LONGEST PATH PROBLEM

Bradley Woodcock &
Dylan Roth

DECISION PROBLEM

Does there exist a simple path in a weighted, directed graph with k edges?

OPTIMIZATION PROBLEM

Find the longest simple path possible given a weighted, directed Graph.

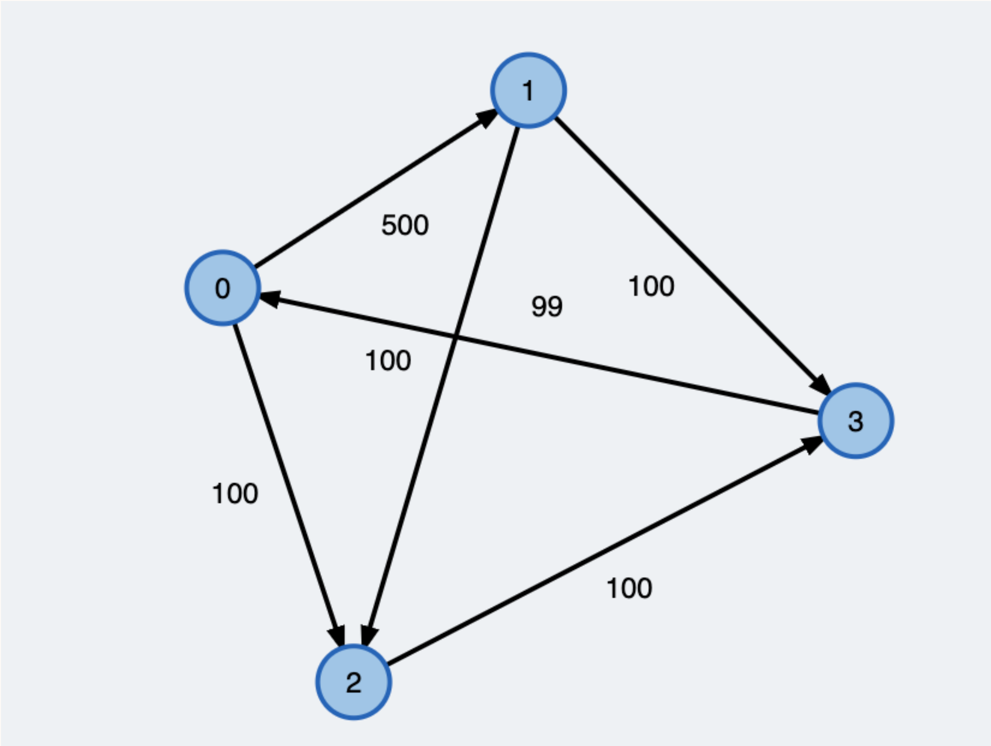
SAMPLE PROBLEM

SAMPLE INPUT

4 6
0 1 500
1 2 100
2 3 100
1 3 100
0 2 100
3 0 99

EXACT OUTPUT

700
0 1 2 3



WHY IS THIS IMPORTANT?

