University of Sargodha

BS 4th Term Examination 2017

Subject: Computer Science **Paper: Design & Analysis of Algorithm (CS: 3143)**

****

Q1. Write short answers of the following.

i. **Big O Notation:** A function t(n) is said to be in O(g(n)), denoted t(n) ∈ O(g(n)), if t(n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that: **$t(n) \leq cg(n)$ for all $n \geq n_0$.**

ii. **What is meant by problem of optimality?**

A problem is said to satisfy the Principle of Optimality if the subsolutions of an optimal solution of the problem are themesleves optimal solutions for their subproblems.

iii. **What are different criteria used to improve the effectiveness of algorithm?**

iv. **Write down the recursive solution for knapsack problem?**

```
int knapSack(int W, int wt[], int val[], int n)
{
    if (n == 0 || W == 0)
        return 0;

    if (wt[n-1] > W)
        return knapSack(W, wt, val, n-1);

    else return max(val[n-1] + knapSack(W-wt[n-1], wt,
                    val, n-1),knapSack(W, wt, val, n-1));
}
```

v. **What is the time complexity of prims algorithm?**

Time Complexity of the above program is **O(V^2)**. If the input graph is represented using adjacency list, then the time complexity of Prim's algorithm can be reduced to **O(E log V)** with the help of binary heap.

vi. **Write the difference between in greedy and dynamic programming?**

Greedy	Dynamic Programming
A greedy algorithm is one that at a given point in time, makes a local optimization.	Dynamic programming can be thought of as 'smart' recursion. It often requires one to break down a problem into smaller components that can be cached.
Greedy algorithms have a local choice of the subproblem that will lead to an optimal answer	Dynamic programming would solve all dependent subproblems and then select one that would lead to an optimal solution.
A greedy algorithm is one which finds optimal solution at each and every stage with the hope of finding global optimum at the end.	A Dynamic algorithm is applicable to problems that exhibit Overlapping subproblems and Optimal substructure properties.

Solved by Talha Shahab

vii. Differentiate between P and NP problems?

A problem which can be solved in polynomial time by a computer are known as P problems. A problem which is not solvable in polynomial time but the solution is verifiable in polynomial time is called NP problems.

viii. What is meant by minimum spanning tree?

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.

ix. How problems are solved using divide and conquer approach?

In divide and conquer approach:

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

x. Define all pair shortest path problem.

The all-pairs shortest path problem is the determination of the shortest graph distances between every pair of vertices in a given graph. The problem can be solved using n applications of Dijkstra's algorithm or all at once using the Floyd-Warshall algorithm.

xi. What is chained matrix multiplication?

Matrix chain multiplication (or Matrix Chain Ordering Problem, MCOP) is an optimization problem that can be solved using dynamic programming. Given a sequence of matrices, the goal is to find the most efficient way to multiply these matrices. The problem is not actually to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved.

xii. What is the purpose of Dijkstra's Algorithm?

The purpose of Dijkstra's algorithm is to find the shortest path in a graph. Dijkstra is concerned with only two nodes.

xiii. Write an algorithm using Recursive function to find sum of n numbers?

```
sum(int n)
{
    if (n == 0)
        return 0;
    else
        return n + sum(n - 1);
}
```

xiv. Write down the ingredients of dynamic programming?

There are two key ingredients of dynamic programming:

1. **Optimal substructure**: problem exhibits optimal substructure if an optimal solution to the proble m contains within it optimal solutions to sub problems.
2. **Overlapping Sub problems:**Dynamic Programming is not useful when there are no common (over lapping) sub problems because there is no point storing the solutions if they are not needed again.

xv. What is activity selection problem?

There are n different activities are given with their starting time and ending time. Select the maximum number of activities to solve by a single person or machine. This is called activity selection problem.

xvi. Write recurrence relation of quicksort?

The recurrence relation for quicksort is:

$$T(n) = 2T(\frac{n}{2}) + \mathcal{O}(n)$$

Subjective Part

Q2. Merge sort

It is one of the well-known divide-and-conquer algorithm. This is a simple and very efficient algorithm for sorting a list of numbers.

Divide: Split A down the middle into two sub-sequences, each of size roughly n/2 .
Conquer: Sort each subsequence (by calling MergeSort recursively on each).
Combine: Merge the two sorted sub-sequences into a single sorted list.

