Algorithms and Data Structures - Assignment 2 (Divide and Conquer)

Introduction

In this report, we present a divide and conquer algorithm to efficiently search for a package based on its code in a 2D grid provided by a fictional company called LogTech. The grid represents a map of locations where packages are left, with the package codes being sorted in both dimensions of the 2D grid. The primary goal of the algorithm is to return the label of the location where the given package code can be found, or None if the package code is invalid.

The problem at hand involves searching the package location on a 2D grid, exploiting the properties of the grid (sorted in both dimensions) to optimize the search process. Our divide and conquer algorithm aims to perform this search operation more efficiently than a linear scan over all cells, thereby improving performance and reducing time complexity.

Our divide-and conquer approach is much faster because, depending on the search value, we subdivide our grid in 3 sub-rectangle. If the search value is greater than the middle value of the grid we search the top right, bottom left and bottom right sub rectangles. If the value is smaller than the middle value of the grid we search the top left, top right and bottom left sub-rectangles. We keep subdividing the various sub grids until we arrive at a problem we can easily verify to be true or false. Because of the subdivision we achieve a much faster running time,

This is more efficient because a linear scan search each and every value in the gird. Where as our approaches subdivides the grid depending on the relation between the current middle value of the grid and the searched value. This way the algorithm can cut away grids and values which are never going to contain the searched value. This cutting away, makes our divide-and-conquer approach much more efficient.

Search Tree

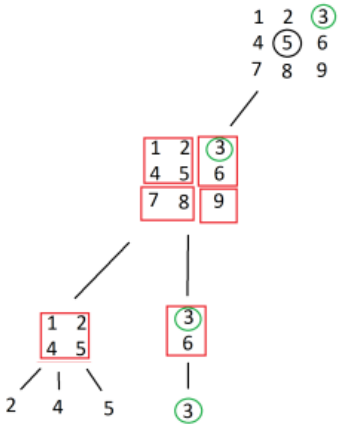

Figure 1: Greedy approach

In figure 1 we illustrate a small example of a problem where we are searching for the value 3. Using our divide-and-conquer algorithm we traverse the search tree. Since our search value is smaller than our mid value we first search through the top left quadrant. After that we search through the top right and bottom left.

The implementation of this divide-and-conquer algorithm is optimal because there is only one solution to find for the problem and it will always find the solution we are searching for. This is because the algorithm traverses the search tree, and by extension, the branches which could contain the search value. In our case, since the search space is ordered in both dimensions we can assume that if the search value is smaller than the mid value the search value can never be in the bottom right quadrant. So, we don't have to search that quadrant. We do search through

all other quadrants and the search value must be in one of these. In the figure we highlight the single cell "9" which is quadrant 4, but we never search through it. It was more meant to show how the matrix is divide into 4 quadrants in our mind.

The Greedy Approach

A greedy approach to search for a given package code could be to always move in the direction where the package code is increasing and closer to the target value. In other words, starting from the top-left corner of the grid, the algorithm would choose either to move right or down at each step, depending on whether the package code in the right cell or the bottom cell is closer to the target value.

The greedy approach can work well in certain cases, but it is not guaranteed to be optimal or even complete. This is because the algorithm might get stuck in a local optimum, choosing sub optimal decisions in pursuit of the target package code, or it might miss the target package code altogether.

Consider the following adversarial test case:

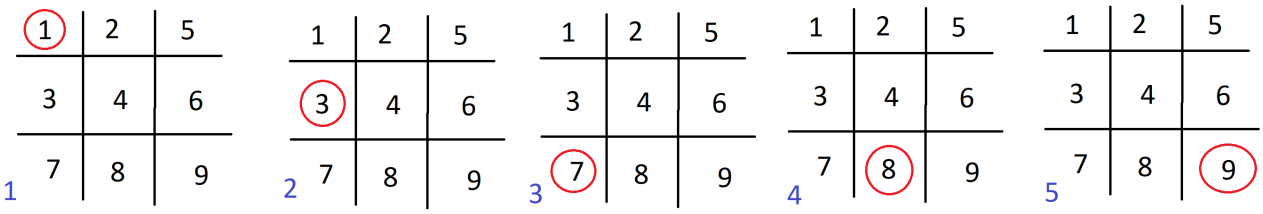

Figure 2: Greedy approach

In figure 2 an adversarial test case of the greedy approach described is shown. Here the pointer moves either to the right or to the cell below. We can see that the greedy approach keeps choosing the cell that has the number closest to 6. This will yield no answer as it keeps moving down until it cannot and afterwards keeps moving right until no further moves can be made. This behaviour clearly showcases the main weakness of the greedy approach; choosing a local optimum without taking into account the global optimum.