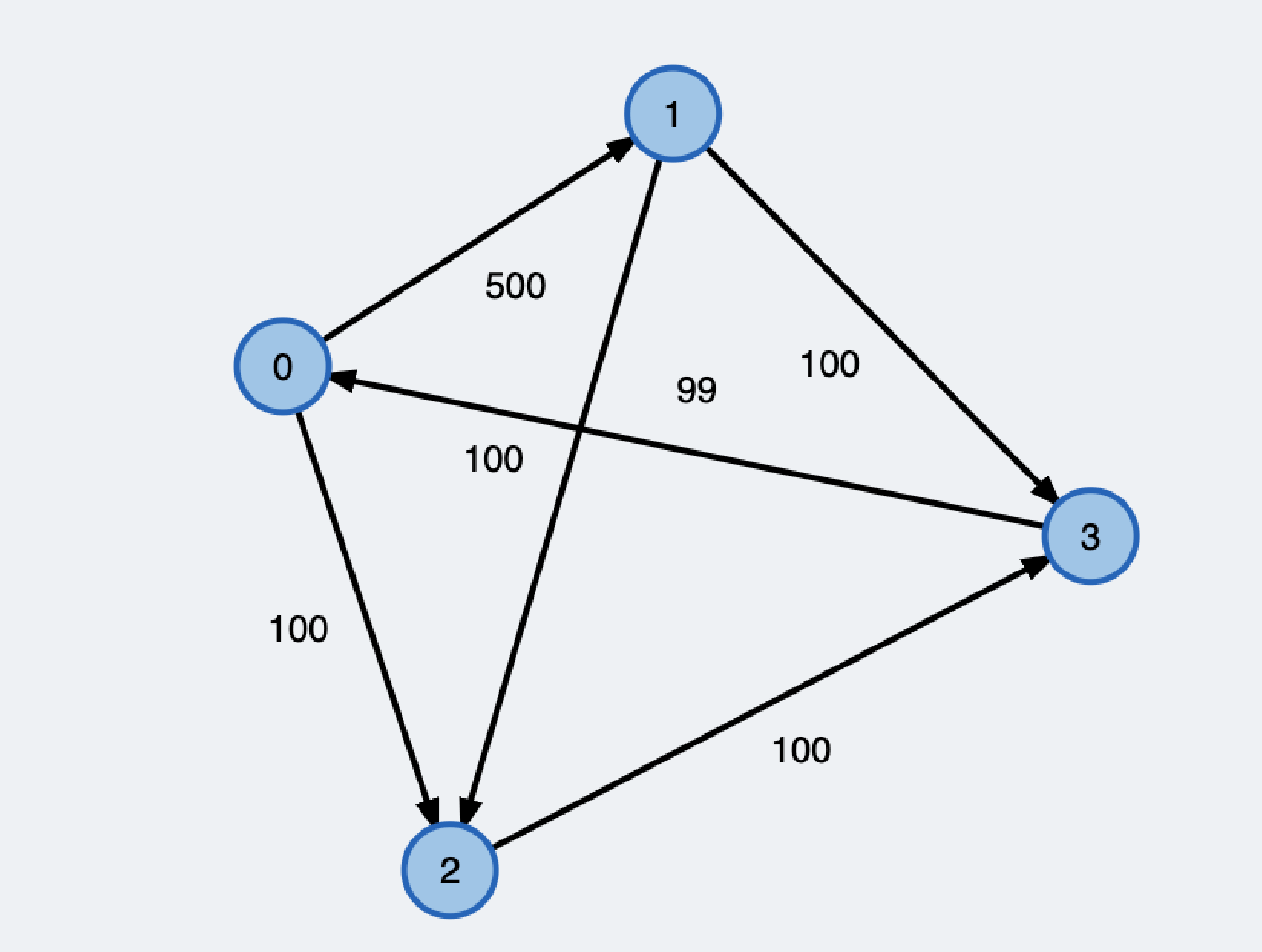
PROJECT TASK SCHEDULING USING LONGEST PATH

- Scheduling a set of activities involves the construction of a directed graph in which the vertices represent project milestones, and the edges represent activities that must be performed after one milestone and before another
- each edge is weighted by an estimate of the amount of time the corresponding activity will take to complete
- In such a graph, the longest path from the first milestone to the last one is the critical path, which indicates the minimum time necessary to project the project.

- Finding the longest paths is useful for analyzing where to place resources (choosing particular edges)
- Example: Which tasks, if they were able to finish slightly early, would help the whole project finish early?

CERTIFIER PROCESS IS POLYNOMIAL

Given a path P in graph G and a length, N, we can go through this path and add its weights in polynomial time. After adding the weights, we can certify the solution by comparing the sum to N.



B

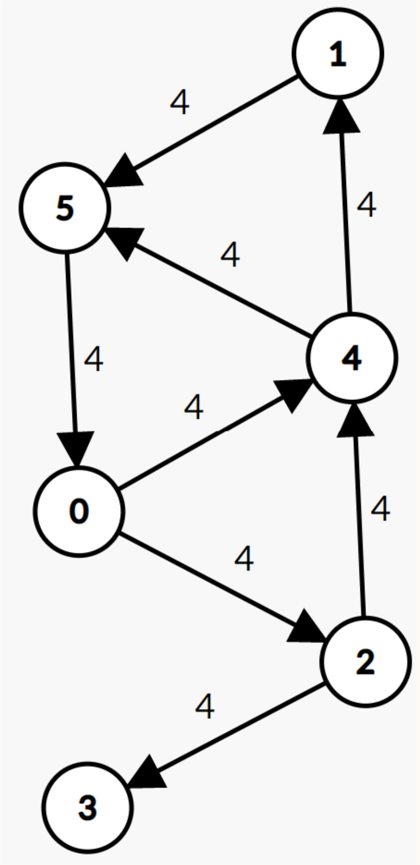
TEST GENERATION

```python
def main():
    # generate the number of vertices and edges
    numVertices, numEdges = input().split(" ")
    numVertices, numEdges = int(numVertices), int(numEdges)

    # generate the edges
    edges = []
    edgesDict = {}
    for _ in range(numEdges):
        u = str(random.randint(0, numVertices-1))
        v = str(random.randint(0, numVertices-1))
        w = str(random.randint(1, 10))
        # make sure node isnt making a loop and that the edge doesnt already exist
        while (str(u) == str(v)) or ((str(u) + str(v)) in edgesDict):
            u = str(random.randint(0, numVertices-1))
            v = str(random.randint(0, numVertices-1))
        edgesDict[str(u) + str(v)] = True

        edges.append((u, v, w))
```

```
3 5
2 0 1
2 1 2
1 0 10
0 2 7
0 1 7
17
1 0 2
Elapsed time with input 3 5 : 4.00543212890625e-05 seconds
```

