Class 11/06: Shortest Paths Solutions

1. **Modifying Bellman-Ford to exit early when all paths are done.**
Suppose we have modified the Relax procedure to return TRUE if it changed an edge weight and FALSE otherwise. (The meaning of TRUE and FALSE returned by Bellman-Ford is different, and does not change.) This enables us to write a better encapsulated version of Bellman-Ford (shown in the problem statement but removed here for brevity).

But Bellman-Ford still does unnecessary work if all shortest paths are significantly shorter than |G.V| - 1. Modify the code above to exit the algorithm with the correct value if there will be no further changes. Do this by modifying Bellman-Ford alone: don't change the Relax procedure.

First Try: :

```
Bellman-Ford(G,w,S)
1  Initialize-Single-Source(G,s)
2  for i = 1 to |G.V| - 1
3       change = FALSE
4       for each edge (u,v) in G.E
5           if Relax(u,v,w)
6               change = TRUE
7       if change == FALSE return TRUE
8  for each edge (u,v) in G.E
9       if Relax(u,v,w)
10          return FALSE
11 return TRUE
```

We can make the code simpler by incorporating the last test for negative weight cycles in the main loop:

```
Bellman-Ford(G,w,S)
1  Initialize-Single-Source(G,s)
2  for i = 1 to |G.V| // one extra pass; should not change
3       change = FALSE
4       for each edge (u,v) in G.E
5           if Relax(u,v,w)
6               change = TRUE
7       if change == FALSE return TRUE
```

```
8  return FALSE
```

2. The **parallel scheduling problem** is to take a set of interdependent jobs with known execution time and determine when to schedule each job so that the last job finishes as soon as possible while respecting the interdependency constraints.

We can model such problems using weighted DAGs as follows:

- Create a DAG with a start node s, a finish node f, and two vertices for each job j: a job start vertex j_s and a job end vertex j_f.
- For each job, add an edge from its start vertex to its end vertex with weight equal to its duration.
- For each precedence constraint where job i must finish before job j starts, add a zero-weight edge from i_f to j_s.
- Also add zero-weight edges from the start node s to every job start node j_s and from every job finish node j_f to the finish node f.

But we need an algorithm to do the scheduling. Rather than write an algorithm from scratch, we will see how we can do this with **DAG-Shortest-Paths**. Some adjustment may be required to make the data structure fit the algorithm.

Some jobs are shown in the table to the right. The first job should start at time 0. Your task is to schedule the jobs such that they are all completed in the minimum amount of time while respecting the constraints.

Job	Duration	Before
j1	30	j2, j3, j6, j7
j2	35	
j3	25	
j4	27	
j5	10	j3, j6
j6	21	j2
j7	18	j3, j4

(a) Do we need to find the shortest paths or the longest paths from s to f, and why?

Longest Paths.

If we found shortest paths we'd be simply finding the least constrained job sequence. For example, suppose we added j8 of duration 5 without constraints. Then the shortest path from s to f would be via j8 and of length 5, but we cannot finish the other jobs in this amount of time, even if they all ran in parallel: all have longer duration.

We need to find the longest paths to find the longest sequence of jobs, each dependent on the previous one finishing, as this gives us the *minimum* amount of time to completion.

(b) Do you need to change the weights in any way to solve it with DAG-Shortest-Paths (without changing the algorithm), and if so how and why?

Yes. Since DAG-Shortest-Paths finds shortest paths, we need to negate the job lengths to find the longest path. Then the shortest path in the negated weight graph will be the most negatively weighted path; i.e., the longest path in the original graph.

(c) After running DAG-Shortest-Paths, how do you extract the solution, specifically the start time of each job and the time at which all jobs will be finished?

The negated distance from s to j_s gives the start time of each job j.

The negated distance of f from s gives the finish time of the entire collection of jobs.

(d) Now let's do it. Solve the job scheduling problem for the jobs shown in the table.
Editing the Google Drawing:

- Fill in the missing edge weights
- Run DAG-Shortest-Paths on the modified DAG

SEE SOLUTION GRAPH

Editing this document below:

- Transform the results into a table for each job of the time it can start running and the time it will finish (the start time of s is 0).
- Write the total time to finish all jobs

job	start	end
j1	0	30
j2	51	86
j3	48	73
j4	48	75
j5	0	10
j6	30	51
j7	30	48

Total time: 86