Student ID: ........................

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA

UNIVERSITY OF WELLINGTON

**EXAMINATIONS – 2015**

**TRIMESTER 1** *** **WITH SOLUTIONS** ***

# COMP 261 ALGORITHMS and DATA STRUCTURES

**Time Allowed:** TWO HOURS

**CLOSED BOOK**

**Permitted materials:** Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.

**Instructions:** [2mm]Attempt ALL Questions.

Answer in the appropriate boxes if possible — if you write your answer elsewhere, make it clear where your answer can be found.

The exam will be marked out of 120 marks.

Non-electronic foreign to English language dictionaries are permitted.

Alphabetic order: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

| Questions | Marks |
|---|---|
| 1. 3D Graphics | [20] |
| 2. Parsing | [25] |
| 3. Articulation Points | [20] |
| 4. B and B+ Trees | [25] |
| 5. String Searching | [5] |
| 6. Maximum Flow | [10] |
| 7. Compression | [15] |

## Question 1. 3D Graphics [20 marks]

**(a)** [10 marks] List all the steps in a 3D Rendering Algorithm (e.g. as described in the lectures) that utilised both the Z-Buffer and the Edge Lists and explain the purpose of each step. Use one sentence or so to describe each major step. Make sure to explain the role of the Edge Lists and the role of the Z-Buffer in the algorithm.

Input:

- set of polygons (position, colour)
- viewing direction
- direction of light source(s)
- size of window

Output: an image
Algorithm:

- rotate polygons and light sources so viewing along z axis
- translate and scale to fit window / clip polygons out of view
- remove any polygons facing away from viewer (normal Z ¿ 0)
- for each polygon
  - compute shading
  - work out which image pixels it will affect (this is where Edge Lists come in to work out which pixels are inside the triangle/polygon)
  - for each pixel write shading and depth to z buffer (retains only the shading of the closest surface - this is the role of the Z-Buffer!)
- convert z-buffer to image

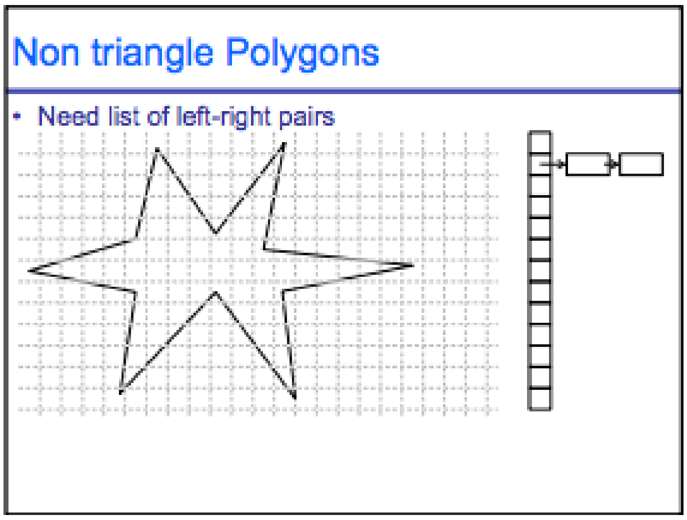
NB! The above is almost verbatim from the lecture slide (e.g. in lecture 13 on 3D graphics part 1 of 2) so extra explanations and clarification would earn more marks to make up for the case if you miss any bits above.

(Question 1 continued on next page)

**(Question 1 continued)**

**(b)** [5 marks] We used triangles as polygons in our lectures and assignment. What changes would be required to the rendering algorithm to make it work for more general polygons?

Main Idea: Edge list needs to have multiple entries, not just two. We discussed it at the end of the last lecture on 3d rendering (lecture 16). So for example see this picture:

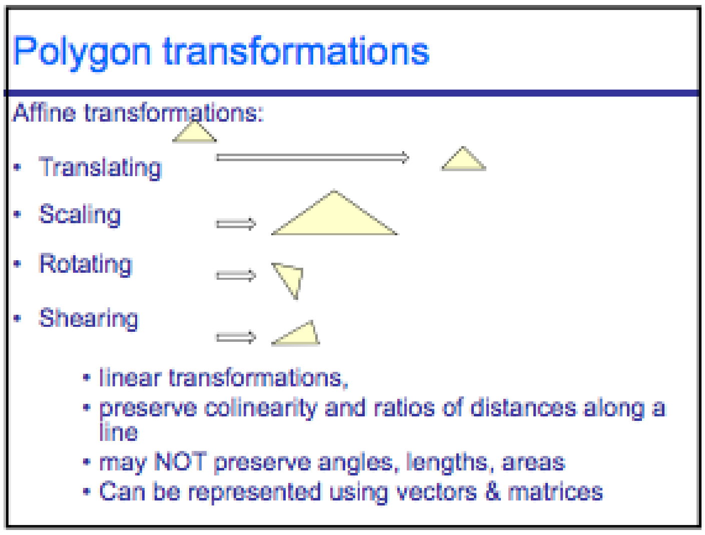


## Non triangle Polygons

- Need list of left-right pairs

**(c)** [5 marks] Explain what *affine* transformation means and why they are preferred in computer graphics.

As in lectures: because you can represent them using matrices and combine them together they are really easy to work with on modern GPU's.



## Polygon transformations

Affine transformations:

- Translating
- Scaling
- Rotating
- Shearing

- linear transformations,
- preserve colinearity and ratios of distances along a line
- may NOT preserve angles, lengths, areas
- Can be represented using vectors & matrices

## Question 2. Parsing [25 marks]

**(a)** [5 marks] Describe the difference between Concrete Parse Tree and Abstract Syntax Tree in parsing.

CPT created to exactly match the grammar. You can read off the original sentence by traversing the tree depth first and reading off the leaves. AST is without unneeded information to make it easy to work with and evaluate.

**(b)** [10 marks] For each of the following sentences, circle Y or N (for Yes or No) to state whether it belongs to the language defined by the following grammar. Nonterminals are in uppercase and terminals are in lowercase. Assume that there are going to be no spaces between terminals and nonterminals (they are spaced out in the grammar just for readability). The text is processed one character at a time (even though `java.util.Scannner` doesn't have such method and one needs to use Strings and Patterns, assume for this assignment that `Scanner.nextChar() and Scanner.hasNextChar(char c)` methods exist). Note that nonterminal RETXT uses a regular expression to express the kinds of terminals it accepts.

```
WEIRDLANG ::= r FOO | t BAR | RETXT BAZ | h BAZ
FOO ::= g | h | BAZ
BAR ::= t RETXT
BAZ ::= te RETXT
RETXT ::= gh+zy*g+
```

```
Y    N    yes    rg

Y    N    no     rteghhhzgggg

Y    N    yes    hteghzg

Y    N    no     ghhhzyyygteghzyyy

Y    N    yes    ttghhhzyg
```

(Question 2 continued on next page)

