

NAME: **ALGORITHMICS UNIT 3 & 4****Trial Exam 2: 2020
SOLUTIONS****Reading Time: 15 minutes
Writing time: 120 minutes (2 hours)****QUESTION AND ANSWER BOOK**

<i>Section</i>	<i>Number of questions</i>	<i>Number of questions to be answered</i>	<i>Number of marks</i>
A	20	20	20
B	8		80

- Students are permitted to bring into the examination room: pens, pencils, highlighters, erasers, sharpeners, rulers and one scientific calculator.
- Students are NOT permitted to bring into the examination room: blank sheets of paper and/or correction fluid/tape

Materials supplied

- Question and answer book of 20 pages
- Answer sheet for multiple-choice questions

Instructions

- Write your student number in the space provided above on this page.
- Check that your name and student number as printed on your answer sheet for multiple-choice questions are correct, and sign your name in the space provided to verify this.
- All written responses must be in English, point form is preferred.

Students are NOT permitted to bring mobile phones and/or any other unauthorised electronic devices into the test room.

The VCAA Exam will include the Master Theorem in this form.

Use the Master Theorem to solve recurrence relations of the form shown below.

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + kn^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases} \quad \text{where } a > 0, b > 1, c \geq 0, d \geq 0, k > 0$$

$$\text{and its solution } T(n) = \begin{cases} O(n^c) & \text{if } \log_b a < c \\ O(n^c \log n) & \text{if } \log_b a = c \\ O(n^{\log_b a}) & \text{if } \log_b a > c \end{cases}$$

The VCAA form of Master Theorem is equivalent to the form of Master Theorem taught in our class by consideration of log laws.

$$\log_b a = c \Leftrightarrow a = b^c \Leftrightarrow \frac{a}{b^c} = 1$$

$$\log_b a < c \Leftrightarrow a < b^c \Leftrightarrow \frac{a}{b^c} < 1$$

$$\log_b a > c \Leftrightarrow a > b^c \Leftrightarrow \frac{a}{b^c} > 1$$

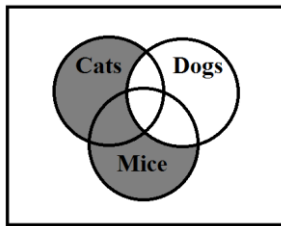
$$T(n) = aT\left(\frac{n}{b}\right) + f(n^k)$$

- $\frac{a}{b^k} < 1$ then $O(n^k)$
- $\frac{a}{b^k} = 1$ then $O(n^k \log_b n)$
- $\frac{a}{b^k} > 1$ then $O(n^{\log_b a})$

SECTION A – Multiple Choice – select one option only

Question 1

Which expression corresponds to elements in the shaded area shown in the Venn Diagram?



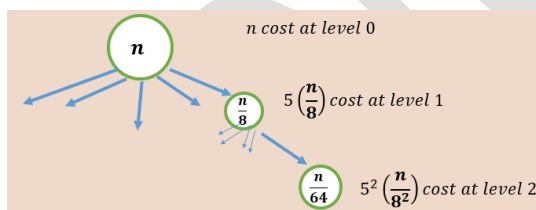
- A. If (NOT DOGS AND (CATS AND MICE)) then
- B. If (NOT DOGS OR (CATS OR MICE)) then
- C. If (NOT DOGS AND (CATS OR MICE)) then
- D. If (NOT DOGS AND NOT (CATS OR MICE)) then

Question 2

Which of the following statements is true about graph properties?

- A. To find the diameter of a graph, first find the shortest path between each pair of vertices. The greatest length of any of these paths is the diameter of the graph.
- B. To find the diameter of a graph, find the longest path between each pair of vertices.
- C. To find the diameter of a graph, first find the shortest path between each pair of vertices. The shortest length of any of these paths is the diameter of the graph.
- D. To find the diameter of a graph, first find the longest path between each pair of vertices. The smallest length of any of these paths is the diameter of the graph.

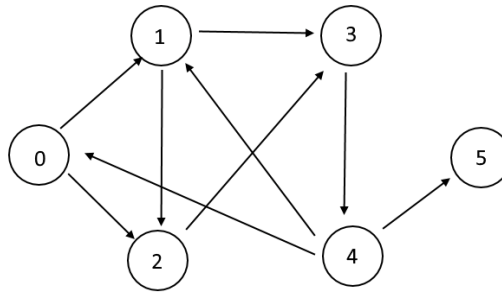
Question 3



The recurrence relation from the call tree shown above where at each level an algorithm makes 5 recursive calls to itself and each recursive call splits the input into eighths could be:

- A. $T(n) = 8T\left(\frac{n}{5}\right) + 8n, T(1) = 1$
- B. $T(n) = T\left(\frac{n}{8}\right) + 5n, T(1) = 1$
- C. $T(n) = 8T\left(\frac{n}{5}\right) + n, T(1) = 1$
- D. $T(n) = 5T\left(\frac{n}{8}\right) + n, T(1) = 1$

Question 4



The Depth first search node processing order on the directed graph shown above starting at node 0 **cannot** be:

- A. 0, 1, 3, 4, 5, 2
- B. 0, 2, 3, 4, 1, 5
- C. 0, 1, 2, 3, 4, 5

D. 0, 1, 3, 2, 4, 5

Question 5

A big- O estimate for the number of operations, where an operation is an addition or a multiplication, used in this segment of an algorithm is:

```
t := 0
for i := 1 to 3
  for j := 1 to 4
    t := t + ij
```

