

**Computer Science 3711 / Engineering 5891 (Winter 2001):  
Midterm Exam  
Answers**

1. (8 marks) Give an **exact** closed form, *i.e.*, without summations, expression for

$$\left(\sum_{k=0}^n k(k+3)\right) - \left(\sum_{k=3}^n k^2\right)$$

**Answer:**

$$\begin{aligned} \left(\sum_{k=0}^n k(k+3)\right) - \left(\sum_{k=3}^n k^2\right) &= \left(\sum_{k=0}^n (k^2 + 3k)\right) - \left(\sum_{k=3}^n k^2\right) \\ &= \sum_{k=0}^n k^2 + \sum_{k=0}^n 3k - \left(\sum_{k=3}^n k^2\right) \\ &= \sum_{k=0}^n k^2 + \sum_{k=0}^n 3k - \left(\sum_{k=0}^n k^2 - \sum_{k=0}^2 k^2\right) \\ &= \sum_{k=0}^n 3k + \sum_{k=0}^2 k^2 \\ &= 3 \sum_{k=0}^n k + \sum_{k=0}^2 k^2 \\ &= 3 \sum_{k=0}^n k + (0^2 + 1^2 + 2^2) \\ &= 3 \frac{n(n+1)}{2} + 5 \\ &= \frac{3}{2}n^2 + \frac{3}{2}n + 5 \end{aligned}$$

2. (12 marks)

- a) (6 marks) Show that  $T(n) \leq n - 1$  by the substitution method, where

$$T(n) = \begin{cases} 0 & n < 3 \\ 3T(\lfloor \frac{n}{3} \rfloor) + 2 & n \geq 3 \end{cases}$$

**Answer:** The inductive hypothesis holds when  $n = 3$  as  $3T(\lfloor \frac{3}{3} \rfloor) + 2 = 3T(1) + 2 = 0 + 2 = 2 \leq 3 - 1 = 2$ . We can then show that the inductive hypothesis

holds for  $n > 4$  as follows:

$$\begin{aligned}
 T(n) &= 3T(\lfloor \frac{n}{3} \rfloor) + 2 \\
 &= 3(\lfloor \frac{n}{3} \rfloor - 1) + 2 \\
 &\leq 3(\frac{n}{3} - 1) + 2 \\
 &= n - 3 + 2 \\
 &= n - 1
 \end{aligned}$$

One other matter remains. As stated above, there is a slight problem in that we claim that the inductive hypothesis also holds over the boundary condition, *i.e.*,  $T(n) \leq n - 1$  for all  $n$  including  $n < 3$ ; though this is true for  $n = 1$  ( $T(1) = 0 \leq 1 - 1 = 0$ ) and  $n = 2$  ( $T(2) = 0 \leq 2 - 1 = 1$ ), it is *not* true for  $n = 0$  ( $T(0) = 0 \not\leq 0 - 1 = -1$ ). However, we can fix this by adjusting the boundary condition such that  $T(n) = 0$  when  $n = 1$  and  $n = 2$  – we don't need to assign a value to  $T(n)$  for any  $n \leq 0$  as these terms will never be invoked during the solution of the recurrence for any positive  $n$  (many thanks to Mike Rowe for noting this problem with the recurrence as originally stated).

**b) (6 marks)** Derive an upper bound for  $T(n)$  by the iteration method, where

$$T(n) = \begin{cases} 0 & n \leq 1 \\ 3T(n-1) + 2 & n > 1 \end{cases}$$

**Answer:** This recurrence is slightly different from those you've seen before because the value of  $T(n)$  goes to 0 at 1 instead of 0, *cf.*, Question #3 on Assignment #2. This ends the series of terms slightly earlier than expected, as shown below:

$$\begin{aligned}
 T(n) &= 2 + 3T(n-1) \\
 &= 2 + 3(2 + 3T(n-2)) \\
 &= 2 + 6 + 9(2 + 3T(n-3)) \\
 &= 2 + 6 + 18 + 27(2 + 3T(n-4)) \\
 &= 2(3^0) + 2(3^1) + 2(3^2) + 2(3^3) + \dots + 2(3^{n-2}) + 3^{n-1}T(1) \\
 &= 2(3^0) + 2(3^1) + 2(3^2) + 2(3^3) + \dots + 2(3^{n-2}) + 3^{n-1}(0) \\
 &= 2 \sum_{k=0}^{n-2} 3^k \\
 &= 2 \left( \frac{3^{n-1} - 1}{3 - 1} \right) \\
 &= 3^{n-1} - 1
 \end{aligned}$$

This suggests that it would be a good idea in general to always run at least two values of  $n$  through a new recurrence to see exactly what terms are in the series.