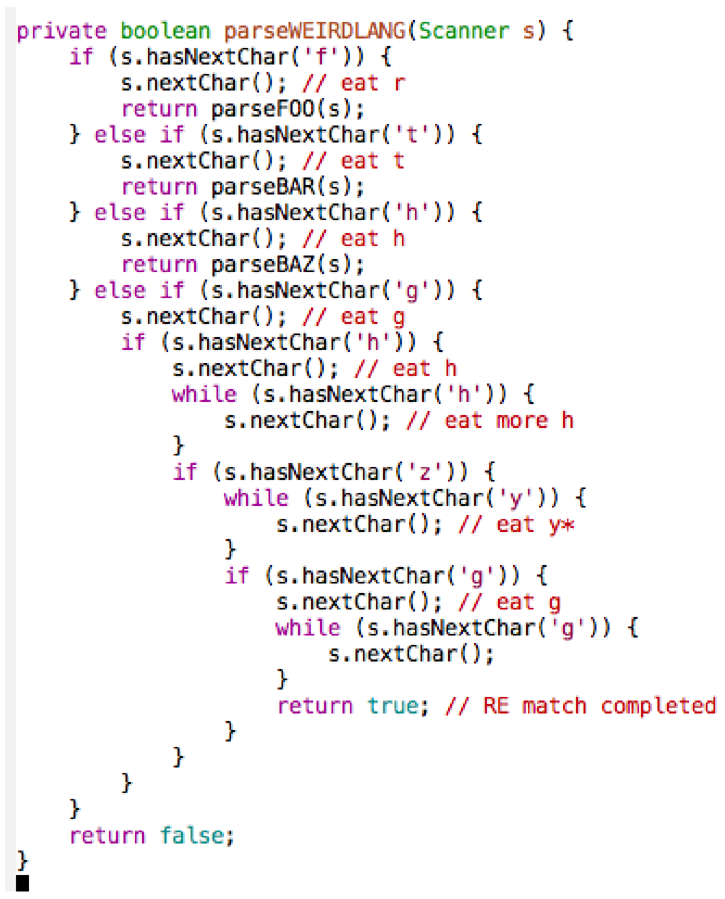
**(Question 2 continued)**

**(c)** [10 marks] Implement the `parseWEIRDLANG` method from the grammar on page 4.

**Important Note:** The text is processed one character at a time (even though `java.util.Scannner` doesn't have such method and one needs to use Strings and Patterns, assume for this exam question that `Scanner.nextChar()` and `Scanner.hasNextChar(char c)` methods exist). Also, assume that the other `parse` methods (e.g. `parseFOO`) are implemented for you and you can use them.

```
    private boolean parseWEIRDLANG(Scanner s){




    }
```

Bonus marks if they use patterns and strings, otherwise using nextChar (which doesn't exist in real life):



```java
private boolean parseWEIRDLANG(Scanner s) {
    if (s.hasNextChar('f')) {
        s.nextChar(); // eat r
        return parseFOO(s);
    } else if (s.hasNextChar('t')) {
        s.nextChar(); // eat t
        return parseBAR(s);
    } else if (s.hasNextChar('h')) {
        s.nextChar(); // eat h
        return parseBAZ(s);
    } else if (s.hasNextChar('g')) {
        s.nextChar(); // eat g
        if (s.hasNextChar('h')) {
            s.nextChar(); // eat h
            while (s.hasNextChar('h')) {
                s.nextChar(); // eat more h
            }
            if (s.hasNextChar('z')) {
                while (s.hasNextChar('y')) {
                    s.nextChar(); // eat y*
                }
                if (s.hasNextChar('g')) {
                    s.nextChar(); // eat g
                    while (s.hasNextChar('g')) {
                        s.nextChar();
                    }
                    return true; // RE match completed
                }
            }
        }
    }
    return false;
}
```

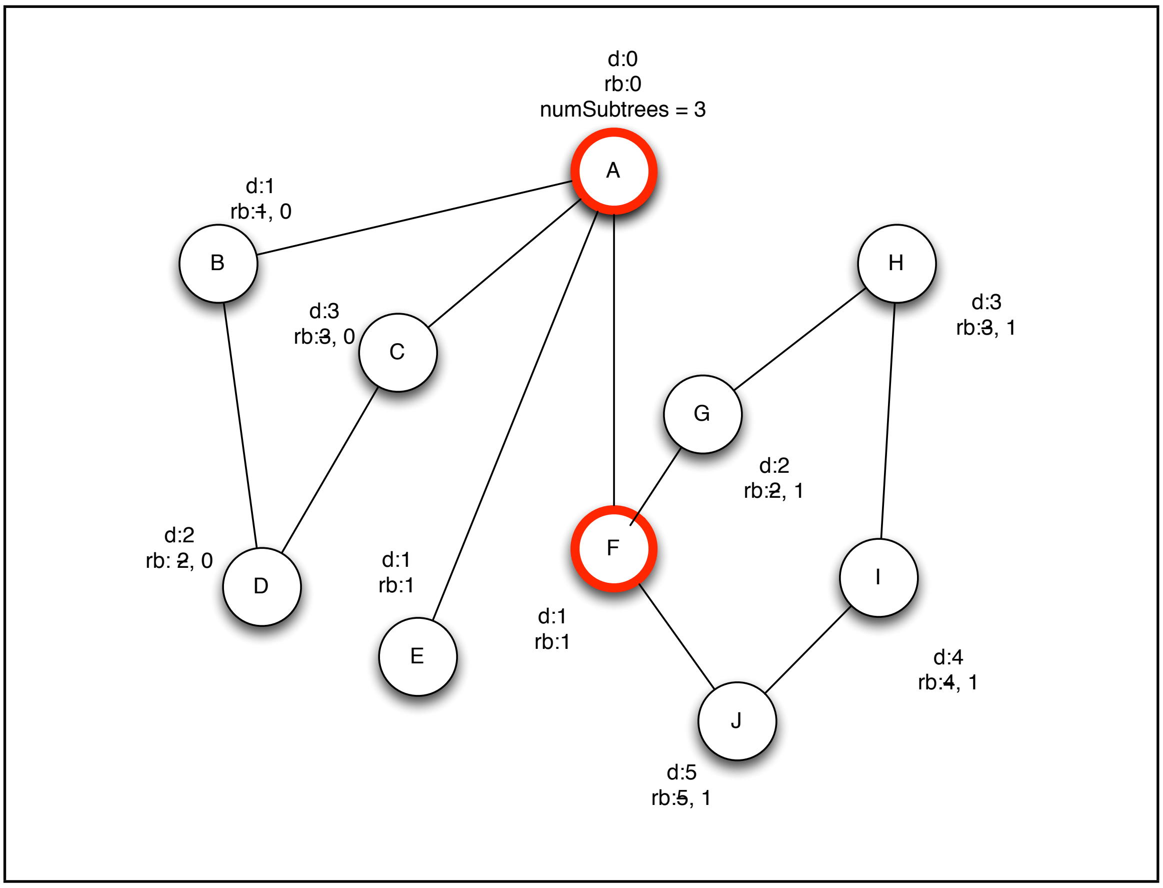
## Question 3. Articulation Points [20 marks]

