Programming Assignment (80%)

A delivery drone is an unmanned aerial vehicle (UAV) utilized to transport packages, food or other goods. In this programming assignment, you will need to implement the following search algorithms to deliver goods from a starting position S to a destination position D in an undirected weighted graph G:

- Breadth-first search
- Depth-first search
- Uniform-cost search using the amount of fuel needed as cost

The drone has an initial amount of fuel, F, and cannot refill until it reaches the destination. The nodes in G represent the places where the drone can travel to. If an edge exists between node A and node B, the drone can travel from A to B. The weight of an edge corresponds to the amount of fuel needed to traverse that edge. So, the route selection will be restricted by the amount of fuel that the drone has. For example, when the drone is in node A, and there exists a path between A and B, the drone can go to B if and only if it has the necessary amount of fuel. Otherwise, B will be considered unreachable from A.

If a path from S to D exists under the fuel constraint, you should return the path; otherwise you should return “No Path”.

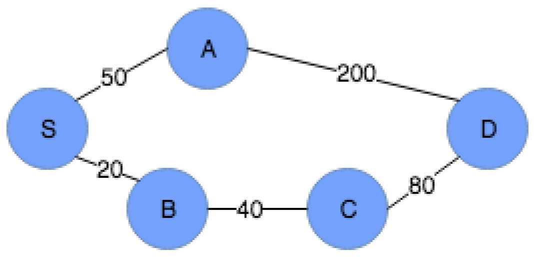


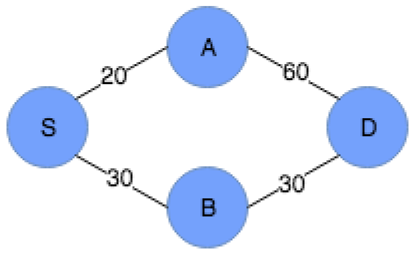
Figure 1. Sample Travelling Graph.

Note: When the costs of two or more nodes are equal, you need to make sure these nodes are popped out of the frontier in alphabetical order. This will resolve ambiguity and ensure that there is only one correct output for each input.

Sample Input and Output:

The input file will contain the search algorithm, the initial fuel amount, the starting and destination positions, and the graph representation. The output file should contain the found path (if one exists, otherwise “No Path”) and the remaining amount of fuel at the destination. We will now see this in greater detail with a few examples.

For BFS: Input and Output



Input File:
```
    BFS
    70
    S
    D
    S: A-20, B-30
    A: S-20, D-60
    B: S-30, D-30
    D: A-60, B-30
```
Output File:
```
    S-B-D 10
```

In the input file, the first line indicates the algorithm to be used, which is either BFS or DFS or UCS. The second line is the initial fuel amount. In the above example, it is 70. The third and fourth lines are the start and goal node names, respectively. The rest represents the graph. For instance, in the above example “S: A-20, B-30” means node S has two neighbors, A and B, the cost between S and A is 20, and the cost between S and B is 30. You could assume the input file is always in the correct format.

The returned path in the above example should be “S-B-D”, because with BFS, from S, both A and B are reachable and get added to the frontier. If we expand the nodes in alphabetical order, we expand A first and the remaining fuel at A will be 70-20=50. Note that because the remaining fuel at A is less than the amount of fuel needed to go to D from A, D does not get added to the frontier at this point. Then we expand B and the remaining fuel at B will be 70-30=40. Now we can add D to the frontier. Lastly, we extract D and the search will be done.

For DFS: Input and output

Input File 1:
```
    DFS
    270
    S
    D
    S: A-50, B-20
    A: S-50, D-200
    B: S-20, C-40
    C: B-40, D-80
    D: A-200, C-80
```

Output File 1:

```
S-A-D 20
```

Input File 2:

```
DFS
100
S
D
S: A-50, B-20
A: S-50, D-200
B: S-20, C-40
C: B-40, D-80
D: A-200, C-80
```

Output File 2:

```
No Path
```

Note that the graph in these two input files is the same as figure 1.

For UCS: Input and output

Input File:

```
UCS
270
S
D
S: A-50, B-20
A: S-50, D-200
B: S-20, C-40
C: B-40, D-80
D: A-200, C-80
```

Output File:

```
S-B-C-D 130
```

Note: Node names can be any strings consisting of only alphanumerical characters [a-zA-Z0-9]. All test cases will follow the same format, including the placement of whitespace and new-line characters. A node can have one or more connecting edges. All edge costs and the initial fuel amount will be integers greater than or equal to zero.