University of New Mexico
Department of Computer Science

Second Midterm Examination

CS 461 Data Structures and Algorithms
Fall, 2003

Name:
Email:

- Print your name and email, *neatly* in the space provided above; print your name at the upper right corner of *every* page. Please print legibly.
- This is an *closed book* exam. You are permitted to use *only* two pages of "cheat sheets" that you have brought to the exam and a calculator. *Nothing else is permitted.*
- Do all problems in this booklet. *Show your work!* You will not get partial credit if we cannot figure out how you arrived at your answer.
- Write your answers in the space provided for the corresponding problem. Let us know if you need more paper.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- If any question is unclear, ask us for clarification.

Question	Points	Score	Grader
1	30		
2	30		
3	20		
4	20		
Total	100		

1. **Short Answer**

Multiple Choice Questions:

True or False: (circle one, 2 points each)

(a) **True or False**: If an operation takes $O(1)$ amortized time, then that operation takes $O(1)$ worst case time. *Solution: False: The worst case time could be larger*

(b) **True or False**: If an operation takes $O(1)$ worst case time then that operation takes $O(1)$ amortized time. *Solution: True*

(c) **True or False**: For a graph G and a node v in that graph, the DFS and BFS trees of G rooted at v always contain the same number of edges. *Solution: True*

(d) **True or False** Any BFS tree for a connected graph G is a spanning tree. *Solution: True*

(e) **True or False** Any shortest path tree for a connected graph G is a spanning tree. *Solution: True*

(f) **True or False** Prim's algorithm is a greedy algorithm but Kruskal's algorithm is not. *Solution: False. They are both greedy*

(g) **True or False** Prim's algorithm runs asymptotically faster than Kruskal's on sparse graphs. *Solution: False. Prim's algorithm is only asymptotically faster on dense graphs*

(h) **True or False** We can determine if an *undirected* graph $G = (V, E)$ has a cycle in $O(|V| + |E|)$ time. *Solution: True. We just compute the DFS tree and look for a back edge*

(i) **True or False** We can determine if an *directed* graph $G = (V, E)$ has a (directed) cycle in $O(|V| + |E|)$ time. *Solution: True. Again we just compute the DFS tree and look for a back edge*

(j) **True or False** Prim's algorithm uses the Union-Find data structure. *Solution: False*

(k) **True or False** In Union-Find with path compression, after we do a Find-Set(x) operation, the height of the tree that x is in always decreases. *Solution: False: x and all its ancestors become children of the root but the height of the tree may stay the same.*

(l) **True or False** The algorithm for finding all the connected components of a graph uses the Union-Find data structure. *Solution: True*

(m) **True or False** In the *best* implementation of the Union-Find data structure, the worst case cost for each operation is $O(\log^* n)$ *Solution: False. We showed that the amortized cost for each operation is $O(\log^* n)$ but the worst case cost could be higher.*

(n) **True or False** In the best implementation of Union-Find, the worst case cost for the Make-Set operation is $O(1)$. *Solution: True*

(o) **True or False**: There exists a graph with 5 nodes where the degree of every node is an odd number. *Solution: False. The sum of the degrees of all the nodes is always an even number and 5 odd numbers add up to an odd number*

2. **Amortized Analysis**

Consider the following two functions over a linked list, L:

```
AddRec(){
  x.value = 2;
  append x to the end of L
}

DecrementRecs(){
  for each x in L{
    x.value = x.value - 1;
    if(x.value==0){
      remove x from L
    }
  }
}
```

(a) Assume we perform n operations over the list L. What is the *worst case* run time of a call to AddRec during this sequence? What is the *worst case* run time of a call to DecrementRecs during this sequence? Justify your answers.

Solution: AddRec is $O(1)$ obviously. DecrementRecs is $O(n)$ which can occur if the first $n-1$ calls in the sequence are to AddRec and the last call is to DecrementRecs

(b) *Accounting Method* Now you will show that the amortized cost of both of these operations is small using the accounting method.

i. First give the amount that you will charge AddRec and the amount that you will charge DecrementRecs.

ii. Next show how you will use these charges to pay for the actual costs

iii. Finally write down the amortized cost per operation

Solution: AddRec gets charged 3 *dollars, DecrementRecs gets charged* 0 *dollars. When a rec is added, we use one dollar immediately to pay the cost of adding the rec. We then store the extra* 2 *dollars with the rec. Whenever the value of the rec is decremented, we use a dollar stored on the rec to pay the cost of that decrement. This shows that the amortized cost per operation is* $O(1)$

