Deep Roots or Broad Roots?

CS110: Computation: Solving Problems with Algorithms
Minerva Schools at KGI

Austin Perzben / Agustín Pérez del Castillo
December 2019

Table of Contents

Deep Roots or Broad Roots?

Computational Solution Contextualization

The campaign

The ArborDay foundation, alongside internet-based content-creators Mark Rober and Jimmy Donaldson, recently run a fundraiser for USD 20,000,000 in order to plant 20,000,000 trees, under the title #TeamTrees. An issue the foundation works to address is the fact that trees cannot be planted in any location if they are intended to survive for extended periods of time. Silvics of the species and soil conditions necessary need to be understood and factored in to find the optimal place to plant a given species of tree.

The computational challenge

The problem is modeled using a graph in which different nodes represent different locations where the ArborDay foundation can plant a species of trees, and their possible approaches to determining which species to plant where is represented through the implementation of two search algorithms programmed to identify the optimal placement for each of the tree species. I implemented graphs of nodes and species of trees as objects with a set of attributes, the values of which are compared to produce a ‘match score’ and the goal of the algorithms is to optimize this match score for each species.

Relevance

➢ Problem

The #TeamTrees campaign represents and an attempt to address an extremely pressing issue, the climate crisis. The silvics-matching issue that the ArborDay foundation faces having agreed to plant the trees raised through the campaign, is extremely relevant to the overall purpose of the campaign because if the trees were planted without this consideration in mind, they would not survive long and the huge effort that was put into #TeamTrees would not have the impact it can have.

➢ Solution

The computational solution I designed, is based on a somewhat simplified, simulated model of how the problem would need to be addressed if dealt with in practice and not virtually. The foundation would need to search each of the places where they can plant the trees, and decide what species to plant in each place. The relevance of a computational solution to this problem is salient in the very least it provides the foundation with the opportunity to decide what search strategy, bread-first or depth-first, is more better suited to find the right place to plant each tree. Furthermore, if the foundation were to run this model using real silvics data for the trees they will plant, and environmental conditions for the areas in the map where they can plant the trees, this computational solution would essentially provide them with a roadmap of how to traverse this map, and which trees to plant where, as they do so.

The Algorithmic Strategies

Implementations

I addressed this problem by programming both a Breadth-First Search (BFS) and a Depth-First Search (DFS) algorithmic strategy. They both take the same inputs and provide the same outputs, which I did intentionally, in order to simplify their comparative analysis. The only factor underlying performance differences is without variance the search strategy as it is implemented since both approaches use the same 'match score' calculation method when deciding whether or not to plant a tree in a certain place, receive data from the same random generation functions, and have the same possible values for every input.

The main contrast points are identifiable in the fact that the DFS implementation has to account for fall-back scenarios and therefore utilize the track-record of the order in which each node is visited. In the spirit of fair play, I also implemented the counting system for the BFS, because in practice it is not used and the added memory cost is simply that of substituting a boolean value for an integer, which is updated the same number of times. In order to save memory, I used a function-wide iterator instead of local variables as the tracker. Another somewhat significant difference is that I add a conditional statement to the 'while' loop within which DFS iterates. Instead of simply checking for the existence of queued values, my DFS also checks for the existence of non-visited nodes; this I chose to include because it reduced the number of transitional traversals that the algorithm went through, as in some cases, there would be left-over visited nodes in the queue and the DFS would go through them all (without double-marking them) at the end of the search.

Inputs

- ‘theMap’: a Map object (or a tuple containing both ‘theMap’ and ‘theTrees’) containing the nodes to be visited and considered for planting the Tree objects.
- ‘theTrees’: a Python dictionary containing key-value pair where the keys are the names given to the Tree object values.
- ‘s’: an optional source node for the search; must be given by key, not index.
- ‘animate’: an indicator for the visual output style, either 0 for no output, 1 for a static display of the traversed map, or 2 for an animated traversal.
- ‘max_depth’: an integer value determining an optional desired limit to the depth of the search, relative to the source.

Outputs

A single Python dictionary object, with key-value pairs:

- {‘table’: t}: a text-based table formatted to display the planted trees.
- {‘path’: visited}: a dictionary with node keys and values representing the order in which they were visited.
- {‘route’: covered}: a dictionary with edge keys and values representing whether they where covered.
- {‘trees’: planted_trees}: a dictionary containing tree name keys and values with the node where they were planted and their happiness score for that node.

The Algorithm

Implementation

Link to Code

See Appendix for the full Python Implementation.

Complexity Analysis

I. Comparison and Conclusions

Both implementations are surprisingly similar at first glance, which can seem counterintuitive, considering BFS and DFS are an all-time classic example of contrasting algorithmic strategies. However, the similarities make sense in the context of their running time and computational cost. Both implementations' performance is directly dependent on the number of nodes in the graph, and the number of vertices connecting the available nodes. The resulting associated behavior can be seen through both the running time analysis I performed and the estimation of their computational complexity. Both curves stayed considerably close together in the running time plot, as the input size grows, and both functions generally qualify within an estimated complexity between quadratic $O(n^2)$ and linearithmic $O(n * \log n)$, consistently never approaching neither exponential nor linear behavior. In synthesis, although my BFS and DFS implementations performed generally in similar ranges, BFS was consistently better within the range that different tests provided. BFS was often either quadratic or cubic, while DFS was generally cubic, logarithmic, and sometimes linearithmic.

