



American University of Beirut



CMPS 256 Final Exam

January 21, 2003

2 hour exam



Student Name: _____ Student ID: _____

Signature: _____

- There are *twelve* pages, including this one. The test is out of 100 marks, and the value of each question is provided; please use this information to manage your time effectively.
- Your handwriting should be neat and readable OR I will not mark the answer!

Question 1: _____/10
Question 2: _____/10
Question 3: _____/10
Question 4: _____/10
Question 5: _____/10
Question 6: _____/10
Question 7: _____/10
Question 8: _____/10
Question 9: _____/10
Question 10: _____/10
Total: _____/100

Instructor: *Marcel R. karam*



Question 1 [10 points] Multiple Choice

True or false? No explanation required. Each correct answer is worth 1 points, but 1 points will be subtracted for each wrong answer, so answer only if you are reasonably certain.

- 1- **T** **F** By the master theorem, the solution to the recurrence: $T(n) = 3T(n/3) + \log n$ is: $O(n \log n)$
- 2- **T** **F** Given a graph $G = (V, E)$ with cost on edges, and a set S in V . Let (u, v) be an edge such that (u, v) is the minimum cost edge between any vertex in S and any vertex in $V - S$. Then, the minimum spanning tree of G must include the edge (u, v) . (You may assume the costs on all edges are distinct, if needed.)
- 3- **T** **F** Let P be a shortest path from some vertex s to some other vertex t in a graph G . If the weight of each edge in the graph is increased by one, P remains a shortest path from s to t .
- 4- **T** **F** The breadth first search algorithm makes use of a stack.
- 5- **T** **F** The Bellman-Ford algorithm is not suitable if the input graph that has negative-weight edges
- 6- **T** **F** Depth-first search of a graph is asymptotically faster than breadth-first search.
- 7- **T** **F** Dijkstra's algorithm is an example of a greedy algorithm.
- 8- **T** **F** The maximum number of edges that can be removed from a graph with n nodes in order to construct a spanning tree is: $n(n-1)/2 - (n-1) = (n-1)(n/2 - 1)$
- 9- **T** **F** You are given a graph in which all edges have non-negative weights, except those leaving the source vertex s . Then Dijkstra's algorithm may fail to correctly determine the shortest paths from s .
- 10- **T** **F** Suppose T is a minimum spanning tree for a graph G . Then the largest edge weight of T is no larger than the largest edge weight of any other spanning tree of G .



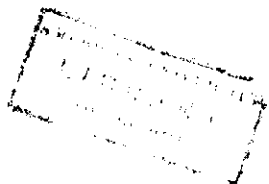
Question 2: Asymptotic and Recurrences [10 points]

The following is a Divide-and-Conquer algorithm for finding the minimum value in an (unsorted) array $S[1..n]$. The library function $\min(a, b)$ used in the algorithm returns the smaller between a and b .

```
function minimum(S, x, y)
  if  $y - x \leq 1$ 
    then return  $\min(S[x], S[y])$ 
  else
     $\text{min1} \leftarrow \text{minimum}(S, x, \lfloor (x + y)/2 \rfloor)$ 
     $\text{min2} \leftarrow \text{minimum}(S, \lfloor (x + y)/2 \rfloor + 1, y)$ 
  return  $\min(\text{min1}, \text{min2})$ 
```

a) Write a recurrence relation for the worst-case number of comparisons used by $\text{minimum}(S, 1, n)$. (You may assume that n is a power of two.)

b) What is the running time of $\text{minimum}(S, 1, n)$. Justify your answer



Question 3: Hash Tables [10 points]

You are given a hash table $m = 10$, where all elements are stored in the hash table itself (i.e., open addressing). On insertion, collisions are resolved using the following reverse linear probe auxiliary hash function: $h(k, i) = (h'(k) - i) \bmod m$ for $i = 0, \dots, m-1$. Namely, if a collision occurs, the immediately previous slot is examined. If that slot is not empty, preceding slots are examined, one by one, until an empty slot is found. If you reach the beginning of the table, you then wrap around to the end and continue from there.

