

**EXAMINATIONS — 2010**

**END-OF-YEAR**

**COMP 261  
ALGORITHMS  
and  
DATA STRUCTURES**

**Time Allowed:** 3 Hours **\*\*\*\*\* WITH SOLUTIONS \*\*\*\*\***

**Instructions:** 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.

Non-programmable calculators without a full alphabetic key pad are permitted.

Non-electronic foreign language dictionaries are permitted.

Useful formulas are listed on the last page of the exam.

| <b>Questions</b>               | <b>Marks</b> |
|--------------------------------|--------------|
| 1. File Structures             | [22]         |
| 2. Union Find                  | [17]         |
| 3. Searching in Graphs         | [26]         |
| 4. String Searching            | [16]         |
| 5. Graphics Rendering          | [9]          |
| 6. Advanced Tree Structures    | [30]         |
| 7. Text Processing             | [40]         |
| 8. File Structures and Hashing | [20]         |

## ANSWERS

### Question 1. File Structures

[22 marks]

Suppose a file contains 100,000 fixed length records describing individual students. Each record has the following fields:

**StudentID:** (length = 5 characters),

**Name:** (length = 50 characters),

**Address:** (length = 120 characters),

**DoB:** (length = 10 characters).

Assume that the file blocks are stored contiguously and that the block size for the file is 1024 characters.

(a) [4 marks] Calculate the record size  $L$  in characters. Show your working.

record has  $(5 + 50 + 120 + 10) = 185$  characters,  
1 byte per character.  
therefore 185 bytes

(b) [4 marks] Calculate the blocking factor  $f$  and the number of file blocks  $b$ . Assume an unspanned file organisation. Show your working.

Blocking factor =  $\lfloor 1024/185 \rfloor = 5$   
Number of blocks =  $100,000/5 = 20,000$  blocks

(c) [4 marks] Calculate the *average* number of block accesses needed to perform a linear search for a random record in the file given its StudentID. Show your working.

Assume have to look at half the records on average = 50,000 records  
Number of blocks =  $50,000/5 = 10,000$  blocks

(Question 1 continued on next page)

**(Question 1 continued)**

(d) [4 marks] Calculate the *worst case* number of block accesses needed to perform a binary search for a random record in the file given its StudentID. Assume the file is ordered by StudentID. Show your working.

May have to look at  $\lfloor \log_2(100,000) \rfloor + 1 = 17$  records.  
Assume that every record we look at is in a different block from the previous one, and that we don't remember earlier blocks, then need 17 blocks.  
Note that it is not possible for the last three records you look at to be in more than two different blocks.

(e) [6 marks] Explain the differences between primary file organisation and secondary file organisation.

Primary file organisation is the way the records are organised in the file on the disk. Secondary file organisation is the additional indexing structures that enable fast access to items in the file, but is not part of the file itself.

## ANSWERS

### Question 2. Union Find

[17 marks]

(a) [4 marks] Explain why Kruskal's algorithm for finding minimum spanning trees needs to keep track of a set of sets of nodes.

Kruskal

An Efficient Union-Find algorithm:

---

MakeSet(x):

    x.parent  $\leftarrow$  x

    x.rank  $\leftarrow$  0

    add x to collection of sets.

Find(x):

**if** x.parent = x **then return** x

**else**

        x.parent  $\leftarrow$  Find(x.parent)

**return** x.parent

Union(x, y):

    xroot  $\leftarrow$  Find(x)

    yroot  $\leftarrow$  Find(y)

**if** xroot.rank < yroot.rank **then**

        xroot.parent  $\leftarrow$  yroot

        remove xroot from collection of sets.

**else**

        yroot.parent  $\leftarrow$  xroot

        remove yroot from collection of sets.

