Programming, Problem Solving, and Algorithms

CPSC203, 2019 W1

Announcements

Project 3 released soon. Due 11:59p, Nov 29.

“Problem of the Day” continues!

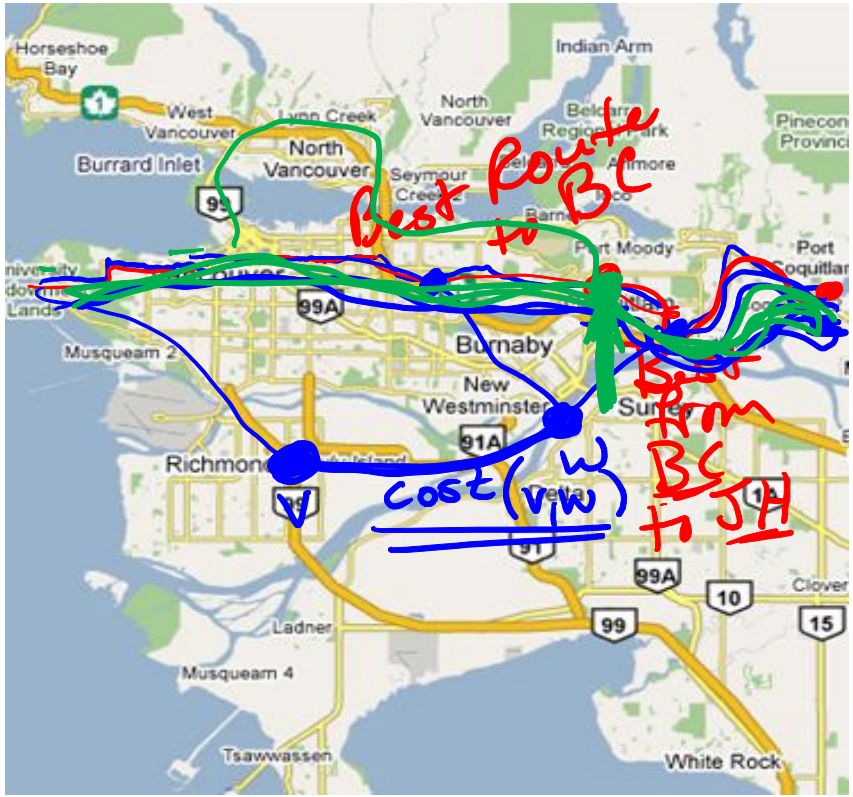
Today:

Shortest Path

Maps!

How many Starbucks are in Vancouver?

Single Source Shortest Path



Given a start vertex (source) s, find the path of least total cost from s to every vertex in the graph.

cost metric is time but we could choose distance, steps, traffic, transit switches, # of starbucks on rt.

Need to elucidate "cost".

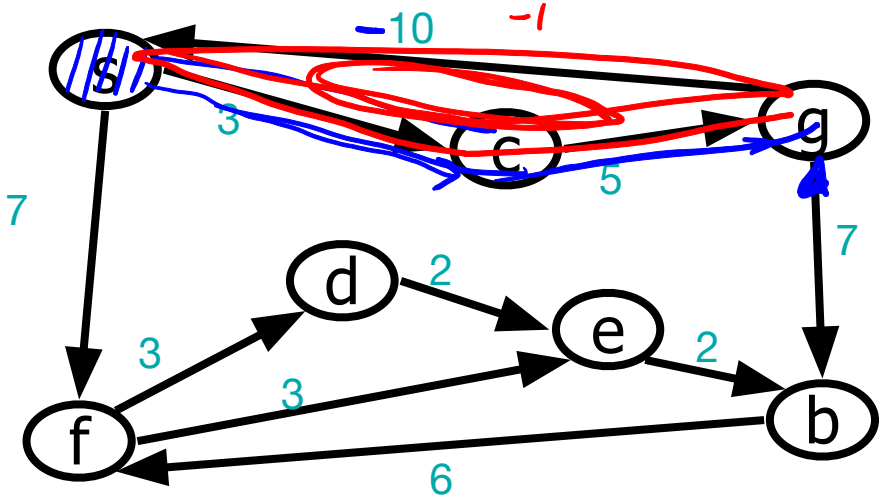
Single Source Shortest Path

Input: directed graph G with non-negative edge weights, and a start vertex s. "source"

neg cycle no alg works

neg edge (no neg cycles) Dijkstra's doesn't work.