A. $O(n)$

B. $O(1)$

C. $O(\log n)$

D. $O(n^2)$

Question 6

Which of the following statements is **not** true

- A. **greedy algorithm:** an algorithm that makes the best choice at each step according to some specified condition
- B. **tractable problem:** a problem for which there is a worst-case polynomial-time algorithm that solves it
- C. **intractable problem:** a problem for which no worst-case polynomial-time algorithm exists for solving it

D. **undecidable problem:** a problem that can be solved by an algorithm

Question 7

The missing logic from the Floyd Warshall algorithm shown at the right here for determining transitive closure are in consecutive order of the empty boxes:

A. 1, 1, (T[i,k] AND T[k,j])

B. 1, 0, (T[i,k] AND T[k,j])

C. 1, 0, (T[i,j] AND T[j,k])

D. 0, 1, (T[i,k] AND T[k,j])

```
Algorithm FloydWarshallTransitiveClosure(Input: G)
// let T be a  $|V| \times |V|$  boolean array of Transitive Closure)
// initialized to run on directed graph  $G=(V,E)$ 
for i from 1 to  $|V|$  do
    for j from 1 to  $|V|$  do
        if ((i is equal to j) OR (edge i-j exists)) then
            T[i,j] := 
        else
            T[i,j] := 
        End if
    End do
End do
// build up the array of reachability
for k from 1 to  $|V|$  do
    for i from 1 to  $|V|$  do
        for j from 1 to  $|V|$  do
            T[i,j] := T[i,j] OR 
        End do
    End do
End do
End Algorithm
```

Question 8

Which of the following techniques uses randomness for avoiding getting trapped in local maxima?

A. Best first search

B. Local beam search

C. Simulated annealing

D. Gradient descent

Question 9

The following operations may be used on a Dictionary Abstract Data type.

- new(dictionary)
- insert(key,value,dictionary)
- delete(key,dictionary)

Another common operation on a Dictionary Abstract Data type is:

A. find(key,dictionary)

B. enqueue(key,dictionary)

C. pop(key,dictionary)

D. front(key,dictionary)

Question 10

Consider the following function defined in pseudocode:

```
function fibonacci(integer n) {  
    if (n < 2) {  
        return n;  
    }  
    if (value=find(key=n,dictionary) exists) {  
        return value;  
    }  
    fiboNumber = fibonacci(n - 1) + fibonacci(n - 2);  
    insert(key=n,fiboNumber,dictionary);  
    return fiboNumber;  
}
```

The space complexity of the fibonacci function above is:

A. $O(n)$

B. $O(n^3)$

C. $O(n!)$

D. $O(k^n)$

Question 11

What is the time complexity of the brute force algorithm used to solve the 01- Optimal Knapsack problem?

A. $O(n)$

B. $O(n!)$

C. $O(2^n)$

D. $O(n^3)$

Question 12

In a modified Mergesort, the input array is split into 3 thirds of the length(n) of the array, resulting in a split of $\frac{n}{3}$, $\frac{n}{3}$ and $\frac{n}{3}$ elements respectively. Which of the following is the tightest upper bound on time complexity of this modified Mergesort?

A. $O(\log_3 n)$

B. $O(n \log_2 n)$

C. $O(n \log_3 n)$

D. $O(n^3)$

Question 13

A greedy algorithm uses a heuristic function to

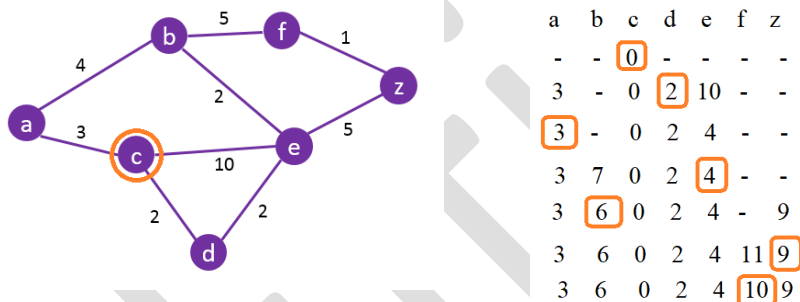
- A. expand the node that appears to be closest to the goal**
- B. expand the node that is closest to the goal
- C. expand the node that is the most expensive
- D. expand the leftmost node

Question 14

Which of the following problems **cannot be** solved by backtracking method?

- A. n-queen problem
- B. a sudoku puzzle
- C. hamiltonian circuit problem
- D. travelling salesman problem**

Question 15



Running Dijkstra's algorithm on the weighted graph above starting at node C will result in the following order of node expansion and processing:

- A. c, d, a, b, e, z, f
- B. c, d, e, a, b, z, f
- C. c, d, e, a, b, z, f
- D. c, d, a, e, b, z, f**

Question 16

The missing terms for the following statement

"Prim's algorithm is a _____ algorithm, that initialises with _____."

- A. Divide and conquer, vertex
- B. Greedy algorithm, vertex**
- C. Dynamic Programming, edge
- D. Backtracking, vertex

Question 17

Problems that can be solved in polynomial time are known as?

- A. intractable
- B. tractable**
- C. decision
- D. complete

Question 18

Who did **not** invent the Turing machine?

- A. Alan Turing
- B. Turing
- C. Mr. Turing
- D. None of the above**

Question 19

Which of the following problems **is not** NP complete?