**if** xroot.rank = yroot.rank **then**

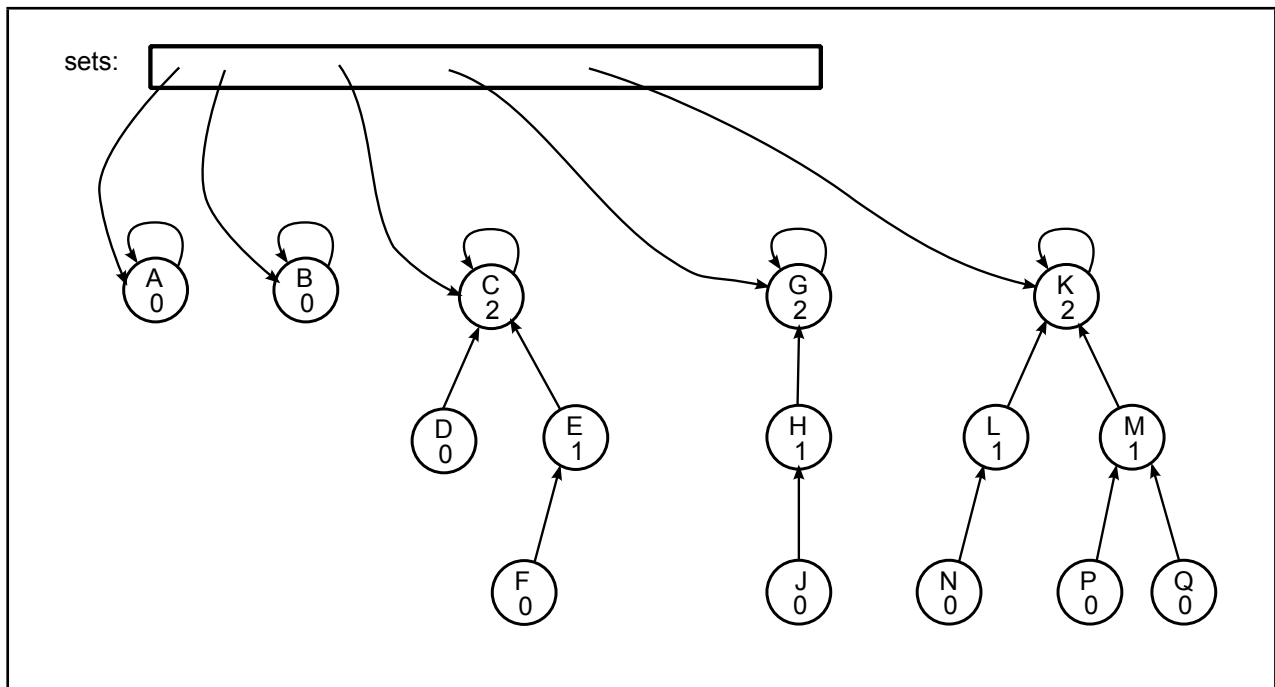
        xroot.rank++

---

(Question 2 continued on next page)

**(Question 2 continued)**

**(b)** [8 marks] Consider the following diagram of a collection of four sets represented using the Union-Find data structure. The numbers are the ranks of the nodes.



Using the efficient Union-Find algorithm on the facing page, show the changes that will be made to the graph by the following *sequence* of operations.

- (i) Find(Q)
- (ii) Union(A, B)
- (iii) Union(B, F)
- (iv) Find(J)
- (vi) Union(N, J)

Draw the changes on the diagram above.

Hint: note the order of the arguments of Union carefully.

(Question 2 continued on next page)

**(Question 2 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 2 continued on next page)

**(Question 2 continued)**

(c) [5 marks] The efficient Union-Find algorithm moves nodes around in the tree when searching for a value in order to make future searches in the tree more efficient. Could a similar technique be used for improving binary search trees? If you think it could, briefly suggest how and the conditions under which it might help. If you think it couldn't, explain why not.

Yes. Every time you find a value in the binary search tree, you could move it to the top of the tree. That involves changing all the nodes of the tree along the path to the node so that they become descendants of the node instead of ancestors. But it is possible, and only nodes along the path need to be changed.

It would help if a few nodes in the tree are searched for much more frequently than the rest of the nodes, because these few nodes would tend to be near the top of the tree. However, it wouldn't help (and would cost a lot more because of all the tree restructuring) if all nodes were roughly equally likely to be searched for.

## ANSWERS

### Question 3. Searching in Graphs

[26 marks]

(a) [5 marks] Prim's algorithm for minimum spanning trees is very similar to Dijkstra's algorithm for single source shortest paths. Explain the key way in which Prim's algorithm differs from Dijkstra's algorithm, and explain why this difference results in two different trees.