### 3. (10 marks)

As part of the proofs below, please give appropriate  $c$ - and  $n_0$ -values.

a) (6 marks) Prove that  $f(n) = (n - 5)^2$  is  $\Theta(n^2)$ .

**Answer:** This answer has two parts:

- i)  $f(n) = (n - 5)^2$  is  $O(n^2)$ : Observe that  $(n - 5)^2 = n^2 - 10n + 25 \leq n^2 + 10n^2 + 25n^2 = 36n^2$ . Hence, the required inequality holds when  $c = 36$  and  $n_0 = 1$ . Alternatively, by inspection, one can verify that the required inequality holds when  $c = 1$  and  $n_0 = 3$ .
- ii)  $f(n) = (n - 5)^2$  is  $\Omega(n^2)$ : This is a bit tricky. The required inequality seems to hold when  $c = n_0 = 1$ ; however, there is a problem when  $n = 5$ , as  $n^2 - 10n + 25 = 25 - 50 + 25 = 0$ . The simplest solution is to let  $c = 1$  and  $n_0 = 6$ .

b) (4 marks) Prove that for any graph  $G = (V, E)$ ,  $f(G) = \frac{|E|}{|V|} \log |E|$  is  $O(|V| \log |V|)$ .

**Answer:** The key here is to remember that for any graph,  $|E| \leq \frac{|V|(|V|-1)}{2} \leq 2|V|^2$ . Hence, we get

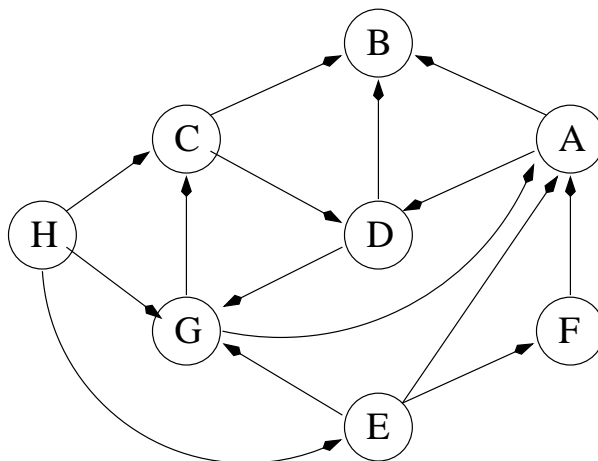
$$\begin{aligned} f(G) &= \frac{|E|}{|V|} \log |E| \\ &\leq \frac{2|V|^2}{|V|} \log 2|V|^2 \\ &= 2|V|(\log 2 + \log |V|^2) \\ &= 2|V|(1 + 2 \log |V|) \\ &\leq 2|V|(3 \log |V|) \\ &= 6(|V| \log |V|) \end{aligned}$$

Hence,  $f(G) = \frac{|E|}{|V|} \log |E|$  is  $O(|V| \log |V|)$  when  $c = 6$ . Where, you may very rightly ask, is the  $n_0$  value? There isn't one, as  $f$  is actually defined over the space of all graphs rather than the set of integers. Is this an abuse of notation? Yes it is, but it is a common one – it is routine to express algorithm time complexities in terms of several parameters of the input, so there actually isn't any one input size such that an asymptotic result holds for all input sizes above a particular  $n_0$ . In such cases, it is common to assume that there are actually multiple  $n_0$ , one for each parameter of the given function, all of which are set to 1.

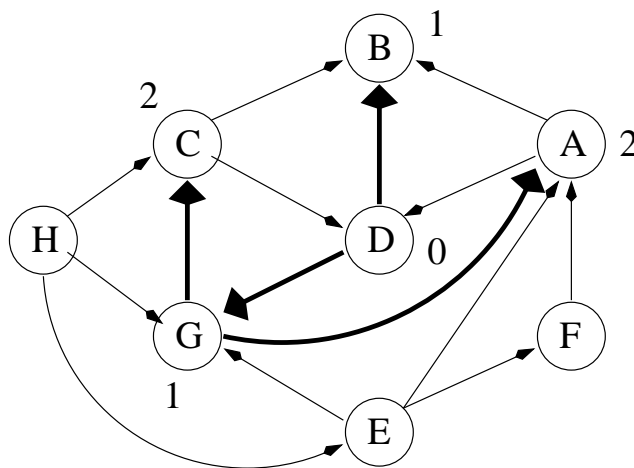
## 4. (15 marks)

Assume that the algorithms cited below consider vertices in alphabetical order and that each adjacency list is ordered alphabetically.

a) (5 marks) Consider the following directed graph:



