COMP271: Design & Analysis of Algorithms
Mid-term Examination

Solution

<u>Problem 1</u> (15 pts)

Given the following 10 functions of *n*:

$f_1(n) = n^n$,

$f_2(n) = (n+4)^3$,

$f_3(n) = 2n^3$,

$f_4(n) = 3^n$,

$f_5(n) = 2^n$,

$f_6(n) = n \log n$,

$f_7(n) = n \log \log n$,

$f_8(n) = n \log_{10} n$

$f_9(n) = n^{1.5}$

$f_{10}(n) = (\log n)^n$

Sort the 10 functions in asymptotic ascending order. In your answer,
if $f_1 = O(f_2)$, *and* $f_1 \neq \Theta(f_2)$, simply write $f_1 < f_2$;
if $f_1 = \Theta(f_2)$, simply write $f_1 = f_2$, etc.
For example, a possible answer may be:
$f_3 < f_4 < f_1 = f_2 = f_5 < f_6 < f_8 = f_7 < f_9 < f_{10}$

Answer:
$f_7 < f_8 = f_6 < f_9 < f_3 = f_2 < f_5 < f_4 < f_{10} < f_1$

<u>Problem 2</u> (20 pts)

Solve the following recurrence equation:

$$T(n) = \begin{cases} 1, & if \ \ n = 1 \\ 2T(n/2) + n \lg n, & if \ \ n > 1 \end{cases}$$

You can use any method you wish.

Answer:

$$T(n) = 2T(n/2) + n \lg n$$

$$= 2(2T(n/4) + \frac{n}{2} \lg \frac{n}{2}) + n \lg n$$

$$= 4T(n/4) + n \lg \frac{n}{2} + n \lg n$$

$$= 8T(n/8) + n \lg \frac{n}{4} + n \lg \frac{n}{2} + n \lg n$$

$$\vdots$$

$$= nT(n/n) + n \lg \frac{n}{n/2} + \ldots + n \lg \frac{n}{4} + n \lg \frac{n}{2} + n \lg n$$

$$= n + n \lg n \lg n + n \lg(\frac{1}{n/2} \frac{1}{n/4} \frac{1}{n/8} \cdots \frac{1}{8} \frac{1}{4} \frac{1}{2})$$

$$= n + n \lg n \lg n + n \lg((\frac{1}{n})^{\frac{1}{2} \lg n})$$

$$= n + n \lg n \lg n - \frac{1}{2} n \lg n \lg n$$

$$= \Theta(n \lg^2 n)$$

A recursive tree also works. This is an example that master method fails.

<u>Problem 3</u> (20 pts)

Given n integers in the range of [1..k]. Note that k can be smaller, bigger or equal to n. Describe an algorithm that preprocesses these n integers and then answers any query about how many of the n integers fall into a range [a...b] in O (1) time, here $1 \le a \le b \le k$. Your algorithm should use O (n + k) preprocessing time. Argue briefly the correctness and running time of your algorithm. (Hint: you may think about the counting sort algorithm.)

Answer:

```
preprocesses (A[], C[], k) {
  for i=0 to k do C[i]=0;        //O(k)
  for j=1 to n do C[A[j]]++;    //O(n) (1)
  for i=2 to k do C[i]+=p[i-1]; //O(k) (2)
}
```

Total time complexity = O(n+k) // (3)

```
range (C[],a,b) {
  return(C[b]-C[a-1]);          //O(1) (4)
}
```

<u>Problem 4a</u> (6 pts)
Enumerate six real-world problems that can be represented by graph.

Answer:
There are many real-world problems just as the lecture notes shown.

<u>Problem 4b</u> (13 pts)
Traverse the following directed graph $G=(V,E)$ by using Depth-First Search starting from vertex a. You must use the adjacent list representation given below. Show the tree edges, back edges and time-stamps produced by the search.

$a :\to c, f$

$b :\to a, e$

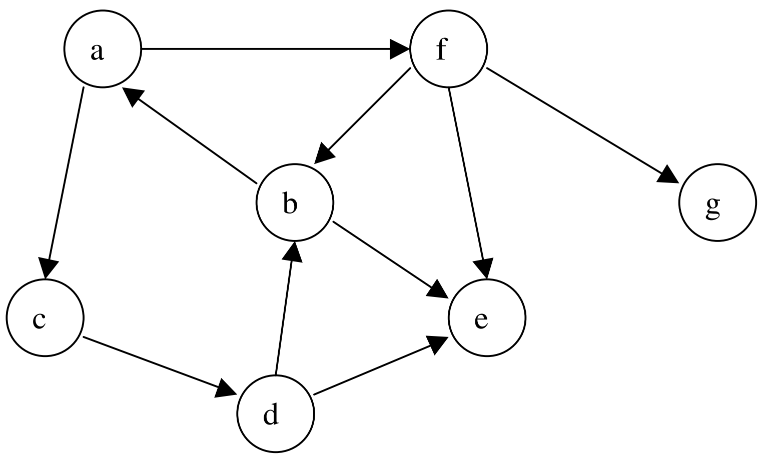
$c :\to d$

$d :\to b, e$

$e :\to \phi(empty)$

$f :\to b, e, g$

$g :\to \phi(empty)$



Answer:

Time-stamps
a: 1/14 b: 4/7 c: 2/9 d: 3/8 e: 5/6 f: 10/13 g: 11/12
Tree edge: a->c, c->d, d->b, b->e, a->f, f->g
Back edge: b->a

<u>Problem 4c</u> (6 pts)

Is the above graph a directed acyclic graph (DAG)? If yes, give a possible topological sort ordering for the graph. If no, delete edges as fewest as you can to make the new graph a DAG, then give a possible topological sort ordering for the new graph.

Answer:

No. Delete edge b->a

There are many possible answers for topological ordering.

a, f, c, g, d , b, e

or

a, c, f, g, d , b, e

or

a, c, f, d , g , b, e

or

a, f, c, d , g , b, e

<u>Problem 5</u> (20 pts)

Let $G=(V,E)$ be a connected undirected graph in which all edges have weight either 1 or 2. Give an $O(|V|+|E|)$ algorithm to compute a minimum spanning tree of G. Justify the running time of your algorithm. (Note: You may either present a new algorithm or just show how to modify an algorithm taught in class.)

Answer:

We modify the Prim's algorithm. The main idea is, instead of using a $O(\log n)$ running time data structure (e.g., heap) to maintain the priority queue, we use three linked lists to store the value of the vertices. Since the key value of each vertex can only be 1, 2 or infinite, we can use the three linked lists to store all vertices. INSERT operation due with appending the vertex in a linked list which only takes O(1) time; DECREASE_KEY operation involves a delete operation and a INSERT operation, we can implement the delete operation carefully in O(1) time, hence DECREASE_KEY takes O(1) time; and EXTRACT_MIN operation involves an extraction of an vertex from linked-list 1 or linked list 2, in both case, it takes only O(1) time. Therefore, the INSERT, DECREASE_KEY and EXTRACT_MIN operation can be done in O(1) time.

Hence, the running time of modified algorithm is :

$$T(n) = \sum_{u\in V} (O(1) + \deg(u)O(1))$$
$$= O(1)\sum_{u\in V} (1 + \deg(u))$$
$$= O(|V| + |E|)$$