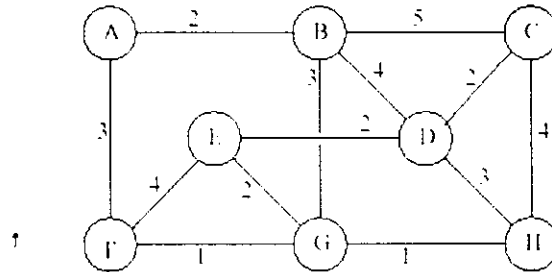
(a) The hash table is initialized ($T = 0$) and the n following sequence of insertions is made:

insertion order \rightarrow	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th
key $k =$	a	b	c	d	e	f	g
primary hash $h'(k) =$	2	4	4	9	3	3	2



Question 4: Minimum Spanning Trees [10 points]

Find a minimum spanning tree for the following graph using Prim's algorithm. Start with vertex *A* and show only the final stage of the constructed tree.



Question 5: Shortest path [10 points]

Does Dijkstra's shortest path algorithm produce a correct shortest path for input graphs with negative edge weights. Prove using an example:

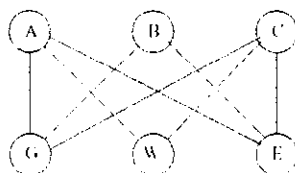


Question 6: Graphs [10 points]

(a) Draw the following graph:

$=(\{1, 2, 3, 4, 5, 6\}, \{\{1, 2\}, \{1, 4\}, \{2, 5\}, \{2, 6\}, \{3, 4\}, \{3, 5\}, \{3, 6\}, \{4, 5\}, \{4, 6\}, \{5, 6\}\})$

(b) Redraw the following graph in such a way that no two lines cross.



(c) Add one or more edges to the above graph to create one which cannot be redrawn in such a way that no two lines cross (a proof is not required).



Question 7 [10 points]

Given a specific edge (u, v) of a graph $G = (V, E)$ devise an algorithm which constructs a spanning tree of G containing (u, v)



Question 8 [10 points]

You are given a connected directed graph $G = (V, E)$ where every edge has the same positive integer weight k . There is a distinguished vertex t in this graph and we want to know the shortest distance from each of the other vertices to this distinguished vertex t . Present an algorithm, with the best asymptotic upper bound for the worst case running time, that solves this problem.

Hint: You can solve this in a couple of ways, using algorithms and data structures we have covered in this course. Your first approach may, or may not, be the best asymptotic one. So, think twice.



Question 9 [10 points]

Recall Kruskal's algorithm to produce a Minimum Spanning Tree (MST). For convenience, the algorithm from the book is copied here:

```

MST-KRUSKAL( $G, w$ )
1.  $A \leftarrow \emptyset$  //  $A$  will be MST.
2. for each vertex  $v \in V[G]$ 
3.   do MAKE-SET( $v$ ) // Create forest of singleton trees.
4. sort the edges  $E$  into nondecreasing order by weight  $w$ 
5. for each edge  $(u, v) \in E$ , taken in nondecreasing order by weight
6.   do if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) // Are  $u$  and  $v$  in disjoint trees?
7.     then  $A \leftarrow A \cup \{(u, v)\}$  // Yes. Add edge  $(u, v)$  to MST.
8.         UNION( $u, v$ ) // Join disjoint trees connected by edge  $(u, v)$ 
9. return  $A$ 

```

Describe how to modify Kruskal's algorithm to find a Maximum Spanning Tree, instead of a Minimum Spanning Tree.

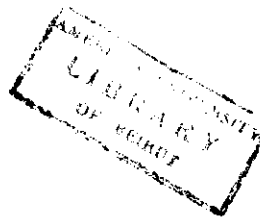


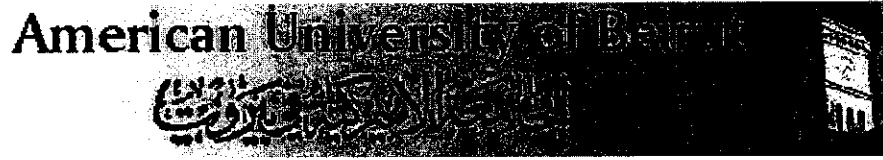
Question 10: B-trees [10 points]

show the result of inserting the following keys:

F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E

in order into an empty B-tree with minimum degree 2. Only draw the final result.





