Student ID: . . . .

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2013

TRIMESTER 2

COMP 261
ALGORITHMS
and
DATA STRUCTURES

Time Allowed: THREE HOURS

Instructions: Closed Book

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 180 marks.

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

Non-electronic foreign 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. Tries	[10]
2. Articulation Points	[20]
3. Graphics	[30]
4. Graph Algorithms	[34]
5. Parsing	[40]
6. B+ Trees	[26]
7. Maximum Flow	[20]

Question 1. Tries [10 marks]

(a) [5 marks] Draw a Trie containing the following set of strings.
(Note: label the links of the Trie, not the nodes.)

```
small    mall     all      malt     smile    smiles
snapped  snack    ma       smack    snap
```

(Question 1 continued on next page)

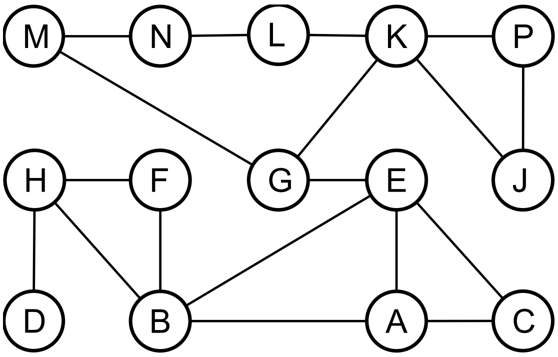
(Question 1 continued)

(b) [5 marks] Give an example of a problem where using a Trie would be a more efficient implementation of a set of strings than a HashSet, and explain why the Trie would be more efficient.

Question 2. Articulation Points [20 marks]

(a) [4 marks] Briefly explain the problem that the Articulation Points algorithm solves.

(b) [4 marks] Circle the articulation points in the following graph.



(c) [4 marks] Outline two examples of applications or problems where the Articulation Points algorithm would be useful.

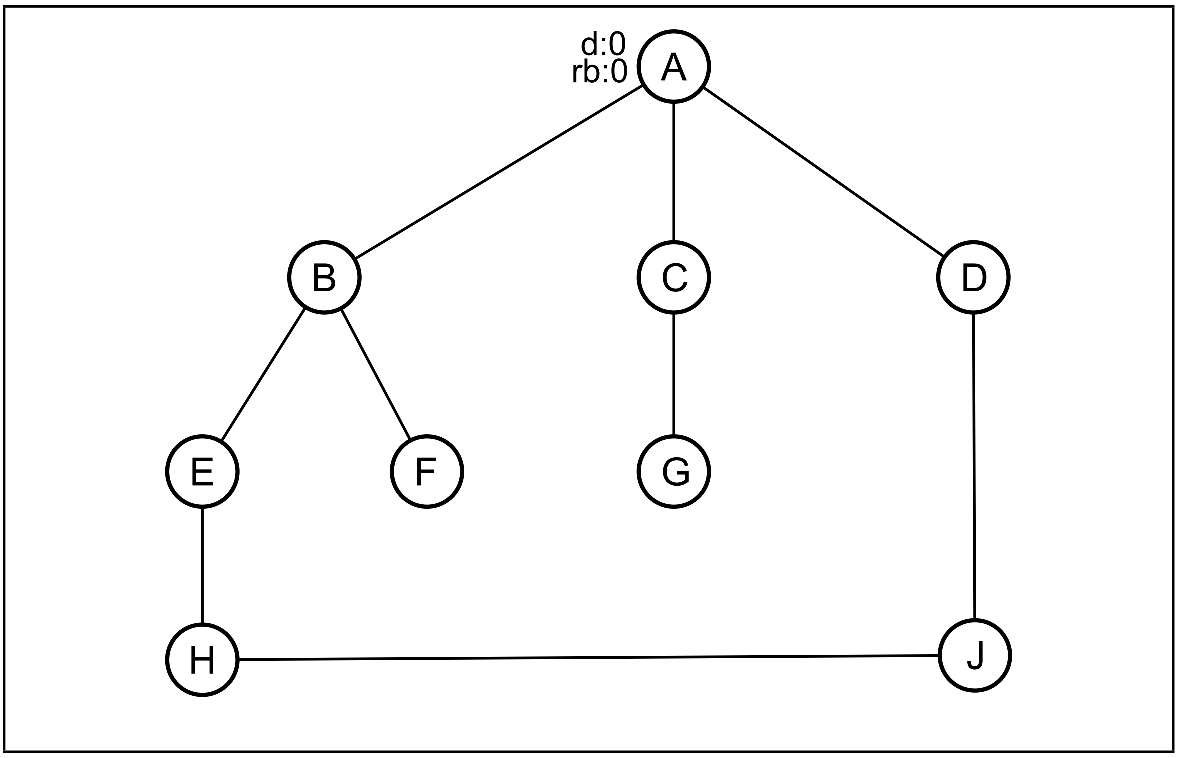
(Question 2 continued on next page)