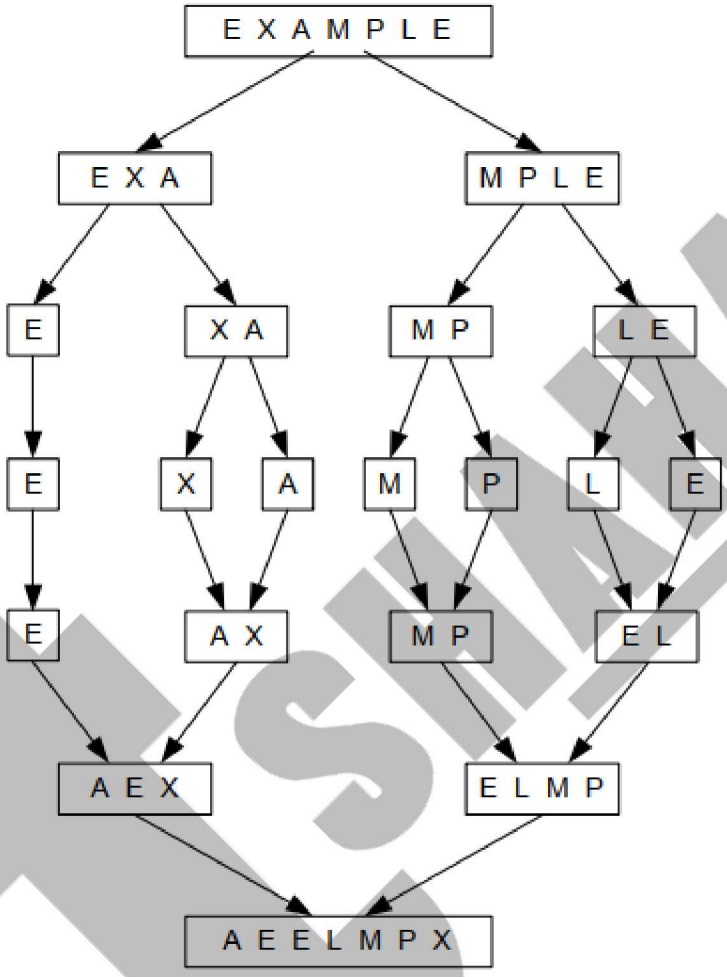
```
MergeSort(A, p, r)
      If p > r
            return;
      q = (p+r)/2;
      mergeSort(A, p, q)
      mergeSort(A, q+1, r)
      merge(A, p, q, r)
```

```
Merge(array A, int p, int q, int r) {          // merges A[p..q] with A[q+1..r]
      array B[p..r]
      i = k = p                                // initialize pointers
      j = q+1
      while (i <= q and j <= r) {              // while both subarrays are nonempty
      if (A[i] <= A[j]) B[k++] = A[i++]        // copy from left subarray
      else B[k++] = A[j++]                     // copy from right subarray
      }
      while (i <= q) B[k++] = A[i++]           // copy any leftover to B
      while (j <= r) B[k++] = A[j++]
      for i = p to r do A[i] = B[i]            // copy B back to A                }
```

T (n) =1 if n = 1 ,

2T (n/ 2) + n otherwise.

Solving the above recurrence we can see that merge sort has a time complexity of Θ (n log n) .



Q3.

Ans of 3(b):Merging order is not important, You can select any, this will affect only to bit-sequences, assigned to this or that symbol, but: a) compressed file will has same size. b) decompressed file will be equals to original one.

Q4.

a. What does this algorithm compute? (5pts)

b. What is its basic operation and how many times is the basic operation executed? (5pts)

c. What is the efficiency class ($\Theta(g(n))$) the function belongs to) of this algorithm? (5pts)

d. Make as many improvements as you can in this algorithmic solution to the problem. You must write down the pseudocode for your new algorithm. What is the efficiency class of your new algorithm? If you cannot improve this algorithm, explain why you cannot do it. (10pts)

Solution:

a. Check if the input matrix is a symmetric matrix. It returns "true" if it is a symmetric matrix; and returns "false" otherwise.

b. Comparison of two matrix elements. $C(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n^2$

