

Family Name:

Other Names:

Signature:

ID Number:

COMP261: Test

16 April 2014

Instructions

- Time allowed: **45 minutes** .
- Answer **all** the questions. There are 45 marks in total.
- Write your answers in the boxes in this test paper and hand in all sheets.
- If you think some question is unclear, ask for clarification.
- This test contributes 15% of your final grade
- You may use paper translation dictionaries, and non-programmable calculators.
- You may write notes and working on this paper, but make sure your answers are clear.

Questions

Marks

1. Data structures

[10]

2. A* search

[10]

3. Minimum spanning tree

[5]

4. Graphics

[15]

5. Articulation points

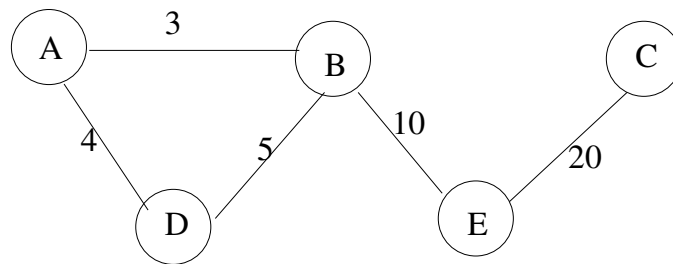
[5]

TOTAL:

Question 1. Data structures

[10 marks]

(a) [5 marks] Show an adjacency matrix representing the following graph.



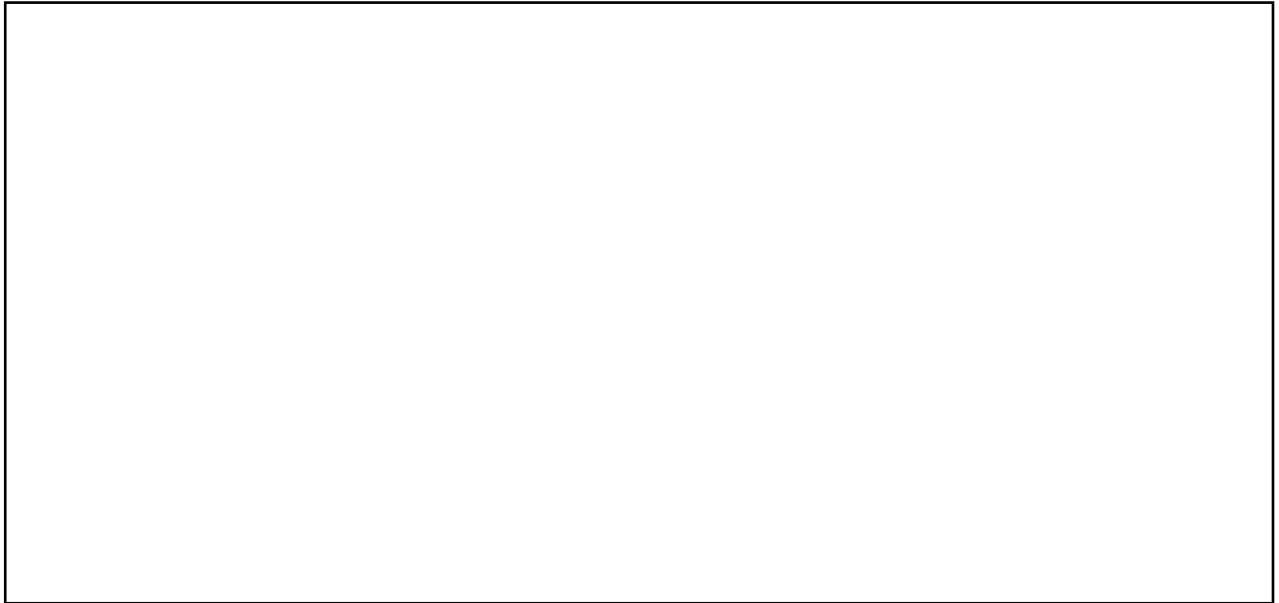
--

(Question 1 continued on next page)