Experimental design

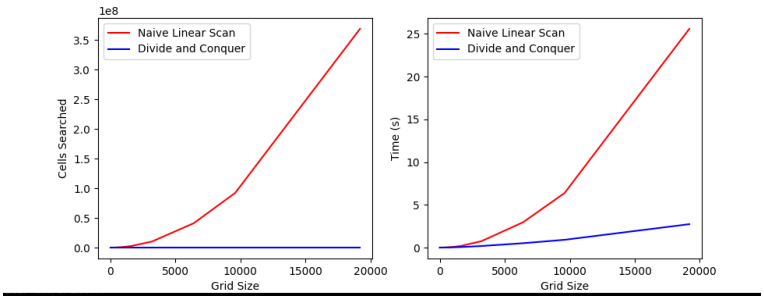

Figure 3: Experimental design graph

The design description:

1. A linear search algorithm consisting of a double for loop.
2. A modified divide-and-conquer algorithm that keeps track of cells searched
3. A method that generates an n x n grid
4. A for loop that executes both the search algorithms for x amount of grids of increasing size and keeps track of cells searched and execution time.
5. A code block dedicated to plotting 2 graphs from figure 3 (using matplotlib).

The grids are comprised of values from {1,2,3 .... } ordered, increasing where the grid is square. The searched grid sizes are; 3, 5, 10, 50, 100, 200, 400, 800, 1600, 3200, 6400, 9800, 19600, 39200.

So for example a 10x10 grid where the values start with 1 and go up to 100.

We set the search value as the last value in the grid to get a worst-case scenario for both search algorithms.

We keep track of the running time and cells searched for both algorithms and the results can be observed in figure 3.

What can be observed from this figure is that the linear scan clearly searches through every cell in the grid, which is quite logical since it loops through every value in the grid. On the contrary, our divide and conquer algorithm barely searches any cells since it's constantly dividing it's search space. Here it looks like the line is 0, but when the values are printed they are increasing slowly but the difference between cells being searched in divide-and-conquer and cells being searched in a linear scan are so massive that, in the graph, it seems like divide-and-conquer is not searching any cells.

We can also observe that the running time of our linear scan goes up way faster than our divide and conquer approach. This is logical since the linear scan searches through every value in the grid, whilst the divide and conquer approach searches through a fraction of these cells since it's constantly dividing it's search space.

From this experiment we can conclude that our divide and conquer algorithm is a lot faster than our linear scan whilst also not traversing the entire search tree. This way it also saves a lot of memory.

Summary and Discussion

The goal in this assignment was to understand how a divide-and-conquer algorithm works and how it differs from a linear scan.

We can conclude that the divide-and-conquer algorithm implemented here for this given problem is optimal, since it always finds the solution if it is present.

An observation can be made about the greedy approach described in this report, The greedy approach isn't always optimal, since the greedy approach is at any time looking for a local optimal choice whilst not taking into account the global optimum. So in the case of this report it may or may not find a solution, depending on the given 2D Grid (in case of the example given, it will definitely not find a solution).

From the experiment the conclusion can be made that there is a clear difference between a linear scan and the implemented divide-and-conquer algorithm. The linear scan searches all possible cells in the grid whilst the divide-and-conquer approach keeps subdividing the grid until it finds a solution whilst, every time shrinking it's search space. The running time of a linear scan is also a lot longer than a divide and conquer algorithm which can clearly be observed from figure 3 of the Experimental design section.

Contributions

Rajiv Jethoe (3490750): Code 50/50, Report 50/50
Steven Wang (3730816): Code: 50/50, Report: 50/50