Searches

Havish Malladi

October 2019

1 BFS

The first topic we will talk about is BFS. BFS, or Breadth-First Search, checks each layer of nodes with the same depth before moving onto the next layer. As a result, it is very useful for shortest path problems on unweighted graphs.

Algorithm 1 BFS

```
fringe = new Queue()
fringe.append(start)
visited = new Set()
while fringe is not empty do current = fringe.pop()
    if current is goal then return goal
    for k in current children do
        if k not in visited then
            fringe.append(k)
            visited.add(k)
```

Example 1 The black king is standing on a chess field. We will denote a cell of the field that is located in the i-th row and j-th column as (i,j). You know that some squares of the given chess field are allowed. All allowed cells of the chess field are given as n segments. Each segment is described by three integers r_i, a_i, b_i ($a_i \leq b_i$), denoting that cells in columns from number a_i to number b_i inclusive in the r_i-th row are allowed. Your task is to find the minimum number of moves the king needs to get from square (x_0, y_0) to square (x_1, y_1), provided that he only moves along the allowed cells. In other words, the king can be located only on allowed cells on his way (Codeforces 242C).

Using the BFS algorithm described above, this problem becomes very simply. We will define a node as the combination of the current position and the path travelled so far. For the children of each node, we will simply compute all 8 possible combinations and then verify if any of these combinations are in our set of possible moves. For each one of the 8 adjacent squares, if we can move there, we do and add this node to the Queue. We continue this process until we reach the end in which we return the distance of the path travelled so far.

Obviously, there are changes that can be made to a BFS to make it more efficient, such as a Bidirectional BFS, in which you search from the start and the finish and try to meet up at the middle. This effectively cuts the number of iterations of the while loop by about a half, making your code much faster.

2 DFS

The second topic we will be talking about is DFS, or Depth-First Search. Instead of exploring the lowest depth nodes, DFS does the exact opposite. It checks the deepest nodes first, which results in it following a path all the way down before moving on to a new path. DFS is preferable over BFS for its potentially lower memory usage. Additionally, its recursive implementation can be extremely short.

Algorithm 2 DFS

```
fringe = new Stack()
fringe.append(start)
visited = new Set()
while fringe is not empty do current = fringe.pop() visited.add(current)
    if current is goal then return goal
    for k in current children do
        if k not in visited then
            fringe.append(k)
```

As you can see, BFS and DFS are very similar except for changing the Stack to a Queue and moving the location where nodes are added to the visited set.

3 Binary vs Linear Searches

Searching is a powerful tool that will come up often in the USACO competition. Searching is the process of retrieving information stored within some data structure. Here we will look into two possibilities of this and discuss the advantages and disadvantages of both methods.

3.1 Linear Search

Linear Search is the simplest type of search. It has complexity $O(N)$ and searches through an array until it can find the goal element. While this might work for small amounts of numbers, when working with data this process is time consuming.

Algorithm 3 Linear Search

```
for x in [0, len(array)] do
    if array[x] = goal then return x
```

3.2 Binary Search and Modifications

Binary Search is a more efficient search when dealing with large amounts of **ordered** data. It has a time complexity of $O(\log N)$. Each iteration of the loop the algorithm will split the data in half based on whether the middle number is higher or lower than the goal element. This explains it's $O(\log N)$ time complexity. One modification of the binary search is the ternary search. Instead of splitting the data in half each iteration, a ternary search splits it into thirds, making it more efficient. Modifications of the binary search may be required in some USACO problems. **Note: BINARY SEARCHES AND MODIFICATIONS OF IT ONLY WORK WITH ORDERED DATA**

Algorithm 4 Binary Search

```
high = len(array)
low = 0
mid
while low < high do
    mid = (high + low) / 2
    if array[mid] < goal then
        low = mid + 1
    else
        high = mid

```

4 Problems

Problem 1 CodeForces 1057A Once upon a time there was only one router in the well-known company Bmail. Years went by and over time new routers were purchased. Every time they bought a new router, they connected it to one of the routers bought before it. You are given the values p_i the index of the router to which the i^{th} router was connected after being purchased ($p_i < i$). There are n routers in Boogle in total now. Print the sequence of routers on the path from the first to the n^{th} router.

Problem 2 2016 USACO December Silver Problem 3 Instead of mooing at each-other over long distances, the cows decide to equip themselves with walkie-talkies, one for each cow. These walkie-talkies each have a limited transmission radius – a walkie-talkie of power P can only transmit to other cows up to a distance of P away (note that cow A might be able to transmit to cow B even if cow B cannot transmit back, due to cow A's power being larger than that of cow B). Fortunately, cows can relay messages to one-another along a path consisting of several hops, so it is not necessary for every cow to be able to transmit directly to every other cow. Due to the asymmetrical nature of the walkie-talkie transmission, broadcasts from some cows may be more effective than from other cows in their ability to reach large numbers of recipients (taking relaying into account). Please help the cows determine the maximum number of cows that can be reached by a broadcast originating from a single cow

Problem 3 2017 USACO Feburary Silver Problem 2 The long road through Farmer John's farm has N crosswalks across it, conveniently numbered 1 to N ($1 \leq N \leq 100,000$). To allow cows to cross at these crosswalks, FJ installs electric crossing signals, which light up with a green cow icon when it is ok for the cow to cross, and red otherwise. Unfortunately, a large electrical storm has damaged some of his signals. Given a list of the damaged signals, please compute the minimum number of signals that FJ needs to repair in order for there to exist some contiguous block of at least K working signals.

Problem 4 2018 BioCode Problem O Rahul is an avid lover of math and when discussing sequences, he exclaimed to the class that, "Strictly increasing sequences are srue(trait of being extremely cool)." Given a sequence please print the length of the longest subsequence that are strictly increasing (there can be multiple).
Input The input will consist of two lines. The first line will consist of the number N which is the length of the given sequence. The second line of input will consist of the sequence (of length N) itself.
Output Print out one line. The line should contain the length of the longest strictly increasing subsequence.