- A. Hamiltonian circuit
- B. Decision version of the Travelling Salesman problem
- C. Searching a network graph
- D. Halting problem**

Question 20

The missing words that are most appropriate as a description of Hilbert's Program in the following sentence:

"Hilbert's Program calls for a [] of all of mathematics in [] form, together with a proof that this axiomatization of mathematics is [], using [] methods."

are:-

- A. differentiation, theoretical, consistent, finitary
- B. formalization, axiomatic, consistent, infinitary
- C. formalization, axiomatic, consistent, finitary**
- D. classification, basic, consistent, infinitary

SECTION B – Extended Response Questions Answer all questions in the space provided.

Question 1 (10 marks)

A Sudoku puzzle is created in a matrix of **k elements** × **k elements** with some values already set. The solution to the puzzle has the integers **1 to k** appear once in each row, column and cage.

Example: k=9 rows, columns, cages, where each cage is a sub-array of 3x3 cells.

5			1					4
2	7	4				6		
	8		9		4			
8	1		4	6		3		2
		2		3		1		
7		6		9	1		5	8
			5		3		1	
		5				9	2	7
1				2				3

With numbered cages

1	2	3
4	5	6
7	8	9

The top right hand corner row=*a*, column=*b* of each cage numbered sequentially from left to right, top to bottom is given by the provided modules **geta(cageNumber)** and **getb(cageNumber)**

(*a*,*b*)

cageNumber

Function **geta(cageNumber, k)**

// returns the top right hand corner cell
 // row number for a cage number
 // of a $k \times k$ Sudoku
 // the **ceil** of a real number will round
 // it up to the next highest integer
 return $\left(\sqrt{k} \times \text{ceil}\left(\frac{\text{cageNumber}}{\sqrt{k}}\right) - \sqrt{k} + 1\right)$

End function

Function **getb(cageNumber, k)**

// returns the top right hand corner cell
 // column number for a cage number
 // of a $k \times k$ Sudoku
 // the **mod** returns the integer remainder
 // of the division of two integers
 If $\left(\text{mod}\left(\frac{\text{cageNumber}}{\sqrt{k}}\right) \text{ is } 0\right)$ then
 return $(k - \sqrt{k} + 1)$
 else
 return $\left(k - \left(\sqrt{k} - \text{mod}\left(\frac{\text{cageNumber}}{\sqrt{k}}\right) + 1\right)\sqrt{k} + 1\right)$
 End if

End function

- a) Complete the module **checkRow** below in structured pseudocode that will check a Sudoku matrix row for correct values. (2 marks)

Module **checkRow(S,row,k)**

//Input: Sudoku matrix S, row of Sudoku, size k of Sudoku

Create/Reset a minimum priority queue MinPQ to empty
 for j=1 to k do
 enqueue value of S[row,j] into MinPQ
 end do
 if (dequeued values from MinPQ are in order 1..k) then
 ok=ok+1
 else
 report "Error in row", row
 end if

End module

Question 1 (Continued)

- b) Complete the module **checkColumn** below that will check a Sudoku matrix column for correct values (2 marks)

Module checkColumn(S,column,k)

//Input: Sudoku matrix S, column of Sudoku, size k of Sudoku

```
Create/Reset a minimum priority queue MinPQ to empty
for j=1 to k do
    enqueue value of S[j,column] into MinPQ
end do
if (dequeued values from MinPQ are in order 1..k) then
    ok=ok+1
else
    report "Error in column", column
end if
```

End module

- c) Complete the module **checkCage** using the provided modules/functions **geta(cageNumber)** and **getb(cageNumber)** (from previous page) to check a Sudoku cage for correct values. (3 marks)

Module checkCage(S,cage,k)

//Input: Sudoku matrix S, cage number of Sudoku, size k of Sudoku

a:=geta(cage) //Find top right hand corner row=a of given cage number (see previous page)

b:=getb(cage) //Find top right hand corner column=b of given cage number (see previous page)

```
Create/Reset a minimum priority queue MinPQ to empty
for i=1 to square root of k do
    for j=1 to square root of k do
        enqueue value of S[a+i-1,b+j-1] into MinPQ
    end do
end do
if (dequeued values from MinPQ are in order 1..k) then
    ok=ok+1
else
    report "Error in cage", cage
end if
```

End module

- d) Combine the modules **checkRow**, **checkColumn**, **checkCage** to make a complete Sudoku solution checking algorithm **checkSudoku** in structured pseudocode. (3 marks)

Algorithm checkSudoku(sudoku,k)

```
Set variable ok=0
For i=1 to k do
    checkRow(sudoku,i,k)
    checkColumn(sudoku,i,k)
    checkCage(sudoku,i,k)
end do
if (ok equals k cubed) then
    report "Sudoku is all good"
End if
```

End algorithm

Question 2 (8 marks)

- a) For any connected graph, by definition there exists a path from any one node to another. Give a structured proof that Depth First Search (DFS) is always correct, being that it will always find a possible path, or return that there isn't one for any given graph input. **(4 marks)**

DFS Proof by Induction: Note that if it is possible to reach one node during a DFS, it is also possible to reach all neighbouring nodes. Let $v_1, v_2, \dots, v_n$ be the nodes in a path from the starting node v_1 in the DFS to any target node v_n .

We proceed by induction to prove DFS is always correct. We call a node "reachable" if a DFS will search the node.

- **Base case:** v_1 is trivially reachable by definition
- **Inductive hypothesis:** Suppose the node v_k is reachable ($1 < k < n$) then since and are connected by an edge, the node v_{k+1} is also reachable

Hence the induction is complete and so the node v_n is reachable. Which means a DFS is able to find a path from any node to any node in a connected graph.

