CSCE 5150 – Analysis of Computer Algorithms

Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

Q1. Activity-selection problem. Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities. (Ex. 16.1-2)

a) Describe how this approach is a greedy algorithm [3 points]

1.a) Algorithm:

```
greedy-activity-selection (s, f)
  n = s.length
  A = {a_n}
  k = n
  for m = n-1 to 1
      if f[m] ≤ s[k]
          A = A ∪ {a_m}
          k = m
  return A
```

Here, $s = \{a_1, a_2, \ldots a_n\}$ are the set of activities

→ activities are sorted in the order of their finishing time

→ The goal is to find the set of activities that do not overlap with each other, we must find the largest set with such activities starting from the end.

→ The starting time and finishing time are passed to the algorithm

→ The logic scans through the activities (starting from the last activity) and adds all the non overlapping activities to the set, trying to find the optimal solution at each and every stage. Therefore, it is a greedy algorithm, because we select the best solution at each step.

b) Prove that it yields an optimal solution [2 points]

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

1 b) Ideally the activity-selection algorithm finds an optimal solution by selecting the first finishing activity first, i.e the activities are selected in ascending order. Similarly, the new algorithms also selects the activities based on finishing time but in a descending order. As the old algorithm produces an optimal solution, we can say that the new one also produces the optimal solution because the only difference here is the order of selecting the activities, the old one starts from the first finishing activity & the new one starts from the last finishing activity.

Q2. Huffman codes. What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers? [4 points]

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers? [3 points] (Ex. 16.3-3)

CSCE 5150 – Analysis of Computer Algorithms

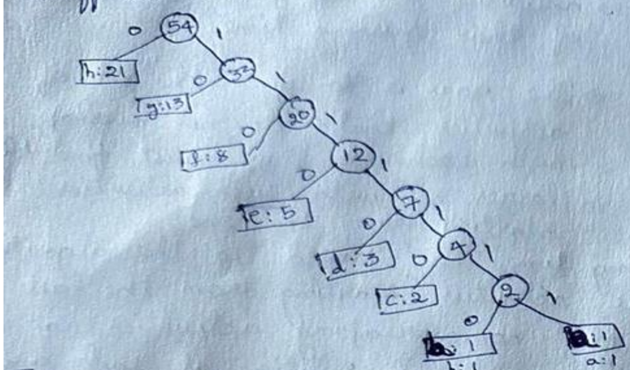
Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

2) Huffman codes:

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

Huffman tree would be as shown below: 1 1 2 3 5 8 13 21



a:1 = 1 1 1 1 1 1 1

b:1 = 1 1 1 1 1 1 0

c:2 = 1 1 1 1 1 0

d:3 = 1 1 1 1 0

e:5 = 1 1 1 0

f:8 = 1 1 0

g:13 = 1 0

h:21 = 0

For 'n' numbers in the form of fibonacci series, we can say that for the first number it should be $(n-1)$ 1's and for rest of the numbers it should be $(n-i)$ 1's and 0 $(n >= i >= 2)$ where i is the position of the number in the series

Eg: → f:8 is in the position 6. So the huffman code will be $(n-i)$ 1's and 0 $= (8-6)$ 1's and 0 $=$ 110

→ d: 3 is in 4th position, so $(n-i)$ 1's + 0 $= (8-4)$ 1's + 0 = 11110

a:1, for 1st number it is $(n-1)$ 1's $= (8-1)$ 1's = 1111111

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

Q3. Coin-changing problem. Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer. (Problem 16-1)
a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution. [4 points]

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

3a) The algorithm uses the same technique that we use in real time, i.e by using the greatest value coins first to sum up to the total amount and we use the remainder as the new amount and further repeat the same steps to sum upto the total amount.

As we apply the best possible solution at each step, we can say that this algorithm is greedy.

The greedy algorithm is said to provide optimal solution because providing optimal solution at each step leads to providing optimal solution on the whole.