When choosing the next node to explore, Dijkstra's algorithm chooses the unvisited node with the shortest length path to the node from the start node; Prim's algorithm just chooses the unvisited node with the shortest edge to the node. This means that Dijkstra's algorithm always finds the shortest path to each node, whereas Prim's algorithm will happily add a very short edge to a node, even if there is a much shorter path to the node if that path includes longer edges.

(b) [5 marks] Give an example of a small graph, including a start node and a goal node, and an *inadmissible* heuristic estimate of remaining path length for each of the nodes and show how A\* would find the wrong path to the goal node using this heuristic. Show the path that A\* would find and show the shortest path that it should have found.

Note: show all the edge lengths and show the heuristic estimate on each node.

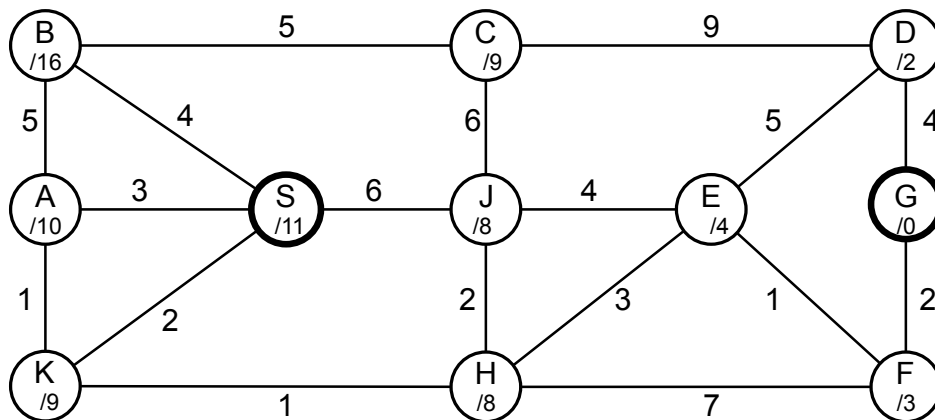


(Question 3 continued on next page)



**(Question 3 continued)**

(c) [8 marks] Suppose you are using A\* to search for the shortest path from **S** to **G** in the graph below, where the heuristic estimate for each node is shown in the node (the heuristic is admissible). Show the order in which nodes will be *added* to the queue, and the order in which they are *removed* from the queue. (When visiting a node, consider the neighbours of the node in alphabetic order.)  
Hint: keep track of the queue, along with the total path length (path so far plus heuristic estimate) for all the nodes on the queue.



Added to Queue: [shown as Node/priority(pathLength+estimate)]  
 [initialise:] S/11(0+11)  
 [from S:] A/13(3+10), B/20(4+16), J/14(6+8), K/11(2+9)  
 [from K:] A/13(3+10), H/11(3+8)  
 [from H:] E/10(6+4), F/15(10+5), J/13(5+8)  
 [from E:] D/13(11+2), F/10(7+3)  
 [from F:] G/9(9+0), (H is visited)

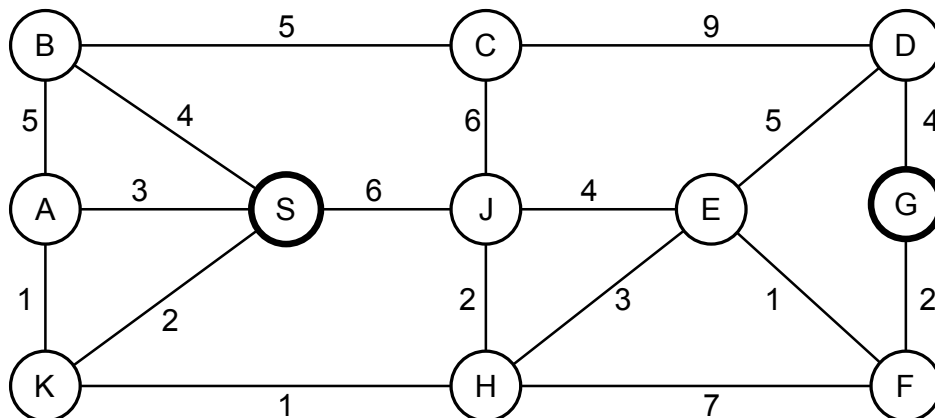
Removed from Queue: S/11, K/11, H/11, E/10, F/10, G/9

(Question 3 continued on next page)

**(Question 3 continued)**

**(d)** [8 marks] Suppose you are using Dijkstra's algorithm to find the shortest path from **S** to **G** in the graph below (the same as in part (c), but without heuristic estimates). Show the order in which nodes will be *added* to the queue, and the order in which they are *removed* from the queue. (When visiting a node, consider the neighbours of the node in alphabetic order.)

