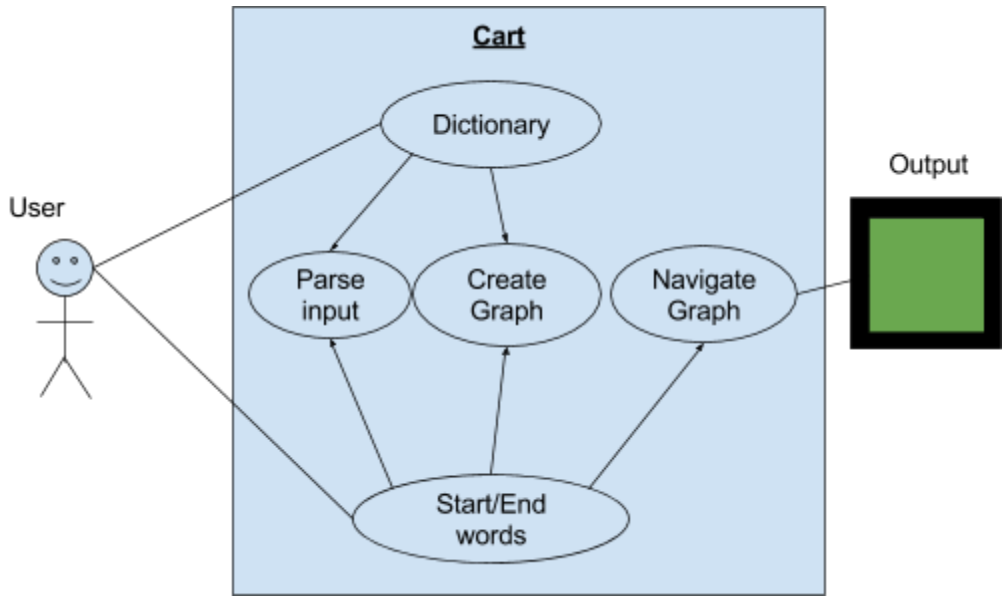
Andrew Ferrari acf2282]
Brian Gao bcg833

Design Paragraph:

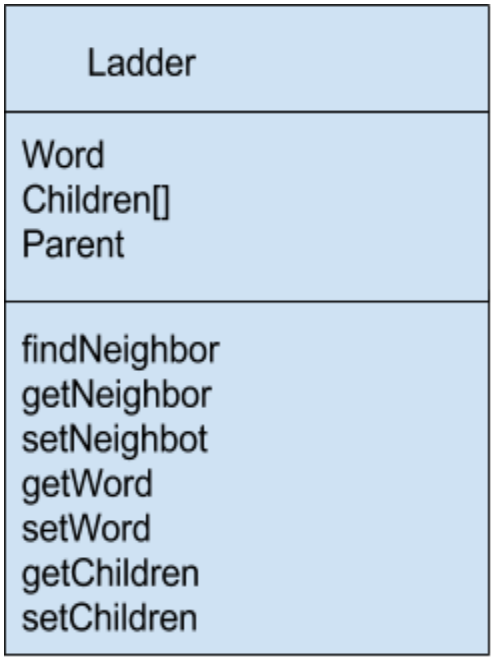
Our design represents the dictionary as a String ArrayList taken from the dictionary input file. We then take the starting word and construct a graph of words with adjacent nodes having one letter differences. The graph is constructed from the top down, so duplicate nodes are not created. We decided to use this breadth first design as opposed to a depth first design because a BFS always finds the shortest path between two nodes. Our design adheres to good principles of design by protecting information such as the dictionary, but allowing getter and setters within the internals to manipulate our objects.

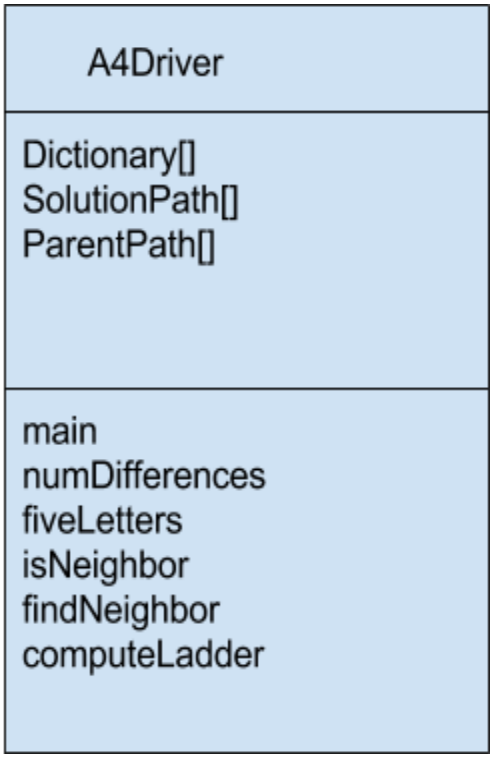
Use Case:



UML Model:

Andrew Ferrari acf2282]
Brian Gao bcg833





Driver Algorithm:

Andrew Ferrari acf2282]
Brian Gao bcg833

boat

```
int found=0;
arrayLinked = new arrayList();
bfs = new Queue();
bfs.enqueue(startword);
while (bfs ≠ empty) {
    currentWord = bfs.dequeue();
    if (currentWord is not in the arrayList) {
        add currentword to arrayLinked;
        for (every child childNode[] of currentword) {
            if (childNode[i] == endword) {
                found = 1;
                break;
            }
            for loop for j
            if (childNode[i] ≠ arrayLinked[j]) {
                *childNode.parent = currentword;
                bfs.enqueue(&childNode)
```

pop

push

Find a way to do this

Make an array list of all children of x called childNode(Rank) this array list based on difference in letters to target word

how many children

pop

push

```
if (found == 1)
    break;
```

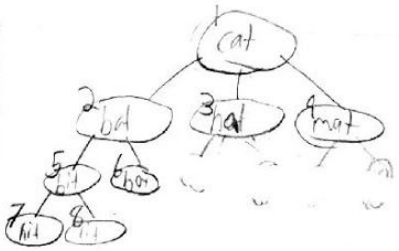
bfs = queue

Actors:

1
2
3
4
5
6
7
8

push 2

push 5



arrayLinked.get(parentList.get(currentIndex))

	arrayLinked
0	cat
1	bat
2	hat
3	mat
4	bit
5	hot
6	hit
7	hit

	parentList
0	-1
1	0
2	0
3	0
4	1
5	1
6	4
7	4