SIG Algorithm Challenges

Week 5: Greedy Algorithms

Dane’s Contact Email: dziema2@uic.edu

Join our Slack channel, #sig_algorithm_chal on the UIC ACM slack (uicacm.slack.com)

Mock Technical Interviews Available!

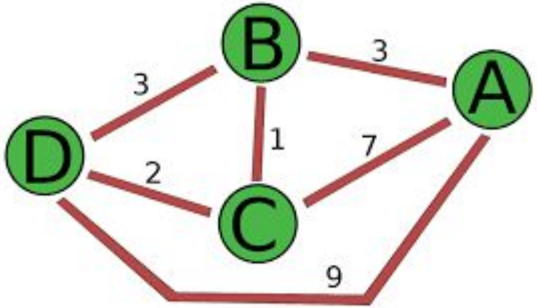
Sign up here: http://bit.ly/SIG_AC_TECH

Greedy Algorithm Example: Djikstra’s

Before we dive too deep into greedy algorithms, it is important to have a prototypical algorithm to look at.

Dijkstra’s Algorithm (for finding shortest path)

- Start off with your root node
- Set the root node distance equal to 0, initialize all others to infinity
- Maintain a list of all nodes you have visited (start with just the root)
- Look at all the edges (u,v) going out of the nodes the set of all nodes you have visited in order by cost from smallest to biggest
- If the cost of edge (u,v) + the distance of the node u is less than the current distance to v, update the distance to v.
- Once finished with the node u, move on to the next node with the lowest distance.



What Makes An Algorithm Greedy?

An algorithm is considered a greedy algorithm if it does the following.

- Makes decisions based off some deterministic ordering or scoring system
  - Djikstra’s: Decides which nodes/edges to explore based off min distance

- This scoring system only considers local conditions (e.g. it doesn’t look into the future and evaluate the decision
  - Djikstra’s: Only looks at distances of nodes based off the fringe.

- Once a decision is made, it is never corrected
  - Djikstra’s: Once a node is expanded its cost never gets updated again (it can be updated before being expanded, but never after)

Are Greedy Algorithms Useful?

- Pros: Greedy algorithms are much faster and require less memory than other approaches for similar problem (such as Dynamic Programming or backtracking)

- Cons: Not all problems have Greedy Algorithm solutions
  - Even wore: sometimes there are problems that look similar, with one having a greedy solution but the other not having one.

- Example: Djikstra’s algorithm is optimal for graphs with no negative edges, but is not if the graph does have negative edges. There is also no greedy algorithm for longest path

When to Use Greedy Algorithms

- In general, it isn't easy to identify when a given problem has a greedy solution

- Greedy algorithms are generally intuitive, and quick to think of
  - Generally when given a decision problem with no prior knowledge, people will devise a greedy algorithm as a first attempt

- Solution: Just dive in! When confronted with a problem that you think may be a candidate for greedy algorithms.

- Once you think of an algorithm, try to think of counterexamples to test its robustness

General Tips for Using Greedy Algorithms

- Brainstorm first
  - Take a couple minutes to think up a few different greedy approaches
  - Try to think of counter examples
  - If you can’t think of any counter examples, try to see if you can reason that this solution is optimal
  - Note: all 3 of these steps take practice, and will not be easy if you haven’t practiced
  - Generally a Queue, PriorityQueue, or some sorted data structure are good ideas.

- Think about what criteria to be greedy on
  - Often times, there are different criterias you can use
  - Example: Interval scheduling: First interval? Shortest interval first? Interval ending first?

- Interview Hack: if problem is super hard (e.g. has an exponentially sized solution space), consider greedy algorithms
  - Interviews are generally ~30 minutes, if a problem looks like it would be exponential even with dynamic programming, that probably would take more than 30 minutes to solve, it probably has a greedy solution

Greedy Algorithm Warmup

Standard US dollar bills come in denominations of \$1, \$2, \$5, \$10, \$20, \$50, \$100

Think of a greedy solution to make change for any amount of money while using the fewest number of bills possible

Follow up: The US treasury introduces a \$3 bill (with Bob Ross's face on it), does your greedy approach still work?

Follow up 2: What if instead they introduced \$25 bill (with Kanye West's face on it), does your greedy approach still work?

Sample Greedy Problem

At a lemonade stand, each lemonade costs `$5`.

Each customer will only buy one lemonade and pay with either a `$5`, `$10`, or `$20` bill. You must provide the correct change to each customer, so that the net transaction is that the customer pays $5.

Return `true` if and only if you can provide every customer with correct change.

```
Input: [5,5,5,10,20]

Output: true
Explanation:
From the first 3 customers, we collect three $5 bills in order.
From the fourth customer, we collect a $10 bill and give back a $5.
From the fifth customer, we give a $10 bill and a $5 bill.
Since all customers got correct change, we output true.
```

Problem Approach

Give people the largest possible bill you can

1: If someone gives you a 5, just take it. If they give you a 10, give them a 5 back or return false if have no 5s.

