**Order of attempt: Q1, Q5, Q6**

**To adjust later: Q2 – Wikipedia pseudocode is better maybe use that instead.**

**Q1 c – Reword with lecture slide terms**

**Q3 potential method can be improved**

**Q4 Find out if you only decrease key or a min pop / clean up is required**

**Q5b Write out rb tree proof**

**Long response: Q3, Q1, Q6, Q5**

# SHORT ANSWER

## QUESTION 1 [12 MARKS]

1. **Describe, with an example, the path compression operation for disjoint sets.**
2. **Explain what is meant by an Eulerian circuit in a finite undirected graph.**

A Eulerian circuit refers to a path in a graph that starts at node s and finishes at node s using every edge exactly once.

1. **What simple property of the graph corresponds exactly to the existence of Eulerian circuits?**

A directed graph has an Eulerian cycle if and only if every vertex has equal in degree and out degree, and all of its vertices with nonzero degree belong to a single strongly connected component.

1. **Explain clearly how to compute an Eulerian circuit in such graphs (you need not write a program, but whatever algorithm you choose must be clearly explained).**
2. **Draw the vEB(16) tree that stores the set {1, 6, 8, 9, 10, 11, 12, 13}.**

## QUESTION 2 [5 MARKS]

1. **Give some clear pseudocode for the Ford-Fulkerson method of finding the maximum flow and discuss its running time. (5 Marks)**

Pseudocode:

Initialise flow to 0

While there exists an augmenting path P

For each edge E

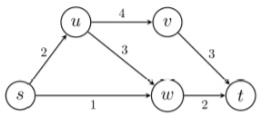
Compute the residual capacity

Augment each edge E and total flow F

Return f

The above algorithm runs at O(E\*F) where E = total number of edges and F = the maximum flow. Given no duplicate edges the worst case in each path we adjust the flow in every iteration

**b) Consider the following flow network G**



**Given an initial flow f with f(s, u) = f(u,w) = f(w, t) = 2, perform one iteration of Ford-Fulkerson; that is, draw the residual graph Gf , specify an augmenting path in Gf, and update the flow f. Is this new flow a maximum flow? Justify your answer.**

Network G after one iteration of Ford-Fulkerson:

Residual Capacity = f(s, u) = f(u, w) = f(w, t) = 2

Flow of Network = f(s, t) = 2

A close up of a map

Description automatically generated

Network G after two iteration of Ford-Fulkerson:

Residual Capacity = f(s, w) = f(w, u) = f(u, v) = f(v, t) = 1

Flow of Network = f(s, t) = 3

A picture containing map, clock, drawing

Description automatically generated

In the second step we make use of the non-empty backward edge (u,w) which has a non-negative flow f(u, w) != 0 to redistribute the f(u,w) = 2 to f(u, w) = 1 and f(u, v) = 1 allowing a total flow f(s, t) = 3. This final Maximum flow network satisfies all 3 constraints.

Since there are no residual capacities coming from the S node the new flow is a maximum flow.

## QUESTION 3 [10 MARKS]

**A queue (FIFO) can be implemented using two stacks (LIFO) where the stack has three operations, push, pop and empty, each with cost 1 and the queue can be implemented as:**

**- enqueue(x): push x onto stack1**

**- dequeue(): if stack2 is empty then pop the entire contents of stack1 pushing each element in turn onto stack 2. Now pop from stack2 and return the result.**

### Aggregate method:

Cost: Enqueue = 1 Dequeue = n + 1

Cursory Analysis:

Each dequeue would require a pop and push for each value giving a cost of n and adding one more pop to get the value with a cost of 1 setting a weak upper bound of .

Finding that each deque will not go n times according to the sequence we can show the amortized cost actual and that the actual cost is greater than amortized cost with the below equation:

or

### Accounting method

Cost:

Enqueue = $3

* $1 pays for x’s insertion into stack 1
* $1 pays for popping x off stack 1
* $1 pays for pushing x over to stack 2

Dequeue = $1

* $1 pays for popping the value off stack 2

|  |  |  |  |
| --- | --- | --- | --- |
| Balance: $2 | | Operation: Enqueue(a) | |
| S1 | a |  |  |
| S2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Balance: $4 | | Operation: Enqueue(b) | |
| S1 | a | b |  |
| S2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Balance: $6 | | Operation: Enqueue(c) | |
| S1 | a | b | c |
| S2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Balance: $ 0 | | Operation: Dequeue() Part A | |
| S1 |  |  |  |
| S2 | c | b | a |

|  |  |  |  |
| --- | --- | --- | --- |
| Balance: $ 1 | | Operation: Dequeue() Part B | |
| S1 |  |  |  |
| S2 | c | b |  |

### Potential Method

**Enqueue:**

Potential change

Amortized change or is it 3 because of the pushes

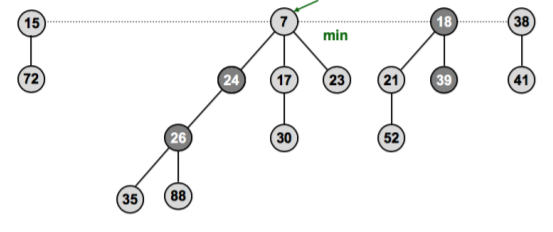
**Dequeue:**

Potential change is it negative n cause they are gone from stack or set – 1 because you only popped 1

Amortized change

## QUESTION 4 [5 MARKS]

**In the following Fibonacci Heap, decrease 35 to 5 and show the resulting Fibonacci Heap. Dark shaded nodes are marked.**



