NAME: _____

CMPSCI 611
Advanced Algorithms
Midterm Exam Fall 2015

A. McGregor 21 October 2015

DIRECTIONS:

- Do not turn over the page until you are told to do so.
- This is a *closed book exam.* No communicating with other students, or looking at notes, or using electronic devices. You may ask the professor or TA to clarify the meaning of a question but do so in a way that causes minimal disruption.
- If you finish early, you may leave early but do so as quietly as possible. The exam script should be given to the professor.
- There are six questions. All carry the same number of marks but some questions may be easier than others. Don't spend too long on a problem if you're stuck – you may find that there are other easier questions.
- The front and back of the pages can be used for solutions. There are also a blank page at the end that can be used. If you are using these pages, clearly indicate which question you're answering. Further paper can be requested if required. However, the best answers are those that are clear and concise (and of course correct).

1	/10
2	/10
3	/8+2
4	/10
5	/10
6	/10
Total	/58+2

Question 1 (The True or False Question). For each of the following statements, say whether they are TRUE or FALSE. No justification is required.

1. If a subset system satisfies the exchange property, it also satisfies the cardinality property.
   Answer: True.

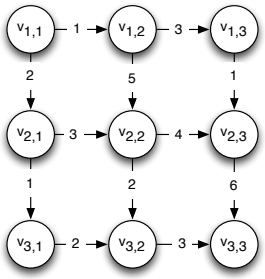
2. Given a connected, undirected graph, let D be the diameter and let M be the size of the maximum matching. Then $M \geq D/2$.
   Answer: True. If the diameter is D there must be a path of length D in the graph that doesn't repeat any nodes. Taking alternating edges on this path gives a matching.

3. Given a matroid $(E, \mathcal{I})$ there is at most one maximal solution $i \in \mathcal{I}$.
   Answer: False. For example, if $E = \{a, b, c\}$ and $\mathcal{I} = \{\{\}, \{a\}, \{b\}, \{c\}\}$ then $\{a\}, \{b\}, \{c\}$ are each maximal.

4. In a bipartite graph there are no cycles with an odd number of edges.
   Answer: True.

5. Given a undirected weighted graph G, let T be a minimum spanning tree. Then the length of the shortest path between u and v in G equals the length of the shortest path between u and v in T.
   Answer: False. For example, consider the complete graph on three nodes with edge weights 2, 2, and 3.

Question 2 (Grid Graphs). In this question, we develop an alternative shortest path algorithm for a specific type of graph. In a *grid graph* there are $n = k^2$ nodes labelled $v_{i,j}$ where i and j range between 1 and k. For each $i < k$ there is a "down" edge of length $d_{i,j}$ from $v_{i,j}$ to $v_{i+1,j}$. For each $j < k$ there is an "across" edge of length $a_{i,j}$ from $v_{i,j}$ to $v_{i,j+1}$. For example, a possible graph for $n = 9$ is



and in this graph, for example, the across edge from $v_{2,1}$ to $v_{2,2}$ has length $a_{2,1} = 3$.

1. What is the length of the shortest path from $v_{1,1}$ to $v_{3,3}$ in the above example?
   Answer: The length of the shortest path is $2 + 1 + 2 + 3 = 8$.

2. Let $d[i, j]$ be the length of the shortest path from $v_{i,j}$ to get$v_{k,k}$. If $i > k$ or $j > k$, let $d[i, j] = \infty$. Give a formula for $d[i, j]$ in terms of $d[i + 1, j]$ and $d[i, j + 1]$. Justify your answer.

Answer:

$$d[i,j] = \begin{cases} 0 & \text{if } i = k, j = k \\ a_{i,j} + d[i, j+1] & \text{if } i = k, j < k \\ d_{i,j} + d[i+1, j] & \text{if } i < k, j = k \\ \min(a_{i,j} + d[i, j+1], d_{i,j} + d[i+1, j]) & \text{if } i < k, j < l \end{cases}$$

because the shortest path from $v_{i,j}$ to $v_{k,k}$ is either via $v_{i,j+1}$, and will have length $a_{i,j} + d[i, j+1]$, or via $v_{i+1,k}$, and will have length $d_{i,j} + d[i+1, j]$. If both nodes exist $d[i, j]$ is the minimum of these two quantities. If only one of these nodes exist, $d[i, j]$ equals the corresponding distance. If neither exists $i = k, j = k$ and hence $d[i, j] = 0$.

