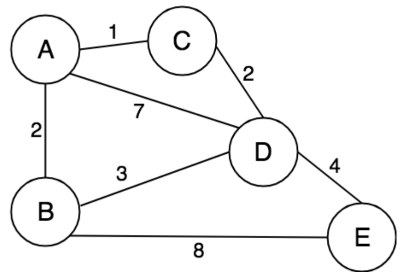
The $E \log V$ term has to do with the *relaxation* step of Dijkstra's algorithm. Each edge examined may result in a relaxation of the neighboring node in the heap; in other words, an update key operation that is $O(\log V)$. We know that the number of vertices examined in line 8 above is bounded by the total degree of all vertices, as each vertex is added and popped exactly once from the min-heap. This value is $2|E|$ by the Handshake lemma, so in the worst case we have $2|E|$ decrease-key operations, for a total of $O(E \log V)$.

This bound is good for easily proving our run-time, but it is not tight. Each edge (u, v) can only cause one relaxation, not two as the handshake lemma suggests. This is because (u, v) is explored only when node

u is popped from the min-heap. This means that when (u, v) is explored from node v node u has already been removed, so it's key cannot be decreased.

Example

Trace through Dijkstra's on this graph, starting at A.



Dijkstra's algorithm produces the following state:

Node	Distance from A
A	0
B	2
C	1
D	3
E	7

Node	Parent node
A	NULL
B	A
C	A
D	C
E	D

Directed Acyclic Graphs

The algorithm for shortest path on edge weighted DAGs from a source vertex s is simpler and faster than Dijkstra's algorithm. Instead of considering vertices by priority of their distance estimates, we consider the vertices of the DAG in a topological order. (Why must a DAG always have a topological order?) Then we just relax each vertex in the topological ordering.

Pseudocode

```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, taken in topologically sorted order
4      for each vertex v ∈ G.Adj[u]
5          RELAX(u, v, w)
```

This algorithm runs in $O(|V| + |E|)$ since we process each vertex only once in Line 3-5 ie. $O(|V|)$ and in aggregate, the loop in Lines 4-5 only runs $|E|$ times. Lastly, for a formal POC see Section 24.2 CLRS.

Problems

Problem 1

Does Dijkstra's Algorithm work with negative weights? Why or why not?

Solution

No, Dijkstra's Algorithm will not work on negative weighted graphs. First, if there exists a negative cycle, the concept of shortest path does not exist.

Secondly, a negative weight breaks an important assumption in the canonical proof of correctness for Dijkstra's algorithm.

Proof (adapted from CLRS). Induct on the size of the shortest path tree S with source s. Assume that Dijkstra's algorithm correctly computes the shortest path for a tree of size $|S| = k$, for some $k \geq 1$. We must show that if u is the $k+1$-st vertex brought into S, then $dist[t]$ is the weight of the shortest path from s to u. Let p be a shortest path from s to u. Let y be the first vertex along p such that $y \in V - S$, and let $x \in S$ be the predecessor of y. Path p can be deconstructed as $s \rightsquigarrow x \to y \rightsquigarrow u$. Let $\delta(\bullet, \bullet)$ represent the actual shortest path distance between two vertices. Because y appears before u and all edge-weights are non-negative, $dist[y] = \delta(s, y) \leq \delta(s, u) \leq dist[u]$. But since both u and y were in $V - S$ when u was taken off of the priority queue, it must be that $dist[u] \leq dist[y]$. So u is in fact the vertex with its distance estimate $dist[u]$ exactly equal to the shortest path distance $\delta(s, u)$.

Problem 2

True or false: Dijkstra's algorithm will not terminate if run on a graph with negative edge weights.

Solution

False. The algorithm will terminate, but it might return a wrong answer.

Problem 3

True or false: If we double the weights of all the edges in a graph, then Dijkstra's algorithm will produce the same shortest path. What about squaring?

Solution

True. Any scaling by a positive factor on the weights will not affect the calculation of shortest paths. You can think of it as unit-conversion. For instance, if you converted weights from expression in miles to kilometers, that would not affect the relative ordering of shortest paths.

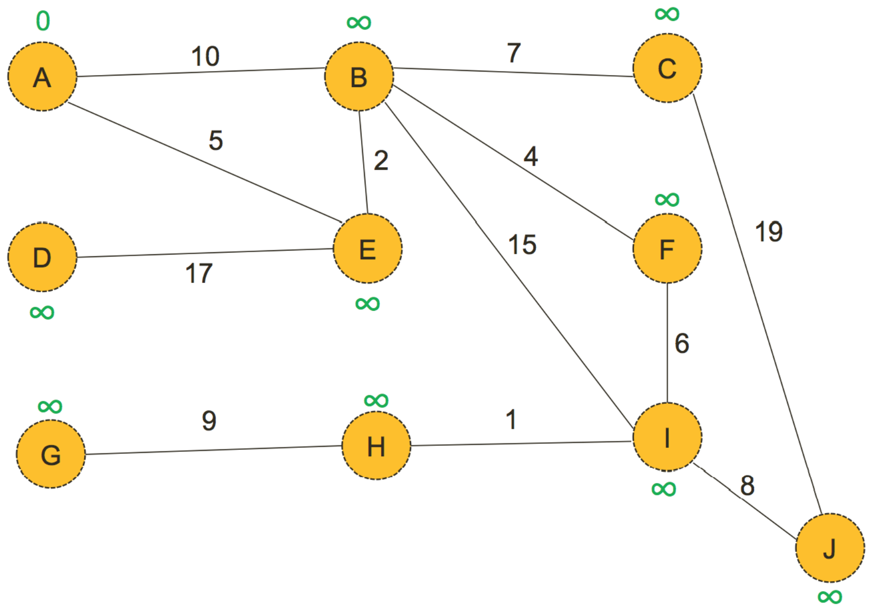
Problem 4

Explain why Dijkstra's algorithm is a greedy algorithm.

A greedy algorithm makes the best choice that is currently available. Dijkstra's algorithm follows this paradigm by using a priority queue structure that, when polled, always produces the node with the shortest distance from the source node.

Problem 5

Find the shortest path between vertices E and G.



Solution

Dijkstra's algorithm produces the following state:

Node	Distance from E
A	5
B	2
C	9
D	17
E	0
F	6
G	22
H	13
I	12
J	20

Node	Parent node
A	E
B	E
C	B
D	E
E	NULL
F	B
G	H
H	I
I	F
J	I

We can use the mapping from nodes to parent nodes to find the shortest path from E to G, which is $E \to B \to F \to I \to H \to G$.