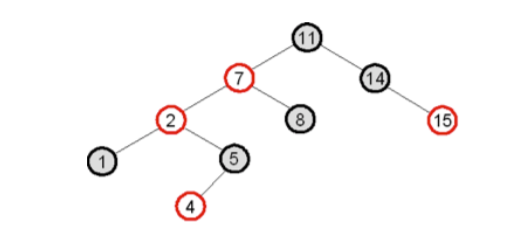
Find out if you only decrease key or a min pop / clean up is required

## QUESTION 5 [8 MARKS]

**a) List the six properties of a Red-Black Tree**

**b) Prove that a Red-Black Tree with n internal nodes has height at most 2 lg(n + 1).**

**c) What Red-Black Tree property is violated in the tree below (black nodes are shaded with bold outline). Describe how you would correct this and draw the resulting tree.**



**d) Show how a red-black tree develops as you insert the keys 41, 38, 31, 12, 19, 8 into an initially empty tree, then show the red-black trees as you successively delete the keys in the order 8, 12, 19, 31, 38, 41.**

## QUESTION 6 [5 MARKS]

**Step by step create the suffix tree for the word “banana”**

## QUESTION 7 [5 MARKS]

**An example of an augmented data structure is the Interval Tree.**

**a) Discuss the underlying data structure and the enhancements required to generate an Interval Tree.**

**b) Write the pseudo-code (Python level) for the IntervalSearch function.**

## QUESTION 8 [5 MARKS]

**Write the pseudo-code for the 2-approximation algorithm for the Travelling Salesman problem and prove that it is a polynomial 2-approximation algorithm.**

# LONG ANSWER

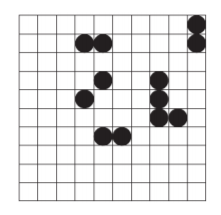
## QUESTION 1 [15 MARKS]

**Consider an N-by-N grid in which some squares are occupied by black circles. Two squares belong to the same group if they share a common edge. In the figure below, there is one group of four occupied squares, three groups of two occupied squares, and two individual occupied squares. Assume that the grid is represented by a two-dimensional array. Write a C++ program that does the following:**

**(a) Computes the size of a group when a square in the group is given.**

**(b) Computes the number of different groups.**

**(c) Lists all groups.**



## QUESTION 2 [15 MARKS]

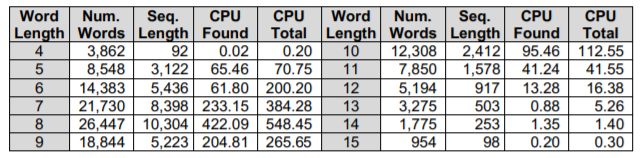
**Assignment 1 required you to write a program to generate a maze.**

**(a) Write a C++ program that computes enough information to output a path in the maze. Give output in the form SEN... (representing go south, then east, then north, etc.).**

**(b) Modify your program to find the shortest path in the maze.**

## QUESTION 3 [15 MARKS]

**The file dictionary.txt contains one word per line. Subsets of these words can ordered such that, with the exception of the first word, the second and third letter of each word is identical to the third last and second last of the preceding word. Words may only be used once within a sequence. Design algorithms and implement C++ programs that find such sequences separately for words of length 4 through to words of length 15 characters. Target sequence lengths and CPU times in seconds are shown in the following tables. You may be able to find longer sequences. Analyse in detail the problem and the performance of each of your algorithms.**



## QUESTION 4 [15 MARKS]

**The goal of a ladder-gram is to transform a source word into the target word on the bottom rung in the least number of steps. During each step, you must replace one letter in the previous word so that a new word is formed, but without changing the positions of the other letters. All words must exist in the supplied dictionary (dictionary.txt). For example, we can achieve the alchemist's dream of changing LEAD to GOLD in 3 steps or HIDE to SEEK in 6 steps. Minimise the number of steps.**

## QUESTION 5 [20 MARKS]

**The object of the Kevin Bacon Game is to link a movie actor to Kevin Bacon via shared movie roles. The minimum number of links is an actor’s Bacon number. For instance, Tom Hanks has a Bacon number of 1; he was in Apollo 13 with Kevin Bacon. Sally Fields has a Bacon number of 2, because she was in Forrest Gump with Tom Hanks, who was in Apollo 13 with Kevin Bacon. Almost all wellknown actors have a Bacon number of 1 or 2. Given a list of actors, with roles, write a C++ program that does the following:**

**(a) Finds an actor’s Bacon number.**

**(b) Finds the actor with the highest Bacon number.**

**(c) Finds the minimum number of links between two arbitrary actors.**

## QUESTION 6 [50 MARKS]

**Design an algorithm and write a program to identify the minimum vertex covers within the complement graphs of the supplied graphs. The table below shows the output from a current minimum vertex cover solver for these complement graphs.**

**(a) You must actually design your own algorithm and write all program code submitted. The program must be able to be run from the command line passing the target word length as an argument. The naming convention for your program file is \_mvc (without the <>) and try and keep the program within a single file.**

**(b) As this a computationally intensive algorithm, you must use C/C++, written using all possible optimisations.**

**(c) You must produce a detailed Latex/Word paper containing an Introduction, Literature Review, Algorithm Description, Experimental Results and Comparisons and a Conclusion.**

**(d) You must produce a Power-point presentation summarising your algorithms, implementation, testing and the results of the problem and performance analysis. Do not include any code in your powerpoint presentation**