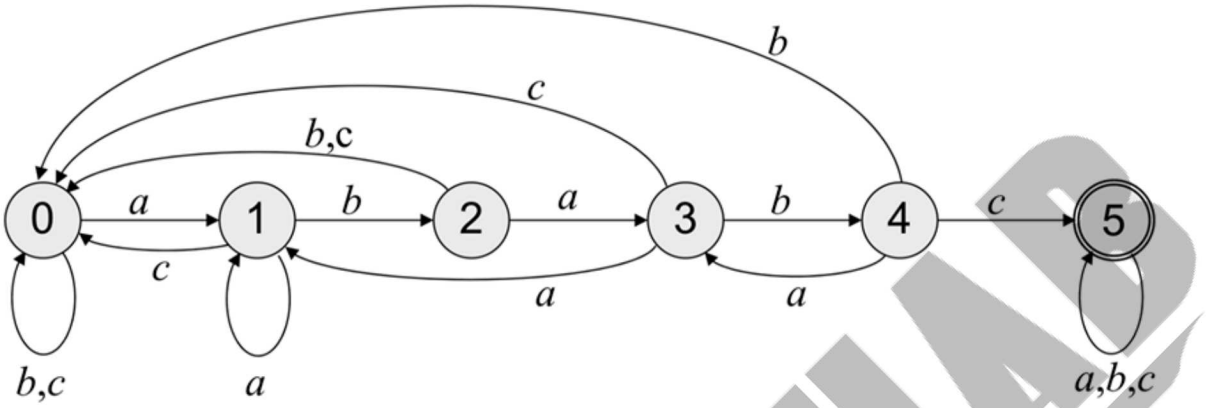
c. $C(n) = n^2 \in \Theta(n^2)$

d.

Algorithm Enigma(A[0..n − 1, 0..n − 1])
    //Input: A matrix A[0..n − 1, 0..n − 1] of real numbers
    for i ← 0 to n − 2 **do**
        for j ← i+1 to n − 1 **do**
            if A[i, j] ≠ A[j, i]
                return false
return true

Q5.

Example: A *finite automaton* to match pattern *ababc* over alphabet $\Sigma = \{a,b,c\}$.



Matching pattern *ababc* in text *caabaabcababababccb*.

char read		***c***	***a***	***a***	***b***	***a***	***a***	***b***	***c***	***a***	***b***	***a***	***b***	***a***	***b***	***c***	***c***	***b***
new state	0	0	1	1	2	3	1	2	0	1	2	3	4	3	4	5	5	5

Q6.

Graph

	v1	v2	v3	v4
v1	0	3	8	INF
v2	INF	0	5	1
v3	INF	4	0	INF
v4	2	5	INF	0

Distances

	v1	v2	v3	v4
v1	0	3	8	4
v2	3	0	5	1
v3	7	4	0	5
v4	2	5	10	0

Predecessors

	v1	v2	v3	v4
v1	0	0	0	1
v2	3	1	1	1
v3	3	2	2	1
v4	3	3	0	3

The sortest path from a to c is: **v1 v3** The sortest path from a to c is: **v2 v4** 8

Q7.

$f_{i,j}$

	1	2	3	4	5	6
1	9	18	20	24	32	35
2	12	16	22	25	30	37

$f^* = 38$

$I_{i,j}$

	2	3	4	5	6
1	1	2	1	1	2
2	1	2	1	2	2

$I^* = 1$

$I^* = 1$

Solution:

$l^* = 1 \Rightarrow$ Station $S_{1,6}$

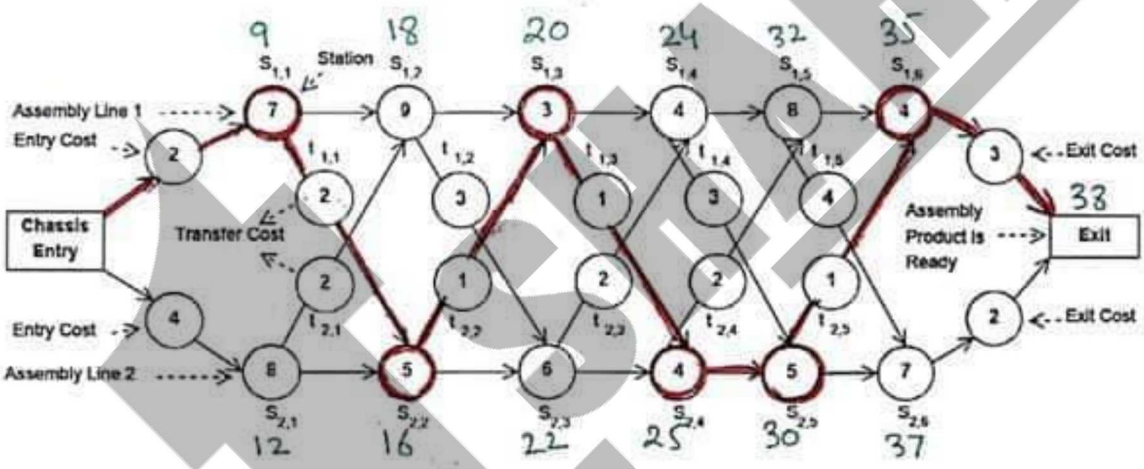
$l_1[6] = 2 \Rightarrow$ Station $S_{2,5}$

$l_2[5] = 2 \Rightarrow$ Station $S_{2,4}$

$l_2[4] = 1 \Rightarrow$ Station $S_{1,3}$

$l_1[3] = 2 \Rightarrow$ Station $S_{2,2}$

$l_2[2] = 1 \Rightarrow$ Station $S_{1,1}$

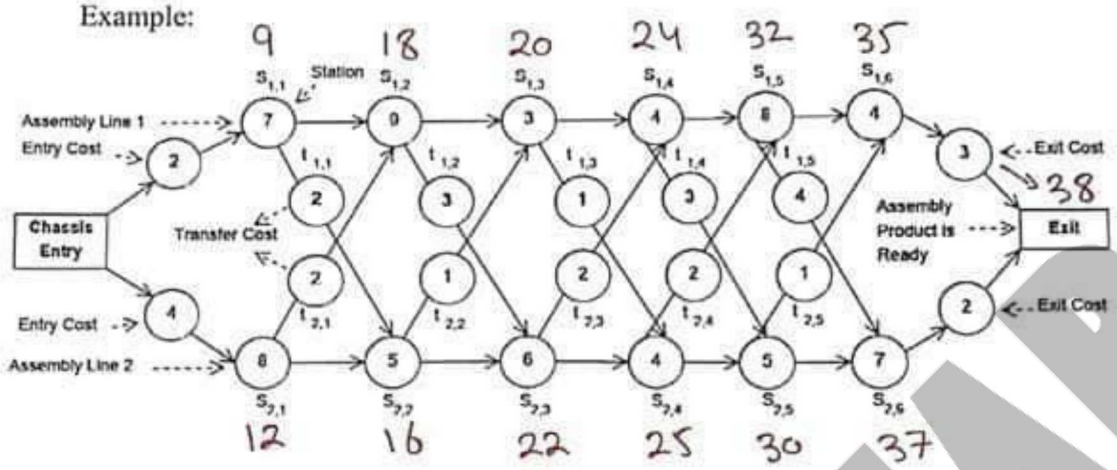


The algorithm: **Fastest-way**(a, t, e, x, n) Source: CLRS

where a denotes the assembly costs, t denotes the transfer costs, e denotes the entry costs, x denotes the exit costs and n denotes the number of assembly stages.

1. $f_1[1] = e_1 + a_{1,1}$
2. $f_2[1] = e_2 + a_{2,1}$
3. for $j = 2$ to n
4. if $((f_1[j-1] + a_{1,j}) \leq (f_2[j-1] + t_{2,j-1} + a_{1,j}))$ then
5. $f_1[j] = f_1[j-1] + a_{1,j}$ and $l_1[j] = 1$ /* l_p denotes the line p */
6. else
7. $f_1[j] = f_2[j-1] + t_{2,j-1} + a_{1,j}$ and $l_1[j] = 2$
8. if $((f_2[j-1] + a_{2,j}) \leq (f_1[j-1] + t_{1,j-1} + a_{2,j}))$ then
9. $f_2[j] = f_2[j-1] + a_{2,j}$ and $l_2[j] = 2$
10. else
11. $f_2[j] = f_1[j-1] + t_{1,j-1} + a_{2,j}$ and $l_2[j] = 1$
12. end for
13. if $(f_1[n] + x_1 \leq f_2[n] + x_2)$ then
14. $f^{OPT} = f_1[n] + x_1$ and $l^{OPT} = 1$
15. else
16. $f^{OPT} = f_2[n] + x_2$ and $l^{OPT} = 2$