American University of Beirut
CMPS 256

Advanced Algorithms and Data Structures
 Fall 2003-2004

Final Exam



Date: Jan. 24th, 2004. 11:00 – 1:00pm.
Instructors: Chiraz Benabdelkader & Jihad Boulos

ID #: -----

Section: -----

This is **NEITHER** an open-book, **NOR** an open-notes exam. You are allowed to have 6 A4 format papers (front-and-back) on which you can write whatever you like.

Your exam should have 14 pages, and there are 7 questions totaling 100 points. Your answers should be concise, and when possible should be a list of important points rather than prose. Solve as many problems as you can. We recommend that you start with problems you think look the easiest. We also advise you to spend time on understanding what is being asked by each problem and **budget your time** wisely to be able to solve all problems.

Beware, wordy and/or irrelevant answers might reduce your score for that problem. Your answers should be the summary of work done on scratch paper that you do not hand in. Also, do not expect that the instructor will spend much time trying to decipher your hand writing. S/he will give a **ZERO** to **illegible** answers. The space allocated for answers should be sufficient. If not, use your own additional papers.

	Prob. 1	Prob. 2	Prob. 3	Prob. 4	Prob. 5	Prob. 6	Prob. 7	Total
Max Grade	21	20	10	10	12	12	15	100
Your Grade								



Exercise 1 (21 points): True or False, and Justify, 3 points each

Circle T or F for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. NO points will be granted if no justification is given.

T F The solution to the recurrence

$$T(n) = T(n/3) + T(n/6) + n^{\sqrt{\log n}}$$

is $\Theta(n^{\sqrt{\log n}})$ (assume $T(n) = 1$ for n smaller than some constant c).

T F Radix sort works in linear time only if the elements to sort are integers in the range $\{1 \dots cn\}$, for some $c = O(1)$, and n is the number of elements.

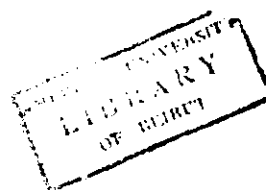
T F Suppose we use HEAPSORT instead of INSERTION-SORT to implement BUCKET-SORT. Then to sort n elements, this modified BUCKET-SORT still runs in average-case linear time, but its worst-case running time is now $O(n \log n)$.

T F To determine if two binary search trees are identical trees, one could perform an inorder tree walk on both and compare the output lists.

T F Assume you are given a magical priority-queue data structure, which performs EXTRACT-MIN, INSERT and DECREASE-KEY in constant time. Then Dijkstra's algorithm can be implemented to run in $O(E + V)$ time using this magical priority-queue.

T F Given a graph with negative weights, we modify it as follows: we find the smallest edge weight, say $w < 0$, and add $-w$ to all the edges, thus making all edge weights positive. We claim that if we run any MST algorithm on the graph with the modified weights, the algorithm outputs a correct MST for the original graph.

T F A maximum spanning tree (i.e., a spanning tree that maximizes the sum of the weights) can be constructed in $O(E \log V)$ time.



Exercise 2 (20 points): Short-answer questions, 4 points each

- (a) Does there exist an undirected graph with 5 vertices, each of degree 3? If so, draw such a graph. If not, explain why no such graph exists.
- (b) Consider the method discussed in class for classifying edges in a DFS. Briefly explain why it cannot distinguish between *cross* and *forward* edges. Describe a modification of this algorithm that can distinguish *cross* and *forward* edges. (Hint: use the discovery and finishing timestamps, $d[u]$ and $f[u]$.)
- (c) Consider step 3 of the STRONG-CONNECTED-COMPONENT(G) algorithm discussed in class (see below). Explain using an example graph why we cannot simply call $DFS(G)$ instead of $DFS(G^T)$.

STRONG-CONNECTED-COMPONENT(G)

1. Call $DFS(G)$ to compute finishing times $f[u]$ for each vertex u
 2. Compute $G^T = \text{Transpose}(G)$
 3. Call $DFS(G^T)$ based on ordering of vertices in step 2
 4. Output vertices of each tree in the DFS forest
- (d) After exactly k iterations of the outer for loop of the Bellman-Ford algorithm, which vertices v are guaranteed to have converged, i.e. $d[v] = \delta(s, v)$? Explain.
- (e) Given a path p composed of n vertices in some graph (for e.g., $p = \langle v_1, v_2, \dots, v_n \rangle$), and suppose the vertices' names are just integer numbers. Give an efficient algorithm that determines whether there is a cycle in the path.

