1. Shortest shortest paths

a) Suppose we are given a a directed, **weighted** graph $G = (V, E)$ with only positive edge weights. For a source vertex s, design an algorithm to find the shortest path from s to all other vertices with the fewest number of edges. That is, if there are multiple paths with the same total edge weight, output the one with the fewest number of edges.

Complete the function `shortest_shortest_path` and test with the example graph given in `test_shortest_shortest_path`. Note that the `shortest_shortest_path` function returns both the weight and the number of edges of each shortest path.

.
.
.

b) What is the work and span of your algorithm?

put in answers.md

.
.
.

.
.
.

2. Computing paths

a) We have seen how to run breadth-first search while keeping track of the distance of each node to the source. Let's now keep track of the actual shortest path from the source to each node. First, observe that the order in which BFS visits nodes implies a tree over the graph:

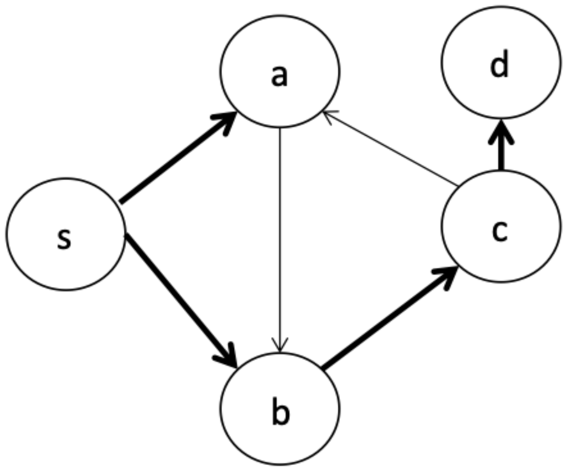

Figure 1: bfs.png

Here, the dark edges indicate all the shortest paths discovered by BFS. To keep track of the paths, then, we just need to represent this tree. To do so, we can store a `dict` from a vertex to its parent in the tree. In the above example, this would be:

```
{'a': 's', 'b': 's', 'c': 'b', 'd': 'c'}
```

Complete the `bfs_path` function to return this parent `dict` and test it with `test_bfs_path`. Your algorithm should not increase the asymptotic work/span of BFS.

b) Next, complete `get_path`, which takes in the parent `dict` and a node, and returns a string indicating the path from the source node to the destination node. Test with `test_get_path`.

3. Improving Dijkstra

In our analysis of the work done by Dijkstra's algorithm, we ended up with a bound of $O(|E| \log |E|)$. Let's take a closer look at how changing the type of heap used affects this work bound.

a) A d-ary heap is a generalization of a binary heap in which we have a d-ary tree as the data structure. The heap and shape properties are still maintained, but each internal node now has d children (except possibly the rightmost leaf). What is the maximum depth of a d-ary heap?

put in answers.md

.
.
.

b) In a binary heap the `delete-min` operation removes the root, places the rightmost leaf at the root position and restores the heap property by swapping downward. Similarly the `insert` operation places the new element as the rightmost leaf and swaps upward to restore the heap property. What is the work done by `delete-min` and `insert` operations in a d-ary heap? Note that the work differs for each operation.

put in answers.md

.
.
.

c) Now, suppose we use a d-ary heap for Dijkstra's algorithm. What is the new bound on the work? Your bound will be a function of $|V|$, $|E|$, and d and will account for the `delete-min` and `insert` operations separately.

put in answers.md

.
.
.

d) Now that we have a characterization of how Dijkstra's algorithm performs with a d-ary heap, let's look at how we might be able to optimize the choice of d under certain assumptions. Let's suppose that we have a moderate number of edges, that is $|E| = |V|^{1+\epsilon}$ for $0 < \epsilon < 1$. What value of d yields an overall running time of $O(|E|)$?

put in answers.md

.
.
.

4. All-pairs shortest paths using Dynamic Programming

In class we looked at the all-pairs shortest path (APSP) problem, and used a graph transformation so that we could essentially run Dijkstra with all possible starting source vertices. This resulted in an algorithm for APSP with work $O(|V||E| \log |E|)$. Let's consider a more direct approach using dynamic programming.

Suppose that we label the vertices from 0 to $n-1$, and let $APSP(i, j, k)$ be the weight of the shortest path between vertices i and j such that only vertices $0, 1, \ldots, k$ are allowed to be used. Then the cost of the shortest path between vertices i and j is then given by $APSP(i, j, n-1)$.

a) Consider the following graph with 3 vertices.

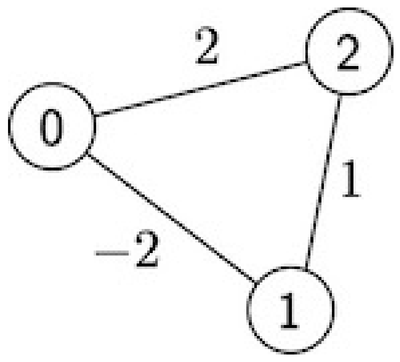

Figure 2: apsp_example.jpg

Compute $APSP(i, j, k)$ for all i, j, k.

put in answers.md

.
.
.

b) Do you see a relationship between $APSP(i, j, 1)$ and $APS(i, j, 2)$? Can you write $APS(i, j, 2)$ in terms of $APS(i, j, 0)$ and $APS(i, j, 1)$ only?

put in answers.md

.
.
.

c) Suppose that an oracles makes available to us all possible values for $APSP(i, j, k - 1)$ for all i, j and some particular value of $k - 1 < n$. Then what is the shortest path cost $APSP(i, j, k)$? Well, it is either $APSP(i, j, k - 1)$, or some other path from i to j that has length k. Generalize your observation from b) above to give an optimal substructure property for $APSP(i, j, k)$.

put in answers.md

.
.
.

d) As usual naively implementing this optimal substructure property to compute $APSP(i, j, n - 1)$ for all i, j will be inefficient. Suppose we perform top-down memoization so that we only ever compute each subproblem from scratch once. How many distinct subproblems will be computed from scratch, and what is the resulting work of this dynamic programming algorithm?

put in answers.md

.
.
.

e) Compare the work of this algorithm against that of Johnson's algorithm. In what cases is our dynamic programming algorithm preferable?

put in answers.md

.
.
.

5. Spanning trees

a) Consider a variation of the MST problem that instead asks for a tree that minimizes the maximum weight of any edge in the spanning tree. Let's call this the minimum maximum edge tree (MMET). Is a solution to MST guaranteed to be a solution to MMET? Why or why not?

put in answers.md

.
.
.

b) Suppose that the optimal solution to MST is impossible to use for some reason. Describe an algorithm to instead find the next best tree (pseudo-code or English is fine). That is, return the tree with the next lowest weight.

put in answers.md

.
.
.

c) What is the work of your algorithm?

put in answers.md

.
.
.