Example:



Iteration 1: $j = 1$ *and* $j = 2$

$f_1[1] = e_1 + a_{1,1} = 2 + 7 = 9$ *and start* $= S_{1,1}$

$f_2[1] = e_2 + a_{2,1} = 4 + 8 = 12$ *and start* $= S_{2,1}$

$f_1[2] = \min\{f_1[1] + a_{1,2}, f_2[1] + t_{2,1} + a_{1,2}\} = \min\{9 + 9, 12 + 2 + 9\} = \underline{18}, l_1[2] = 1$

$f_2[2] = \min\{f_2[1] + a_{2,2}, f_1[1] + t_{1,1} + a_{2,2}\} = \min\{12 + 5, 9 + 2 + 5\} = \underline{16}, l_2[2] = 1$

Iteration 2: $j = 3$

$f_1[3] = \min\{f_1[2] + a_{1,3}, f_2[2] + t_{2,2} + a_{1,3}\} = \min\{18 + 3, 16 + 1 + 3\} = \underline{20}, l_1[3] = 2$

$f_2[3] = \min\{f_2[2] + a_{2,3}, f_1[2] + t_{1,2} + a_{2,3}\} = \min\{16 + 6, 18 + 3 + 6\} = \underline{22}, l_2[3] = 2$

Iteration 3: $j = 4$

$f_1[4] = \min\{f_1[3] + a_{1,4}, f_2[3] + t_{2,3} + a_{1,4}\} = \min\{20 + 4, 22 + 2 + 4\} = \underline{24}, l_1[4] = 1$

$f_2[4] = \min\{f_2[3] + a_{2,4}, f_1[3] + t_{1,3} + a_{2,4}\} = \min\{22 + 4, 20 + 1 + 4\} = \underline{25}, l_2[4] = 1$

Iteration 4: $j = 5$

$f_1[5] = \min\{f_1[4] + a_{1,5}, f_2[4] + t_{2,4} + a_{1,5}\} = \min\{24 + 8, 25 + 2 + 8\} = \underline{32}, l_1[5] = 1$

$f_2[5] = \min\{f_2[4] + a_{2,5}, f_1[4] + t_{1,4} + a_{2,5}\} = \min\{25 + 5, 24 + 3 + 5\} = \underline{30}, l_2[5] = 2$

Iteration 5: $j = 6$

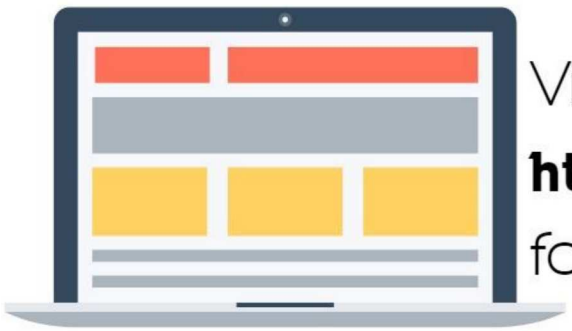
$f_1[6] = \min\{f_1[5] + a_{1,6}, f_2[5] + t_{2,5} + a_{1,6}\} = \min\{32 + 4, 30 + 1 + 4\} = \underline{35}, l_1[6] = 2$

$f_2[6] = \min\{f_2[5] + a_{2,6}, f_1[5] + t_{1,5} + a_{2,6}\} = \min\{[illegible]\} = \underline{37}, l_2[6] = 2$

$\{30 + 7, 32 + 4 + 7\}$

Optimal Schedule:

$f^* = \min\{f_1[6] + x_1, f_2[6] + x_2\} = \min\{35 + 3, 37 + 2\} = (38)$ *AND* $l^* = (1)$



Visit

https://tshahab.blogspot.com

for more.

Solved by Talha Shahab