Hint: keep track of the queue, along with the priority for all the nodes on the queue.



Added to Queue: [shown as Node/priority]

[initialise:] S/0

[from S:] A/3, B/4, J/6, K/2

[from K:] A/3, H/3

[from A:] B/8, (K visited)

[from H:] E/6, F/10, J/5

[from B:] C/9

[from J:] C/11, E/9, (H visited)

[from E:] D/11, F/7, (J visited)

[from F:] G/9

Removed from Queue:

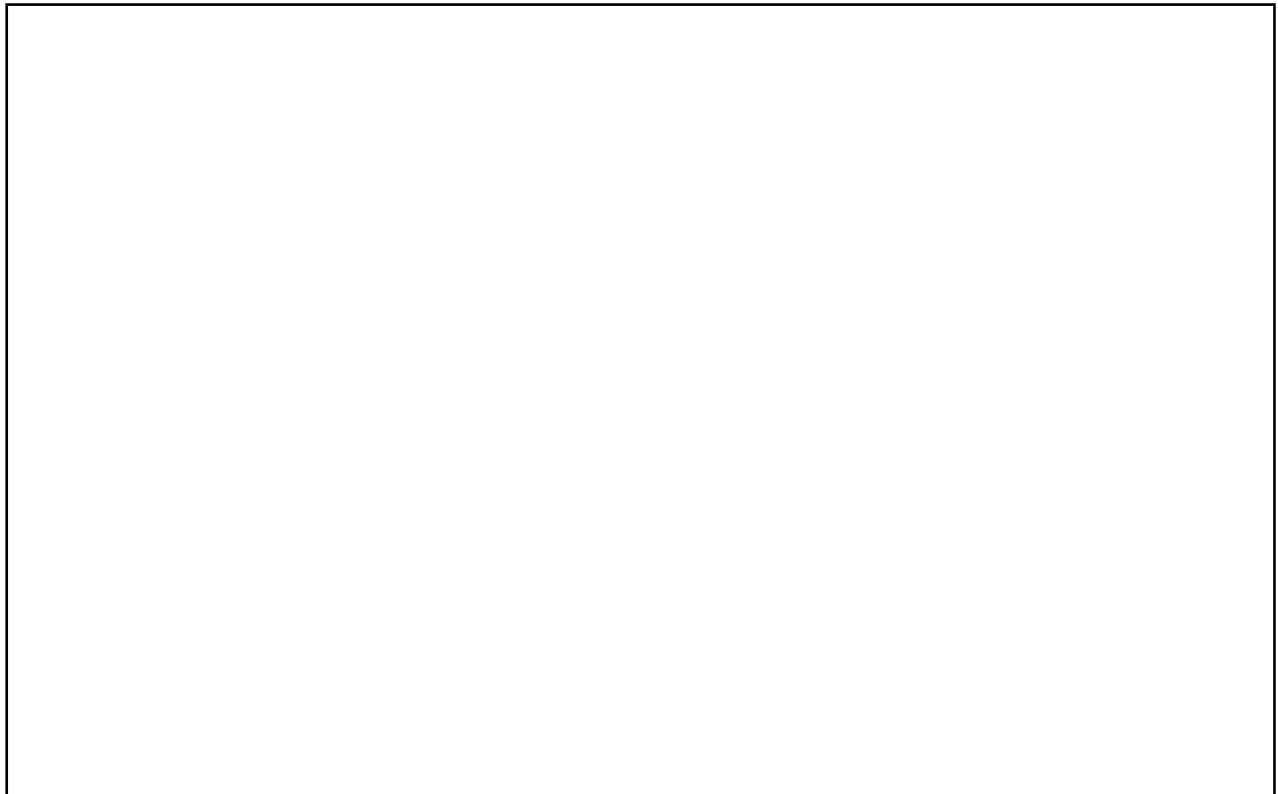
S/0, K/2, A/3, H/3, B/4, J/5, E/6, F/7, B/8(visited), G/9

**ANSWERS****Question 4. String Searching****[16 marks]**

**(a)** [5 marks] To search a text for an occurrence of any of a set of words, it is efficient to construct a trie of the words. Draw a trie for the following set of words.

