Answer every question, whether or not you think your answer is complete or correct. This is a closed book/closed notes examination. Ensure that no papers or electronic devices can be seen by any class member.

Part of the exercise of doing the questions is understanding the questions. Read each carefully so as not to waste time.

All questions are equally weighted (but not necessarily equally difficult.)

<table>
<thead>
<tr>
<th>Question</th>
<th>Mark</th>
<th>Out Of</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>

1. Clarifications

Read the sample solutions carefully before you ask questions. If, after reading them, you need clarification, talk with the TA or instructor.

If you want to question a grade on a portion of your paper, proceed as follows. If you think it is just an addition error, see the instructor. If you think it has not been graded correctly, write a clear, short explanation of what you think has not been graded correctly. Turn in the explanation together with the original of the entire test, and the entire test will be regraded. This may result in you receiving more, less, or the same number of marks currently assigned. Please remember that 101 papers have been graded, using the same criteria throughout. If you did not get as many points as you think you should, probably everyone feels that way. Consequently, it is very unusual for me to change a grade. *Any request for reevaluation must be received by the start of class on 10/21/15.*
2. General Comments

If you are trying to prove that a method always works, it is a waste of your time and mine to give examples where it works. One example says nothing in such cases. And two or three examples still say nothing. If you are making a general statement, it requires a general proof.

On the other hand, if you are showing that an algorithm does not work, one example suffices to do this. But you must be sure that your example shows what you intend, and that you are applying the algorithm correctly, for the example to work.

Be very careful about circular reasoning in which you assume what you are trying to prove. If what you are proving is the correctness of Dijkstra’s algorithm, for example, you cannot use facts that were derived as a consequence of the correctness of the algorithm.

Finally, make sure you are proving what you need to prove. Showing that a maximum weight spanning tree of \( G' \) yields a spanning forest of the same weight in \( G \), for example, does not show that the forest has maximum weight.

3. Question 1.

On Homework # 1 we saw that the Gale-Shapley propose-and-reject algorithm for \( n \) TAs and \( n \) courses can require \( \Theta(n^2) \) offers.

(1) Show that \( O(n^2) \) offers are sufficient. In the GS algorithm, every course makes an offer to each TA at most once, so there can be no more than \( n^2 \) offers. The issue is with correctness. If some TA \( T \) was not matched then \( T \) was never offered a course, and there is a course \( C \) that has no TA. But \( C \) continues to make offers until it has a final assignment, so \( C \) must have made an offer to \( T \), which cannot be. Now suppose to the contrary that all courses and TAs are matched but \( C - T \) is an unstable pair. If \( C \) was not offered to \( T \) then \( C \) prefers its current TA to \( T \) so \( C - T \) is not unstable. On the other hand, if \( C \) was offered to \( T \) and declined at any point, then \( T \) prefers its current course to \( C \) so \( C - T \) is not unstable. In either case, we have a contradiction so there can be no unstable pairs.

Only a few students had proved that the offers made in GS suffice to produce a stable matching, instead most assumed that GS is correct. I had allocated 2 of the points to showing correctness, but because almost no one read the question that way, I adjusted the grades. So no marks were assigned to showing correctness.

(2) Show that \( \Omega(n^2) \) offers are necessary in the worst case. (One example does not suffice!) This is from the homework 1 solutions.
GS with courses making offers would assign course \( i \) to TA \( i - 1 \) for \( 2 \leq i \leq n \), and course 1 to TA \( n \), which is stable. It also has the remarkable property that if offers are made by courses in strict rotation for courses 1, \ldots, \( n \) repeatedly, \( n^2 - n + 1 \) offers will be made.

Another easier way to show \( \Omega(n^2) \) is to have every TA prefer the courses in decreasing order \( C_n, C_{n-1}, \ldots, C_2, C_1 \) and every course to prefer TAs in decreasing order \( T_1, T_2, \ldots, T_{n-1}, T_n \). Make offers from unassigned courses in cyclic order starting with \( C_1 \). In the first \( n \) offers, \( C_1 \) is offered to \( T_1 \), which takes the offer every time. So after \( n \) offers, \( C_n \) is assigned to \( T_1 \), its first choice. This will remain stable, so we have an identical problem remaining on \( n - 1 \) courses and \( n - 1 \) TAs. It follows that the number of offers is \( \sum_{i=1}^{n} i = \binom{n+1}{2} \), which is \( \Omega(n^2) \).

