

UNIVERSITY OF CANTERBURY

# Mid-Year Examinations 2008

Prescription Number(s): COSC329

Paper Title: Algorithms and Artificial Intelligence

Time Allowed: THREE hours

Number of pages: 10

1. Answer *all* questions.
2. This exam is worth a total of 100 marks.
3. Open book examination. Calculators permitted.
4. Check carefully the number of marks allocated to each question. This suggests the degree of detail required in each answer, and the amount of time you should spend on the question.
5. Use the separate *Answer Booklet* for answering *all* questions.

**Question 1 [14 marks for the whole question]**

- (a) [4 marks] When writing rules for the JESS rule engine one problem is that the rules can “fire” in a different order to what they were coded. Explain why this is the case and describe *two* ways of ensuring rules fire in the desired order.
- (b) [2 marks] Explain the role of *forward chaining* in the following JESS code fragment:

```
(defrule rule2
  (room (x ?x) (y ?y))
  (nextDoor (x1 ?x) (y1 ?y) (x2 ?xtest) (y2 ?ytest))
  (room (x ?xtest) (y ?ytest) (dangerous "Y"))
  =>
    (printout t "HELP!" crlf)
)

(defrule rule2
  (room (x ?x1) (y ?y1))
  (room (x = (+ ?x1 1) (y ?y1))
  =>
    (bind ?x2 (+ ?x1 1))
    (assert (nextDoor (x1 ?x1)(y1 ?y1)(x2 ?x2)(y2 ?y1))
  )
```

- (c) [8 marks] Consider the following JESS program:

```
(reset)

(assert (father "Bob" "Kate"))
(assert (mother "Kate" "Wilbur"))

(defrule related1
  (or (father ?p1 ?p2)
      (mother ?p1 ?p2)
  )
  =>
    (printout t ?p1 " is related to " ?p2 crlf)
)
```

The rule “Related1” currently will report that two people are related if one is the father or mother of the other. Use forward chaining to extend this rule so that it reports that two people are “related” for all pairs where one is the ancestor of another (i.e parent, grandparent, great-grandparent, etc). For example, in this case it should also discover that Bob is related to Wilbur. You may write as many rules as you need.

**Question 2 [15 marks for the whole question]**

Cosc121	Cosc122	Cosc208	Class: Honours?
B	A	A	Y
B	B	C	N
A	C	A	Y
B	D	C	N
D	E	B	N
A	C	A	N
E	B	A	Y
D	A	B	Y

- (a) [10 marks] The COSC department is trying to predict the likelihood that students will study for an honours degree based on their performance in their first three courses, and gathers the data listed in the table above. Use the ID3 algorithm to determine the attribute a decision tree for this prediction should branch on first. Show your working.

Some logarithms you *may* need are:

$$\log_2(1/2) = -1; \log_2(1) = 0; \log_2(1/4) = -2; \log_2(3/4) = -0.415; \log_2(1/3) = -1.585;$$

$$\log_2(2/3) = -0.585; \log_2(1/8) = -3; \log_2(3/8) = -1.415; \log_2(5/8) = -0.678;$$

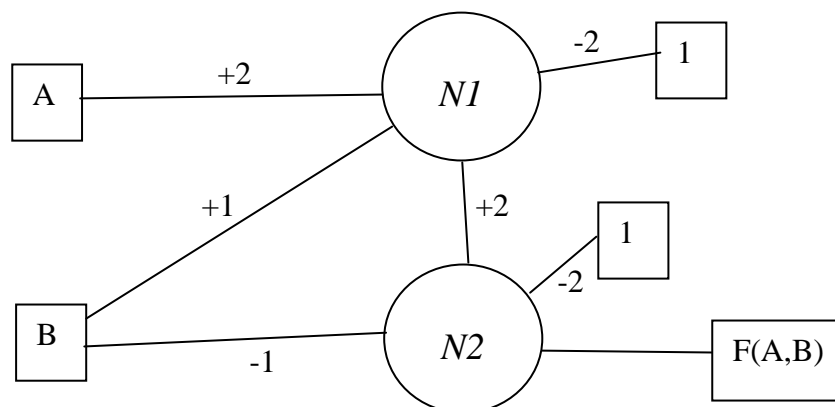
$$\log_2(7/8) = -0.193.$$

- (b) [5 marks] Using the nearest neighbour algorithm, predict whether or not a student with the following grades will take Honours, assuming that the distance function for symbolic attributes is  $d=0$  when the attribute has the same value, otherwise  $d=1$ . Show your working.

$$\text{Honours}(\text{cosc121:D, cosc122:A, cosc208:A}) = ?$$

**Question 3 [11 marks for the whole question]**

- (a) [5 marks] Determine the output  $F(A,B)$  from the following perceptron network for all possible binary inputs  $A$  and  $B$ . Each perceptron has a threshold such that the output will be 1 if  $\text{sum}(X) \geq 0$ . Show your working.