If there are two nodes in a graph such that there are no paths from one to the other then the graph is either directed or disconnected, in which case a DFS will traverse all nodes within the connected subgraph of the starting node, returning "no path found" when all reachable nodes have been visited.

Consider the pseudocode for the Floyd-Warshall's shortest path algorithm.

(NOTE: you probably won't be given the F-W pseudocode in VCAA exam)

```
Floyd-Warshall's Algorithm
// initialise distance |V|×|V| matrix
set all distance[ , ] cells to infinity
for all weighted edges i-j in G update distance[i,j] with weight
for all nodes in G i update distance[i,i]=0
// relaxation
for k = 1 to n do
  for i = 1 to n do
    for j = 1 to n do
      if (i != k AND j != k) then
        if (distance[i,j] > distance[i,k] + distance[k,j]) then
          distance[i,j] = distance[i,k] + distance[k,j];
        end if
      end if
    enddo
  enddo
enddo
```

- b) Give a proof by contradiction of the correctness of the Floyd-Warshall algorithm for finding shortest paths between pairs of nodes in weighted, connected and undirected graphs. **(4 marks)**

By contradiction say that Floyd Warshall (F-W) does not find the shortest path between nodes i to j in graph G . At iteration $(k-1)$ of F-W for node i to node j , $\text{distance}[i,j]$ is containing only vertices in the vertex set $\{1, \dots, (k-1)\}$ for path A, node i , and node j are in the graph G , and $\text{distance}[i,j]$ does not contain the length of a shortest path between $i \rightarrow j$.

In the iteration k , the length of path A of $\text{distance}[i,j]$ is compared to the length of a path B composed of two subpaths,

- B1: from node $i \rightarrow k$ node path of $\text{distance}[i,k]$ with "intermediate vertices" only in $\{1, \dots, (k-1)\}$, and
- B2: from node $k \rightarrow j$ node path of $\text{distance}[k,j]$ with "intermediate vertices" only in $\{1, \dots, (k-1)\}$.

The shorter of these two A or $(B1 + B2)$ is always chosen. A contradiction to the initial premise.

The shortest path from $i \rightarrow j$ in G containing only vertices in the vertex set $\{1, \dots, k\}$ either

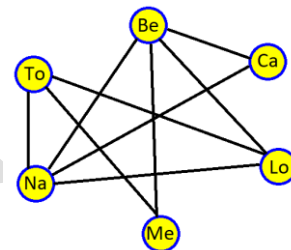
- a) the shortest path does not contain k , and hence is the one found in iteration $(k-1)$ or
 - b) the shortest path does contains k , and then can be decomposed into an $i \rightarrow k$ followed by a $k \rightarrow j$ path
- Hence the update ensures the correctness of F-W for finding the shortest path in $\text{distance}[i,j]$ after all iterations of k . A contradiction to the initial premise.

Question 3 (14 marks)

The “**Cities Puzzle**” tries to place citizens of cities into a seating plan so that citizens **are not seated** next to other citizens in adjacent horizontal or vertical grouping of seats if the name of their city **shares any letter(s)**. For example citizens of Tokyo **can be** seated adjacent to citizens of Berlin since their cities do not share any of the same letters, but citizens of Berlin and Paris cannot be seated near each other since they have the letters “i” and “r” in common.

Consider the following 6 cities, abbreviated in brackets: Berlin (Be), Caracas (Ca), London (Lo), Metz (Me), Nairobi (Na) and Tokyo (To).

These 6 cities are represented by a graph model (at the right) that indicates cities that **do share** at least one same letter in their full name, which are connected by an edge.



This model does not immediately solve the “**Cities Puzzle**”.

a) Give the degree of each node in the graph above.

(1 mark)

Deg(Be)=4, Deg(Ca)=2, Deg(Lo)=3, Deg(Me)=2, Deg(Na)=4, Deg(To)=3

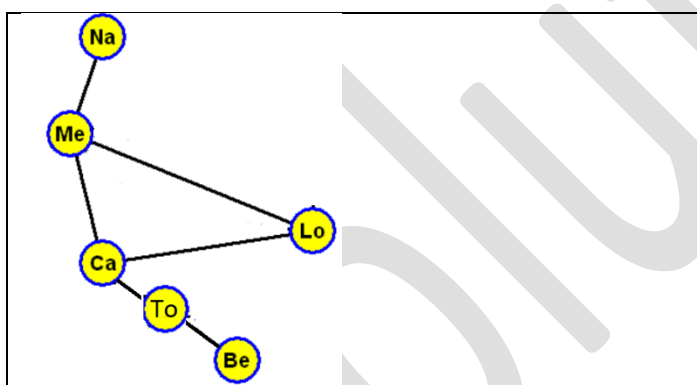
b) What property of graphs can be used to solve the “**Cities Puzzle**” for this example with the provided information?

(1 mark)

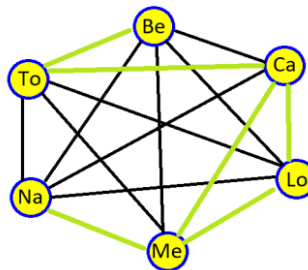
Complete the graph to find the cities that don’t share any letters.

c) Place the names of these 6 cities into a graph Abstract Data Type model that indicates cities that **do not share** any same letters in their name.