**(a)** [12 marks] Recursive Articulation Points

Show how the *Recursive Depth-First-Search (DFS) Articulation Points* algorithm would find the articulation points in the following graph, assuming that the search starts with node A and considers neighbours of nodes in alphabetical order. Mark the following on the graph

- the depth ("d:") and the reach-back ("rb:") of each node (node A is done for you). If the reach-back is updated, give all the values e.g "rb: 3,1"
- the return values from each recursive call.
- the nodes that are articulation points, indicating why they were marked.

The algorithm is given on the facing page for your reference.



(Question 3 continued on next page)

**(Question 3 continued)**

## Recursive DFS Articulation Points Algorithm

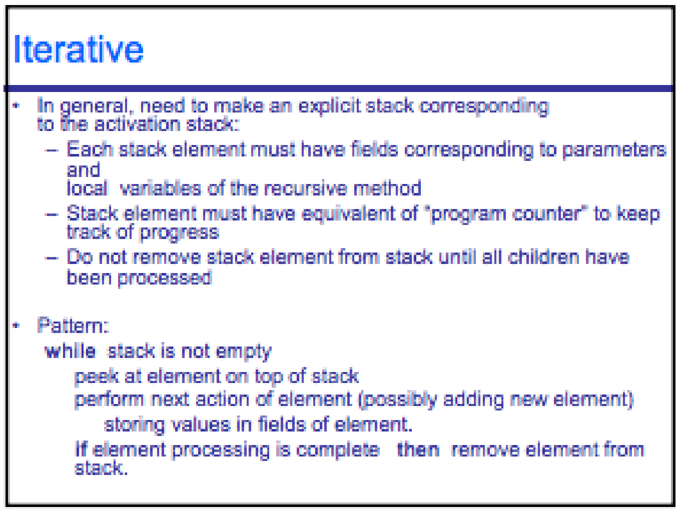
```
FindArticulationPoints (graph, start ):
    for each node: node.depth $\leftarrow \infty$, articulationPoints $\leftarrow \{\ \}$
    start .depth $\leftarrow$ 0, numSubtrees $\leftarrow$ 0
    for each neighbour of start
        if neighbour.depth = $\infty$ then
            RecursiveArtPts( neighbour, 1, start )
            numSubtrees ++
    if numSubtrees $>$ 1 then add start to articulationPoints

RecursiveArtPts(node, depth, fromNode):
    node.depth $\leftarrow$ depth, reachBack $\leftarrow$ depth,
    for each neighbour of node other than fromNode
        if neighbour.depth $< \infty$ then
            reachBack $\leftarrow$ min(neighbour.depth, reachBack)
        else
            childReach $\leftarrow$ recArtPts(neighbour, depth +1, node)
            reachBack $\leftarrow$ min(childReach, reachBack )
            if childReach $\geq$ depth then add node to articulationPoints
    return reachBack
```

(Question 3 continued on next page)

**(Question 3 continued)**

**(b)** [8 marks] Iterative Articulation Points

Describe the changes required to turn the *Recursive Depth-First-Search (DFS) Articulation Points* algorithm into the ***Iterative*** *Depth-First-Search (DFS) Articulation Points* algorithm. Provide an outline of the ***Iterative*** *Depth-First-Search (DFS) Articulation Points* algorithm.