- (b) [2 marks] Explain why a single layer perceptron network is unable to learn non-linear functions such as XOR.
- (c) [4 marks] What are the *two main features* of a backpropagation feedforward network that allow it to learn non-linear functions? Explain how they do this.

**Question 4 [10 marks for the whole question]**

Use the best-first algorithm A\* to find the shortest path from START to FINISH. Use the Manhattan distance as your cost estimate (i.e. squares across + squares up or down). Show your working.

1	Start	2	3
4			5
6	7	8	9
10			
11	12	Finish	

TURN OVER

**Question 5 [20 marks for the whole question]** We design algorithms for generating a Gray code for permutations. To start off, we generate permutations of (1, 2, 3) in the following with the underlying tree structure. We call permuted numbers “items”.

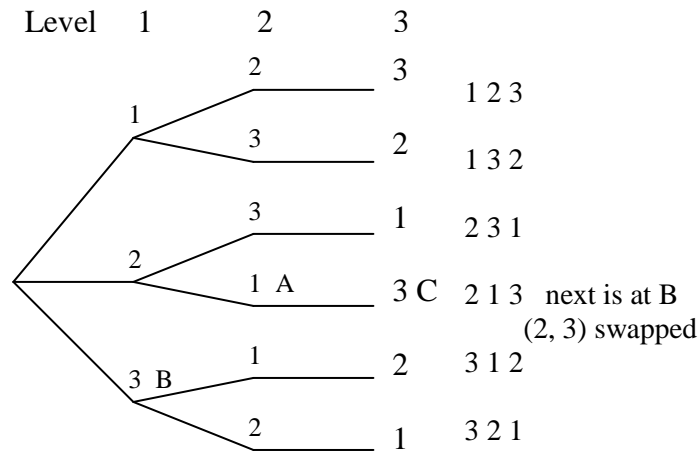


Figure 1

At level 1, items increase. At level 2, items increase, decrease, and increase. Level 3 shows the opposite pattern of level 2. Suppose we stand at point A in the above figure. We identify next[1] at level 1 is 3 at point B, and position of item 3 in the current permutation is at level 3 located at point C. Then we swap items at level 1 and level 3, and have the next permutation. This item 3 at point B was obtained while we examine 3 and 1 at level 2 and identify the one that is greater than the parent 2 and is minimum.

This idea is summarized in the recursive algorithm in Algorithm A. The meanings of variables and functions are summarised below.

a : array used as a container for permutations. a[i] is the i-th element in the permutation.  
d : array used to show the direction, increasing or decreasing at level i.  
pos : array used to hold the position for swapping at level i. We swap a[i] and a[pos[i]].  
next : array used to hold the next item. next[i] is the next item at level i.  
swap(i, j) : function for swapping a[i] and a[j] with parameters i and j.  
find\_pos(i) : To find the position of next[i-1] in the current array a.  
update\_next(i) : This is to update the candidates for next at one level up, the next[i-1].  
While we are moving at level i, we update the next item for level i-1 so that the item next[i-1] is obtained as follows:

$$\begin{aligned} \text{next}[i-1] &= \min x \text{ such that } x > a[i-1], \text{ if } d[i-1] > 0 \\ &= \max x \text{ such that } x < a[i-1], \text{ if } d[i-1] < 0 \end{aligned}$$

In the program, next[i-1] is re-initialized to 99 if d[i-1]>0, and 0 if d[i-1]<0 for each call of perm(i). Those two values are meant to be very large and very small.

perm(i) : This is the main recursive function to permute items in a[i .. n]. It re-initializes next[i-1], and do the following n-i+1 times. Call perm(i+1) to generate all permutations in a[i+1 .. n] and update next[i-1] for level i-1. Then swap a[i] and a[pos[i]], if the current position is not a last child and level is not n. Before ending perm(i), find the position for next[i-1] and reverse d[i]. By preparing pos[i-1], we can perform swapping when we go back to the calling point at level i-1. Also read comments in the code.