3. Design an algorithm that computes $d[1, 1]$. Prove the running time and correctness.

   Answer:

   (a) Let $d[k, k] = 0$

   (b) For $r = 2k - 1, \ldots, 2$

      i. For i, j such that $i + j = r$, let $d[i, j]$ be updated using the formula above.

   (c) Return $d[1, 1]$.

   The correctness of the formula for $d[i, j]$ is argued above and by computing $d[i, j]$ in decreasing order of $i + j$ we ensure $d[i+1, j]$ and $d[i, j+1]$ have already been computed when there values are required to set $d[i, j]$. The running time is $O(n)$ since there are n values $d[i, j]$ to compute and each can be computed in $O(1)$ time given the previously computed values.

4. Any path from $v_{1,1}$ to $v_{k,k}$ will use exactly $k - 1$ down edges and $k - 1$ across edges. Suppose we add the constraint that we only allow a path where the sub-path from $v_{1,1}$ to any intermediate node on the path uses at least as many across edges as down edges. How would you modify your algorithm to find the shortest such path?

   Answer: For $i > j$, we change the formula to $d[i, j] = \infty$. This ensures that $d[i, j]$ for $i \leq j$ always corresponds to path that does not visit a node $v_{k,\ell}$ where $k > \ell$.

Question 3 (Planning Your Working Vacation). Suppose you are a ski instructor and that you plan to work at a European ski resort for some part of the n-day ski season next year. Unfortunately, there isn't enough work for you to be paid every day and you need to cover your own expenses. Fortunately, you know in advance that if you are at the resort on the ith day of the season, you'll make p_i euros where p_i could be negative (if expenses are more than earnings) or positive (if expenses are less than earnings). To maximize your earning you should choose carefully which day you arrive and which day you leave; the days you work should be consecutive and you don't need to work all season. For example, if $n = 8$ and

$$p_1 = -9\ ,\ p_2 = 10\ ,\ p_3 = -8\ ,\ p_4 = 10\ ,\ p_5 = 5\ ,\ p_6 = -4\ ,\ p_7 = -2\ ,\ p_8 = 5$$

then if you worked from day 2 to day 5, you would earn $10 - 8 + 10 + 5 = 17$ euros in total.

1. Give a divide and conquer algorithm for this problem that runs in $O(n \log n)$ time. You may assume n is a power of two. If your algorithm takes longer or doesn't use divide and conquer you might get partial credit.

Answer: Let a and b be the maximum possible earning from the first and second half of the season respectively. The c be the maximum possible earning that includes both days $n/2$ and $n/2+1$. Then the maximum possible earning is $\max(a, b, c)$. We compute a and b recursively where the base case when considering single day i is $\max(p_i, 0)$. To compute c, let c_1 be the earning from an interval that ends with the $n/2$th day and let c_2 be maximum earning from an interval that starts with the $(n/2+1)$th day. Let $c = c_1 + c_2$. We compute c_2 as follows

(a) Let $total = p_{n/2+1}$ and $c_2 = p_{n/2+1}$.
(b) For $i = n/2+2, \ldots, n$, let $total = total + p_i$ and $c_2 = \max(c_2, total)$.

and c_1 is computed analogously.

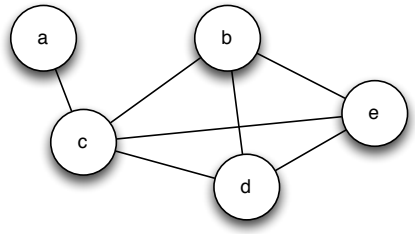
2. Justify why your algorithm is correct and analyze the running time.
   Answer: Let $T(n)$ be the time to compute $\max(a, b, c)$. Then $T(n) \leq 2T(n/2) + O(n)$ since computing a and b takes $2T(n/2)$ time and computing c takes $O(n)$ time. Hence, by the Master Theorem, $T(n) = O(n \log n)$. Correctness: The fact that the max possible earning is $\max(a, b, c)$ follows since every possible interval is either a subset of the first or second half of the season or includes day $n/2$ and $n/2+1$. $c = c_1 + c_2$ because the best interval that includes day $n/2$ and $n/2+1$ and be decomposed into the best interval that ends on day $n/2$ and the best interval that starts on day $n/2+1$.