Exercise 3 (10 points)

For the following program segment compute the worst-case asymptotic time complexity as a function of n . You can assume the running time of `loop-body` is $O(1)$. Show all your work!

```
for (i=0; i<=n-1; i++)
    loop-body

for (i=0; i<=n^2; i++)
    for (j=0; j<=sqrt(n); j++)
        for (k=j; k>=1; k=floor(k/5))
            loop-body
```

Exercise 4 (10 points) BFS

Suppose we run the BFS algorithm on a given graph $G = (V, E)$ and some source vertex s .

(a) Show that: $\forall (u, v) \in E$, if (u, v) is a non-tree edge (i.e. it is not contained in the BFS tree) then $|\delta(s, u) - \delta(s, v)| < 2$

(b) Is the converse true? i.e., $\forall (u, v) \in E$, if $|\delta(s, u) - \delta(s, v)| < 2$ then (u, v) is a non-tree edge. Justify your answer.

Exercise 5 (12 points) DFS

Suppose we run the DFS algorithm on a graph $G = (V, E)$, where $|V|=n$, and let d and f be the two n -element arrays output by the algorithm and containing respectively the discovery and finishing timestamps for each vertex in V .

(a) Write an efficient algorithm that determines the root vertex of each DFS tree in the DFS forest based only on arrays d and f . Your algorithm should return the result in an array R containing the indices of all root vertices.

(b) Write an efficient algorithm that determines all the leaves in the DFS forest based only on arrays d and f . Again, your algorithm should return the result in an array L containing the indices of the leaf vertices.

Exercise 6 (12 points): Pattern Matching

Assume you are given a *text* array $T[1 \dots n]$ containing letters from the standard English alphabet. In other words, $T[i] \in \{a, b, z\}$ for $i = 1 \dots n$. In addition, you are given a *pattern* array $P[1 \dots m]$, $m < n$, which also contains letters from the English alphabet. For any sub-array $T_i^m = T[i \dots i + m - 1]$ of T , we say that T_i^m is an *anagram* of P if there is a way of permuting symbols in T_i^m so that the resulting array is equal to P . For example, if $T = \text{"solutionislimited"}$ and $P = \text{"time"}$ then T_{13}^4 is an anagram of P .

- Give an $O(m)$ algorithm that, given an index i (and m), determines whether T_i^m is an anagram of P .
- Design an algorithm, which given T and P as an input, reports all i 's such that T_i^m is an anagram of P . Ideally, your algorithm should run in $O(n + m)$ time. However, partial credit will also be given for less efficient solutions.

Exercise 7 (15 points):

Imagine you found yourself stranded on an island in the middle of the ocean. The island is connected by bridges to other islands, which are in turn connected to other islands by other bridges, and eventually connected to the land. Unfortunately, it turns out that these bridges are rickety, i.e. each has some probability of failure. Furthermore, every bridge is marked at both entrances with some reliability measure stating its probability of failure, i.e. a value between 0 and 1 that measures the likelihood that the bridge will collapse (it is close to 0 if the bridge is very unlikely to collapse and close to 1 if it is very likely to collapse).

Your goal is ultimately to get from that island s to the land t safely, by trying to cross the most reliable bridges. To this end, you've created a weighted graph with each island as a vertex and each bridge as an edge.

- a) For any edge (u,v) , let $p(u,v)$ be the probability of failure of the corresponding bridge, and for any path $p = \langle v_1, v_2, \dots, v_k \rangle$, let $\lambda(p)$ be the probability of failure of that path. Assuming that a path fails if any of the bridges on that path fails, and that bridges fail independently of each other, express $\lambda(p)$ as a function of $p(v_1, v_2)$, $p(v_2, v_3)$, ..., $p(v_{k-1}, v_k)$.
- b) To maximize your chances of reaching land safely, you want to find the path from s to t that has the minimum probability of failure. Design a polynomial-time algorithm that finds such a path. Briefly analyze the running time of your algorithm.

Hint: The shortest-paths algorithms discussed in class work when the weight of a path $w(p)$ is equal to the *sum* of all the edge weights on path p .