### Algorithm A

```
int n, a[10], next[10], pos[10], d[10];
```

```

out(){
int i;
for(i=1;i<=n;i++)printf("%d ", a[i]);
printf("\n");
}
swap(int i, int j){int w;
w=a[i]; a[i]=a[j]; a[j]=w;
}
int find_pos(int i){
int j;
for(j=i;j<=n;j++)if(a[j]==next[i-1])return j;
}
update_next(int i){
if(d[i-1]>0){
if((a[i]>a[i-1])&&(a[i]<next[i-1]))next[i-1]=a[i]; // minimizing next[i-1]
} else{
if((a[i]<a[i-1])&&(a[i]>next[i-1]))next[i-1]=a[i]; // maximizing next[i-1]
}
}
}
perm(int i){
int k;
if(i<=n){
if(d[i-1]>0)next[i-1]=99; else next[i-1]=0; /*** Re-initialization for next ***/
for(k=1;k<=n-i+1;k++){
perm(i+1);
update_next(i);
if(k<n-i+1) if(i<n)swap(i, pos[i]); // pos[i] has been prepared by perm(i+1)
}
} else out();
pos[i-1]=find_pos(i); // This takes O(n) time
d[i]=-d[i];
}
}
main(){int i;
scanf("%d",&n); getchar();
for(i=1;i<=n;i++)a[i]=i;
for(i=0;i<=n;i++)d[i]=1;
for(i=1;i<=n;i++)pos[i]=i;
printf("Start\n");
perm(1);
}

```

(A) [5marks] Following the above description, we give a partial output for  $n=4$ . Finish the output, and draw the tree structure similar to Figure 1.

```

1 2 3 4
1 2 4 3
1 3 4 2
1 3 2 4
1 4 2 3
1 4 3 2
2 4 3 1
2 4 1 3
2 3 1 4
2 3 4 1
2 1 4 3
2 1 3 4

```

TURN OVER

(B) [8 marks] The next project is to convert Algorithm A to an iterative one, Algorithm B, that can generate permutations in the Gray code order in  $O(1)$  time. The following is a partial list of the program. Fill the vacant positions labelled by /\*A\*/ and /\*B\*/. The algorithm is based on the tree traversal, similar to the iterative Johnson-Trotter. Array “up” navigates where to go up after swapping is done. The re-initialization for “next” is slightly different from Algorithm A, that is, it is done for level  $i$ . The functions “update\_next” and “swap” are the same. Array “c” is used for counting the number of branches from each node in the tree so that we can identify a last child.

**Algorithm B**

```
main(){
    scanf("%d",&n); getchar();
    for(i=1;i<=n;i++){a[i]=i; up[i]=i; d[i]=1; c[i]=1; pos[i]=i; next[i]=99;}
    up[0]=0;
    printf("Start\n"); i=n;
    out();
    while(i!=0)
    {
        i=up[n]; up[n]=n;
        if(d[i]>0)next[i]=99; else next[i]=0; /*** Re-initialization for next ***/
        update_next();
        if((i>0)&&(c[i]<n-i+1))swap(i, ___/*A*/); // Swap if not last child
        if(c[i]==n-i+1){ // last child check
            update_next();
            up[i]=up[i-1]; up[i-1]=i-1;
            if(next[up[i]]==___/*B*/)pos[up[i]]=i;
            d[i]=-d[i]; c[i]=1;
        }
    }
}
```

(C) [7 marks] Prove that the last permutation is  $(n, n-1, 1, 2, \dots, n-2)$  for odd  $n$ , and  $(n, 1, 2, \dots, n-1)$  for even  $n$ . Note that the initial permutation is  $(1, 2, \dots, n)$  in either case. Hint. Observe the increase/decrease pattern at each level.

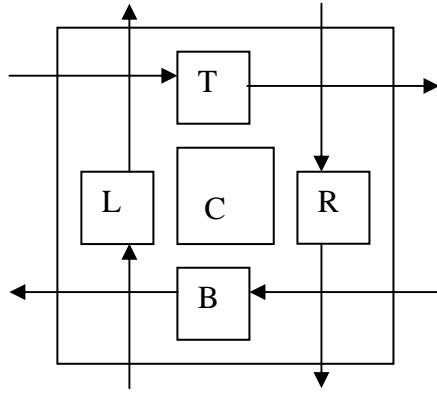
**Question 6 [15 marks for the whole question]**

(A) [5 marks] The following is a distance matrix.

0	1	4	The meaning of this matrix is that the distance from
7	0	2	vertex 1 to vertex 3 is 4, etc.
3	5	0	

Obtain all pairs shortest distances by any method, including inspection.