Note: The characters should be attached to the edges in the trie, and all terminal nodes should be indicated clearly.

|       |         |        |        |      |        |
|-------|---------|--------|--------|------|--------|
| red   | strike  | sell   | search | read | sea    |
| ready | strings | strict | reach  | seat | string |



(Question 4 continued on next page)

**(Question 4 continued)**

The simplified Boyer-Moore string searching algorithm shown on the facing page constructs a “Bad-Character” table and then uses it while matching to work out how far to move the string forward when there is a mismatch. It does not use a “GoodSuffix” table.

If the string being searched for is `agkga`, the BadCharacter table would be the following:

BC:

|     |     |   |   |   |   |   |   |   |   |   |   |   |   |     |
|-----|-----|---|---|---|---|---|---|---|---|---|---|---|---|-----|
| 0   | 5   | 5 | 5 | 5 | 5 | 1 | 5 | 5 | 5 | 2 | 5 | 5 | 5 | ... |
| 'a' | 'b' | c | d | e | f | g | h | i | j | k | l | m | n | ... |

**(b) [11 marks]** Show the first 8 sets of values of  $i$ ,  $k$ , and  $T[i]$  at the point marked \*\*\* (the test in the while loop) in the simplified Boyer-Moore algorithm when it is used to search for the string `agkga` in the following text. (The first set of values is given.)

a g k b b a g k g k k g a g k g a m c d k g a a k a  
0                      5                      10                      15                      20                      25

| <u>i</u> | <u>k</u> | <u>T[i]</u> |
|----------|----------|-------------|
| 4        | 4        | b           |
| 9        | 4        | k           |
| 11       | 4        | g           |
| 12       | 4        | a           |
| 11       | 3        | g           |
| 10       | 2        | k           |
| 9        | 1        | k           |
| 13       | 4        | g           |

(Question 4 continued on next page)

**(Question 4 continued)**

A simplified version of the Boyer-Moore algorithm which only uses the “BadCharacter” table, not the “goodSuffix” table.

---

Boyer–Moore Search:

**Input:** *string*  $S[0 \dots m-1]$ ,  
           *text*  $T[0 \dots n-1]$

**Output:** the position in  $T$  at which  $S$  is found, or  $-1$  if not present

**Variables:**  $i$                // position of current character in  $T$   
                $k$                // position of current character in  $S$   
                $BC[char]$  // Bad Character table

**Actions:**

$BC \leftarrow \text{computeBadCharTable}(S)$

$i \leftarrow m-1$

**while**  $i < n$  **do**

$k \leftarrow m-1$

**while**  $k \geq 0$  and  $S[k] = T[i]$  **do** \*\*\* // record  $i$ ,  $k$ , and  $T[i]$  at this point

$i \leftarrow i-1$ ,

$k \leftarrow k-1$

**if**  $k = -1$  **then return**  $i+1$

$i \leftarrow i + \max(BC[T[i]], m-k)$        // mismatch  $\Leftarrow$  advance

**return**  $-1$

---

### **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.

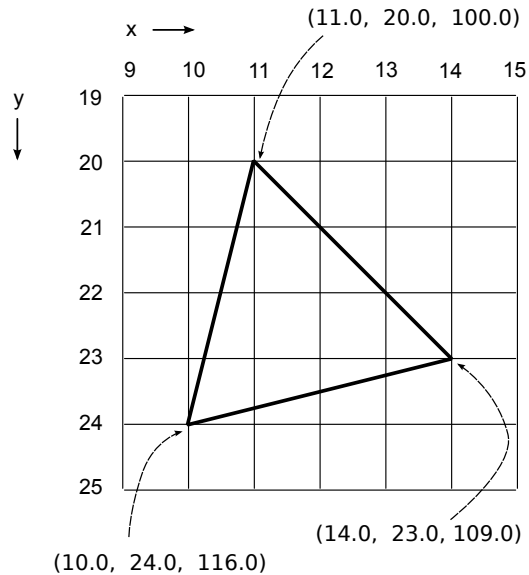
## ANSWERS

## Question 5. Graphics Rendering

[9 marks]

(a) [6 marks] Show the values in the edge-lists that would be constructed when rendering the following polygon. The  $(x, y, z)$  coordinates of the vertices are shown.

