Recitation 11 – *Shortest Paths: Dijkstra's and Bellman Ford Algorithms*

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2015)

March 24th, 2015

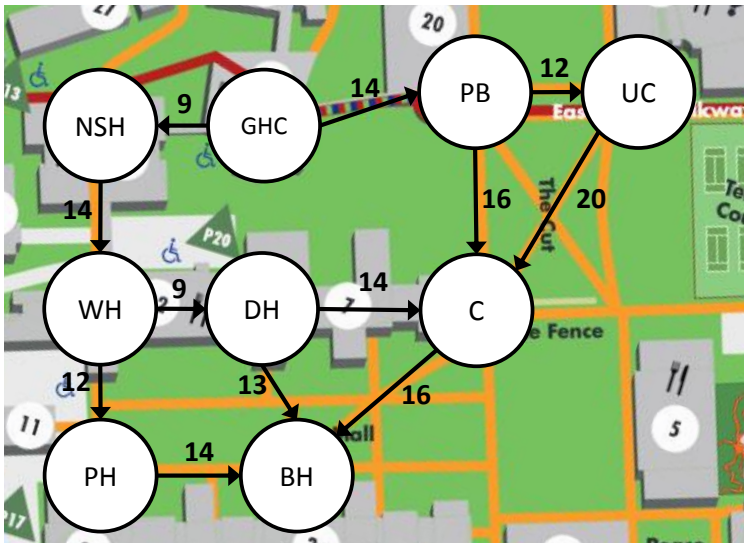
1 Announcements

- *ThesaurusLab* is due tomorrow! In addition to that
- *A sudden turnabout!*: Barring something miraculous, Exam 2 will probably be in 2 weeks. As a consequence, expect AbridgedLab to be out soon! (It's relatively tricky)

2 Dijkstra's Algorithm

We will now review Dijkstra's single-source shortest path algorithm by way of an example. It's approaching noon on a Tuesday morning, and Professors Umut and Guy need to get from their offices in GHC to a very important super-secret conference in Baker. There are many ways to get there: they could take the Pausch Bridge to the cut, walk straight across to the Doherty entrance and then diagonally across to Baker, or they could take the bridges to Newell Simon and Wean, and then walk across the mall to Baker. Other routes are possible as well. However, they would like to find the fastest route.

You realize that by representing the paths around CMU's campus as a graph, you can use Dijkstra's Algorithm to make sure the professors get to the super secret conference on time.



Let's work through what Dijkstra's algorithm does on the graph of CMU above, with edge weights given by arbitrarily-scaled straight-line distances.

Recall that Dijkstra's Algorithm requires 2 components. A table and a priority queue.

Q: What information is stored in the table? How about the Priority queue?

For each timestep, the table below shows the node we visit at that timestep, the priority queue operations we perform during the step, and the values in the priority queue after the step. Fill in the table for all of the timesteps! The first 3 timesteps set have been given to you.

The priority queue operations available are `deleteMin`, which removes the minimum valued node in the priority queue and `insert(elem, priority)`, which adds the element to the priority queue with the specified priority.

	Node	Priority Queue Operations	Priority Queue (*lowest values only*)
1	*GHC*	`deleteMin`, `insert(PB, 14)`, `insert(NSH, 9)`	$\{NSH \mapsto 9, PB \mapsto 14\}$
2	*NSH*	`deleteMin`, `insert(WH, 23)`	$\{PB \mapsto 14, WH \mapsto 23\}$
3	*PB*	`deleteMin`, `insert(UC, 26)`, `insert(C, 30)`	$\{WH \mapsto 23, UC \mapsto 26, C \mapsto 30\}$
4			
5			
6			
7			
8			
9			

All nodes have now been visited, so we can stop here.

Q: What does our table of shortest-paths look like?

Vertex	Distance
GHC	
NSH	
PB	
UC	
C	
WH	
DH	
PH	
BH	

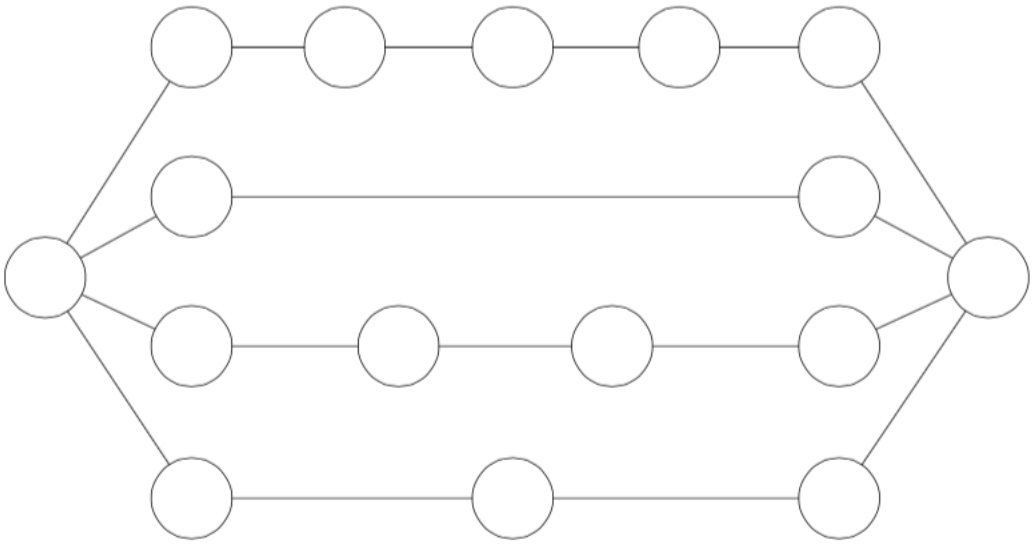
Q: What route gives you precisely the shortest-path to Baker

Q: Can you modify Dijkstra to give you the route you need to take along with the shortest path?

