Department of Computer Science

Final Examination

CS 362 Data Structures and Algorithms
Spring, 2005

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 the 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. The questions are weighted equally. 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	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **True or False**

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

(a) **True or False**: There is a greedy algorithm for $0-1$ knapsack which runs in $O(n \log n)$ time. *Solution: False. The greedy algorithm fails for* $0-1$ *knapsack on the example we went over in class. The greedy algorithm only works for fractional knapsack.*

(b) **True or False**: If there is an activity with start time earlier than all other activities, then the greedy algorithm for activity selection will always choose this activity.*Solution: False - the algorithm only considers finish times of the activities.*

(c) **True or False**: Let T be the MST for some graph G. Then for any pair of vertices x and y, the shortest path from x to y in G is the same as the path from x to y in T.*Solution: False*

(d) **True or False**: Consider a graph $G = (V, E)$, with negative weight edges. We can solve the single source shortest paths problem on G in $O(|V||E|)$ time. *Solution: True. Bellman-Ford takes* $O(|V||E|)$ *time.*

(e) **True or False**: Consider a graph $G = (V, E)$, with negative weight edges. The fastest time to determine if there is a negative cycle in G is $O(|V||E|)$. *Solution: True. We can do this with Bellman-Ford, taking* $O(|V||E|)$ *time.*

(f) **True or False**: For a connected graph G, the BFS, DFS, MST and shortest path trees will all have the same number of edges. *Solution: True. If* G *has* n *vertices, all spanning trees of* G *will have* $n-1$ *edges.*

(g) **True or False**: Consider a graph $G = (V, E)$, with negative weight edges. We can solve all pairs shortest paths on G in $O(|V|^3)$ time. *Solution: True. We can do this with Floyd-Warshall, taking* $O(|V|^3)$ *time.*

(h) **True or False**: If there is a polynomial time algorithm for some problem in NP, then all problems in NP can be solved in polynomial time *Solution: False. P is a subset of NP but this does not imply that all problems in NP can be solved in polynomial time.*

(i) **True or False**: We know of a problem in the class NP that is not in the class P. *Solution: False. If we knew of such a problem, we would know that P!=NP which is not know.*

(j) **True or False**: All problems are either in the class P or in the class NP. *Solution: False. There are some problems which are neither in P nor NP e.g. the halting problem, which are undecidable.*

2. **Short Answer (5 points each) Where appropriate, circle your final answer.**

(a) Solve the following recurrence using the master method: $T(n) = 2T(n/2) + 1$

Solution: $f(n) = 1$ and $af(n/b) = 2$ so $af(n/b) \geq cf(n)$ for $c > 1$. Thus the leaf nodes dominate. Thus $T(n) = \Theta(n)$.

(b) Solve the following recurrence using annihilators: $T(n) = 4T(n-1) - 4T(n-2)$. Give the solution in general form i.e. do not solve for the constants

Solution: The annihilator is $\boldsymbol{L}^2 - 4\boldsymbol{L} + 4$ which factors to $(\boldsymbol{L} - 2)(\boldsymbol{L} - 2)$. This implies that the general solution is $T(n) = (c_1 + c_2 n)2^n$

(c) Assume a data structure supports an operation *foo* such that a sequence of n calls to *foo* takes $O(n^2)$ time in the worst case. Answer the following: 1) What is the amortized cost of a *foo* operation and 2) What is the highest possible cost of a single call to *foo* and 3) What is the lowest possible cost of a single call to *foo*? Give your answers in Θ notation.

Solution: Amortized cost is $\Theta(n)$. Single cost can be as high as $\Theta(n^2)$ and as low as $\Theta(1)$.

(d) Prove that $10n + 10 = O(n^2)$.

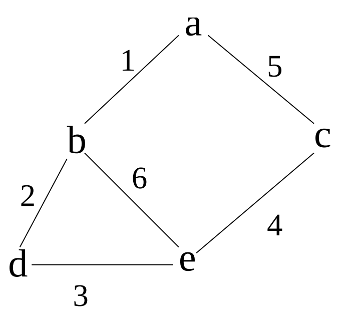
Solution: Goal: Give positive constants c and n_0 such that $10n \leq cn^2$ for all $n \geq n_0$. The inequality we want then is:

$$\begin{aligned} 10n + 10 &\leq cn^2 \\ 10/n + 10/n^2 &\leq c \\ c &\geq 10/n + 10/n^2 \end{aligned}$$

The right hand side of this inequality is decreasing as n grows large. Thus if we choose $n_0 = 1$ and $c = 20$, it satisfies the inequality for all $n \geq n_0$. In other words, for $c = 20$ and $n_0 = 1$, it's the case that $10n + 10 \leq cn^2$ for all $n \geq n_0$.

