School of Information Technology, Deakin University

SIT320 — Advanced Algorithms

Module Eight — Greedy Algorithms

GitHub Link

ChatGPT Link

Summary

- **The Tough Choice** - Simplicity vs Optimality
  - Dynamic programming algorithms are often the best choice for **optimisation problems**, and more often(than greddy) find the optimal solution. The only caveat is that they tend to be over kill. The greddy approach will often make the **best choice in the moment**, but it may turn out not be the best choice in the long run. We often use the local optimal choice and **hope** that it makes the best choice. So it should be made clear, greddy algorithms **do not** always yield the correct solution. They are often used to find an **approximate** solution to a problem. Dynamic programming is **more complicated**, and often will have worse time complexity, unless more advanced techniques are used to reconstruct the solution. The trade off is that dynamic programming will **always** find the **optimal solution**.

Activity Selection Problem

- A classic optimization and decision making porblem. We are given a set of task with a start and end time, and we want to find the **maximum number of tasks** that can be completed without any overlap. If we have two activites A and B,we say an activity **overlaps** when activity B starts before activity A ends. Assuming all acticities are **sorted** by end time, we can use a greedy algorithm(or better yet, dynamic programming) to find the maximum number of activities that can be completed.
- **Algorithm** - This is a six step algorithm
  1. Sort the activities by end time
  2. Create a DP table of n dimension where n is the number of activities
  3. Initialise the DP table with the base case
  4. Iterate through the DP table and calculate the optimal solution for each subproblem
  5. Reconstruction the optimal solution
  6. Return the result - that is, the last element in the DP table
- **Complexity**

- **Greedy Algorithm**
    - The complexity of the greedy algorithm O(n log n) due to sorting the greedy selection is O(n)
    - Space complexity is O(n) due to the need to store the sorted activities
- **Dynamic Programming**
    - The complexity of the dynamic programming algorithm is O(n^2) as we need to iterate through the DP table. This is a lazy approach to the problem, and it is possible to solve the problem in O(n log n) time if we use a binary search to find the optimal solution from the DP table.
    - Space complexity of the DP table is O(n) n being the number of activities

Optimized Prim's Algorithm

- At the heart of Prim's algorithm is the Minimum Spanning Tree(MST) and operation much like **Djikstra's**. It is a blazingly fast implementation using the heap data structure. MST are all about connecting a bunch objects as cheaply as possible. Real world examples are connected computer servers, or a network of roads. The connections corespond to similar pairs of objects. MST's are a special case of trees that comprose of connected undirected nodes and edges. The edges are weighted, and the weight of the MST is the sum of the weights of the edges. The MST is the tree with the lowest weight. **NOTE** If the graph is not connected, or has cycles then there is no MST. And the algorithm may end up looping forever. Once again we can use a greedy algorithm or DP to find the MST.
- **Algorithm** - The algorithm is a greedy algorithm, and it is a six step algorithm
    1. Create an array to store the MST and add the first node to the MST
    2. Create a heap and add all the edges from the first node to the heap
    3. While the heap is not empty
        1. Pop the edge with the lowest weight from the heap
        2. If the edge connects two nodes that are not in the MST, add the edge to the MST
        3. Add all the edges from the node that was just added to the MST to the heap
    4. Return the MST
- **Complexity**
    - When constructed using a heap the running time of Prim's is O(E log V) where E is the number of edges and V is the number of vertices. The space complexity is O(V) as we need to store the MST and the heap.

Aha Moments

- **Union is appending**
  - I was confused about the **union operation** in the pseudocode for Kruskal's algorithm. I thought it was a union of sets, but it's actually just **appending the two sets**. I'd never seen the union operation used in this way before.
- **The weighted sum**
  - The weighted sum was a confusing concept to me. Intuatiedly I felt as though tasks could be completed in **any order**. Once i realised that the weighted sum is the sum of the weights of the tasks that are completed, I understood that this meant that the **order of the tasks matters**. And the order of the tasks is determined by the greedy/DP algorithm.
- **Coding Time Vs Research** -The time complexity of actually writing code is O(log n) - where is n is the time speant reading and thikning =)

Conclusion

- Greedy algorithms are a great way to find an approximate solution to an optimisation problem. They are often used when a DP solution is too slow or the doesn't 100% need an optimal solution. This moudle was a great introduction to greedy algorithms and I'm impressed by the power of DP. Again it was interesting that such small and simple algorithms can be so powerful. I'm now starting to understand what people say when they say that **algorithms are beautiful**.

Reading and Resources

- **Algortihms Iluminated- Part 3**: Tim Roughgarden
  - Chapter 13: Introduction to Greedy Algorithms
  - Chapter 15: Minimum Spanning Trees
  - Chapter 16: Introduction to Dynamic Programming
- **Introduction to Algorithms**: Cormen, Leiserson, Rivest, Stein
  - Part IV: Advanced Design and Analysis Techniques
    - Chapter 15: Dynamic Programming
    - Chapter 16: Greedy Algorithms
  - Part VI: Graph Algorithms
    - Chapter 23: Minimum Spanning Trees