(Note that the  $z$  values are chosen carefully to make the interpolation easy.)



| Edge-Lists |                   |                   |                    |                    |
|------------|-------------------|-------------------|--------------------|--------------------|
|            | $x_{\text{left}}$ | $z_{\text{left}}$ | $x_{\text{right}}$ | $z_{\text{right}}$ |
| 20         |                   |                   |                    |                    |
| 21         |                   |                   |                    |                    |
| 22         |                   |                   |                    |                    |
| 23         |                   |                   |                    |                    |
| 24         |                   |                   |                    |                    |

(b) [3 marks] In constructing and using the edge-lists, you had to convert floating point numbers to integers. Explain why this can introduce errors unless you are careful.

The interpolation process can introduce very small errors. This can mean that two numbers that are extremely close (eg 12.999999 and 13.000001, or 22.49999 and 22.500001) may be converted to different integers, producing “holes” or artifacts in the images, where a pixel is missed out or an extra pixel is added. It doesn’t matter whether you round numbers to the closest integer, or use floor and ceil to round up or round down, two numbers just either side of the decision value can be converted to different integers.

## ANSWERS

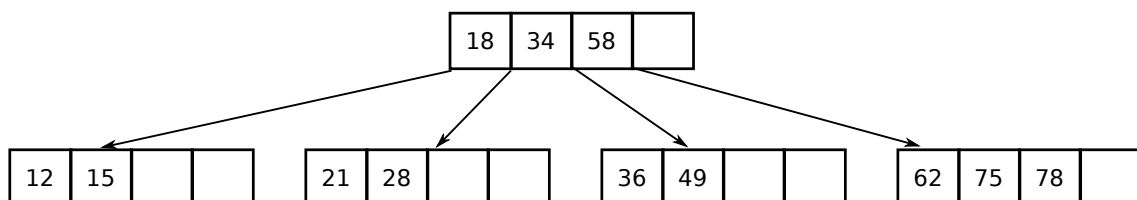
### Question 6. B-Trees

[30 marks]

(a) [10 marks] State the constraints that must be maintained by a  $B$ -tree with order  $p = 2m + 1$  and height  $h$ . Indicate which constraints ensure that a  $B$ -tree will be kept well-balanced.

.

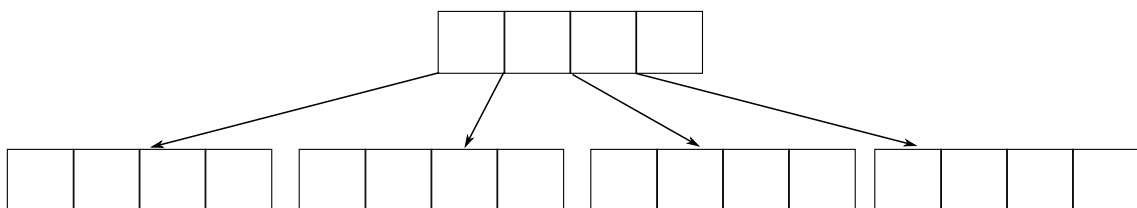
(b) [10 marks] Consider the  $B$ -tree of order 5 illustrated below.



Update the  $B$ -tree by successively deleting the key values 75, 15, 62, 21. In your answer, show the  $B$ -tree after each deletion and briefly describe what you have done.

Note, the empty trees below are to save you time; you may modify their structure if you choose.

