CSCE 629 - 602 Analysis of Algorithms

September 19, 2016

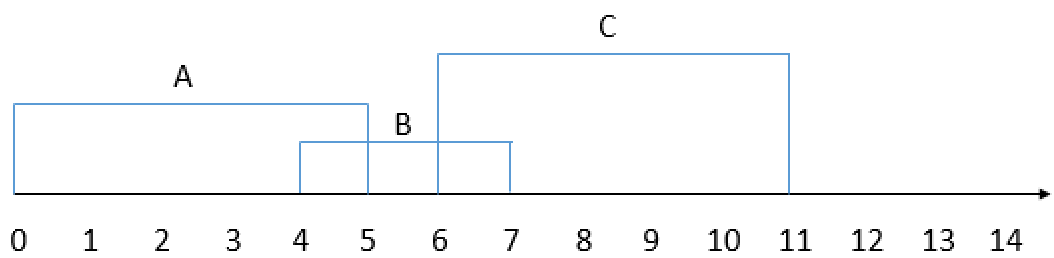
Homework II **Rishabh Yadav**
Prof. Anxiao (Andrew) Jiang UIN: 425009824

1A. 16.1-3 Solution

Part 1

Let us take activities with start time (s_i) and finish time (f_i) as per the following table.

$Activity$	A	B	C
$Start(s_i)$	0	4	6
$Finish(f_i)$	5	7	11

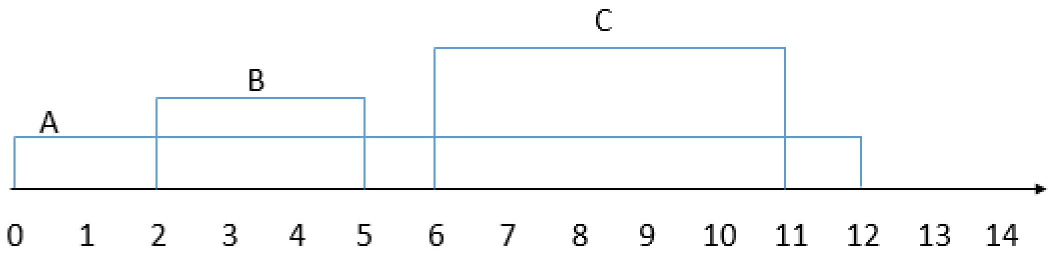


For greedy approach by selecting activity with least duration we will choose activity B with duration 3 units. After making this choice we cannot choose any other activity as both activities A and C overlap with this choice. hence our solution in 1.
This is sub-optimal because when we make choice by by taking first finish time we choose activity A and then activity C which gives us 2 activities in solution which is more than solution from previous approach of picking activities with least duration.

Part 2

Let us take activities with start time (s_i) and finish time (f_i) as per the following table.

Activity	A	B	C
$Start(s_i)$	0	2	6
$Finish(f_i)$	12	5	11

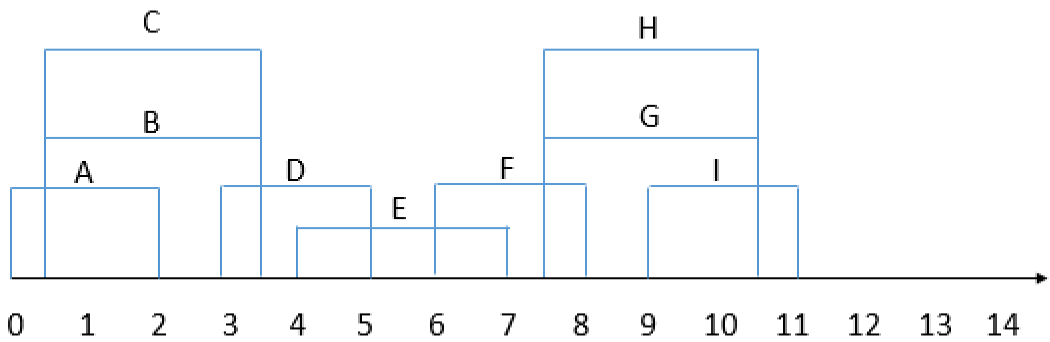


For greedy approach by selecting activity with earliest start time we will choose activity A with start time 0. After making this choice we cannot choose any other activity as both activities B and C overlap with this choice. hence our solution in 1.
This is sub-optimal because when we make choice by by taking first finish time we choose activity B and then activity C which gives us 2 activities in solution which is more than solution from previous approach of picking activities with earliest start time.

Part 3

Let us take activities with start time (s_i) and finish time (f_i) as per the following table.

$Activity$	A	B	C	D	E	F	G	H	I
$Start(s_i)$	0	1.5	1.5	3	4	6	7.5	7.5	9
$Finish(f_i)$	2	3.5	3.5	5	7	8	10.5	10.5	11



For greedy approach by selecting activity with least overlap we will choose activity E with overlap with 2 other activities (D and E). After making this choice we choose activity A with overlap 2 (B and C). Now we can make one more choice of activity I with overlap 2 (H and G). Hence our solution in 2 activities.
This is sub-optimal because when we make choice by by taking first finish time we choose activity A and then activity D, then activity F and finally activity I. This gives us 4 activities in solution which is more than solution from previous approach of picking activities with least overlap.