Q: What are the limits of Dijkstra in regards to weights? Runtime considerations of span?

3 Bounding Dijkstra

Consider the following. You are given a weighted, undirected graph which consists of k paths that are connected at the ends. For example, the following graph has k=4 paths (edge weights omitted).



Q: Running Dijkstra's starting from any node, bound the size of the priority queue at any point during the algorithm. Give your answer in terms of k.

Now consider if we had the following function

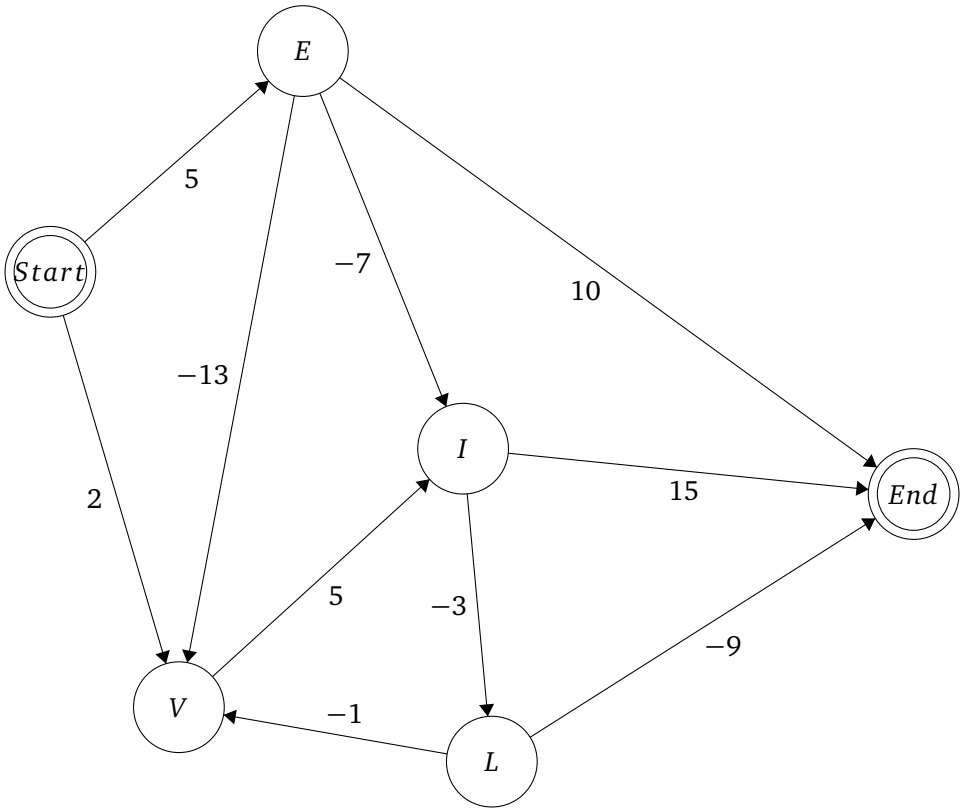
```
decreaseKey:  PQ -> (key, priority) -> PQ
```

which takes in a priority queue PQ and a key priority pair (k, p). It adds (k, p) to PQ if k is not already a key in PQ, otherwise, it updates the priority of k in the PQ if p < p', where (k, p') is in the PQ. The runtime of this is $O(\log n)$ work span.

Q: What does this let us bound the size of our priority queue to? How does this affect runtime?

4 Bellman Ford

Upon making it to the super-secret conference in Baker, Umut and Guy are tasked with operating the super secret **E**xam **V**alidating **I**ntrinsic **L**egibility machine. The machine is laid out like a finite state machine, and in order to make the exams as legible as possible, the operators of the machine must take it from its starting point to the end with the shortest edge-weight possible. However, there can be negative edge weights, which throws Umut's and Guy's previous strategy out the window. The machine is laid out below



You realize that you can instead use the Bellman Ford Algorithm to make the exams as legible as possible!

The table below shows the nodes of the finite state machine and the distances from the source node to each of them at each timestep of the algorithm. Fill out the table! The first 2 timesteps set have been given to you.

Vertex	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
Start	0	0	0				
E	∞	5	5				
V	∞	2	-8				
I	∞	∞	-2				
L	∞	∞	∞				
End	∞	∞	15				

Q: When will the algorithm end?

Q: How do we handle negative cycles?

Q: What other benefits does Bellman Ford afford over Djikstra?