(Question 1 continued)

(b) [5 marks] Show a Trie containing the following set of strings.

apple	all	bat	ball	base
ballet	bit	wise	west	wall



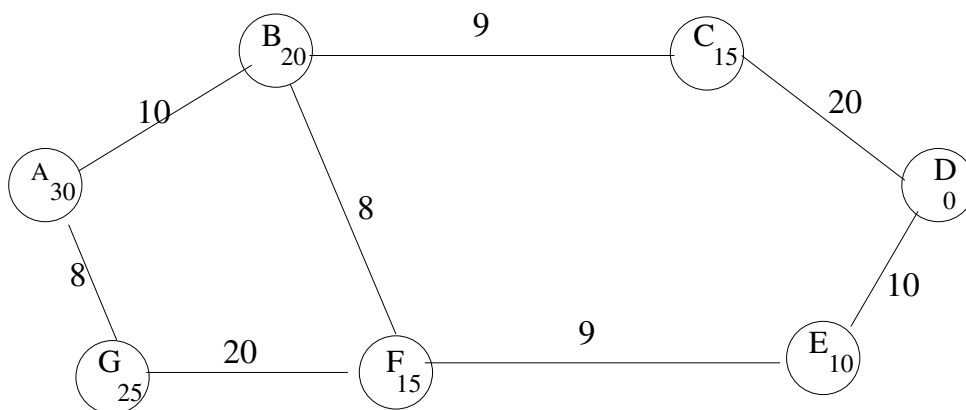
Question 2. A* search

[10 marks]

Show how A* finds the shortest path from **A** to **D** in the weighted undirected graph below, where the heuristic estimate of the cost from each node to the goal **D** is shown in the node. You may assume that the heuristic estimate is both admissible and consistent.

On the facing page, you should show the queue and the solution:

- list each element that is added to the priority queue, showing the node, from-node, cost to node, and total estimated cost.
(The first element put on and removed from the priority queue is shown as an example.)
- add neighbours in alphabetical order
- indicate the order the elements are removed from the priority queue (eg, by numbering them).
- list the nodes of the shortest path found.



Elements of the queue:

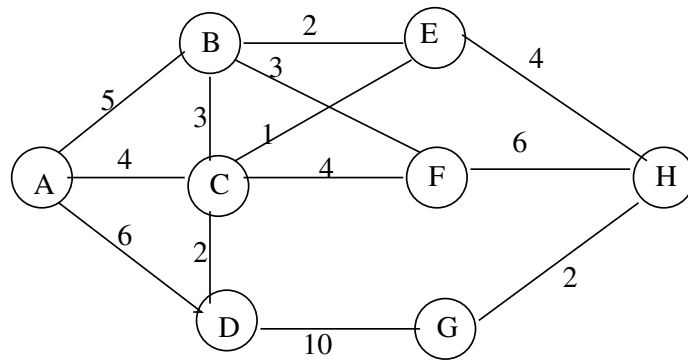
[Node ← From: Cost, TotEst] When removed

[A ← null: 0, 30] #1

The Shortest Path Found:

Question 3. Minimum spanning tree

[5 marks]



Suppose you are using Prim's algorithm to search for the minimum spanning tree in the graph and you start from node A. Show the order in which nodes/edges will be added to the tree.

Question 4. Graphics**[15 marks]****(a) [2 marks]**

The normal is used for which of these purposes?