(2 marks)

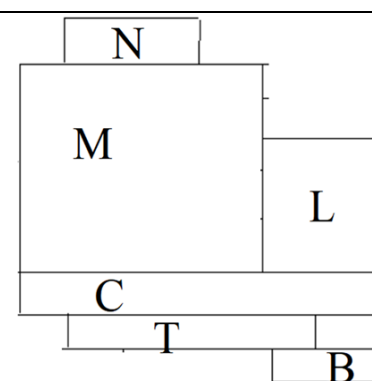
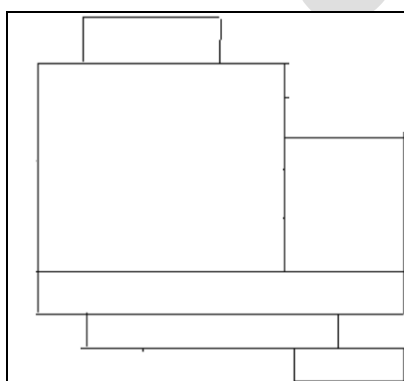


Found easily by adding edges to make a complete graph on the graph with countries that share at least one letter.



d) If the 6 cities are represented as 6 seating areas, label the adjacent seating areas with the cities that **do not share any** of the same letters in their name and solve the “**Cities Puzzle**” for this example.

(2 marks)



Question 3 (continued)

- e) What well known and studied problem has the “**Cities Puzzle**” been transformed into? (1 mark)

Graph Colouring Problem

- f) Is it possible to solve the “**Cities Puzzle**” for larger number of cities, say 50 cities? With consideration of your responses in parts a), b), c) and d) of this question. Explain what kind of information, abstract data types and associated algorithms could be used **to prepare the information** prior to finding a solution. Explain if there are any limitations on those algorithms. (3 marks)

Process for information could be:

Step1: create a network graph of cities for $n=50$, with letter in common this would take n^2 comparisons to compare each city with every other for common letters, creating a graph of 50 nodes.

Step 2: find the cities with no edges connecting them to others and add an edge with attribute “don’t have letters in common”. This would take n^2 comparisons to compare each city in a simple way.

Step 3: Information is now ready to solve the “**Cities Puzzle**”

Space complexity for a complete graph of n nodes is $\frac{n(n-1)}{2}$ edges + n nodes.

Preparing the information can be done in polynomial time in polynomial space.

- g) Discuss the time complexity for solving the “**Cities Puzzle**” problem for a large number of cities, with an explanation of the feasibility of finding the solution. (2 marks)

Once the information has been collected into a graph ADT, which takes polynomial time, we have created a classic Graph Colouring problem, colours are assigned to the nodes of the graph subject to constraints. If the minimum colours are used then we can determine the exact size and extent of the conflict.

The time complexity of a brute force approach colouring with a minimum of “ k ” colours would be $O(k^n)$, an exponential time complexity, that may be infeasible to wait for.

A heuristic approach such as a Greedy colouring algorithm could be used to find a “good” solution in tractable, polynomial time.

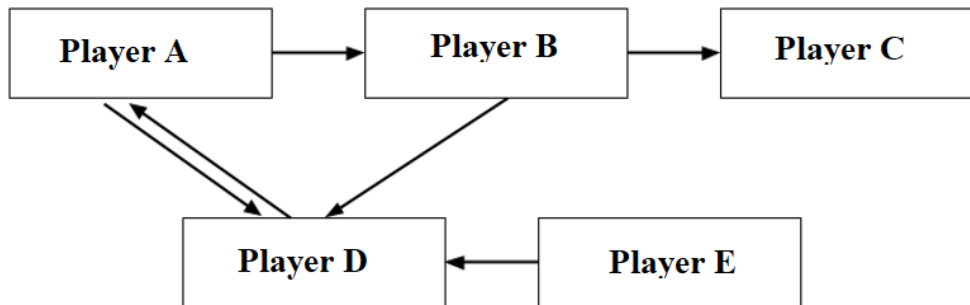
- h) What class of problem is the “**Cities Puzzle**”? Explain and justify your conclusions. (2 marks)

The “**Cities Puzzle**” can be transformed in polynomial time into the classic Graph Colouring problem where colours are assigned to the nodes of the graph subject to constraints modelled with connecting edges. The Graph Colouring problem which is a known NP-Complete problem, hence the “**Cities Puzzle**” is also NP-Complete.

Question 4 (9 marks)

Player A	Player B	Player C	Player D	Player E
				

In one particular Soccer season the player to player passing rate of 85% for 5 players A, B, C, D and E is shown by the directed graph below.



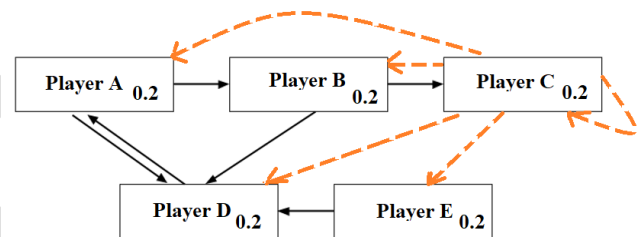
- a) Using the Page Rank algorithm show all the initialisation steps and values for each player's node that are actioned by the algorithm. **(2 marks)**

Here are the initialisation steps

The sink node (Player C) is redistributed to all the other nodes including itself.