The  $B$ -tree after deleting key value 75:

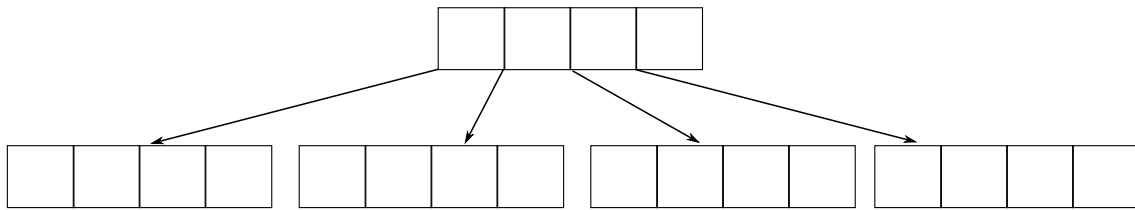


(Question 6 continued on next page)

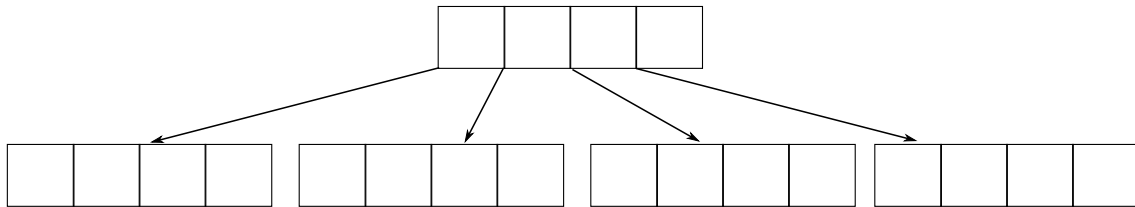


**(Question 6 continued)**

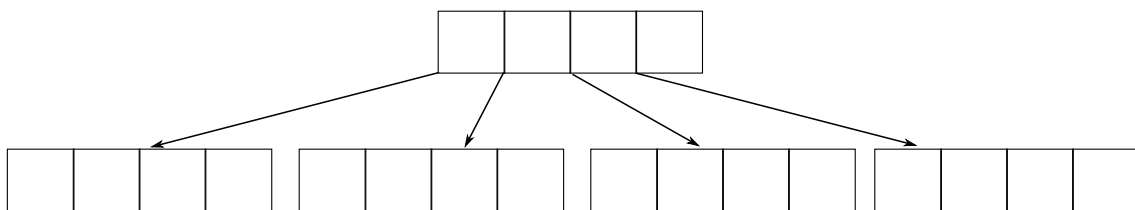
The *B*-tree after deleting key values 75 and 15:



The *B*-tree after deleting key values 75, 15, and 62:



The *B*-tree after deleting key values 75, 15, 62, and 21:



(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)**

(c) [5 marks] Consider a  $B$ -tree file of order  $p = 2m + 1 = 199$  that contains  $r = 10^7$  records. How many disk accesses will it take to retrieve a random record from the file using the  $B$ -tree in the worst case?

You may assume that the block size is larger than the size of the  $B$ -tree nodes and larger than the size of the records.

Show your working.

(d) [5 marks] Explain the difference between  $B$ -tree and  $B^+$ -tree index structures.

## ANSWERS

### Question 7. Text Processing

[40 marks]

(a) [5 marks] Explain the difference between an *Abstract Syntax Tree* and a *Concrete Parse Tree*.

.

(Question 7 continued on next page)

**(Question 7 continued)**

**(b)** [10 marks] Write a parser for the simple Query grammar below that returns `true` or `false`. Assume nonterminals are always in quotes and there are spaces separating tokens for simplicity.

```
QUERY ::= "SELECT" "*" ["FROM" NAME] ["WHERE" NAME "=" DATA] ";"  
NAME  ::= [A-Za-z]+  
DATA  ::= [A-Za-z0-9]*
```

Hint: It is easier to write the parser in Java, but you may use clear pseudocode if you prefer.

```
public boolean parseQuery(Scanner s) .
```

(Question 7 continued on next page)

**(Question 7 continued)**

Consider the following grammar where nonterminals are in uppercase and terminals are enclosed in quotation marks. Assume that tokens will be separated by spaces.