(c) *Potential Method* You will next use the potential method to get the amortized cost per operation. Let L_i be the linked list after the i-th operation. You will use the following potential function:

$$\Phi_i = \text{Sum of the value fields of all records in } L_i$$

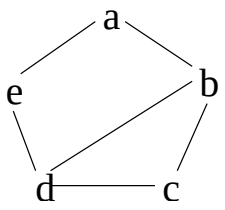
i. First show that this potential function is *valid* (i.e. $\Phi_0 = 0$ and $\Phi_i \geq 0$ for all i)

ii. Next use this potential function to calculate the amortized costs of AddRec and DecrementRecs. (Recall that $a_i = c_i + \Phi_i - \Phi_{i-1}$ where a_i is the amortized cost of the i-th operation and c_i is the actual cost of the i-th operation)

Solution: L_0 is the empty list so $\Phi_0 = 0$. The values fields of the recs are always nonnegative so $\Phi_i \geq 0$ for all i. Next we compute the amortized costs of the operations. First we calculate the amortized cost of an AddRec at time i. Note that c_i is 1 and $\Phi_i - \Phi_{i-1} = 2$. Thus $a_i = 3$. Next we calculate the amortized cost of DecrementRecs at time i. Note that in this case $\Phi_i - \Phi_{i-1}$ equals the total number of records which were decremented in the call to DecrementRecs. Thus $c_i = \Phi_i - \Phi_{i-1}$ and so $a_i = 0$. So again we've shown that the amortized cost of these operations is $O(1)$

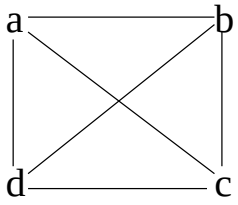
3. **BFS and DFS**

(a) Give BFS and DFS trees for the following graph. Assume that BFS and DFS are initially called with the vertex a and that the edges are stored in the adjacency lists in alphabetical order. Make sure you label which tree is a BFS tree and which is a DFS tree



Solution: The edge set $\{(a,b),(b,c),(c,d)(d,e)\}$ *is the DFS tree. The edge set* $\{(a,b),(b,c),(b,d)(d,e)\}$ *is the BFS tree*

(b) Give BFS and DFS trees for the following graph. Assume that BFS and DFS are initially called with the vertex a and that the edges are stored in the adjacency lists in alphabetical order. Make sure you label which tree is a BFS tree and which is a DFS tree

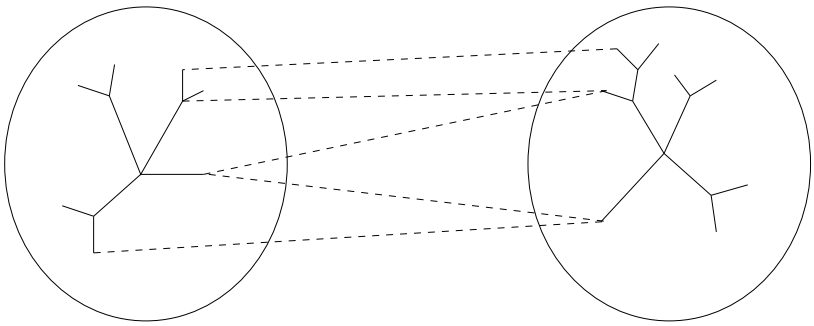


Solution: The edge set $\{(a,b),(a,c),(a,d)\}$ *is a BFS tree. The edge set* $\{(a,b),(b,c),(c,d)\}$ *is a DFS tree.*

4. **Minimum Spanning Tree**

Assume we have two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Also assume that we have T_1 which is a MST of G_1 and T_2 which is MST of G_2. Now consider a new graph $G = (V, E)$ such that $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup E_3$ where E_3 is a new set of edges that all cross the cut (V_1, V_2).

Following is an example of what G might look like. The dashed edges are E_3, the solid edges in G_1 (on the left) are T_1, and the solid edges in G_2 (on the right) are T_2.



Now assume we want to find a MST of the new graph G. Consider the following algorithm which tries to do this:

Algorithm: Maybe-MST(T_1, T_2, E_3)

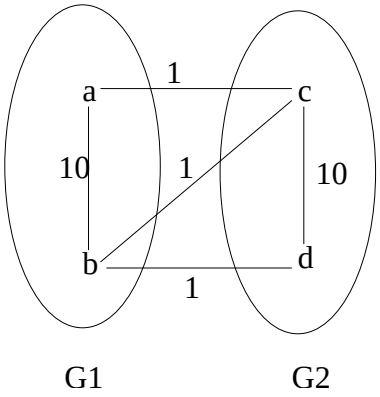
(a) e_{min} = *a minimum weight edge in* E_3

(b) $T = T_1 \cup T_2 \cup \{e_{min}\}$

(c) return T

Question: Does this algorithm always return a MST for G? If so, prove that the algorithm is correct using the "safe edge" theorem. If not, give an example input for which the algorithm fails (i.e. give T_1, T_2 and E_3 for which the algorithm fails)

Solution: The algorithm is not *correct. Consider the following graph:*



$T_1 = \{(a, b)\}$ *and* $T_2 = \{(c, d)\}$. *However the MST of* G *is* $\{(a, c), (c, b), (b, d)\}$, *which contains no edge from* T_1 *or* T_2.