II. Consideration of Additional Aspects

The importing of libraries, generation of data, and visualization aspects affect both of my algorithmic implementations in the exact same manner, which makes their contribution to the

overall computational complexity analysis irrelevant to the comparative analysis of the two solutions' performance at solving the problem. It is important to note, however, that the cost of running these pieces of code would have an impact on the utility of the solution for the users. For example, if the cost of generating the model and presenting its results is of exponential order, even a constant-time algorithmic strategy implementation would result in a highly ineffective and inefficient solution to the problem. I did run empirical estimations for the asymptotic behavior of the most important additional functions and found that the most inefficient implementation in my entire program, with an estimated $O(k^n)$, which showed up in tests as closely matching an exponential scaling form.

List LOs and HCs Applied

- **#emotionaliq:** Working on this project, I honestly had to make very consciously remain aware of my emotions to self-regulate in an effective way, as an overwhelming amount of stressors combined, resulting in me effectively choosing to prioritize my mental state over timely submission of this assignment. This is relevant not to the content of the assignment, but to the way I worked on it.
- **#purpose:** Directly addressing the purpose behind designing this computational solution and taking into account the principles underlying the intended user's motivation in order to provide a solution that is not only effective at addressing the problem, but also would be useful to the foundation.
- **#dataviz:** Produced data visualizations based on a creative and intuitive, animation-based approach, intended to reveal significant and otherwise hidden insights to the foundation, particularly regarding the processes guiding each of the algorithmic search implementations.
- **#algorithms:** Identifying the appropriate algorithmic strategies for a search-related scenario and conducting the necessary analyses to justify the choice between the proposed solutions. Implementing the two versions of major search algorithms, alongside a suitable explanation of the code produced.
- **#modeling:** Innovating on the process used to generate and interpret a model, which is meant to describe a system, identify patterns and make directly useful, justified, predictions for the choice that the ArborDay Foundation faces. Accurately determining the relevant factors in the given context and thus justifying the factors chosen to model

with when generating the graphs and the tree species in the virtual implementation. Effectively developing a programmed model appropriate to the context that provides insights on the scenario, by effectively interpreting the modeled results.

- **#optimization:** Accurately implementing and interpreting two comparable optimization techniques with explicit justifications, including the feasibility, efficiency, and effectiveness of each approach. Interpreting the solutions that result from applying the two search algorithms for the same optimization task, accounting for the solution's acceptability in the context of #TeamTrees, which characterizes and addresses a complex, sophisticated optimization problem related to a real issue.
- **#PythonProgramming:** Producing a Python program that correctly, accurately and efficiently implements the required functionality for the proposed task, which requires deep knowledge of Python in order to properly implement each search algorithm and integrate the underlying data structures of different objects into a cohesive, readable program with definite instructions on how to navigate an operate the implementation. Iteratively running tests and using advanced analyses to produce well-documented performance results for operational algorithmic implementations.
- **#ComputationalSolutions:** Applying a creative and effective algorithmic solution to achieve an improvement over a standard approach to the problem of deciding where the ArborDay Foundation should plant different tree species. In addition, ensuring the algorithmic solution is efficient and implements purposeful and suitable data structures and algorithmic strategies, contrasting clearly and thoroughly between the proposed alternatives.

- **#ComplexityAnalysis:** Performing the appropriate complexity calculations and reporting the relevant resulting metrics for the context of the search algorithms implemented. Engaging in an accurate and detailed analysis of the asymptotic behavior of the algorithmic strategies developed, as it is necessary for the real-world scenario in which the problem emerges to be able to efficiently work with large inputs.

References

ArborDay Foundation. (n.d.). Now is the time.The time for trees. Retrieved December 20, 2019, from https://www.arborday.org/

Burns, R. M., & Honkala, B. L. (1965). Silvics of North America. United States Department of Agriculture. Retrieved December 12, 2019, from https://www.srs.fs.usda.gov/pubs/misc/ag_654/table_of_contents.htm

Help Us Plant 20 Million Trees - Join #TeamTrees. (n.d.). Retrieved December 20, 2019, from https://teamtrees.org/

Pberkes. (2019, December 4). pberkes/big_O. Retrieved December 20, 2019, from https://github.com/pberkes/big_O

SmarterEveryDay. (2019, October 25). How to Plant 20 MILLION TREES - Smarter Every Day 227. Retrieved December 14, 2019, from https://www.youtube.com/watch?v=7bsuXsxX950

Wikipedia. (2019, December 25). Team Trees. Retrieved December 20, 2019, from https://en.wikipedia.org/wiki/Team_Trees

Appendix - Full Code

Link to Code: http://tiny.cc/DeepRootsOrBroadRoots