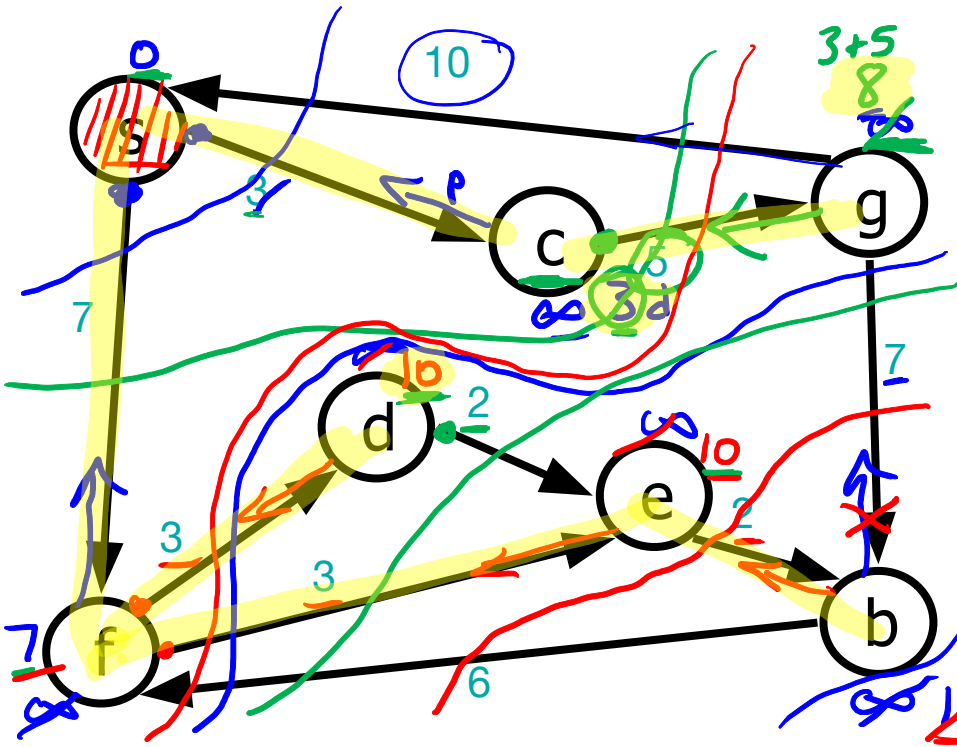
Output: A subgraph G' consisting of the shortest (minimum total cost) paths from s to every other vertex in the graph.



Dijkstra's Algorithm (1959)

Shortest path doesn't exist if there is a negative wt cycle.

Single Source Shortest Path



Given a source vertex s, we wish to find the shortest path from s to every other vertex in the graph.

Initialize structure: Priority Queue d[v]

dist[v]: current best from s to v.

pred[v] = w if (w,v) is in soln

Repeat these steps:

1. Label a new (unlabelled) vertex v, whose shortest distance has been found scfgdeb
2. Update v's neighbors with an improved distance

Single Source Shortest Path

Initialize structure:

current best distance

1. For all v, d[v] = “infinity”, p[v] = null

predecessor

2. Initialize source: d[s] = 0
3. Initialize priority (min) queue of the d[v]

Repeat these steps n times:

PQ operation

- Find minimum d[] unlabelled vertex: v
- Label vertex v done
- For all unlabelled neighbors w of v,

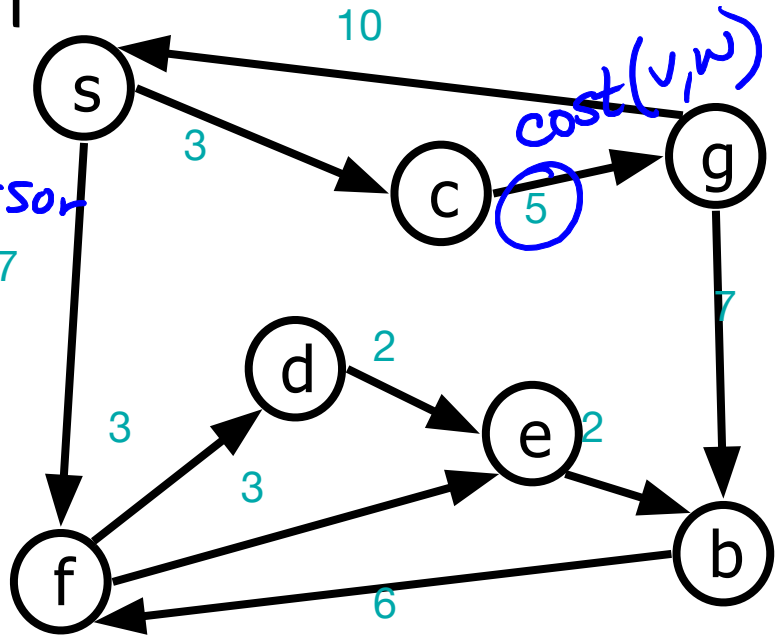
  If (d[v] + cost(v,w) < d[w]) "is better than"

  d[w] = d[v]+cost(v,w)