3. **Spanning Trees**

(a) Give the minimum spanning tree and the shortest path tree rooted at the vertex a for the following graph. Make sure that you label which tree is the MST and which is the shortest path tree rooted at a.



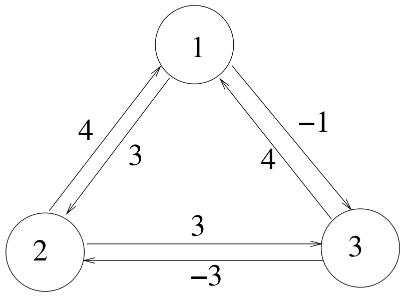
Solution: The MST is the following edges $\{(a,b),(b,d),(d,e),(e,c)\}$. *The shortest path tree rooted at a is the following edges:* $\{(a,b),(b,d),(d,e),(a,c)\}$

(b) Assume you are given an undirected, connected graph G with n nodes and at least n edges. Give an algorithm to return a cycle in G. (Your algorithm should return a minimal set of edges in G that form a cycle.)

Solution: Let T *be a spanning tree of* G *e.g. the BFS (or DFS) tree. Let* (x,y) *be an edge in* G *which is not in* T. *The cycle returned is the edge* (x,y) *plus all edges in* T *which are on the path between* x *and* y.

4. **Graph Theory**

Recall that in the Floyd-Warshall algorithm $dist(u, v, r)$ is defined to be the shortest path from u to v where all intermediate vertices (if any) are numbered r or less. For the following graph, fill in the distance arrays computed by Floyd-Warshall for all values of r. *In the distance arrays, let the row be the vertex the path starts at and let the column be the vertex the path ends at.*



$r = 0$

	1	2	3
1			
2			
3			

$r = 1$

	1	2	3
1			
2			
3			

$r = 2$

	1	2	3
1			
2			
3			

$r = 3$

	1	2	3
1			
2			
3			

Solution:

$r = 0$

	1	2	3
1	0	3	-1
2	4	0	3
3	4	-3	0

$r = 1$

	1	2	3
1	0	3	-1
2	4	0	3
3	4	-3	0

$r = 2$

	1	2	3
1	0	3	-1
2	4	0	3
3	1	-3	0

$r = 3$

	1	2	3
1	0	-4	-1
2	4	0	3
3	1	-3	0

5. **NP-Hardness and Approximation Algorithms**

Your first job out of college is with the rapidly growing consulting company Downsize.com which specializes in "downsizing solutions". When Downsize.com consults for an organization, it is given a list of all employees of that organization and a list of all pairs of these employees that have personality conflicts. Downsize.com then seeks to find the *smallest* set of employees that must be fired in order to remove all personality conflicts in the organization. This problem is called the *Downsize* problem.

On your first day at work, your boss gives you the assignment of designing an efficient algorithm to solve the Downsize problem. In particular, he wants your algorithm to be given as input a list of employees and a list of personality conflicts and to output the minimum number of employees that must be fired to remove all personality conflicts.

(a) Show that you can not be expected to design an efficient algorithm to solve the Downsize problem. In particular, show that the problem is NP-Hard by doing a reduction from one of the following NP-Hard problems: 3-SAT, Clique, Vertex Cover, or TSP (the especially brave student can also try a reduction from Tetris :)

Solution: To show this problem is NP-Hard, we can reduce from Vertex Cover. We assume that we have a polynomial time algorithm for the Downsize problem and we want to show that we can use this algorithm to solve vertex cover. In vertex cover, we are given a graph G and an integer k and we should return yes iff G has a vertex cover of size k. Given a graph G and an integer k, we convert this to a Downsize problem as follows. For each node, x, in G we enter an employee, "Employee x", on the employee list. For each edge (x, y) in G we enter the conflict "Employee x and Employee y" on the conflict list. We then feed these lists into the algorithm for Downsize and get back as output the minimum number of employees which must be fired to remove all conflicts. If this number is no more than k, then we output "yes G has a vertex cover of size k" otherwise we output "no G does no have a vertex cover of size k". The reason this works is that the nodes of G associated with the set of fired employees must form a vertex cover in G since each edge in G is associated with a conflict that involves at least one of the fired employees.

(b) Your boss accepts the fact that the Downsize problem is NP-Hard but is still unwilling to give you the day off. He now wants you to design an approximation algorithm for the Downsize problem. Can you do this? Explain.

Solution: Yes you can design a 2-approximation algorithm for this problem. You first transform the problem to a graph G as discussed in the previous soln and you then use the approximation algorithm discussed in class for finding a vertex cover which is no more than twice as big as the optimal vertex cover. The employees associated with the nodes in this vertex cover are the ones that get the axe.