REDUCTION FROM HAMILTONIAN PATH TO LONGEST PATH

- Hamiltonian Path is a path that visits every node once in a given graph G
- If a Hamiltonian Path exists, then the longest path is of length (n-1) vertices in the graph
- Make a weighted graph into an unweighted graph:
  - Set the weights of all edges to the same length which takes polynomial time: O(n)
- Find the longest simple path in the G
- If the length of that path is n-1, then a Hamiltonian Path does exist



WHY NEGATING ALL EDGE WEIGHTS AND USING BELLMAN-FORD DOESN'T WORK

- Does not work because Longest Path Problem asks for a simple path
- Bellman-Ford computes the shortest path in a graph but can repeat vertices
- Bellman-Ford does not solve the shortest SIMPLE path
- Therefore, a negative cycle would allow the program to continue walking around the cycle forever and never find a SIMPLE path

OUR CODE

- Psuedocode:

```
LongestLen = 0
Path = None
For each path in graph:
    CurrLen = 0
    CurrPath = path[0]
    For each vertice in path:
        If edge from vertice to vertice+1:
            CurrLen += edge
            CurrPath.append(vertice+1)
        Else:
            Break
    If currLen > longestLen:
        LongestLen = currLen
        Path = currPath
```

B

BIG-O

O(v!)

Permutations are size v!

```python
# find all permutations of the vertices
permutations = itertools.permutations(adjlist.keys())

# find the longest path from each permutation
for permutation in permutations:
    currLength = 0
    currPath = []
    currPath.append(permutation[0])
    for i in range(len(permutation)-1):
        if adjlist.get(permutation[i]) and adjlist.get(permutation[i]).get(permutation[i+1]):
                currLength += adjlist[permutation[i]][permutation[i+1]]
                currPath.append(permutation[i + 1])
        else:
            break
    if currLength > maxLength:
        maxLength = currLength
        path = currPath.copy()
return maxLength, path
```

WALL CLOCK RUNTIME

Wallclock Runtime (s)	
	Exact Time
2	5.29E-05
3	8.87E-05
4	0.000132799
5	0.000427008
6	0.001636744
7	0.009511232
8	0.045943737
9	0.301304102
10	2.934784889
11	29.64515686
12	367.1866448
13	7303.976574

D

APPROXIMATION SECTION

Bradley Woodcock & Dylan Roth

APPROXIMATION PSEUDOCODE

```
findLongestPath(adjList):
    Choose a random start vertex from all vertices

    initialize path to [start vertex]
    initialize pathLength to 0
    initialize currVertex to start vertex
    initialize visited to [start vertex]
    initialize unvisited to all vertices
    remove start vertex from unvisited
    while there is still elements in unvisited:
        initialize maxWeight to -10000
        initialize longestNeighbor to None
        for neighbor, weight in adjacency list:
            if neighbor not visited:
                if weight > maxWeight: # choose largest edge
                    set maxWeight to weight
                    set longestNeighbor to neighbor
        if no more unvisited neighbors:
            done.
        add maxLengthNeighbor to path
        add neighbor edge weight to pathLength
        add maxLengthNeighbor to visited
        remove maxLengthNeighbor from unvisited
        update currVertex to maxLengthNeighbor
    return pathLength, path
```

```
main():
    Initialize attempts to 1000
    Initialize longestLength to 0
    Initialize longestPath to None
    for all attempts:
        call findLongestPath(adjList)
            to get currLength, currPath
        if currLength larger than longestLength:
            update longestLength
            update longestPath
```

BIG O TIME COMPLEXITY: O(ATT * N^2)