Many students tried to tell me that \( n^2 \) offers are made in the worst case. This is not true, and here is the proof. No TA declines an offer unless they have a more preferred course. So the algorithm must terminate when each TA has received an offer. Hence the maximum number of offers cannot exceed \( n^2 - n + 1 \).

4. Question 2.

Recall from Homework #1 that I have a large amount of work that to get done this semester. Suppose that I have \( n \) tasks, each with a start date and a due date; each task gets me a reward of 10 points if I complete it, and I lose a point if I do not. Once I decide to do a task, I must complete it without interruption. I can do six tasks at a time. I need to get as many points as I can.

(1) There is an algorithm to select tasks under the assumption that I can do only one task at a time. A greedy algorithm repeatedly uses this “one-task” algorithm six times to select six sets of tasks (each time deleting those tasks already selected from the ones available). Then I take the union of the six sets to get the solution. Show that this need not be optimal. This is from the homework 1 solutions, but many solutions are possible.
Recall that the “one-task” algorithm sorts tasks in nondecreasing order of due date and selects greedily. Then the six-times method does not get the optimal solution on the following data:

To avoid drawing a picture, here’s a different, general solution. Suppose that I can do \( t \) tasks at once. I will form a set of \( 2t \) tasks named \( 1, \ldots, 2t \). For \( 1 \leq i \leq t \), task \( i \) has start date 0 and due date \( i \). For \( 1 \leq i \leq t \), task \( t + i \) has start date \( i \) and due date \( 5t - i \). It is possible to do all \( 2t \) tasks on \( t \) processors by assigning tasks \( \{i, t + i\} \) to processor \( i \) for \( 1 \leq i \leq t \). What does the one–set–at–a–time method do? It first picks tasks \( \{1, 2t\} \), but now the \( t \) tasks \( \{2, \ldots, t, t + 1\} \) all remain and all have to be done on day 1, so we cannot do the rest with only \( t - 1 \) processors.

(2) Instead I sort the tasks in nondecreasing order by due date and process them in this order. For each I make a decision whether to do the task or not, and I do not change a decision once it is made. As each task \( T \) is considered, if I have so far chosen fewer than six tasks that are active on \( T \)’s start date, I choose \( T \); otherwise I do not choose \( T \). Does this method get me the maximum number of points? Answer yes or no, and explain.

The answer is yes. The point structure is such that we want to maximize the number of tasks completed. Suppose to the contrary that the subset \( S = (S_1, \ldots, S_r) \) is selected by the algorithm, sorted in nondecreasing order by due date, and that \( S^* = (T_1, \ldots, T_m) \) is an optimal solution with \( m > \ell \), also in nondecreasing order by due date, for which \( S_i = T_i \) for \( 1 \leq i < r \) and \( r \) is as large as possible.

If \( r = \ell + 1 \), then the algorithm would not stop at \( S_\ell \) but would also include \( T_{\ell+1} \) because it is feasible (because it appears in \( S^* \) at this point). So \( r \leq \ell \); the algorithm selects \( S_r \) while the optimal solution selects \( T_r \). If the due date of \( S_r \) is no later than that of \( T_r \), replacing \( T_r \) by \( S_r \) in \( S^* \) yields an optimal selection with larger \( r \) than \( S^* \) has, a contradiction. Therefore the due date of \( S_r \) is later than that of \( T_r \). But because \( T_r \) is feasible at this point, the
algorithm cannot have rejected it, again a contradiction. Hence the selection found must be optimal.

5. Question 3.

On Homework #2 we looked for a maximum weight spanning forest of a given undirected graph \( G \) in which each edge has an integer weight. \( G \) can be disconnected, so some effort was needed to adapt the minimum spanning tree algorithms for this problem.  

