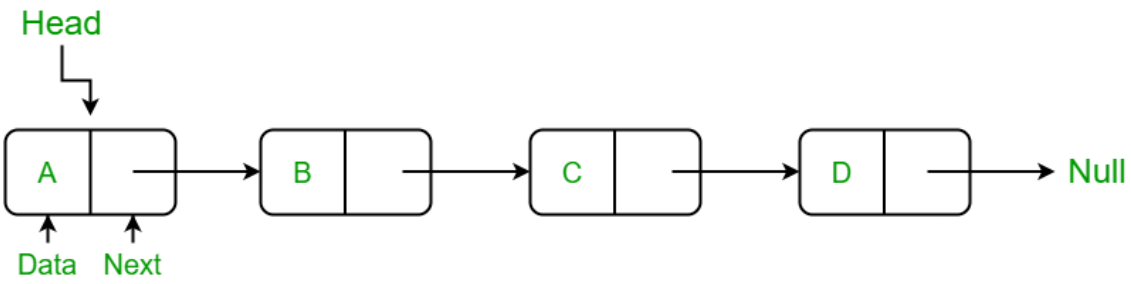
Lab 21: Tree Search

Assignment 6 will have you implement breadth first search (BFS) and depth first search (DFS), so this lab will not cover them in detail. Instead, it will cover some useful tools that will help you implement the DFS and BFS.

1. **Reversing a linked list (also finding the start of a linked list)**. The simplest tree structure (and in extension graph structure) is a linked list. Each node has a single pointer to the next element, or in some cases to its parent element. The most basic case of having a single link is called a singly linked list, while those with both parent and next element are called doubly linked lists. In relation to BFS and DFS, at the end of running these algorithms, you need to extract the solution path from the different nodes. The linked list starting from the goal node traces its parent all the way back to the starting node.



In some cases, you may actually need to reverse the links rather than just get the data in reversed order (in which case you should just traverse the list then use `std::reverse`). Try to do this in an iterative manner (the recursive method is much less intuitive). You should implement the `reverseList` function below, which requires a pointer to the new head (previous tail). It should reverse the list in-place, meaning you shouldn't create a duplicate list.

```
#include <iostream>
#include <memory>

class Node {
public:
    int index;
```

```
    std::shared_ptr<Node> next;
};
using NodePtr = std::shared_ptr<Node>;

void printList(const NodePtr root) {
    auto n = root;
    while (n->next != nullptr && n->next != root) {
        std::cout << n->index << "->";
        n = n->next;
    }
    std::cout << n->index << std::endl;
}

NodePtr reverseList(NodePtr root) {
    // --- Your code here
    // ---
}

int main() {
    auto root = std::make_shared<Node>(Node{});
    root->index = 0;
    auto n = root;
    for (int i = 1; i < 10; ++i) {
        n->next = std::make_shared<Node>(Node{});
        n->next->index = i;
        n = n->next;
    }

    printList(root);
    root = reverseList(root);
    printList(root);
}
```

This should print out

```
0->1->2->3->4->5->6->7->8->9
9->8->7->6->5->4->3->2->1->0
```

2. **Implicit Graph Searches and Neighbors**. Tree search algorithms rarely search a graph that is explicitly defined. It is much more common to work with implicit graphs instead of explicit graphs. Implicit graphs do not have all the possible nodes stored in memory ahead of time, instead generating them by performing actions on a node to find its neighbors. In this problem, we explore such a case.

   We return to the familiar least squares problem `Ax = b` and this time we will solve a simplified form of it by forming it as a graph search problem. Suppose `x` is

restricted to having integer values. We can then define an action as `+-1` along a single dimension. Implement the function below:

```
#include <iostream>
#include <vector>
#include <eigen3/Eigen/Eigen>

using State = Eigen::Vector3d;

std::vector<State> neighbors(const State& start) {
    std::vector<State> neighbors;
    // --- Your code here
    // ---
    return neighbors;
}

int main() {
    State x;
    x << 3, 4, 0;
    auto n {neighbors(x)};
    for (const auto& xx : n) {
        std::cout << xx.transpose() << std::endl;
    }
}
```

Since the space of integers is infinite, we also have a possibly infinite-depth search tree; however, the branching factor is limited by the number of neighbors we can generate at each state.

Often there are constraints on the state that limit which states are valid. For example, in this problem, suppose that we require the third element must be smaller than the absolute difference between the first and second elements. How would you modify `neighbors` to account for this constraint?

Tree search algorithms start with a list of states of explore, then add onto that list the neighbors of the explored states. This problem stops here, but you are two steps away from implementing BFS or DFS (just need a data structure to store exploring and explored states).