3. **Extra Credit:** Design an $O(n)$ time algorithm for this problem.
   Answer: Let $e(t)$ be the earnings of the best interval that is either empty or ends on the tth day. Then, $e(t) = \max(e(t-1) + p_t, 0)$ since the best corresponding interval is either empty or includes the interval corresponding to $e(t-1)$ and $\{t\}$. The best interval is then $s = \max_{1 \leq i \leq n} e(i)$. We compute s as follows:

   (a) Let $e = 0$ and $s = 0$.
   (b) For $i = 1, \ldots, n$, let $e = \max(e + p_i, 0)$ and $s = \max(s, e)$.

   At the end of iteration t, $e = e(t)$ and $s = \max_{1 \leq i \leq t} e(t)$. Hence, at the end of the algorithm $s = \max_{1 \leq i \leq n} e(i)$ as required. The running time is $O(n)$ since there are n iterations and each takes $O(1)$ time.

Question 4 (Triangles and Four Cycles). Given an undirected graph G on n nodes, a triangle in the graph is a set of three different nodes $\{u, v, w\}$ such that there are edges between each of these nodes. For example, in the following graph $\{b, c, d\}$ is a triangle.



1. List the other triangles in the above graph.
   Answer: The other triangles are $\{b, d, e\}$, $\{b, c, e\}$, and $\{c, d, e\}$.

2. Design an $O(n^\omega)$ time algorithm that returns the number of triangles in a graph where $O(n^\omega)$ is the running time of an algorithm that multiplies two $n \times n$ matrices. Recall $\omega < 2.38$. Analyze the running time and prove your algorithm is correct. **Hint:** If M is the adjacency matrix of G, first consider computing $M \times M$.

   Answer: First note that

$$M^2[i,j] = \sum_k M[i,k]M[k,j] = (\#\text{number of length two paths from } i \text{ to } j)$$

   and therefore the number of triangles that include the nodes i and j is $M^2[i,j]M[i,j]$ since there are $M^2[i,j]$ triangles if there is an edge between i and j and 0 triangles otherwise. Hence the total number of triangles is

$$\frac{\sum_{i<j} M^2[i,j]M[i,j]}{3}$$

   where the division by three is because there are three ways to pick two nodes in the triangle. The time to compute M^2 is $O(n^\omega)$ and computing the sum takes $O(n^2)$ time. (Another way to solve the problem is to compute M^3; the some of the diagonal elements equals 6 times the total number of triangles.)

3. A four-cycle is a set of four different nodes $\{u, v, w, x\}$ such that there is an edge between u and v, v and w, w and x, and x and u (there may also be additional edges between the nodes). For example, in the example earlier, $\{b, c, d, e\}$ is a four-cycle. Design an $O(n^\omega)$ time algorithm that returns the number of four-cycles in a graph.

   Answer: As above $M^2[i,j]$ is the number of length two paths between i and j. Hence the number of four cycles where i and j are not adjacent in the cycle is $\binom{M^2[i,j]}{2}$. Hence, the number of four cycles is

$$\frac{\sum_{i<j} \binom{M^2[i,j]}{2}}{2}$$

   where the division by two is to adjust for double counting because there are two ways to pick two non-adjacent nodes on a four cycle. The time to compute this quantity is $O(n^\omega + n^2) = O(n^\omega)$ as above.

Question 5 (Planning A Party). Alice wants to throw a party and is deciding whom to call. She has n (which is at least 11) people to choose from, and she has made up a list of which pairs of these people know each other. She wants to invite as many people as possible subject to the following two constraints:

1. Every person invited should know at least five other people that are invited.
2. Every person invited should not know at least five other people that are invited.