For Loop nested inside a *While* Loop

APPROXIMATE SOLUTION IS NOT ALWAYS CORRECT

SAMPLE INPUT

```
7 14
0 1 8
5 6 3
5 2 1
6 0 4
0 6 3
5 3 9
4 0 5
2 6 4
6 5 7
1 6 6
4 3 9
3 5 8
1 2 10
6 1 5
```

EXACT OUTPUT

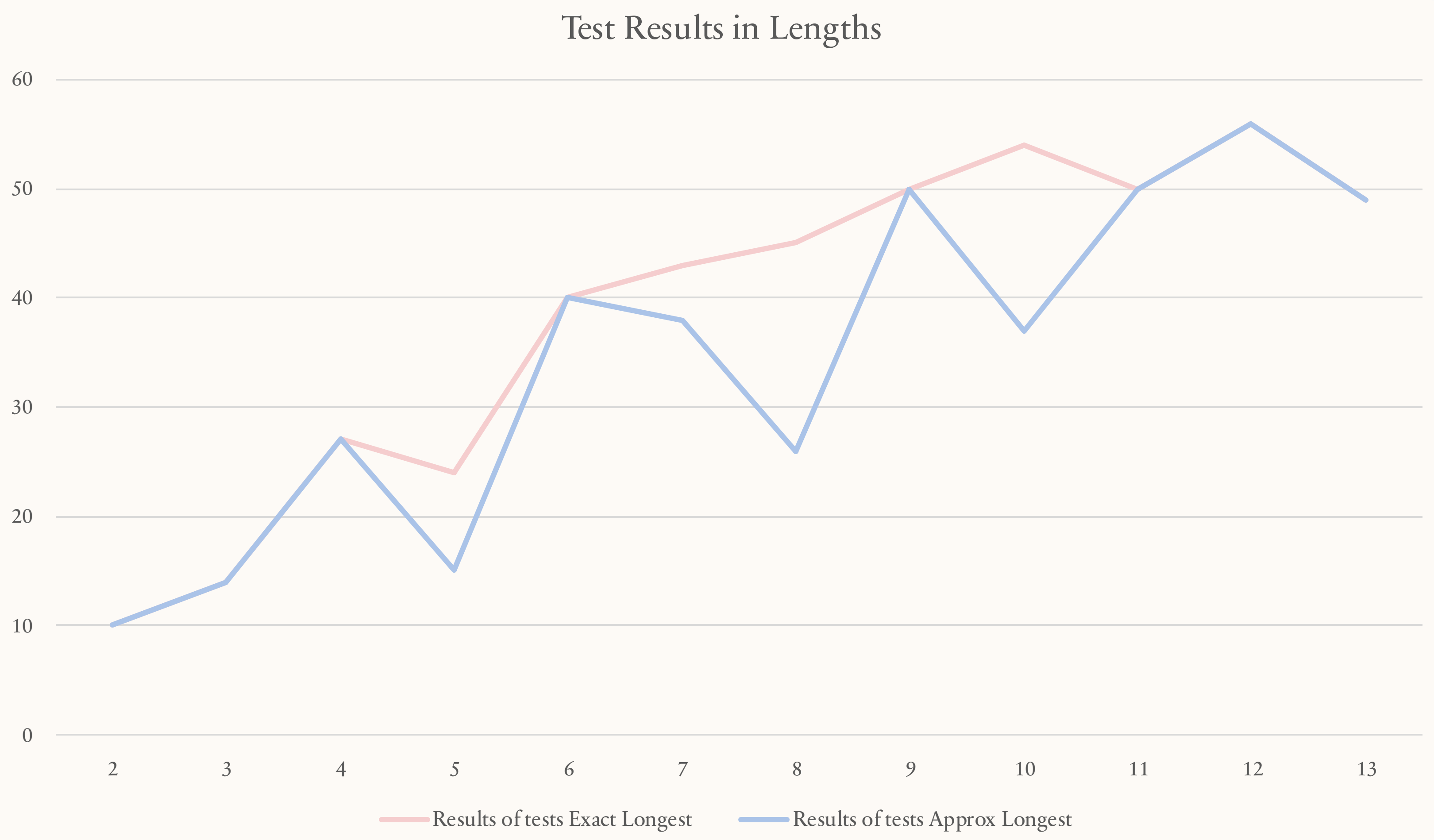
```
43
4 0 1 2 6 5 3
```

APPROX OUTPUT

```
38
0 1 2 6 5 3
```

EXACT VS. APPROXIMATION SOLUTION

	Results of tests	
	Exact Longest	Approx Longest
2	10	10
3	14	14
4	27	27
5	24	15
6	40	40
7	43	38
8	45	26
9	50	50
10	54	37
11	50	50
12	56	56
13	49	49



WALL CLOCK RUNTIME

	Wallclock Runtime (s)	
	Exact Time	Approx Time
2	5.29E-05	0.000305891
3	8.87E-05	0.000353098
4	0.000132799	0.000432014
5	0.000427008	0.000426054
6	0.001636744	0.000545025
7	0.009511232	0.000441074
8	0.045943737	0.000658035
9	0.301304102	0.000568151
10	2.934784889	0.000658035
11	29.64515686	0.000720263
12	367.1866448	0.000638008
13	7303.976574	0.000512123