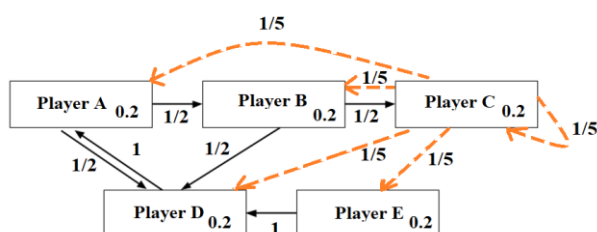
Each node is assigned an initial probability of $\frac{1}{|V|}$, in this case the number of nodes $|V|=5$
So initialised with

$$\Pr(A_0) = \Pr(B_0) = \Pr(C_0) = \Pr(D_0) = \Pr(E_0) = \frac{1}{5}$$



- b) Show recurrence relations used by the Page Rank Algorithm for each player $\Pr(A_{i+1})$, $\Pr(B_{i+1})$, $\Pr(C_{i+1})$, $\Pr(D_{i+1})$, $\Pr(E_{i+1})$ for finding the next iteration $(i+1)^{th}$ page rank of each player. **(3 marks)**

Looking at the outgoing edges from each player, the probability of passing to the next player is assigned, note for this season that this probability applies only 85% of the time, and random chance ($\frac{15\%}{|V|}$) of the time.



$$\Pr(A_0) = \Pr(B_0) = \Pr(C_0) = \Pr(D_0) = \frac{1}{5}$$

Recurrences

$$\Pr(A_{i+1}) = 0.85 \left(\frac{1}{5} \Pr(C_i) + \Pr(D_i) \right) + \frac{0.15}{5}$$

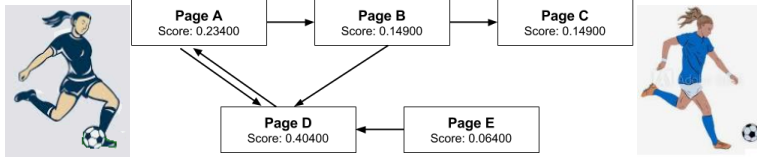
$$\Pr(B_{i+1}) = 0.85 \left(\frac{1}{5} \Pr(C_i) + \frac{1}{2} \Pr(A_i) \right) + \frac{0.15}{5}$$

$$\Pr(C_{i+1}) = 0.85 \left(\frac{1}{5} \Pr(C_i) + \frac{1}{2} \Pr(B_i) \right) + \frac{0.15}{5}$$

$$\Pr(D_{i+1}) = 0.85 \left(\frac{1}{2} \Pr(A_i) + \frac{1}{2} \Pr(B_i) + \frac{1}{5} \Pr(C_i) + \Pr(E_i) \right) + \frac{0.15}{5}$$

$$\Pr(E_{i+1}) = 0.85 \left(\frac{1}{5} \Pr(C_i) \right) + \frac{0.15}{5}$$

Question 4 (continued)



c) Using the Page Rank algorithm find the ranking for each player after the **first iteration**. (2 marks)

$\Pr(A_1) = 0.85 \left(\frac{1}{5} \Pr(C_0) + \Pr(D_0) \right) + \frac{0.15}{5}$ $\Pr(B_1) = 0.85 \left(\frac{1}{5} \Pr(C_0) + \frac{1}{2} \Pr(A_0) \right) + \frac{0.15}{5}$ $\Pr(C_1) = 0.85 \left(\frac{1}{5} \Pr(C_0) + \frac{1}{2} \Pr(B_0) \right) + \frac{0.15}{5}$ $\Pr(D_1) = 0.85 \left(\frac{1}{2} \Pr(A_0) + \frac{1}{2} \Pr(B_0) + \frac{1}{5} \Pr(C_0) + \Pr(E_0) \right) + \frac{0.15}{5}$ $\Pr(E_1) = 0.85 \left(\frac{1}{5} \Pr(C_0) \right) + \frac{0.15}{5}$ $\Pr(A_0) = \Pr(B_0) = \Pr(C_0) = \Pr(D_0) = \Pr(E_0) = \frac{1}{5}$	Using your scientific calculator which is permitted in the VCAA exam $\Pr(A_1) = 0.85 \left(\frac{1}{5} \times \frac{1}{5} + \frac{1}{5} \right) + 0.03 = 0.234$ $\Pr(B_1) = 0.85 \left(\frac{1}{5} \times \frac{1}{5} + \frac{1}{2} \times \frac{1}{5} \right) + 0.03 = 0.149$ $\Pr(C_1) = 0.85 \left(\frac{1}{5} \times \frac{1}{5} + \frac{1}{2} \times \frac{1}{5} \right) + 0.03 = 0.149$ $\Pr(D_1) = 0.85 \left(\frac{1}{2} \times \frac{1}{5} + \frac{1}{2} \times \frac{1}{5} + \frac{1}{5} \times \frac{1}{5} + \frac{1}{5} \right) + 0.03 = 0.404$ $\Pr(E_1) = 0.85 \left(\frac{1}{5} \times \frac{1}{5} \right) + 0.03 = 0.064$
---	--

The Soccer coach has statistics from a previous season, where the **player to player** passing rate for the same players in the same directions was **70%**.

d) Explain how the Page Rank algorithm applies in detail for **Player A** in the previous season. (2 marks)

The initialisation remains the same for any rate of favoured passing in PR. The chance of random passing to any of the 5 players is $\frac{0.3}{5}$, while the intentional passing rate is 0.7 and is reflected in adjusted recurrence calculations for Player A.	Recurrence for previous season $\Pr(A_{i+1}) = 0.7 \left(\frac{1}{5} \Pr(C_i) + \Pr(D_i) \right) + \frac{0.3}{5}$
---	---

Question 5 (9 marks)

The optimal Travelling Salesman problem seeks to travel to each city in a connected network once only by the cheapest tour before returning home.



a) Why is the Travelling Salesman Problem (TSP) studied so extensively in Computer Science? (1 mark)

It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, many heuristics and exact algorithms are known,

b) What other real world application can be formulated as the Travelling Salesman Problem? (1 mark)

Planning, logistics, and the manufacture of microchips. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources.
--

Question 5 (continued)



- c) What exact algorithms exist to solve the optimal Travelling Salesman Problem? Briefly explain the time complexity of the exact algorithm and the amount of work it will do for 20 cities. **(2 marks)**

The most direct solution would be to try all permutations and see which one is cheapest, by using brute-force search. The running time for this approach $O(n!)$, the factorial of the number of cities, so this solution becomes impractical even for only 20 cities, resulting in a factor of $\frac{(20-1)!}{2}$ comparisons.