2A. Solution 16.3-3

Part 1

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

To assign prefix free code to the above symbols we follow the following steps:

Step 1: Pick symbols with least frequency, which is a and b combine them and add there frequency. the table after this operation is:

Symbols	ab	c	d	e	f	g	h
Frequency	2	2	3	5	8	13	21

Step 2: Continuing with the same approach till we get one symbols. Picking ab and c combine them and add their frequency. the table after this operation is:

Symbols	abc	d	e	f	g	h
Frequency	4	3	5	8	13	21

Step 3: Continuing with the same approach till we get one symbols. Picking abc and d combine them and add their frequency. the table after this operation is:

Symbols	abcd	e	f	g	h
Frequency	7	5	8	13	21

Step 4: Continuing with the same approach till we get one symbols. Picking $abcd$ and e combine them and add their frequency. the table after this operation is:

Symbols	abcde	f	g	h
Frequency	12	8	13	21

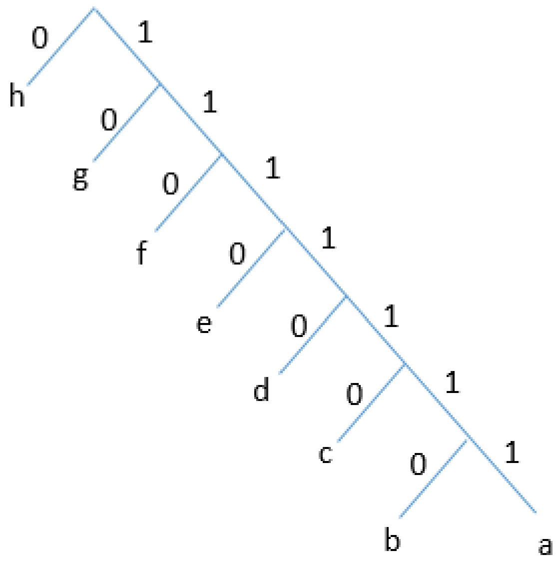
Step 5: Continuing with the same approach till we get one symbols. Picking $abcde$ and f combine them and add their frequency. the table after this operation is:

Symbols	abcdef	g	h
Frequency	20	13	21

Step 6: Continuing with the same approach till we get one symbols. Picking $abcdef$ and g combine them and add their frequency. the table after this operation is:

Symbols	abcdefg	h
Frequency	33	21

Step 7: Since this is a trivial case where we need to assign code to two symbols, h and $abcdef$. On backtracking our way we get the following binary tree structure.



*

The optimal Huffman code for the given symbols and corresponding Fibonacci frequency are:
h: 0
g: 10
f: 110
e: 1110
d: 11110
c: 111110
b: 1111110
a: 1111111

Part 2
We can see that after merging two lower frequency symbols the resultant frequency is linearly increasing such that they in picked in the same order as their occurrence in Fibonacci series.

General Rule

for n symbols series:
1. for lowest frequency symbol
$(n-1)$ times 1
2. for other symbols
$(n-i-1)$ times 1 followed by a 0, where $0 < i < n$, is the index of occurrence in the series.

3A. Solution 16-1

Part a

Pennies	1
Nickel	5
Dime	10
Quarter	25

Idea

The Greedy algorithm proposed works such that we take the highest denomination, c_k, possible less than equal to the given amount sum n, and continue the coin change algorithm on the difference $n - c_k$. We continue till we reach 0.

Algorithm

Change(n):

1. denominations = [25, 10, 5, 1] // they are sorted in decreasing order
2. $returnValue = \{\}$
3. while $n > 0$:
   (a) for coin, denominationValue in enumerate(coins):
   (b) if $n > 0$ and denominationValue <= n:
      i. returnValue.add(coin)
      ii. $n-$ =denominationValue
   (c) else return $returnValue$
4. return $returnValue$

Proof of Correctness

We will give a proof by contradiction.
Let S denote the solution set. Let us say that the Greedy choice for any given number n is d_{greedy}. We do not pick d_{greedy} as part of the solution but pick another denomination d_i which as part of supposed optimal solution. We can show that some sequence in S can be replaced by a higher denomination, d_{greedy}. There can be following cases:

1. CASE I:
   $1 \leq n < 5$. Since this a basic case any value of n in this case would have only pennies in the solution set S.
2. CASE II:
   $5 \leq n < 10$. In this case value of $d_{greedy} = 5$, hence our "supposed" optimal solution S should only contain pennies. And since $n \geq 5$ there should be at least 5 pennies

in supposed solution which can be replaced by a single nickel, which is the greedy choice d_{greedy}, and hence giving us a more optimal solution.