2: If they give you a 20, give them a 10 and a 5 if possible, otherwise give 3 5s, or return false if less than 3 5s

Pseudocode for this problem

```
MakeChange(Customers):
    fiveCount = 0
    tenCount = 0
    For every c in customer
        If c = 5 fiveCount++
        If c = 10
            If fiveCount = 0 return false
            Else fiveCount--, tenCount++
        If c = 20
            If fiveCount = 0 or fiveCount < 3 and tenCount = 0 return false
            If tenCount = 0 fiveCount-=3 else fiveCount--, tenCount--
```

Java Code for this problem (Part 1)

```
class Solution {
    public boolean lemonadeChange(int[] bills) {
        int fiveCount = 0;
        int tenCount = 0;
        for (int i=0; i<bills.length; i++) {
            //if they gave us a 5 increase our count
            if (bills[i] == 5) {
                fiveCount++;
            }
            //if they give us a ten we have to give a 5
            else if (bills[i] == 10) {
                fiveCount--;
                if (fiveCount < 0) {
                    return false;
                }
                tenCount++;
            }
            //if they give us a 20
            else {
                if (fiveCount == 0) {
                    return false;
                }
```

Java Code for this problem (part 2)

```java
            }
            //if they give us a 20
            else {
                if (fiveCount == 0) {
                    return false;
                }
                //only give 3 fives if you have no tens
                else if (tenCount == 0) {
                    if (fiveCount < 3) {
                        return false;
                    }
                    fiveCount-=3;
                }
                else {
                    fiveCount--;
                    tenCount--;
                }
            }
        }
        return true;
    }
}
```

Greedy Problems to Work On

Links to level 1-3 can be found on my Github account (Dane8373)

Direct Link: https://github.com/dane8373/SIG_Algorithm_Challenges/tree/master/Week5

Level 1) You have a bunch of cookies you wish to share with children. Each cookie has a size, and each child has a minimum size they will accept. Given an array of cookie sizes and children's preferences, return the maximum number of children who you could give a cookie

Level 2) Given a collection of intervals, find the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

Level 3) N couples sit in 2N seats arranged in a row and want to hold hands. We want to know the minimum number of swaps so that every couple is sitting side by side. A *swap* consists of choosing any two people, then they stand up and switch seats.

Problem Approach for Level 1 problem

Sort the children and cookies from smallest to largest

1: Give the least greedy child the smallest cookie possible to make him happy

2: Continue for as long as you have cookies/children left

Java Code for level 1 problem

```
class Solution {
    public int findContentChildren(int[] g, int[] s) {
        int count = 0;
        int cookie = 0;
        int child = 0;
        //sort the arrays
        Arrays.sort(g);
        Arrays.sort(s);
        //while we still have children or cookies left
        while (cookie < s.length && child < g.length) {
            //keep advancing until we have a cookie that the
            //least greedy child will accept
            while (cookie < s.length && s[cookie] < g[child]) {
                cookie++;
            }
            //check to see if we got out of the loop because
            //of a match or because we ran out of cookies
            if (cookie < s.length) {
                child++;
                cookie++;
            }
        }
        return child;
    }
}
```

Problem Approach for Level 2 problem

The main problem of this one is to decide what to be greedy upon. In this case, use ending time

1: Sort the intervals from earliest ending time to latest

2: For each interval end, check to see if any intervals start before the interval ends, if they do then delete them

3: Continue until there are no overlapping intervals

Java Code for level 2 problem

```java
class Solution {
    public int eraseOverlapIntervals(Interval[] intervals) {
        if (intervals.length == 0 || intervals.length == 1) {
            return 0;
        }
        //sort the arrays by end time
        Arrays.sort(intervals, Comparator.comparing((Interval i) -> i.end));
        int currEnd = intervals[0].end;
        int count = 0;
        int i = 1;
        while (i<intervals.length) {
            //keep removing any intervals that start before our current one ends
            while (i<intervals.length && intervals[i].start < currEnd) {
                count++;
                i++;
            }
            //move on to the next one if there are any left
            if (i<intervals.length) {
                currEnd=intervals[i].end;
            }
            i++;
        }
        return count;
    }
}
```

Problem Approach for Level 3 problem

Turns out to be simpler than it seems, you can just process the people from left to right

1: Starting from the left, if this person is not seated next to their partner, move their partner next to them

2: Continue until all partners are together

Java Code for level 3 problem (part 1)

```java
class Solution {
    public int minSwapsCouples(int[] row) {
        //maps used to describe the pairs and locations of all numbers
        HashMap<Integer,Integer> locations = new HashMap<Integer, Integer>();
        HashMap<Integer,Integer> pairs = new HashMap<Integer, Integer>();
        locations.put(row.length,-1);
        //set up the pair/location mpas
        for (int i=0; i<row.length; i++) {
            locations.put(row[i],i);
        }
        for (int i=0; i<row.length; i+=2) {
            pairs.put(i, i+1);
            pairs.put(i+1, i);
        }
```

Java Code for level 3 problem (part 2)

```java
        int count = 0;
        //process people from left to right
        for (int i=0; i<row.length; i+=2) {
            //if they are already seated together do nothing
            if (i-locations.get(pairs.get(row[i])) == -1) {
                continue;
            }
            //otherwise swap the person to the right with the pair
            else {
                int myRight = row[i+1];
                row[i+1] = pairs.get(row[i]);
                row[locations.get(pairs.get(row[i]))] = myRight;
                int temp = locations.get(myRight);
                locations.put(myRight, locations.get(pairs.get(row[i])));
                locations.put(pairs.get(row[i]), temp);
                count++;
            }
        }
        return count;
    }
}
```

Next Week: Dynamic Programming