I have an idea! Given graph \( G = (V,E) \), I form a new graph \( G' \). To do this, add a new vertex \( \nu \) and edges \( \{\nu,x\} : x \in V \}. \) Each new edge has weight 0. To (try to) find a maximum weight spanning forest \( F \) of \( G \), I find a maximum weight spanning tree \( T \) of \( G' \), and delete vertex \( \nu \) and all edges containing \( \nu \) from \( T \) to get \( F \).  

**Prove or disprove:** \( F \) is a maximum weight spanning forest of \( G \) with weight equal to the weight of \( T \).  

I will **prove** it. Adding or removing edges incident at \( \nu \) does not change the weight of a subgraph, because all such edges have weight 0. Using my method I find a maximum weight spanning tree \( T \) of \( G' \) and hence a spanning forest \( F \) of \( G \) having the same weight. Most students only proved what we have done so far, and then asserted without a proper proof that \( F \) has maximum weight. But more is needed to do that. Suppose to the contrary there is a spanning forest \( F^* \) of \( G \) that has larger weight than \( F \). In each component of \( F^* \), choose a vertex \( x \) and add the edge \( \{x,\nu\} \). This forms a spanning tree \( T^* \) of \( G' \), because no cycle is formed, and the added edges connect the graph. Now \( T^* \) has the same weight as \( F^* \) whose weight is larger than that of \( F \), which has the same weight as \( T \). But \( T \) has maximum weight of any spanning tree in \( G' \), so the weight of \( T \) is at least that of \( T^* \), which is a contradiction. Hence \( F \) is a maximum weight spanning forest of \( G \).  

This does not rely on any knowledge of the modified Kruskal algorithm or the proof of its correctness.


On Homework #2, we found paths from servers to clients in a directed graph in which every arc has a known (nonnegative) delay. An apparently easier case is when there is only one server, so assume that \( s \) is the only server. We have informed one of the clients, \( c \), that we cannot find a path from \( s \) to \( c \) that has total delay at most \( \delta \). And now \( c \) is very angry, and has demanded an explanation from management!  

Devise an efficient method to produce a **short, complete, and correct** proof that no path from \( s \) to \( c \) with delay at most \( \delta \) exists. Explain carefully.  

**You cannot just say that Dijkstra’s algorithm works. Imagine the following conversation:**
Client: Why can’t you find me an $s,c$-path with delay at most $\delta$?

Algorithm Designer: Dijkstra’s algorithm says there isn’t one.

Client: So? What is Dijkstra?

Algorithm Designer: He’s a who not a what. Dijkstra’s algorithm is correct. Every textbook says so. It is obvious! You could read about it to see.

Client: Or I could hire someone who can explain why I cannot have my path, which I now plan to do.

You also cannot just prove that Dijkstra’s algorithm works. You would also need to show an execution trace on the directed graph to show that it was executed correctly. After all, the client wants a short, complete proof.

If there is no path at all from $s$ to $c$, I show the client a cut $(S, S')$ with $s \in S$, $c \in S'$, and no directed edges from a vertex in $S$ to one in $S'$.

If there is a path but it is too long, I show the client a shortest path tree $T_s$ for the directed graph rooted at $s$. I label each vertex $x$ with its distance $d_x$ from $s$ in the shortest path tree. The client can see that I have computed these distances correctly because there is only one path to each vertex in $T_s$. I show the client, for each directed edge $(x, y)$ not in $T_s$ in turn, that $d_y - d_x$ is at most the delay of the directed edge $(x, y)$. I argue that if $(x, y)$ were on a shortest path, it would be on a shortest $s,y$-path ending in $(x, y)$. But this path is no shorter than the one in $T_s$, so $(x, y)$ cannot be needed to make a shortest path. So I can delete it, and the client agrees. But in this way all directed edges not in $T_s$ are deleted. And the client has already agreed that the distance to $c$ in $T_s$ is exactly $d_c$ – remember, only one path has to be checked. Finally I ask the client to agree that $d_c > \delta$, and we are done.

And I proclaim: Quod erat demonstrandum. And the client says: What?

Notice that I never said the word Dijkstra in my explanation. Indeed it does not matter what algorithm was used to find the answer. That is a red herring.