09/26/16

ALGORITHMS

Home Work - 2

Kiran C. Shettar
UML ID - 01605800

16.1.1

Recursive algorithm to solve activity selection problem:

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = 0 \\ \max\{c[i,k] + c[k,j] + 1 & \text{if } S_{ij} \neq 0 \end{cases}$$

Let 'f' be the finish time where $f_1 \leq f_2 \leq \ldots \leq f_n$

```
RECURSIVE_DYNAMIC_ACTIVITY_SELECTOR (s, f, k, n)
n = length(s)
for i = 0 to n
    c[i,j] = 0

for m = 2 to n
    for i = 1 to (n - m + 1)
    j = (i + m - 1)
    c[i,j] = ∞
for k = (i + 1] to (j - 1)
  q = c[i,k] + c[k+1, j] + 1
 if q < c(i,j)
      then c(i,j) = q
```

Here, $c(i,j)$ is a 2-D array. Hence it takes $O(n^2)$ to fill the values.

Now we have to compute a_k in S_{ij}. for this we use the algorithm below.

OPTIMAL-ACTIVITY-SELECTION (c, s, f, i, j)

1. Return (a-i) U SUBOPTIMAL-SELECTION (s, f, i, j) U (a-j)

SUBOPTIMAL-SELECTION (c, s, f, i, j)

1. if i<j AND j<a.length
2. k = c[i][j]
3. if i<k and k<j
4. return SUBOPTIMAL-SELECTION(c, s, f, i, j) U (a-k) U SUBOPTIMAL-SELETION (c, s, f, k, j)

16.2.3

Let $i_1, i_2 \dots i_n$ be the items with values $v_1, v_2, v_3 \dots v_n$ and let $w_1, w_2 \dots w_n$ be their weights.

Let 'w' be the maximum knapsack weight.

Now, $w_1 \leq w_2 \leq \dots \leq w_n$ and $v_1 \geq v_2 \geq \dots \geq v_n$

∴ The algorithm that would be efficient for this kind of problem would be :

KNAPSACK - ALGO

1. $W = 0$
2. $S = 0$
3. for $(i = 1;\ i \le n;\ i{+}{+})$
4. if $(W + w_i \le w)$
5. $W = W + w_i$
6. $S = S \cup \{i\}$

Proof for correctness of algorithm:

Greedy-choice: Let 'S' be an optical knapsack load. Let us assume that $i_1 \in S$. Indeed, if $i_1 \notin S$, let k be the smallest index of item of 'S'.

If $S' = S / i_k \cup i_1$. Since $w_1 \le w_k$ & $w(S') \le w(S) \le W$

Hence S' is a legal packing. Also $v_1 \ge v_k$ implies $v(S') \ge v(S)$. Hence S' is also optimal.

15.9

Let us consider that the array 'L' is sorted.

First element $L[0] = 0$

Last element $L[m+1] = n$

Here,

the initial case would be

~~$f(i,i) = 0$ & $g(i,i+1) = 0$~~

$L(i, i+1) = 0 \quad \forall\ i,\ 0 \leq i \leq n$

Recursion:

$L(i,j)$ for $j > i+1$

$$\underset{i<k<j}{L(i,j)} = \text{minimum}$$

Let us consider the cuts at 'c'

where $c_1, c_2 \ldots c_n$.

$L(i,k) + L(k,j) + (c_j - c_i)$

Hence, we can write this as,

$$L(i,j) = \min_{i<k<j} \{ L(i,k) + L(k,j) + (c_j - c_i) \}$$

There are two subproblems here

whose running time will be $O(n^2)$.

To solve each sub problem, it takes

$O(n)$ time.

Hence the total running time will be

$O(n^3)$.

16.3.5

Given that the characters are alphabatically arranged and their frequencies are monotonically decreasing

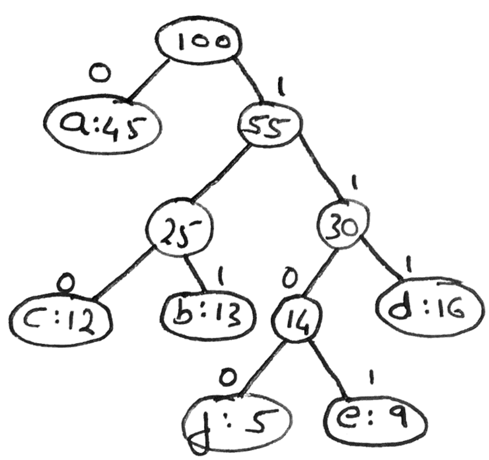
i.e $a.freq > b.freq > c.freq > d.freq > \ldots > z.freq.$

By considering the image shown in the text book figure 16.3

Optimal code for a code word is represented using a full binary tree.

Let us consider alphabets from a-f

	a	b	c	d	e	f
Frequency (in thousand)	45	13	12	16	9	5
Fixed length codeword	000	001	010	011	100	101
Variable length codeword	0	101	100	111	1101	1100



We can see from the above table that, the frequency for alphabets a→f has gone decreasing where as the code word length has monotonically increased.

a:45, b:13, c:12, d:16; e:9, f:5 — <i>

& a=0, b=101, c=100, d=111, e=1101, f=1100 — <ii>

From <i> & <ii> we can prove that if we order the characters in an alphabet so that their frequencies are monotonically decreasing, then there exists an optimal code whose codeword lengths are monotonically increasing