## Iterative

- In general, need to make an explicit stack corresponding to the activation stack:
  - Each stack element must have fields corresponding to parameters and local variables of the recursive method
  - Stack element must have equivalent of "program counter" to keep track of progress
  - Do not remove stack element from stack until all children have been processed
- Pattern:

```
while stack is not empty
    peek at element on top of stack
    perform next action of element (possibly adding new element)
        storing values in fields of element.
    if element processing is complete   then  remove element from
    stack.
```

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

## Question 4. B and B+ Trees [25 marks]

**(a)** [10 marks] A 2-3 tree is a B-tree with a Max Degree $m = 3$. Here is the definition in Wikipedia:

> In computer science, a 2-3 tree is a tree data structure, where every node with children (internal node) has either two children (2-node) and one data element or three children (3-node) and two data elements. Nodes on the outside of the tree (leaf nodes) have no children and one or two data elements.

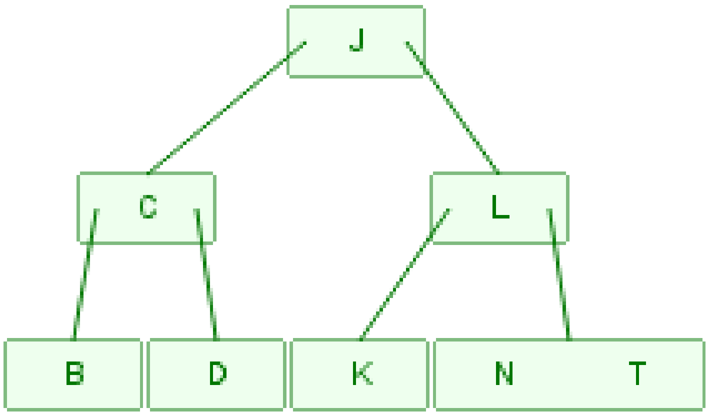
Construct a 2-3 tree by inserting the following key values in this order:

L, J, B, K, C, T, N, D

Show your working and make sure your answer includes a clear copy of the final tree. State any assumptions you have made.

Alphabetic order: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Note: Only the final tree is shown below



(Question 4 continued on next page)

**(Question 4 continued)**

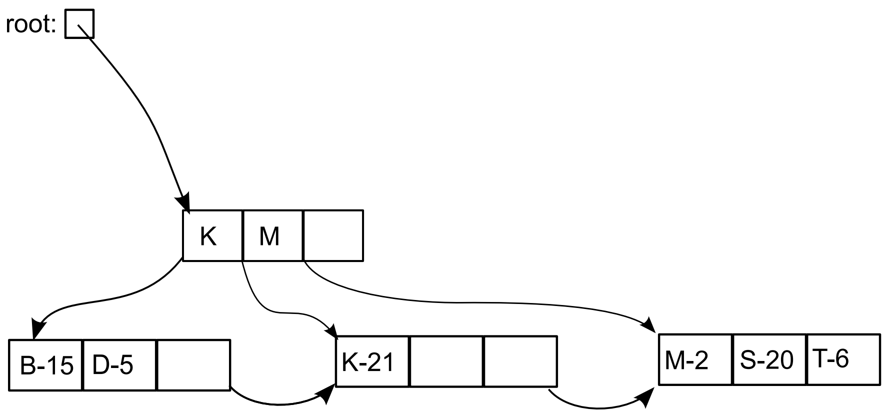
**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

(Question 4 continued on next page)

**(Question 4 continued)**

**(b)** [10 marks] The following figure shows a B+ tree that has internal nodes holding up to 3 keys, and leaf nodes holding up to 3 key-value pairs. The keys are letters; the values are numbers. The tree is created by inserting these items in order: K-21, S-20, B-15, M-2, D-5, T-6. When a node is split, it is split 1-1-2 with the 2nd key being promoted.