- d) What is the classification of the optimal Travelling Salesman Problem? Justify your answer. **(1 marks)**

NP-Hard, as the correct and optimal solution cannot be solved or verified in polynomial time.

- e) What other approaches exist for finding good solutions for the optimal TSP? Describe **three** of these approaches. **(4 marks)**
- Many heuristic and approximation algorithms have been devised that can be used for the optimal TSP, which quickly (in polynomial time) yield good solutions.
 - Nearest Neighbour heuristic . a greedy algorithm lets the salesman choose the nearest unvisited city as his next move. This algorithm quickly yields an effectively short route for N cities randomly distributed on a plane.
 - Hill climbing attempts to maximize or minimize a target function $f(x)$, where $f(x)$ is a vector of continuous and/or discrete values. At each iteration, hill climbing will adjust a single element in $f(x)$ and determine whether the change improves the value of $f(x)$, in greedy progression better moves are kept. Hill Climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution.
 - The Simulated annealing heuristic considers some neighbouring state s^* of the current state s , and uses randomised methods to decide between moving the system to state s^* or staying in state s . Simulated annealing will accept some worse solutions in the short term, jumping back if not promising. This randomised method is repeated while the probability for finding a better solution elsewhere to the current state s is reducing, until the system reaches a state that is good enough for the application, or until the time or space resources allocated has been exhausted.

Question 6 (6 marks)

An iterative algorithm described in three steps for finding a Square Root approximation is shown below. This algorithm accepts a positive number S and finds the square root using an approximation algorithm that dates from ancient Babylonian times.

1. Begin with an arbitrary positive starting value x_0 , that is between 1 and S , the closer to the actual square root of N , the better.
2. Let $x_{(n+1)}$ be the average of $\left(x_n + \frac{S}{x_n}\right)$, which is the arithmetic mean
$$x_{(n+1)} = \frac{1}{2} \left(x_n + \frac{S}{x_n}\right)$$
, and use it to approximate the geometric mean
3. If $(x_{(n+1)})^2 \approx S$ to the required accuracy then you are done, otherwise repeat from step 2

Here are the mathematical representations for the algorithm

It can also be represented as:

$$x_0 \approx \sqrt{S},$$

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{S}{x_n}\right),$$

$$\sqrt{S} = \lim_{n \rightarrow \infty} x_n.$$

- a) Create a **recursive function** in structured pseudocode for square root approximation using the Babylonian algorithmic method that will only recur for a maximum of **maximum** times. (5 marks)

```
Function Babylonian (S, x, max)
// Input S the number to find the square root for
// Input x, a guess of the square root
// Input max, the number of recursions
If (x > 0) AND (x < S) then
    If (x2 equals S to the required accuracy) OR (max > maximum) then
        // Assumption: maximum is a global constant value set to stop the recursion
        Return x
    Else
         $x_{new} := \frac{1}{2} \left(x_n + \frac{S}{x_n}\right)$ 
        Babylonian (S,  $x_{new}$ , max + 1)
    End if
End if
End Function
```

- b) Give an example of how your function from part a) will be called. (1 marks)

An example of how the function is called by:

```
maximum:=100
Babylonian (105, 6, 1)
```

Question 7 (16 marks)

Queue Theory – is a real study, here is a limited model to study the order in which customers will be served by cashiers at a supermarket.



The conditions in this supermarket are:

- Each cashier spends the same amount of time with each customer
- The number of active queues will be between 1 and 3 inclusive.
- The number of customers, will be between 1 and 20 inclusive identified by a unique letter (A,B,C,...)

On Monday at 9am at this supermarket, there were 3 active queues:

- **Queue1**, Cashier number 1 spends 3 minutes on each customer, customers in this queue: **A, B, C**
- **Queue2**, Cashier number 2 spends 2 minutes on each customer, customers in this queue: **D, E, F, G**
- **Queue3**, Cashier number 3 spends 4 minutes on each customer, customers in this queue: **H, I, J**

- a) What was the order of service of customers A, B, C, D, E, F, G, H, I, J across the three queues with this Monday situation? **(1 marks)**

D (q2), A (q1), E (q2), H (q3), B (q1), F (q2), G (q2), I (q3), C (q1), J (q3)

- b) Build a model using appropriate abstract data type(s) and operations to model the information of the Monday queues. **(2 marks)**

Cashier 1	Cashier 2	Cashier 3
Create Array Cashier 3x2 elements Cashier[1][1] :=3 Cashier[1][2] :=Q1 Create Queue Q1 Enqueue A to Q1 Enqueue B to Q1 Enqueue C to Q1	Cashier[2][1] :=2 Cashier[2][2] :=Q2 Create Queue Q2 Enqueue D to Q2 Enqueue E to Q2 Enqueue F to Q2 Enqueue G to Q2	Cashier[3][1] := 4 Cashier[3][2]:=Q3 Create Queue Q3 Enqueue H to Q3 Enqueue I to Q3 Enqueue J to Q3

