PS6

July 2, 2025

Problem 6-1

We would like to computed for every pair s and $v \in V \backslash \{s\}$ the strength of the strongest connection between them with vagueness at most k.

The vagueness is the number of edges connecting them.

The strength is given by $S(p) = \Pi_{i=1}^{k} ER(u_{i-1}, u_i)$, where $ER(u, v)$ is the EgdeRank which is a value in the range $(0, 1)$ which indicates the interest u shows in v.

Since log is a monotonic increasing function, the path which maximizes $S(p)$ also maximizes

$$\log S(p) = \sum_{i=1}^{k} ER(u_{i-1}, u_i)$$

Moreover, the path which maximizes $\log S(p)$ minimizes

$$-\log(S(p)) = -\sum_{i=1}^{k} ER(u_{i-1}, u_i)$$

For each v satisfying the condition we can solve the problem using a modified version of Bellman-Ford.

The Psuedo code for the modified Bellman-Ford is

Modified_Bellman_Ford(G,W,s):
_Initialize()
_for $i = 1$ to $k - 1$:
__for each edge $(u, v) \in E$:
___Relax(u, v)
_for each edge $(u, v) \in E$:
__if $d[v] > d[u] + w(u, v)$
___then report a negative-weight cycle exists

The running time of the initialization is $O(|V|)$, first nested loop is $O(kE)$, and the final loop is $O(E)$. Therefore the total running time of the algorithm is $O(kE + V)$.

Problem 6-2

The library installation problem and relative dependencies can be represented in terms of the directed acyclic graph (DAG). To choose the appropriate graph representation we note that $V^2 \gg E$. In this case three possible graph representatians come to mind, adjecency lists, or it's object oriented variation, which takes $O(E+V)$ space.

(a)

The installation order can be evaluated using topological sort algorithm. The method performs a depth-first search (DFS), records the order by which the nodes have been processed (or more accuratly, finished to be processed) and reverses this order. The resulting order takes into account the various dependencies between nodes and orders them so there is no violation with respect to these dependencies. The DFS algorithm's running time is $O(V+E)$, revering the order is only $O(V)$, therefore generating the installation order running time is the same. If the graph is not connected than the installation procedure involves picking each time a source node which hasn't been processed.

(b)

The problem can be solved using a modified topological sort algorithm. We modify the DFS algorithm to fit to the modified situation.

```
topological_sort(graph,source):
_r = DFSResult()
_DFS_visit(graph, source, r)
_reverse r.finish
_return r.finished
DFS_visit(graph, node, r):
_for n in neighbors of node:
__if (n not in r.parents) and n (not in installed.keys()):
___r.parents[n] = v
___DFS_visit(graph, n, r)
class DFSResult(object):
_parents = {}
_finished = []
```

Since the maximum number of dependencies is D. The loop over the neighbors has at most D iterations. The DFS algorithm runs over. The initialization and inverting the "finished" array runs in $O(P)$. Therefore the total running time is $O(P+DP) = O(DP)$ If the graph is not connected than the installation procedure involves picking each time a source node which hasn't been processed.

Problem 6-3

Solution is given in "solver.py" which can be tested by "test_solver.py". The test file ran in 0.56 seconds on my laptop. All code files were adjusted so to run on Python3.

Problem 6-4

Solution is given in "dijkstra.py" which can be tested by "dijkstra_test.py". All code files were adjusted so to run on Python3. The path from Boston, MA to Berkeley, CA produced the following output:

Path: BOSTON EAST BOST, MA -> BERKELEY, CA
Graph size: 90415 nodes, 250604 edges
Nodes visited: 84458
Path length: 54161784.7363
Number of roads: 732
Time used in seconds: 0.569