This question concerns a new B+ tree with the same tree structure (internal nodes holding up to 3 keys, leaf nodes holding up to 3 key-value pairs, and node splitting at 1-1-2). You are required to create the new B+ tree from scratch by inserting the following items in this order:

P-2, K-1, B-7, M-6, D-3, T-5, G-8, A-11, R-9

Show your working and make sure your answer includes a clear copy of your final tree. State any assumptions you have made.

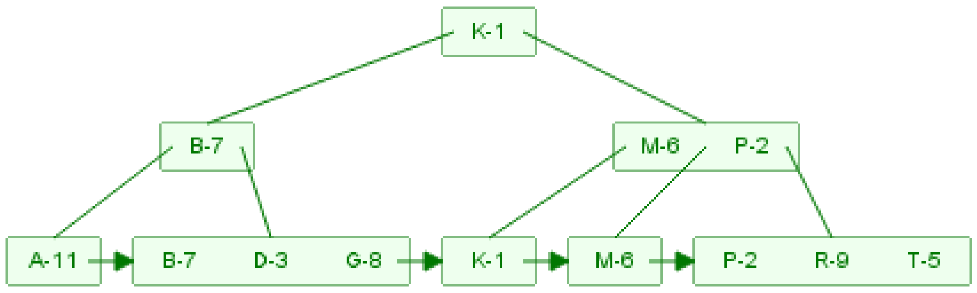
*The answer box is on the next page.*

(Question 4 continued on next page)

**(Question 4 continued)**

Alphabetic order: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Note: only final tree is shown below.
It should be built from scratch, splitting at 1-1-2.
(Partial marks are given for adding the items to the tree on previous page, also for spliting at 2-1-1.)
Note: the internal node should not have the values, should only contain the keys



(Question 4 continued on next page)

**(Question 4 continued)**

**(c)** [5 marks] What are the main differences between a B tree and a B+tree? Include in your answer an explanation of *why is B+ tree usually a better choice*.

B+ trees have all data at leaf level, with links to next, so traversal is very easy.
B trees store both key and value in internal and leaf nodes
B+trees store key value pairs in leaf node. B+ trees only store keys in internal nodes, so it can store more keys in internal nodes and this is helpful when the value is big. Normally B+ tree is "busher" and the search is more efficient.

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

## Question 5. String Searching [5 marks]

The Knuth Morris Pratt (KMP) algorithm searches for a string in a piece of text. It first analyses the string and builds a table, which enables KMP to search more efficiently than the naive string search algorithm.