(Question 2 continued)

(d) [8 marks] Show how the recursive depth-first-search 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 on the graph

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

The algorithm is given below for your reference.



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

RecursiveArtPts(node, depth, fromNode):
    node.depth ← depth, reachBack ← depth,
    for each neighbour of node other than fromNode
        if neighbour.depth < ∞ then
            reachBack ← min(neighbour.depth, reachBack)
        else
            childReach ← recArtPts(neighbour, depth +1, node)
            reachBack ← min(childReach, reachBack )
            if childReach ≥ depth then add node to articulationPoints
    return reachBack
```

Question 3. Graphics [30 marks]

(a) [6 marks] To represent the 3D transformations for the graphics algorithms, we used 4×4 matrices and 4D vectors for the points, even though the points were in 3D space. Explain the advantages of this representation.

(b) [6 marks] The 3D rendering algorithm in the lectures constructed edgelists as part of the process of interpolation to compute the depth (z) at each pixel position on the polygon. Explain the role of the edgelists in the algorithm.

(Question 3 continued on next page)

(Question 3 continued)

(c) [8 marks] Suppose your Z-buffer rendering program has processed some of the polygons in a model, and is now processing a new polygon. The colour of the new polygon is blue. Part of the edgelists for the polygon are:

	left		right	
y	x	z	x	z
20	11	20	16	30
21	10	17	15	27

The current contents of the Z-buffer are the following. Each cell of the Z-buffer contains a colour and a *z* value (*z* increases away from the viewer).

y \ x	10	11	12	13	14	15	16
20	gray ∞	gray ∞	gray ∞	red 27	red 24	red 21	gray ∞
21	green 20	green 20	green 20	gray ∞	yellow 28	yellow 26	yellow 24

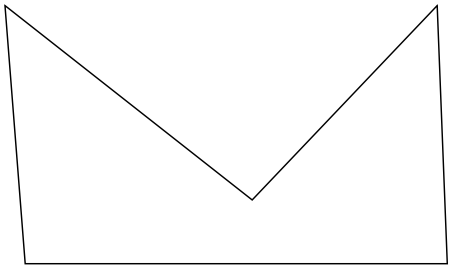
Show the contents of the Z-buffer after your program has rendered the edgelist data into the Z-buffer. State any assumptions you are making.

y \ x	10	11	12	13	14	15	16
20							
21							

(Question 3 continued on next page)

(Question 3 continued)

(d) [5 marks] When all the polygons in a 3D model are triangles, the edgelists just contain two x values (left and right) for each value of y. Explain, using the following diagram, why the edgelists have to hold more values if the polygon can have more than three sides.



(Question 3 continued on next page)

(Question 3 continued)

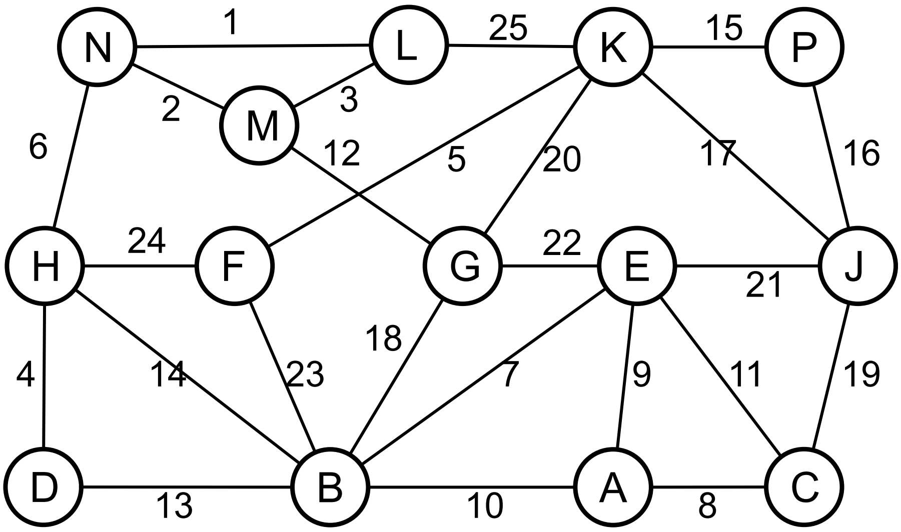
(e) [5 marks] The algorithm for rendering 3D models presented in the lectures produced images showing the individual polygons as flat faces with sharp edges. Explain what would be required to render the model as a smoothly rounded surface.

Question 4. Graph Algorithms [34 marks]

(a) [8 marks] Show how Kruskal's Minimum Spanning Tree algorithm will find a minimum spanning tree for the following graph by marking the edges that the algorithm will add to the spanning tree, and listing the edges in the order they are added.

Note: All the edges have different weights, from 1 to 25. You can list the edges by giving their weights.

Reminder: Kruskal's MST algorithm builds up sets of connected nodes, in contrast to Prim's algorithm which builds outwards from a starting node.



Edges added:

(Question 4 continued)

(b) [12 marks] Write pseudocode for Kruskal's Minimum Spanning Tree algorithm.

(Question 4 continued on next page)

(Question 4 continued)

(c) [8 marks] Explain why the union-find data structure is important for the efficiency of Kruskal's algorithm.

(Question 4 continued on next page)

(Question 4 continued)

(d) [6 marks] Explain how the use of a heuristic can make A* search more efficient than Dijkstra's algorithm for finding the shortest path between two nodes in a graph, and discuss the properties that the heuristic should have.

Question 5. Parsing [40 marks]

Consider the following grammar for part of a simple scripting language for a stock trading system :

