CISC-365*  
Test #1  
October 8, 2015  

Student Number (Required) ______________________

Name (Optional)________________________________

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but will not be reconsidered after they have been marked.

The test will be marked out of 50.

<table>
<thead>
<tr>
<th>Question 1</th>
<th>/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 2</td>
<td>/15</td>
</tr>
<tr>
<td>Question 3</td>
<td>/15</td>
</tr>
</tbody>
</table>

| TOTAL           | /50 |
General marking philosophy: a student who gives enough of an answer to show they understood what they were supposed to do, even if they couldn’t do it (or made lots of errors while doing it) should get at least 50% on that question.

Full marks should be given if a solution is sound and not missing anything important.

Feel free to give marks like 14.5/15 to a solution that is correct but contains a minor error.

A student should only get 0 on a question if they made no attempt to answer it at all.
**Question 1 (20 marks)**

In this question you are required to modify Dijkstra’s Algorithm to solve a new problem.

Here is Dijkstra’s Algorithm, much as presented in class. The input is a graph G consisting of vertices and edges with numeric weights on the edges, and a start vertex A. We use a queue Q to keep track of vertices awaiting processing, and indexed structures (such as arrays) P, D, and M to keep track of Predecessors, Distances, and Marked vertices (vertices are marked when we remove them from Q).

1. **DA(G,A):**
2. \[D[A] = 0\]
3. for all vertices \(x\) other than A:
4. \[D[x] = \text{infinity}\]
5. \[M[x] = \text{"no"}\]
6. add A to Q
7. while Q non-empty:
8. let \(x\) be the vertex in Q that has the smallest \(D\) value
9. remove \(x\) from Q
10. \[M[x] = \text{"yes"}\]
11. for each \(y\) that is a neighbour of \(x\) with \(M[y] == \text{"no"}\):
12. if \(D[x] + \text{weight}(x,y) < D[y]\):
13. \[D[y] = D[x] + \text{weight}(x,y)\]
14. \[P[y] = x\]
15. if \(y\) is not in Q:
16. add \(y\) to Q
17. return \(P\) #\(P\) contains the path information

Now consider this problem: Suppose we have a graph in which the vertices represent banks and the edges represent money-transfer agreements. So an edge between vertex \(X\) and vertex \(Y\) means that Bank \(X\) is willing to transfer money to Bank \(Y\). The banks are evil and skim off a percentage of each transfer – the percentage that is actually transferred on each edge is given as the edge-weight. If money is transferred along a multi-step path, the actual percent of the original amount that goes through to the end of the path is the **product** of the percentages on the edges in the path.

A customer who has her money in bank \(A\) wants to know the best paths along which to transfer her money to other banks – that is, the highest-value paths from \(A\) to all other vertices.

The next page shows a small example to demonstrate the problem.
In this graph there are two paths from A to B. Going through C, the first step reduces the amount to 80% and the second step reduces that amount to 50% of the already reduced value – so the value at the end is just 40% of the original value (80% * 50%). Going through D, the first step reduces the amount to 70% and the second step drops it to 90% of that, giving a final value of 63% of the original. In this example, going through D is better because 63% > 40%.

Modify Dijkstra's Algorithm to solve this problem. Please indicate your changes using the line numbers given in the statement of the algorithm shown above, such as “Change Line 7 to ...”

You only need to change 5 lines.
Solution:

- Change Line 2 to “$D[A] = 1$”
- Change Line 4 to “$D[x] = 0$”
- Change Line 8 to “let $x$ be the vertex in $Q$ that has the largest $D$ value”
- Change Line 12 to “if $D[x] \times \text{weight}(x,y)/100 > D[y]$:
- Change Line 13 to “$D[y] = D[x] \times \text{weight}(x,y)/100$”

Marking:

Solutions that show an understanding of how Dijkstra’s algorithm works should get at least 10/20, even if they cannot solve this problem. This over-rides specific marking instructions shown below.

Notes: Line 4 could also be “$D[x] = -1$” or “$D[x] = -\text{infinity}$” etc

- Solutions that neglect the “$/100$” in lines 12 and 13 should lose 2 marks
- Generally, errors in the changes that affect the correctness of the algorithm should lose 1 or 2 marks
- For each completely missed change, deduct 3 marks
QUESTION 2 (15 marks)

Suppose X and Y are both problems in the class NP. X is known to be NP-Complete, but the complexity of Y is unknown.

Each part of this question provides you with more information, and asks you to explain the significance (if any) of the new information. Each part of this question is independent of all the other parts.

(a) Suppose it is proved that X reduces to Y. What (if anything) does this tell us?

Solution: This would prove that Y is NP-Complete

(b) Suppose it is proved that Y reduces to X. What (if anything) does this tell us?

Solution: This has no significance. Since X is known to be NP-Complete, we already know that Y reduces to X

(c) Suppose an O(n^{16}) algorithm is found for X. What (if anything) does this tell us?

Solution: This would prove P = NP

Marking:
5 marks for each answer. Give part marks for incorrect answers if they show understanding of what the terms “NP”, “NP-Complete” and “reduces” mean
Question 3 (15 marks)

Given a set of n integers, already sorted, give a **Divide & Conquer** algorithm that will find the two values that are closest together in value.

For example, in the set {1, 3, 4, 7, 12, 14, 18, 21} the closest values are 3 and 4.

Divide & Conquer is not necessarily the best way to solve this problem. The purpose of this question is to give you the opportunity to demonstrate familiarity with the Divide & Conquer method.

**Solution:**

_Overview: Split the set into a left half L and a right half R. Solve the problem recursively on L and R. The solution is either_

- the closest pair of values in L
- the closest pair of values in R
- the largest value in L and the smallest value in R
**Pseudocode:**

CV(S):

# S is a sorted set of integers in an indexed structure (eg an array)

n = length(S)

if n <= 3:
    solve problem directly, and assign
    x1, x2: the two closest values
    d: the difference between x1 and x2
    return (x1, x2, d)

else:
    L = S[1..n/2]
        # note: I prefer 1-based indexing
    R = S[n/2+1..n]
    LS = CV(L)
    RS = CV(R)
    OS = (S[n/2], S[n/2+1], S[n/2+1] - S[n/2])

        return LS        # left pair wins
    elif RS[3] <= OS[3]:
        return RS        # right pair wins
    else:
        return OS        # cross-over pair wins

**Marking:**

Pseudo-code is not required (because I forgot to specify!) - a clear text description is acceptable. Pseudo-code (or actual code) does not need to be like mine, or documented.

Solutions that show good understanding of D&C method should get at least 8/15, even if the solution is incorrect. This over-rides specific deductions listed below:

Solutions that omit a base case: penalize 3 marks

Solutions that do not handle the “cross-over” case: penalize 3 marks

Other errors: penalize 1 or 2 marks, depending on severity