Show the tables that the following KMP table building algorithm would construct for the string: "`whatwhywhatwhy`".

```
computeKMPTable(string)
    initialise  table  to  an array  of  integers
    table [0]  ← −1;      table [1]  ← 0;
    pos ← 2;              j  ← 0;
    while  pos < string.length
        if  string [pos−1] = string[j ]
            table [pos] ← j+1
            pos++
            j++
        else if   j  > 0
            j  ← table[ j ]
        else
            table [pos] ← 0
            pos++
    return table
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| w | h | a | t | w | h | y | w | h | a | t | w | h | y |
| -1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | 5 |

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

## Question 6. Maximum Flow [10 marks]

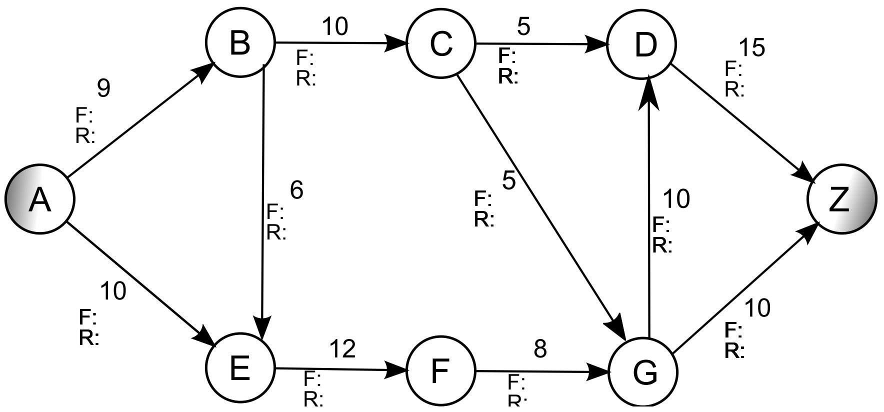
Show how the Edmonds-Karp algorithm for Maximum Flow would find the maximum flow through the graph shown on the next page from the node A to the node Z. Each edge is labeled with its capacity.

1. Label each edge with its flow and its remaining capacity. Show how the flow and remaining capacity change during the algorithm.
2. Below the graph, show the path found in each iteration, along with the flow that can be added along that path.
3. Show the maximum flow from A to Z found by the algorithm.

Hint: Remember that Edmonds-Karp repeatedly uses breadth first search to find a path from source to sink in which every edge has non-zero remaining capacity, sets the new flow to the minimum remaining capacity along the path, and adds this flow to the flow on each edge of the path (and subtracts this flow from each of the reverse edges).

(Question 6 continued on next page)

**(Question 6 continued)**



Note : the flow, remaining flow in the graph should be uppdated

| Path | Flow added along path |
|---|---|
| A B C D Z | 5 |
| A E F G Z | 8 |
| A B C G Z | 2 |
| A B C G D Z | 2 |

if you happened to try any path with B to E first,
then you might need to use phantom edges to reverse it.

Maximum Flow = 17

## Question 7. Compression [15 marks]

**(a)** [5 marks] What is the key idea of Lempel-Ziv 77 compression algorithm? Please explain using an example.

Eliminate repeating patterns
“ababc” is [0,0, a][0,0,b][2,2,c]

**(b)** [5 marks] Construct a trie that can be used to encode the following sentence using Huffman Encoding. (Note: label the links of the trie, not the node.)

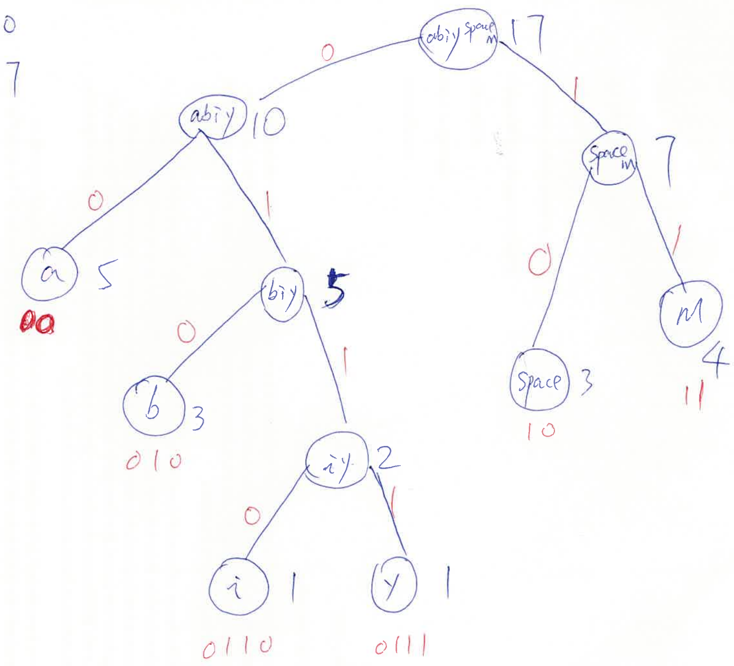
```
mamma mia by abba
```

<abiy 10

<space_m 7

<biy 5

| | |
|---|---|
| a | 5 |
| m | 4 |
| Space | 3 |
| b | 3 |
| <iy | 2 |
| i | 1 |
| y | 1 |



(Question 7 continued on next page)

**(Question 7 continued)**

**(c)** [5 marks] Show the resulting encoding and state how many bits will be required to store the message from part (b) of this question:

```
mamma mia by abba
```

a: 00
m: 11
space:10
b:010
i:0110
y:0111

11 00 11 11 00 10
11 0110 00 10
010 0111 10
010 0110 10
00 010 010 00

41 bits

* * * * * * * * * * * * * * *