Design an efficient algorithm for maximizing the number of people she can invite. Remember to analyze the running time and correctness. **Hint:** Maximizing the number of invitees is the same as minimizing the number of people Alice doesn't invite. Obviously Alice might not be able to invite everyone. For example, if one of the n people knows less than five people out of the n potential invitees then the first constraint can never be satisfied for that person.

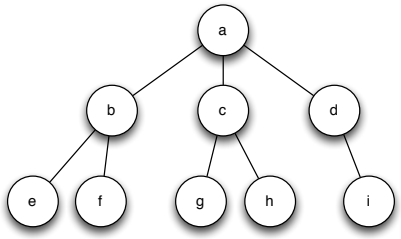
Answer: Label the people that could be invited $1, \ldots n$. For a subset S of $\{1, 2, \ldots, n\}$, define $K_i(S)$ to be the number of people in S that the ith person knows and define $D_i(S)$ be the number of people in S that the ith person doesn't know. Then the algorithm is:

- Let $P = \{1, 2, \ldots, n\}$ be the set of potential invitees.
- While there exists $i \in P$ such that $K_i(P) < 5$ or $S_i(P) < 5$: $P \leftarrow P \setminus \{i\}$.
- Return P.

Correctness: For all people in the final set P (which could be empty), there are at least 5 people they don't know in P and 5 people they do know in P. Hence, inviting all people in P satisfies Alice's constraints. Suppose the optimal set of invitees is P_{opt}. We use induction on the number of people we remove to prove P always contains P_{opt}. Initially, when we have removed 0 people, P contains P_{opt}. Assume after removing k people, P contains P_{opt}. If we remove a $(k+1)$th person then this person can not have been in P_{opt} since if $K_i(P) < 5$ or $D_i(P) < 5$ then $K_i(P_{opt}) < 5$ or $D_i(P_{opt}) < 5$. Hence P_{opt} is still contained in P.

The running time is $O(n^2)$. There are $O(n)$ iterations and in each we need to scan through the $|P| \leq n$ remaining possible invitees to find if there is an i such that $K_i(P) < 5$ or $S_i(P) < 5$. If so, we need to update $|P| - 1$ values of K_j and S_j.

Question 6 (Independent Sets in Trees). Given a graph G, we say a set of nodes S is independent if there are no edges in G between any two nodes in S. In this question we consider the problem of computing independent sets when G is a tree. For example, in the following tree,



the set $S = \{b, g, h, d\}$ is an independent set. Suppose every node v has a weight $w(v)$. Design and analyze an algorithm for finding an independent set S in a tree such that $\sum_{v \in S} w(v)$ is maximized. Remember to analyze the running time and prove correctness. **Hint:** Consider a dynamic problem where there are n subproblems and each corresponds to finding the maximum weight independent set restricted to a node and its decedents. For example, in the subproblem for node c we are interested in finding the maximum weight independent subgraph amongst the nodes $\{c, g, h\}$.

Answer: Let $T(u)$ be the subtree rooted at u and let $S(u)$ be the maximum weight independent set of $T(u)$. Then

$$S(u) = \max \left(w(u) + \sum_{w \text{ is a grandchild of } u} S(w) \, , \sum_{v \text{ is a child of } u} S(v) \right)$$

because either u is in the maximum weight independent set of $T(u)$ or it isn't. If it is, then none of the children of u are also in the set and so the largest weight independent set also includes the largest weight independent sets of $T(w)$ for all w that are grandchildren of u. If it isn't, the largest weight independent set also includes the largest weight independent sets amongst v and its descendants for all v that are children of u.

For each node u, let $height(u)$ be the maximum length of the shortest path from u to a descendent of u. The algorithm is as follows:

1. For $h = 0, 1, 2 \ldots$
   (a) Let $S(u) = \max\left(w(u) + \sum_{w \text{ is a grandchild of } u} S(w)\ ,\ \sum_{v \text{ is a child of } u} S(v)\right)$
2. Return $S(r)$ where r is the root of the tree.

Since we compute $S(\cdot)$ in order of increasing height, we will have computed $S(\cdot)$ for the children and grandchildren of u before we need these values to compute $S(u)$. The running time is $O(n)$ since there are n values of $S(\cdot)$ to compute and each value is needed to compute 2 other values of $S(\cdot)$ (every node has at most one parent and one grand parent).