- c) Create a simple algorithm to process the information in the model you created in part b) and output the order of service of customers A, B, C, D, E, F, G, H, I, J. **(4 marks)**

```
Algorithm Service (Cashier, Q1, Q2, Q3)
// Input Cashier, Q1, Q2, Q3 queue information
Create Minimum Priority Queue ServiceQ
For i=1 to length (Cashier) do
    Interval:= 0
    Time:=Cashier[i][1]
    ThisQueue:=Cashier[i][2]
    While (ThisQueue is not empty) do
        Interval := Interval + Time
        Dequeue person from ThisQueue
        Enqueue person with rank Interval to ServiceQ
    End do
End do
While (ServiceQ is not empty) do
    Print front of ServiceQ
    Dequeue person from ServiceQ
End do
End Algorithm
```

Question 7 (continued)

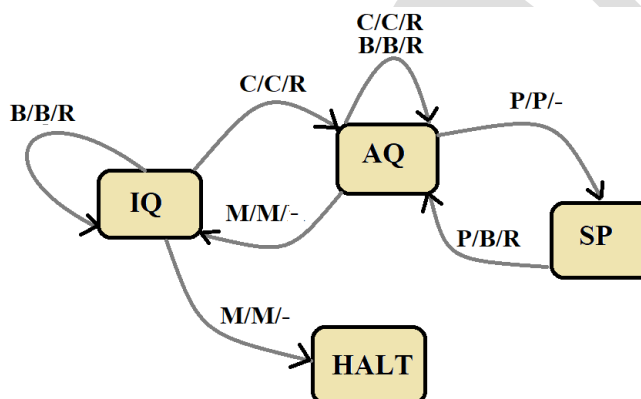
A Turing machine is set up to service one queue.

AT Telling machine is setup to service one queue.

The states are:		The symbols are:	
IQ – Inactive Queue with no cashier		C – Cashier	
AQ – Active Queue with a cashier		B – Blank	
SP – Serving Person		P – Person	
Halt – Supermarket closed		M –Manager	

Input received	If in state	Do this action
C	IQ	Change state to AQ, move to next input
M	IQ	Change state to Halt
B	IQ	Remain in state IQ, move to next input
C	AQ	Remain in state AQ, move to next input
B	AQ	Remain in state AQ, move to next input
M	AQ	Change state to IQ
P	AQ	Change state to SP
P	SP	Overwrite P with B, change to state AQ, move to next input

- d) Convert this Turing Machine to directed graph form. (4 marks)



- e) Why is the Turing machine considered so important in Computer Science with respect to the classification of problems and their algorithms? (3 marks)

Turing Machines (TM)s are important for computer science as they are one of the simplest theoretical models for computers. A TM is used for the computation of functions, the result of the function, which depends on the input, is provided as output.

A TM is a finite system of rules, states and transitions as an abstraction of a computer running an algorithm with input and outputs.

The classification of problems as P class – polynomial time and NP class – non-deterministic polynomial time is determined by the number of steps required to compute the function/solution for the problem on a Turing Machine as a proportion of the input size to the problem.

- f) What is the Church Turing thesis, how is it connected to the Turing machine and to the classification of problems? (2 marks)

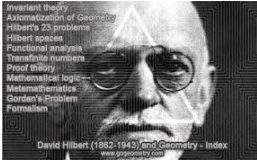
Church-Turing (CT) thesis states that all computers are only as powerful as Turing machines. This can be used to prove if a problem is computable as according to the CT thesis any problem that can be solved (decided) by a TM can be solved by a computer that has a finite amount of memory.

OR

It states that a function on the natural numbers can be calculated by an effective method in finite time if and only if it is computable by a Turing machine provided with unlimited memory

CT will classify $f(x)$ as effectively computable even if it is only computable by a Turing machine

Question 8 (8 marks)



- a) What were the main goals of Hilbert's program, in terms of **Completeness**, **Consistency** and **Decidability**? (3 marks)

The main goal of Hilbert's program was to provide secure foundations for all mathematics. A formulation of all mathematics; in other words all mathematical statements should be written in a precise formal language, and manipulated according to well defined rules, axioms and theorems.

Completeness: a proof that all true mathematical statements can be proved in the formalism.

Consistency: a proof that no contradiction can be obtained in the formalism of mathematics.

Decidability: there should be an algorithm for deciding the truth or falsity of any mathematical statement.

- b) Why was Hilbert's program was shown to be unachievable? Briefly explain why. (3 marks)

Hilbert's attempt to support axiomatized mathematics with no theoretical uncertainties, ended in failure.

Gödel showed that for any consistent **formal axiomatic system** strong enough to formalize what was traditionally regarded as finitary reasoning, it is still possible to define a sentence that expresses the consistency of the **formal axiomatic system**, which is not provable in the **formal axiomatic system**.

Gödel demonstrated that any non-contradictory formal system, which was comprehensive enough to include at least arithmetic, cannot demonstrate its completeness by way of its own axioms.

- c) How did Hilbert's program and Gödel's work contribute to the foundations of Computer Science? (2 marks)

Hilbert's work had started formalisms and logic on this course of clarification of Mathematics; with finitary reasoning a focus for defining the formulations.

Mathematical logic, formalisms and finitary reasoning is the basis for theoretical computer science, in the work of Alonzo Church and Alan Turing, which also grew directly out of this 'debate'.

END OF TRIAL EXAM 2