3. CASE III:
   $10 \leq n < 25$. In this case value of $d_{greedy} = 10$, hence our "supposed" optimal solution S should only contain nickels and pennies. From Case I and II we can say the solution S, should contain atleast 2 nickels and since $n \geq 10$. These 2 nickels in supposed solution can be replaced by a single dime, which is the greedy choice d_{greedy}, and hence giving us a more optimal solution.

4. CASE IV:
   $25 \leq n$. In this case value of $d_{greedy} = 25$, hence our "supposed" optimal solution S should only contain dimes, nickels and pennies. From Case I, II and III we can say the solution S, should contain atleast 2 dimes and a nickel and since $n \geq 25$. These 2 dimes and a nickel in supposed solution can be replaced by a single quarter, which is the greedy choice d_{greedy}, and hence giving us a more optimal solution.

We observer that always making a greedy choice in this particular type of denominations we get an optimal solution.

Time Complexity

Time Complexity= $O(k)$, where k is the number of denominations in the set.

Part b

Idea

The Greedy algorithm proposed works such that we take the highest denomination, c^k, possible less than equal to the given amount sum n, and continue the coin change algorithm on the difference $n - c^k$. We continue till we reach 0.

Algorithm

Change(n):

1. denominations = $[c^n, c^{n-1}, c^{n-2}, ..., c^0]$// they are sorted in decreasing order
2. $returnValue = \{\}$
3. while $n > 0$:
   (a) for coin, denominationValue in enumerate(coins):
   (b) if $n > 0$ and denominationValue $<=$ n:
      i. returnValue.add(coin)
      ii. $n-$ =denominationValue
   (c) else return $returnValue$
4. return $returnValue$

Proof of Correctness

We will give a proof by contradiction.
Let S denote the solution set. Let us say that the Greedy choice for any given number n is d_{greedy}, such that,

$d_{greedy} = c^k \leq n$

Our supposed optimal solution, S, may contain a denomination c^{k-1} since $c^k \leq n$ there must be at least c instances of c^{k-1} in S because highest power of c in n is k. These c instances can be replaced by a single c^k, which is our greedy choice, d_{greedy}. We can extend similar arguments can be give for lower powers of c in the denomination set.

Time Complexity

Time Complexity= $O(k)$, where k is the number of denominations in the set.

Part c

Example is denominations set = [10,6,1]. And $n = 12$.

By greedy choice we take 10 first followed by 1 and 1. Our solution by greedy approach is
$Solution_{greedy} = \{10, 1, 1\}$: 3

But we can clearly see we can have a better solution by picking 6 and 6.
$Solution = \{6, 6\}$: 2

Part d

Idea

The idea is to use dynamic programming to solve the coin change for any set of k denominations with one of the coins is a penny. The approach is that we will pick coin a coin c_i, $(0 \le i \le k-1)$ from the set of k different denominations and see what is the minimum number of coins needed if only we had coins $< c_0, c_1, c_2, ..., c_i >$. We will store the results in table and work our way in bottom up fashion. The recurrence relation to get the minimum number of coins to change sum n from set of k different coins is:

$$T[n] = min(T[i], 1 + T[i] - coinValue(c_j))$$

where,
$T[i]$, is the value when we DON'T take the j^{th} coin (c_j), and
$1+T[i]-coinValue(c_j)$, is the value when we take the j^{th} coin (c_j). We will fill the table in bottom up manner.

Algorithm

Make Change($n, coinSet[k]$)

1. $Table[] = n * INTMAX$
2. Table[0] = 0 //0 ways to make change for sum 0, Trivial case
3. for $j = 0$ to k //length of coinSet[]
   (a) for $i = 1$ to n //bottom up from $1- > n$
      i. if $coinSet[j] \le i$:
         A. $Table[i] = \min(1 + Table[i] - coinSet[i]), Table[i]$
4. return $Table[n]$

Proof of Correctness

Given a set of coinSet = $\{c_1, c_2, , c_k\}$ of coin denominations, $T(n)$ denote the minimum number of coins in coinSet needed to obtain sum n. Trivial case of $T(0) = 0$

If $n > 0$, to obtain n with $T(n)$ coins uses at least one coin c_j with denomination (at least one such j exists), then removing this coin we obtain a way to obtain nc_k and hence conclude,

$T(nc_j) \leq T(n)1$, for at least one $1 \leq j \leq k$

On the other hand,

if $1 \leq j \leq j$ and nc_j we can find n with $T(n - c_j) + 1$ coins by adding a c_j coin to an optimal way to get nc_j. We conclude,

$T(n) \leq T(n - c_j) + 1$, for all $1 \leq k \leq m$ with $n \leq c_j$.

Combining two equation above we get,

$T(n) = min\{T(n), T(n - c_j) + 1, 1 \leq k \leq m$ and $c_j \leq n\}$, for all $n > 1$
This is what the DP algorithm uses to compute $T(n)$ in bottom up fashion.

Time Complexity

As evident from the two loops in the algorithm presented above the outer loops run from 0 to k, which is the length of the coinSet.And the inner loop runs from the 1-n. Hence,
Time Complexity = $O(nk)$.

References

Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest, C. Stein http://math.stackexchange.com/