EXPRESSION ::= FOO "+" BAR "end"

FOO ::= [a-z0-9]+ | EXPRESSION

BAR ::= EXPRESSION FOO | a\*b\*c+

**(c)** [10 marks] For each of the following sentences, state whether it belongs to the language defined by this grammar.

yes

abracadabraend + abc end

yes

0 + c end + c end

yes

aabbc + aacc end

yes

0 + abc end + c end + c end

no

0 + 0 + c end c end c end + c end

**(d)** [5 marks] Explain why the grammar above cannot be parsed by a predictive, one symbol lookahead, left-to-right (LL(1)) parser.

??

(Question 7 continued on next page)

**(Question 7 continued)**

**(e)** [10 marks] Write a new grammar for EXPRESSIONs which specifies the same language as the grammar on the facing page, but which can be parsed using an LL(1) parser.

EXPRESSION ::= .

## ANSWERS

### Question 8. File structures and Hashing

[20 marks]

(a) [10 marks] For each of the following file structures, discuss their advantages and disadvantages by explaining the efficiency of the different file operations (insertion, deletion, search, sequential access) with different structures, and giving examples of when it is appropriate to use each kind of file.

(i) Heap file:

Heap files are very fast to insert into, but slow to search, and therefore delete from. They are also slow for sequential access. They are only good for constructing files that are not read often, or along the way to constructing a sequential file (constructing the file and then sorting it).

(ii) Sequential file.

Sequential files are slow to insert into and slow to delete from, but are fast to search and for sequential access. They are good for files that will be read very frequently, but only infrequently modified.

(iii) Hash file.

Hash files are fast to search, to insert into, and to delete from, but they require additional disk memory. They are slow for sequential access. As long as disk space is not very tight, they are the best choice for random access files, as long as sequential access is not required.



**(Question 8 continued)**

**(b)** [10 marks] Describe how extendible hash files work.

.

\*\*\*\*\*

### **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.

## Useful Formulas

You may tear off this page if you wish. You do not need to hand it in.

### File Performance Formulas

- blocking factor:  $f = \lfloor \frac{B}{L} \rfloor$
- number of blocks:  $b = \lceil \frac{r}{f} \rceil$
- external sort-merge:  $N = 2b \cdot (1 + \lceil (\log_{n-1} b) - 1 \rceil) = 2b \cdot (1 + \lceil (\frac{\log_{10} b}{\log_{10}(n-1)} - 1) \rceil)$   
(where  $n$  is the number of buffers)

### B-tree (worst case)

- height:  $h = 1 + \lfloor \log_{m+1} \frac{r+1}{2} \rfloor = 1 + \lfloor \log_{10} \frac{\frac{r+1}{2}}{\log_{10}(m+1)} \rfloor$
- number of leaves:  $N_{leaves} = 2(m+1)^{h-2} \leq N_{leaves} \leq (2m+1)^{h-1}$

### B<sup>+</sup>-tree (worst case)

- height:  $h = 2 + \lfloor \log_{m+1} \frac{r}{2m} \rfloor = 2 + \lfloor \frac{\log_2 \frac{r}{2m}}{\log_2(m+1)} \rfloor$
- number of leaves:  $N_{leaves} = \lceil \frac{r}{m} \rceil$

### Index-Sequential File with a B-tree

- number of sequence sets:  $s = \lceil \frac{r}{f} \rceil \leq s \leq \lceil \frac{2r}{f} \rceil$

### Logs to base 2

|            |   |   |   |   |    |    |    |     |     |     |       |           |
|------------|---|---|---|---|----|----|----|-----|-----|-----|-------|-----------|
| $n$        | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1,024 | 1,048,576 |
| $\log_2 n$ | 0 | 1 | 2 | 3 | 4  | 5  | 6  | 7   | 8   | 9   | 10    | 20        |

### Logs to base 10

|               |     |    |     |     |     |      |      |        |        |                 |                  |                  |
|---------------|-----|----|-----|-----|-----|------|------|--------|--------|-----------------|------------------|------------------|
| $n$           | 5   | 10 | 50  | 100 | 500 | 1000 | 5000 | 10,000 | $10^6$ | $5 \times 10^6$ | $10 \times 10^6$ | $50 \times 10^6$ |
| $\log_{10} n$ | 0.7 | 1  | 1.7 | 2   | 2.7 | 3    | 3.7  | 4      | 6      | 6.7             | 7                | 7.7              |