CSCI 2300 — Algo
Homework 6
Hayden Fuller

- **Q1** Given an undirected graph G, describe a linear time algorithm to find the number of distinct shortest paths between two given vertices u and v. Note that two shortest paths are distinct if they have at least one edge that is different.

```
Count = 0
Use BFS starting at u:
  if you reach v, Count++
  at the end of each level down, if Count==0, continue
  if Count!=0, Count is your number of shortest paths
```

- **Q2** Given a weighted directed graph with positive weights, given a O($|V|^3$) algorithm to find the length of the shortest cycle or report that the graph is acyclic.
  Use a modified Dijkstra's algorithm, don't count not going anywhere as 0, look for a loop instead, then find the shortest distance from a node to itself

```
create 2D array Distances of size $|V|x|V|$, all values set to infinity
for each node s, start the algorithm at it:
  Heap=queue (V)
  while Heap isn't empty:
    u=deletemMinimum(Heap)
    for all edges from u (to v):
      if Distances[s][v]>Distances[s][u]+length(u,v) or Distances[s][s] == 0:
        Distances[s][v]=Distances[s][u]+length(u,v)

minCycle=infinity
for all v in V:
  if Distances[v][v] < minCycle and Distances[v][v] != 0:
    minCycle=Distances[v][v]

minCycle = infinity means acyclic
```

- **Q3** Given a directed weighted graph G, with positive weights on the edges, let us also add positive weights on the nodes. Let l(x,y) denote the weight of an edge (x,y), and let w(x) denote the weight of a vertex x. Define the cost of a path as the sum of the weights of all the edges and vertices on that path. Give an efficient algorithm to find all the smallest cost path (as defined above) from a source vertex to all other vertices. Analyze and report the running time of your algorithm.
  use Dijkstra's algorithm, just add the weight of the node you're connected to when you look at the length.

```
create array Distances, all values at infinity
Heap=queue (G)
```

```
while Heap isn't empty:
  x=deletemMinimum(Heap)
  for all edges from x (to y) in G:
    if Distances[x] > Distances[y] + l[x,y] + w[y]:
      if Distances[x]==infinity: \\not really necessary since we just find shortest, but it
                         keeps the distances accurate assumming we count the starting node
        Distances[x] = Distances[y] + l[x,y] + w[y] + w[x]
      else:
        Distances[x] = Distances[y] + l[x,y] + w[y]
```

Just like regular dijkstra's algorithm, runs in O($|V|^2$), you just add the weight of what you visit

- **Q4** Given a directed graph G with possibly negative edge weights. Consider the following algorithm to find the shortest paths from a source vertex to all other vertices. Pick some large constant c, and add it to the weight of each edge, so that there are no negative weights. Now, just run Dijkstra's algorithm to find all the shortest paths. Is this method correct? If yes, reason why. If not, give a counterexample.
  no. This hurts longer paths with shorter weights. Here's a counter example, sorry I can only type it out right now, here's the list of edges, and I explain it in words below, it starts at 1 and we look at distances to 5

u	v	L
1	2	1
2	3	1
3	4	1
4	5	1
1	5	10
1	6	-20

  basically two paths from 1 to 5. Directly from 1 to 5 with weight 10, and going through 2, 3, and 4 to get to 5 with total weight 4
  but there's an edge from 1 to 6 with -20. This could be almost anyweher in the graph, here it's not connected to anything but 1
  The proposed idea is that we add a large number to each length to avoid negatives, I use 20 since that's just enough to hit a minimum of 0
  this results in all the 1's becoming 21's and the 10 becoming 30. This increases the path that was 10 to 30 and the path that was 4 to 84
  this obviously results in an incorrect answer, since the 10 path only had 20 added to it while the 4 path had 80 added to it.