(B) [10 marks] The following is a partial trace of Mesh Floyd. Complete the trace. The clock time  $t$  and control signals from the  $k$ -th update are given separately above the main mesh picture. Each cell has five registers. The center, abbreviated as C, is the main one, surrounded by four registers, (top, bottom, right, left), abbreviated as (T, B, R, L). The value of T moves right shown by  $\Rightarrow$  attached right, that of R moves down shown by  $*$  attached below, etc. Control signals trigger to copy the center register value to (L, R) if it comes horizontally, or to (T, B) if it comes vertically. Those register values update the C values. Some examples are explained in the snapshots. A picture of one cell is given below. 99 is meant to be an infinity.



t= 0

```
1 0 0
0 0 0
0 0 0
```

Control signal 1 starts

```
* 99 => * 99 => * 99 =>
99 0 99 99 1 99 99 4 99
<= 99 * <= 99 * <= 99 *
```

The initial distance matrix values are contained in C registers

```
* 99 => * 99 => * 99 =>
99 7 99 99 0 99 99 2 99
<= 99 * <= 99 * <= 99 *
```

```
* 99 => * 99 => * 99 =>
99 3 99 99 5 99 99 0 99
<= 99 * <= 99 * <= 99 *
```

t= 1

```
0 1 0
1 0 0
0 0 0
```

Control signal 1 goes right and down

```
* 99 => * 99 => * 99 =>
99 0 99 1 1 1 99 4 99
<= 99 * <= 99 * <= 99 *
```

Value 1 is copied to L and R

```
* 7 => * 99 => * 99 =>
99 7 99 99 0 99 99 2 99
<= 7 * <= 99 * <= 99 *
```

```
* 99 => * 99 => * 99 =>
99 3 99 99 5 99 99 0 99
<= 99 * <= 99 * <= 99 *
```

t= 2

```
0 0 1
0 0 0
1 0 0
```

```
* 99 => * 99 => * 99 =>
99 0 99 99 1 99 4 4 4
<= 99 * <= 99 * <= 99 *
```

```
* 99 => * 7 => * 99 =>
99 7 99 99 0 1 99 2 99
<= 99 * <= 99 * <= 99 *
```

```
* 3 => * 99 => * 99 =>
99 3 99 99 5 99 99 0 99
<= 3 * <= 99 * <= 99 *
```

t= 3

```
0 0 0
0 2 0
0 0 0
```

Control signal 2 starts

TURN OVER

```

* 99 =>      * 99 =>      * 99 =>
99  0  99    99  1  99    99  4  99
<= 99  *    <= 99  *    <= 99  *

```

```

* 99 =>      * 99 =>      * 7 =>
99  7  99    99  0  99    99  2  4
<= 99  *    <= 99  *    <= 99  *

```

```

* 99 =>      * 3 =>      * 99 =>
99  3  99    99  4  1    99  0  99
<= 99  *    <= 99  *    <= 99  *

```

t= 4

```

0  2  0
2  0  2
0  2  0

```

```

* 99 =>      * 1 =>      * 99 =>
99  0  99    99  1  99    99  4  99
<= 99  *    <= 1  *    <= 99  *

```

```

* 99 =>      * 99 =>      * 99 =>
7  7  7      99  0  99    2  2  2
<= 99  *    <= 99  *    <= 99  *

```

```

* 99 =>      * 4 =>      * 3 =>
99  3  99    99  4  99    99  0  4
<= 99  *    <= 4  *    <= 99  *

```

t= 5

```

0  0  0
0  0  0
0  0  0

```

```

* 99 =>      * 99 =>      * 1 =>
7  0  99    99  1  99    2  3  99
<= 1  *    <= 99  *    <= 99  *

```

The previous value 4 is updated to 3 by 1 from left neighbour and 2 from below neighbour

```

* 99 =>      * 99 =>      * 99 =>
99  7  99    99  0  99    99  2  99
<= 99  *    <= 99  *    <= 99  *

```

```

* 99 =>      * 99 =>      * 4 =>
99  3  7      99  4  99    99  0  2
<= 4  *    <= 99  *    <= 99  *

```

In your trace, comments like those at the right-hand side of each snapshot are not required.

### Question 7 [15 marks for the whole question]

(a) [5 marks] Draw a bitonic sorting network of size 16, that sort the input sequence in decreasing order at the output.

(b) [10 marks] Trace your network with the input sequence

(10, 5, 6, 7, 8, 15, 9, 16, 11, 12, 14, 1, 2, 3, 4, 13)