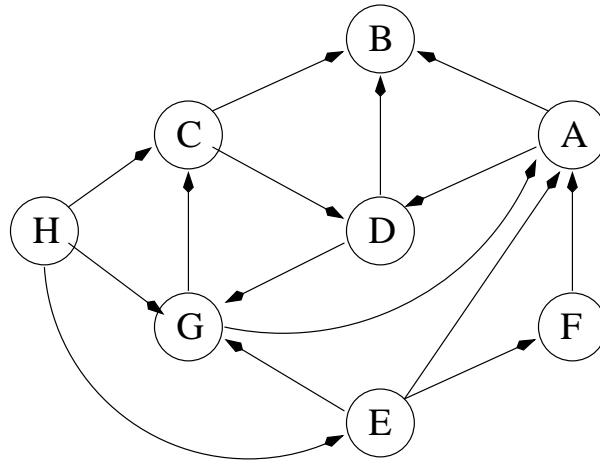
Give the graph at the end of the execution of the BFS algorithm when the search is started at vertex D, with the  $d$ -values for all vertices as well as all BFS-search tree edges clearly marked.



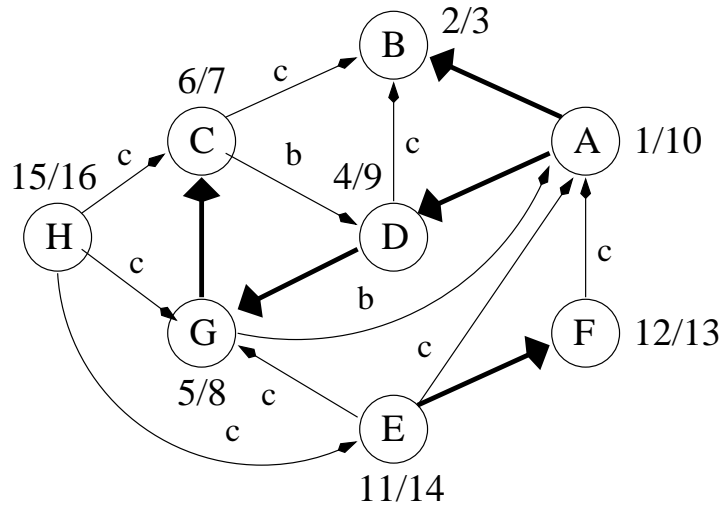
**Answer:**

Each vertex reachable from vertex D is annotated with a number indicating that vertex's  $d$ -value.

b) (10 marks) Consider the following directed graph:



Give the graph at the end of the execution of the DFS algorithm, with the  $d$ - and  $f$ -values of all vertices as well as the types of all edges clearly marked.

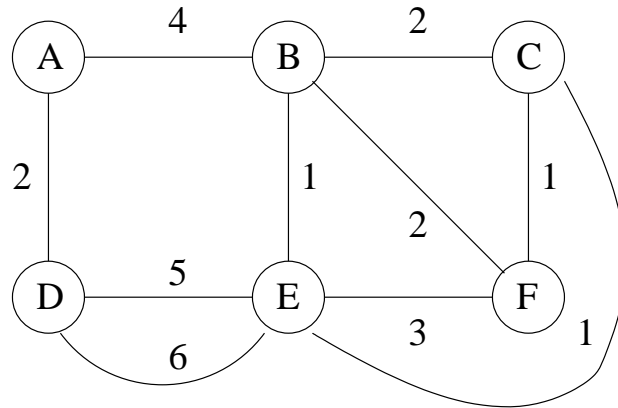


**Answer:**

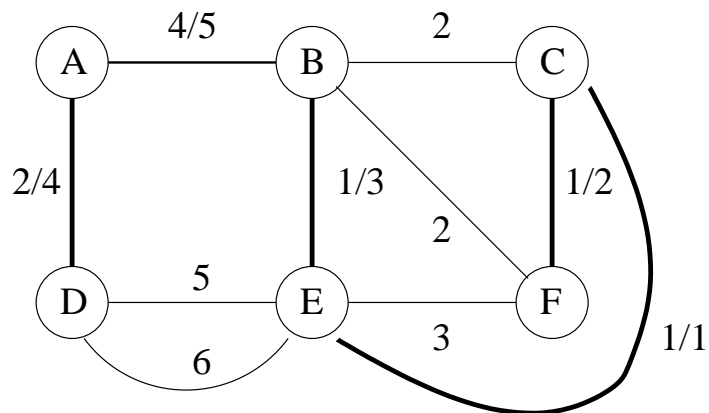
Each vertex is annotated with a string " $x/y$ " where  $x$  and  $y$  are that vertex's  $d$ - and  $f$ -values, respectively. Each edge is either bold face, *i.e.*, it is a tree edge, or is annotated by a letter that indicates that edge's type (back ( $b$ ) / forward ( $f$ ) / cross ( $c$ )). Note that cross edges go between any two vertices such that one is not an ancestor of the other; hence, cross edges are not restricted to being between search trees but can also exist within a single tree, *e.g.*, edges  $(D, B)$  and  $(C, B)$  above.