```
SCRIPT ::= "when" CODE "hits" NUMBER "{" [ CMD ]+ "}"
CMD ::=  SELL  |  BUY  |  SCRIPT
SELL ::= "sell" NUMBER CODE
BUY ::= "buy" NUMBER CODE
CODE ::=  three or four alphabetic characters
NUMBER ::=  a positive integer
```

Note: `[ CMD ]+` means one or more repetitions of CMD.

(a) [7 marks] Give a concrete syntax tree for the following script

```
when MSFT hits 3975 { sell 1000 AAPL buy 2000 IBM }
```

(Question 5 continued on next page)

(Question 5 continued)

(b) [7 marks] Give an abstract syntax tree for the following (different) script, and give a brief justification for what you included and what you ignored in the abstract syntax tree.

```
when AAPL hits 49500 { when IBM hits 18100 { sell 2000 IBM }
                       buy 1000 MSFT }
```

Justification:

(Question 5 continued on next page)

(Question 5 continued)

(c) [20 marks] Suppose you are writing a recursive descent parser for the script language and you have definitions of the classes, patterns, and methods given below.

Complete the three methods, parseCmd, parseBuy and parseScript on the next two pages. All three have a Scanner parameter, and all three should return an appropriate Node. If the Scanner does not contain a valid CMD, BUY, or SCRIPT, the methods should throw an exception using the fail method. (Explanatory error messages are not necessary.)

Assume that the Scanner has an appropriate delimiter for separating the tokens.

Your methods may (if you wish) call the gobble, fail, and parseSell methods given below.