Proof:

→ Say we only have pennies, nickles, dimes & quaters, if we assume to get change for 30, we do not have to use 3 dimes, instead we use a quarter and a nickle, thereby reducing the number of coins used. Hence the algorithm provides optimal solution.

→ Say we have pennies, nickles & dimes, if we have to get change for 15, we do not have to use 3 coins of nickles but we use a dime & a nickle, we will be using only 2 coins instead of 3 nickles. Hence providing optimal solution.

→ Say, we have only pennies & nickles, and we try to change ¢6, we do not have to use 6 pennies, instead we can use 1 nickle & 1 penny.

Therefore, the greedy algorithm provides optimal solution for the above set of coin denominators.

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

b) Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n. [2 points]

3b) The considered algorithm might not provide an optimal solution for few cases.

Ex1: Let the coin denominators be pennies, dimes & quaters
$\{1, 10, 25\}$ and the amount $n = 30$ cents
Using the algorithm the solution $= \{25, 1, 1, 1, 1, 1\}$
i.e one quarter & 5 pennies = 6 coins
But the optimal solution wold be 3 dimes = 3 coins.

Ex 2: Let the coin denominators $= \{1, 3, 4\}$ we have the amount $n = 6$, we get the solution as $\{4, 1, 1\}$ but the optimal solution would be $\{3, 3\}$.

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

c) Write the pseudocode of the greedy algorithm for the coin-changing problem, with an amount n and coin denominations d1 > d2 > d3 > . . . > dm as its input. (Hint. You may use integer divisions in your algorithm) [3 points]

3c) Pseudocode of greedy algorithm for coin-changing problem

```
Change-Coin (n, D[1;..m])        # n is the amount & D is the
                                   coin denominators, in
                                   descending order.
  for i in range (0, m):
      C[i] = int(n / D[i])       # divide 'n' with highest value
                                   coin denominator
      n = n % D[i]               # remainder will be the
                                   new amount
      if (n==0): return C        # C is the solution set
      else: return "no solution".
```

Time complexity = $\Theta(m)$ if we stop at n==0, the time efficiency would be $O(m)$.

```python
def coin_changing(m,n,deno):
    input_ = n
    output = []
    out = []
    for i in range(0, m):
        output.append(int(n/deno[i]))
        for j in range(0,output[i]):
            out.append(deno[i])
        n= n%deno[i]
        if n==0:
            print("The coin denominations needed for",input_, "are :",out)
            break
```

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

Q4. Rumor-spreading problem. There are n people, each in possession of a different rumor. They want to share all the rumors with each other by sending electronic messages. Assume that a sender includes all the rumors he or she knows at the time the message is sent and that a message may only have one addressee. Design a greedy algorithm that always yields the minimum number of messages they need to send to guarantee that every one of them gets all the rumors. (Hint. The minimum number of messages for n = 4 is six) [4 points]

CSCE 5150 – Analysis of Computer Algorithms

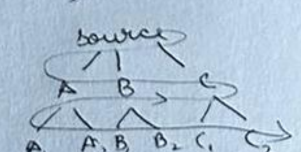
Homework No. 5 – Greedy Algorithms

Neha Goud Baddam
11519516

7.) Rumour spreading Algorithm

We shall be using the BFS (Breadth first search) traversal to generate the algorithm. We consider N person as N nodes of the graph, each node will carry different rumours. The algorithm should make sure that the rumours should visit every node.

```
for node = 1 to n        # In the BFS nodes the message
                           is shared
  BFS(node n , message)
  {
    visited[n]           # we mark the visited array as
                           fase, when we actually visit
    queue q;               the node in future we will mark
                           it as true
    q. enqueue (node);   # here node is the source of
                           rumour.
    while (q is not empty);
      {
        s = q. front); # we share the message of node
                         with s
        q. dequeue ();
        for all adjacent nodes of S
          {
            q. enqueue (adjacent nodes);
          }
      }
  }
```



Complexity $= N*(V+E)$ $(V=N)$

$= O(N*(N+E))$ $(E=O(N))$

$= O(N*(N+O(N))) = O(N^2)$

E : no. of edges / association between people

V : N nodes.