5. (20 marks)

a) (5 marks) Consider the following weighted undirected graph:



Give the graph at the end of the execution of Kruskal's minimum spanning tree algorithm, with all tree-edges and the order in which each tree-edge was added clearly marked.



**Answer:**

Each edge that is part of the minimum spanning tree is annotated with a string " $x/y$ " such that  $x$  and  $y$  are that edge's weight and add-order under Kruskal's algorithm. Note that there is a tie in the weights of first three edges added; hence, depending on the tie-resolution policy adopted by Kruskal's algorithm, any order of addition of these three edges could be valid.

- b) (6 marks) Suppose the graph in part (a) is modified to contain an additional edge  $(B, D)$  with weight (i) 6, (ii) 4, or (iii) 2. For each of these three cases, say whether or not Kruskal's algorithm could produce a different minimum spanning tree from that given in part (a) of this question, as well as whether or not the number of minimum spanning trees of the modified graph is different from the number of minimum spanning trees of the original graph in part (a).

i) Weight = 6	Different min spanning tree possible under Kruskal?	<u>No</u>
	# min spanning trees different?	<u>No</u>
ii) Weight = 4	Different min spanning tree possible under Kruskal?	<u>Yes</u>
	# min spanning trees different?	<u>Yes</u>
iii) Weight = 2	Different min spanning tree possible under Kruskal?	<u>Yes</u>
	# min spanning trees different?	<u>No</u>

**Answer:** You need to read this question carefully because it is slightly tricky. In case i), new edge  $(B, D)$  will not be included in any minimum spanning tree for the modified graph, so you get the same minimum spanning tree and the same number of minimum spanning trees, *i.e.*, 1, as for the original graph. In case ii), there is now a tie between edges  $(A, B)$  and  $(B, D)$ , which results in two minimum spanning trees for the modified graph, and the one produced by Kruskal's algorithm depends on what tie-resolution policy is adopted. In case iii), edge  $(D, B)$  is now selected instead of edge  $(A, B)$  so the modified graph has a totally different minimum spanning tree from the original graph; however, note that this tree is unique so both graphs once again have only one minimum spanning tree.

- c) (9 marks) Consider the following simplified form of Prim's minimum spanning tree algorithm:

```

INIT_Q(V);
pred[r] = nil;
while NOT_EMPTY(Q) do
    u = EXTRACT_MIN(Q);
    for each v that is adjacent to u in G do
        if IN_Q(v) and UPDATE_Q(v) then
            pred[v] = u;

```

As discussed in class and described in the textbook, this algorithm runs in  $O(|E| \log |V|)$  time if priority queue Q is implemented as a binary heap, which allows the operations INIT\_Q, NOT\_EMPTY, EXTRACT\_MIN, IN\_Q, and UPDATE\_Q be done in  $O(|V|)$ ,  $O(1)$ ,  $O(\log |V|)$ ,  $O(1)$ , and  $O(\log |V|)$  time, respectively. Give the time complexity of Prim algorithm when the INIT\_Q and NOT\_EMPTY operations require  $O(|V|)$  and  $O(1)$  time, respectively, and the EXTRACT\_MIN, IN\_Q, and UPDATE\_Q operations require

- i)  $O(|V|)$ ,  $O(|V|)$ , and  $O(1)$  time, respectively;
- ii)  $O(|V|)$ ,  $O(1)$ , and  $O(|V|)$  time, respectively; and
- iii)  $O(1)$ ,  $O(\log |V|)$ , and  $O(|V|)$  time, respectively

**Answer:** The key here is to recognize that the time complexity of the algorithm can be written as  $O(|V|(\text{UPDATE\_Q}) + |E|(\text{IN\_Q} + \text{UPDATE\_Q}))$ . One can then plug in the appropriate operation-complexities and simplify the resulting expressions. Thus, we get  $O(|V|(O(|V|)) + |E|(O(|V|) + O(1))) = O(|V|^2 + |E||V|) = O(|E||V|)$ , for part i),  $O(|V|(O(|V|)) + |E|(O(1) + O(|V|))) = O(|V|^2 + |E||V|) = O(|E||V|)$ , for part ii), and  $O(|V|(O(1)) + |E|(O(\log |V|) + O(|V|))) = O(|V| + |E||V|) = O(|E||V|)$ , for part iii).