```
interface Node {}

class ScriptNode implements Node{
    private String code;
    private int price;
    private List<Node> cmds;
    public ScriptNode(String cd, int pr, List<Node> cs){ ... }
}
class BuyNode implements Node{
    private String code;
    private int count;
    public BuyNode(String cd, int ct){code=cd;count=ct;}
}
class SellNode implements Node{
    private String code;
    private int count;
    public SellNode(String cd, int ct){code=cd;count=ct;}
}

private String CodePat = "[A-Z][A-Z][A-Z][A-Z]?";
private String NumPat = "[1-9][0-9]+";
private String OpenPat = "\\{";
private String ClosePat = "\\}";

private boolean gobble(String pat, Scanner s){
     if (s.hasNext(pat)) {s.next(); return true;}
    return false;
}
private void fail (Scanner s){
   throw new RuntimeException("Parser error at " + s.next());
}
// Parse Methods
private Node parseSell(Scanner s){ ... } //parses a Sell command
private Node parseBuy(Scanner s){ ... } //complete on facing page
private Node parseCmd(Scanner s){ ... } //complete on facing page
private Node parseScript(Scanner s){ ... } //complete on following page.
```

(Question 5 continued on next page)

(Question 5 continued)

The grammar (repeated):

```
SCRIPT ::= "when" CODE "hits" NUMBER "{" [ CMD ]+ "}"
CMD ::=  SELL  |  BUY  |  SCRIPT
SELL ::= "sell" NUMBER CODE
BUY ::= "buy" NUMBER CODE
CODE ::=  three or four alphabetic characters
NUMBER ::=  a positive integer
```

```
    private Node parseBuy(Scanner s){





    }
    private Node parseCmd(Scanner s){





    }


 // Complete parseScript on the next page.
```

(Question 5 continued on next page)

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 continued)

The grammar (repeated):

```
   SCRIPT ::= "when" CODE "hits" NUMBER "{" [ CMD ]+ "}"
   CMD ::=  SELL  |  BUY  |  SCRIPT
   SELL ::= "sell" NUMBER CODE
   BUY ::= "buy" NUMBER CODE
   CODE ::=  three or four alphabetic characters
   NUMBER ::=  a positive integer
```

```
    private Node parseScript(Scanner s){

    }
```

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 continued)

(d) [6 marks] Suppose we wanted to be able to specify that commands must be performed concurrently (specified by ``and'') or one after the other (specified by ``then''). For example, we might want to write:

```
when MSFT hits 3975 { sell 1000 AAPL then sell 2000 IBM
                      and buy 500 HPQ and buy 1000 DELL }
```

A candidate grammar to allow this might be the following:

```
SCRIPT ::= "when" CODE "hits" NUMBER "{" CMDS "}"

CMDS   ::=  CMD  | CMD  "and" CMDS | CMDS "then" CMDS

CMD    ::=  SELL  |  BUY  |  SCRIPT
SELL   ::= "sell" NUMBER CODE
BUY    ::= "buy" NUMBER CODE
CODE   ::=  three or four alphabetic characters
NUMBER ::=  a positive integer
```

Explain why it would now be problematic to construct a recursive descent parser for this grammar, and suggest how you might start addressing the problem(s).

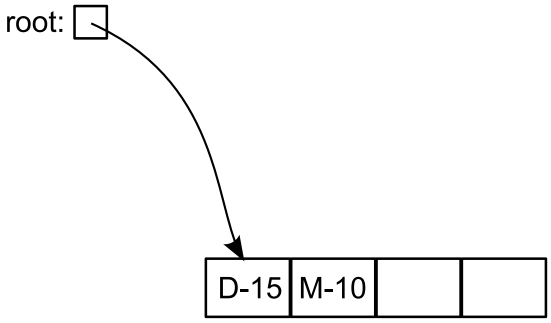
Question 6. B+ Trees [26 marks]

The following subquestions concern a B+ tree that has internal nodes holding up to 3 keys, and leaf nodes holding up to 4 key-value pairs. The keys are letters; the values are numbers.

(a) [5 marks] Show how the B+ tree below would be changed if the following key-value pairs were added to it:

```
    H-5  F-7 K-22 N-12
```

State any assumptions you are making.



Assumptions (if any):

Alphabet: 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

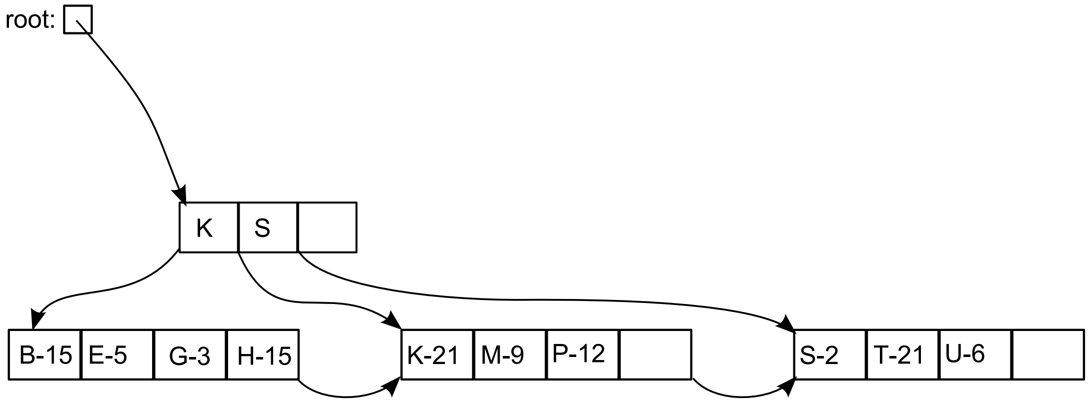
(Question 6 continued on next page)

(Question 6 continued)

(b) [5 marks] Show how the B+ tree below would be changed if the following key-value pairs were added to it:

```
    N-5  Q-18 V-2
```

State any assumptions you are making.



Assumptions (if any):

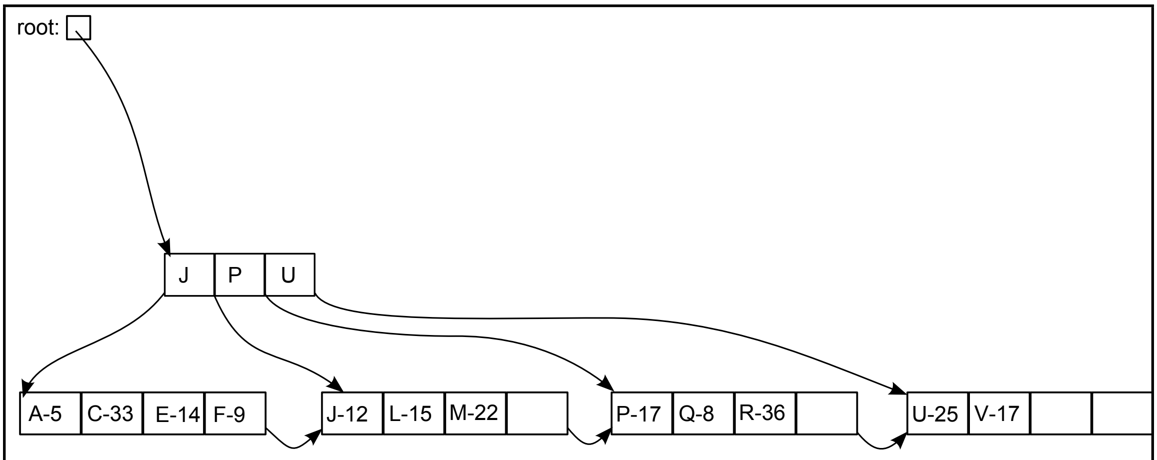
(Question 6 continued on next page)

(Question 6 continued)

(c) [5 marks] Show how the B+ tree below would be changed if the following key-value pair were added to it:

    D-1

State any assumptions you are making.



Assumptions (if any):

(Question 6 continued on next page)

(Question 6 continued)

(d) [5 marks] Suppose the internal nodes of a B+ tree have at most 6 children (*i.e.*, 5 keys), and the leaves contain at most 4 key-value pairs. The maximum number of key value pairs that can be in a tree is 4×6^h if the height of the tree is h. What is the minimum number of key-value pairs that can be in tree of height h? Assume that $h \geq 1$. Show your working!
Note: the height of a tree is the number of edges (= number of internal nodes) on a path from the root to a leaf.

(Question 6 continued on next page)

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 continued)

(e) [6 marks] Suppose you are implementing a B+ tree in a file, storing each node of the tree in one block. The keys of the B+ tree are share market codes (for example "AAPL"), and the values are company names. Each share market code is up to 6 characters long, and the company names are limited to 66 characters. The blocks are 512 bytes long.

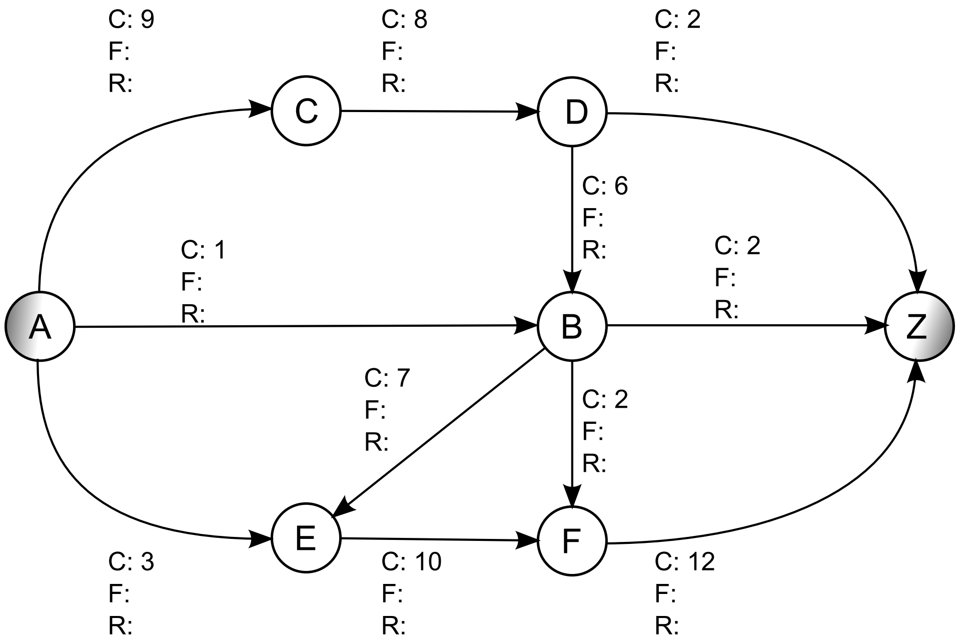
How many key-value pairs can be stored in a leaf node? Explain your reasoning, show your working, and state any assumptions you make about the information stored in the blocks.

Question 7. Maximum Flow [20 marks]

(a) [13 marks] Show how the Edmonds-Karp algorithm for Maximum Flow would find the maximum flow through the following graph from the node A to the node Z. Each edge is labeled with its capacity, its flow, and its remaining capacity.

1. 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 from each of the reverse edges).



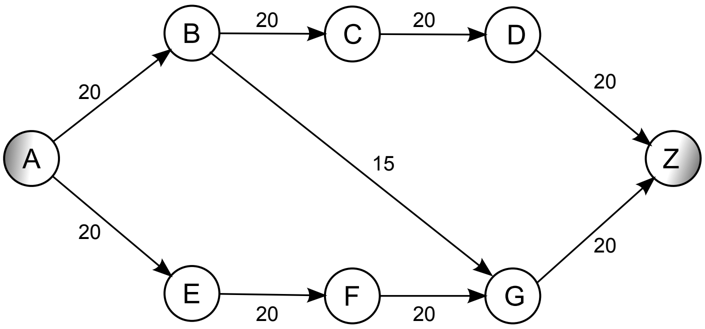
Path Flow added along path

Maximum Flow =

(Question 7 continued on next page)

(Question 7 continued)

(b) [7 marks] The graph in part (a) did not need to use any "phantom edges" (though it was not wrong to use them). This is not always the case. Using the following graph as an example, explain why the Edmonds-Karp algorithm needs "phantom edges", and explain how they are used in the algorithm.



****