  p[w] = v

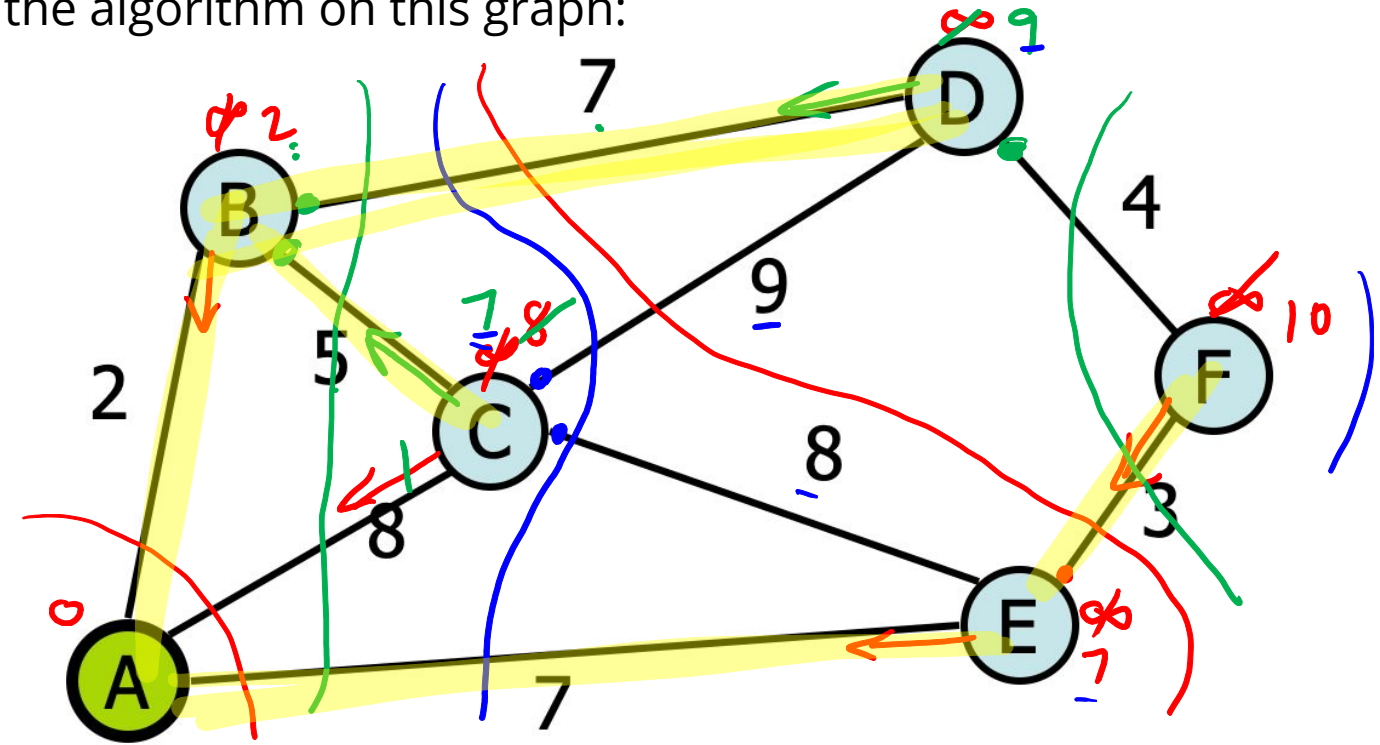
old best dist to v's neighbor.

update step.



Your Turn...

Execute the algorithm on this graph:



Dijkstra's Algorithm PQ

How is this algorithm similar to BFS/DFS?

structure
optimality provided by

How is this algorithm different from BFS/DFS?

no enqueuing of children
'cuz vertices are already there

Initialize structure:

1. For all v, d[v] = "infinity", p[v] = null
2. Initialize source: d[s] = 0
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to ∅.

While PQ is not empty:

Repeat these steps n times:

- Find & remove minimum d[] unlabelled vertex v Q.dequeue
- Label vertex v
- For all unlabelled neighbors w of v,

  Same

  If cost(v,w) < d[w]

  d[w] = cost(v,w)

  p[w] = v

  update

Map applications

Three parts:

1. Assembling the data - OSM, local data stores, statsCan, etc. This is mostly the art of assembling geodataframes.

2. Computing on the data - osmnx simplifies graph algorithms and computation, but also supports other spatial computation.

3. Visualizing the data - matplotlib for static maps, folium for interactive maps.

POTD #36 Thu

https://github.students.cs.ubc.ca/cpsc203-2019w-t1/potd36

Describe any snags you run into:

1. Line ___: _____
2. Line ___: _____
3. Line ___: _____
4. Line ___: _____
5. Line ___: _____

ToDo for next class...

POTD: Continue every weekday! Submit to repo.

Reading: TLACS Ch 10 & 12 (lists and dictionaries)

References:

https://www.youtube.com/watch?v=wsSEKm-rU6U

https://github.com/gboeing/osmnx-examples/tree/master/notebooks

https://gist.github.com/psychemedia/b49c49da365666ba9199d2e27d002d07