HW06 for ECE 9343

Tongda XU, N18100977

December 5, 2018

1 Question 1: Huffman Code Running

One possible result:
{ h: 00, a: 11, b: 100, f:101, d: 110, e: 1111, c: 11100, g: 11101 }

2 Question 2: CLRS Problem Set 16.1

2.1 Describe a greedy algorithm

CC(k)
1 $Change \leftarrow \{25, 10, 5, 1\}$
2 $Result \leftarrow []$
3 **for** each c in $Change$
4 $\quad Result[c] = \text{FLOOR}(k/c)$
5 $\quad k = k \mod c$
6 **return** $Result$

Suppose that we are working with coin set {30, 10, 5, 1}, it is fairly easy to find out that in any step of loop 3-5, if we did not select the greedy choice, which is let the current largest coin fill result first, the reminder would be larger than the current largest coin. Since the largest coin could be divided by the second largest one, the reminder needs at lease $\frac{largest}{secondlargest}$ coin to fill the reminder, which could be merged into at least 1 largest coin to reduce number of coins.

For the case of {25, 10, 5, 1} it becomes tricky. the larger than 30 case could be easily cut into 25 and 5, which is obviously better than 3*10. But for {26, 27, 28, 29}, we have to enumerate this four case to proof, such that 26 = 25 + 1 = 2*10 + 5 + 1, 27 = 25 + 1 + 1 = 2*10 + 5 + 1 + 1, 28 = 25 + 3*1 = 2*10 + 5 + 3*1, 29 = 25 + 4*1 = 2*10 + 5 + 4*1 to complete the proof.

2.2 Proof greedy works for power sequence

Suppose that in any step of loop 3-5, the reminder r is larger than divider c_i, which is not the greedy choice, then it is easy to find out that r requires at least c times c_{i-1}, or c^2 times c_{i-2}, which could be merged into 1 c_i to reduce the number of coins.

2.3 Describe a set of coins greedy does not apply

Consider ={4, 3, 1} dividing 6

2.4 Describe a universal (DP) solution

CC(n)

$DP \leftarrow []$
$Change \leftarrow \{k_k, ..., k_1\}$
return CC-AID(n)

CC-AID(k)

if $DP[k] \neq NIL$
  return $DP[k]$
else
  $DP[k] \leftarrow \min_{k_i \in Change, k-k_i \geq 0} \{\text{CC-AID}(n - k_i)\} + 1$
  return $DP[k]$

This algorithm construct a dictionary of size n, and each time rely on the retrieve of previous k element in total of O(n) time, thus the running time would be O(k + nk) = O(nk).

3 Question 3: CLRS Exercise 15.4-3

See Figure 1

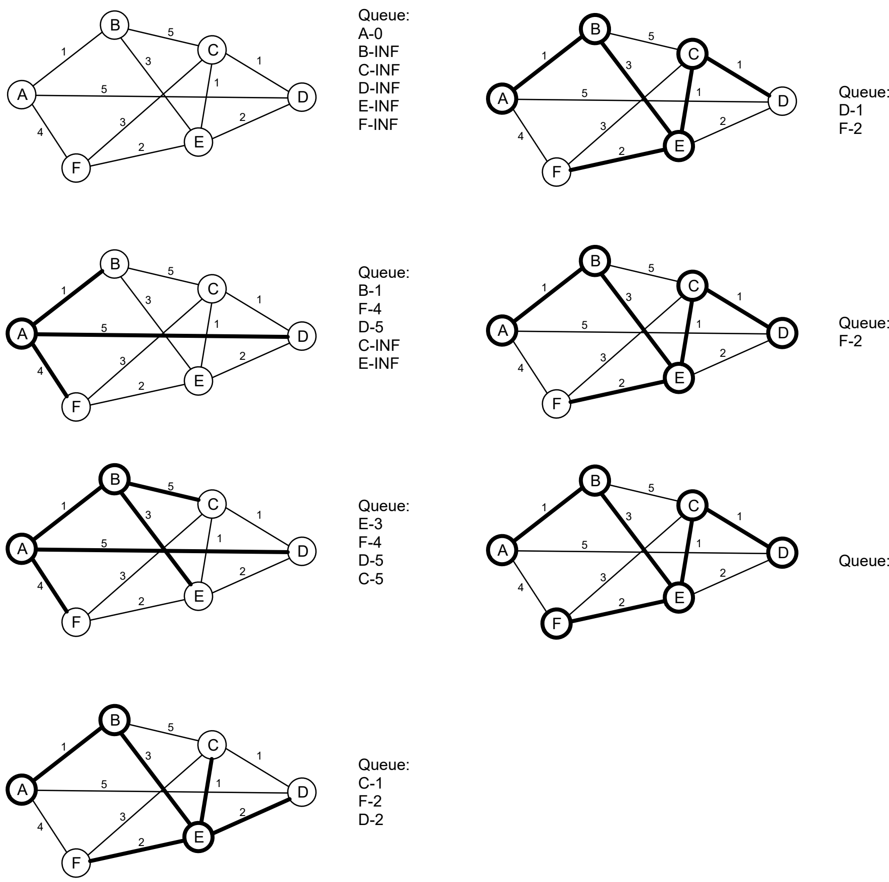

Figure 1: Prism algorithm running procedure