- a) testing if a polygon is backfacing
- b) shading
- c) computing depth
- d) determining if two polygons overlap in the edgelist

Write the letter corresponding to the correct answer(s):

(b) [1 mark]

In the Z-buffer algorithm, the Z (depth) test is done

- a) once per object
- b) once per polygon
- c) once per pixel
- d) The advantage of this algorithm is that it does not need to test depth

Write the letter corresponding to the correct answer:

(c) [2 marks]

If the "handedness" (right-hand versus left-hand convention) is inconsistent in various parts of your program, which of these might happen:

- a) you see the inside or back side of objects
- b) objects are a reflected or mirror image of what they should be
- c) the Z-buffer test will fail

(d) [2 marks]

Suppose that the particular image being rendered does not contain overlapping objects. In other words, there are no objects in the 3D scene that are in front of and hiding other objects when viewed from the camera position. Which would be a correct characterization of the complexity of the rendering algorithm described in class:

- a) it is linear in the number of pixels
- b) it is $O(N^2)$ in the number of polygons
- c) it is $O(N \cdot \log N)$ in the number of polygons
- d) the computation time cannot be characterised in this way

Write the letter corresponding to the correct answer(s):

(e) [1 mark] Suppose a particular vertex located at (1,1,1) is transformed by the following matrix. What is the new position of the vertex?

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Write the 3D position of the transformed vertex in the box below:

(f) [2 marks] Which of these is a reasonable diffuse shading model? In these descriptions, $*$ is regular multiplication, \cdot is the dot product, and \times is the cross product.

- a) color = reflectance + normal \cdot light + ambient
- b) color = reflectance + normal \times light + ambient
- c) color = reflectance $*$ (normal \cdot light + ambient)
- d) color = (reflectance + normal \times light) $*$ ambient

Write the answer in the box below:

(g) [5 marks] The following pseudocode gives the algorithm for processing triangles into the Z-buffer. However, one very important step is missing.

initialise :

Zbuffer.c = array [0.. imageWidth] [0..imageHeight] of *Color*
 Zbuffer.d = array [0.. imageWidth] [0..imageHeight] of *infinity*

for each polygon **do**

 compute edge lists EL, and shading

 (EL is an array of (xleft, zleft, xright, zright))

for y = 0 to edgelist.length-1 **do**

 x = round(EL[y].xleft), z = EL[y].zleft

 mz = (EL[y].zright - EL[y].zleft) / (EL[y].xright - EL[y].xleft)

while x <= round(EL[y].xright) **do**

 // check if x is in bounds somewhere here

if z < Zbuffer.d[x][y] **then**

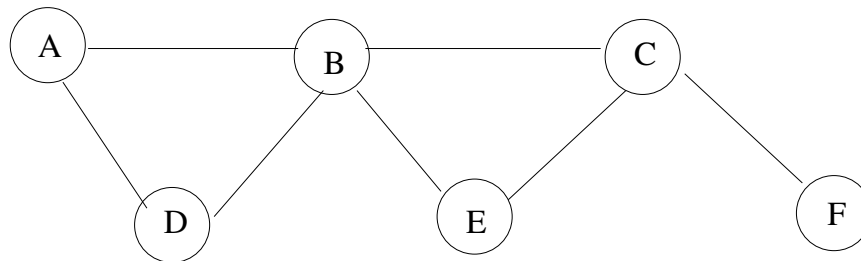
 Zbuffer.c[x][y] = shading

 x++, z = z + mz

Please describe what is missing:

Question 5. Articulation points

[5 marks]



What are the articulation points in this graph? Suppose you are using the recursive version of the articulation points detection algorithm shown below and you pick node **A** as the root node. Show how this algorithm identifies the articulation points. Your answers should include the depth and the reach back of each node. You may make your own assumptions if necessary.

Initialise : **for** each node: node.depth = infinite , articulationPoints = { }
start.depth = 0, numSubtrees = 0

```
for each neighbour of start
  if neighbour.depth == infinite then
    recArtPts( neighbour, 1, start )
    numSubtrees ++
  if numSubtrees > 1 then add start to articulationPoints
```

```
recArtPts(node, depth, fromNode):
  node.depth = depth, reachBack = depth,
  for each neighbour of node other than fromNode
    if neighbour.depth < infinite 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
```

Student ID:

