編號：190、182　　國立成功大學107學年度碩士班招生考試試題

系　　所：電腦與通信工程研究所、電機工程學系

考試科目：資料結構　　考試日期：0205，節次：2

※ 考生請注意：本試題不可使用計算機。　請於答案卷(卡)作答，於本試題紙上作答者，不予計分。

一、 複選題: (25 分　，每題 5 分，全對才給分)

1. Which of the following statement(s) is (are) true?

(A) An abstract data type (**ADT**) is a data type that is organized in a way that specifications of the objects and the associated operations are separated from the representation of the objects and implementation of the operations.

(B) **Data encapsulation** makes modifications of program much easier.

(C) Software development time and money can be saved if **template** is used in a program.

(D) **A derived class** inherits all the non-private members (data and functions) of the base class including constructor and destructor.

2. Which of the following statement(s) is (are) true?

(A) The number of nodes with two children in a binary tree must be smaller than or equal to number of leaf nodes.

(B) The worse time complexity to insert a node into a binary search tree is O(log n) when the tree contains n nodes.

(C) There exists n+1 thread in a threaded binary search tree with n nodes.

(D) If we use a list representation to represent a k-ary tree (i.e., at tree of degree k) for n nodes, where each node in a list has a fixed size k to record its children, then $n(k\text{-}1) + 1$ fields will be wasted in the representation.

3. Which of the following statement(s) is (are) true?

(A) Quick sort and merge sort both apply the divide-and-conquer approach to solve the sorting problem.

(B) Quick sort spends large effort in dividing a sequence into two parts while merge sort spends large effort in combining two unsorted sequence into one sorted sequence.

(C) Both of the two algorithms are stable.

(D) The expected time complexity of merge sort is O(nlogn) which is faster than quick sort in the worse case.

4. Which of the following statement(s) is (are) true?

(A) A **back** edge or a **cross** edge happen when a non-tree edge is found when we apply **DFS** algorithm to search a graph.

(B) We can apply **BFS** based algorithm to traverse a graph in order to find biconnected components in it.

(C) We have to apply **topological sorting** algorithm before we build an activity-on-edge (**AOE**) network.

(D) **Dijkstra's** algorithm can be applied to a directed acyclic graph of edges with negative weights.

5. Which of the following statement(s) is (are) true (assume the number of nodes in a maximum heap and in a binary search trees are both n)?

(A) We have go through a path from the root to a leaf when we insert a node in a maximum heap.

(B) It takes O(n) time to insert a node into a binary search tree.

(C) It takes O(log n) time to obtain the maximum value in a maximum heap.

(D) Max heapfiy is an operation to maintain the maximum heap property which starts from a node and goes through a path to a leaf.

二、問答題: (75 分)

1. **Permutation** (12 **pts**)

a. (3 **pts**) Please show the output of the function in serial. Assume the initial state of the array is {1 2 3} and k=0, m =2.

b. (7 **pts**) Please show the recursive function in order to compute the time complexity of the function and give the answer of the function. Assume the dimension of the array in n and it takes O(1) to swap two elements (i.e., swap(a[k], a[i]).

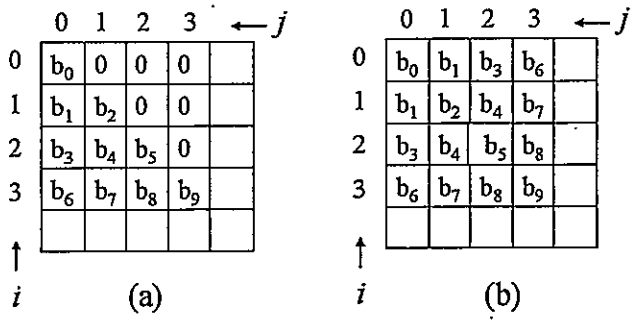
c. (2 **pts**) Is this program a polynomial time algorithm?

```
void Function (char *a, const int k, const int m) {
    if (k == m) { // output permutation
        cout << "[";
        for (int i = 0; i <= m; i++) {
            if (i != m)     cout << a[i] <<", ";
            else            cout << a[i] <<"] "; }
    } else {
        for (i = k; i <= m; i++)) {
            swap(a[k], a[i]);
            Function (a, k+1, m);
            swap(a[k], a[i]); }
    }
}
```

2. **Matrix** (10 **pts**)

a. (5 **pts**) A matrix is said to be **lower triangular** when all elements below the main diagonal of square matrix is zero. Given a lower triangle **A**, each non-zero element is stored by rows in a one-dimensional array **B** with $a[0][0]$ being stored in $b[0]$ (see Fig. (a)). Give a single expression for k in terms of i and j if an element $a_{i,j}$ is stored at $b[k]$.

b. (5 **pts**) A matrix is said to be a **symmetric matrix** if it is a square matrix and its transpose is equal to itself. Similarly, each element of a symmetric matrix is stored by rows in a one-dimensional array **B** with *a*[0][0] being stored in *b*[0]. EX: $a_{10}=a_{01}$ is stored in b(1); a_{11} is stored in b(2); $a_{21}=a_{12}$ is stored in b(3) and so on (see Fig. (b)). Write a single expression for *k* in terms of *i* and *j* if an element $a_{i,j}$ is stored at *b*[*k*] (**hint**: the functions Max(*i*, *j*) and Min(*i*, *j*) could be used in the expression).



3. **A Mazing Problem** (16 **pts**)

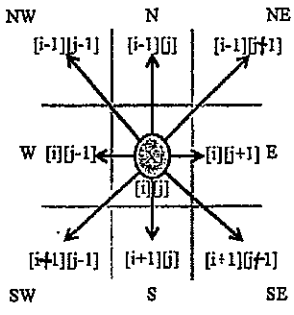
The rat in a maze experiment is a classical one from experimental psychology, where a map is recorded by a two-dimensional array, *maze*[*i*][*j*], where $1 \le i \le m$ and $1 \le j \le p$. A value of 1 implies a blocked path, and a 0 means one can walk right on through. The data types needed are

struct *offsets* { int *a*, *b*; }

enum directions {N, NW, W, SW, S, SE, E, NE};

offsets *move*[8];

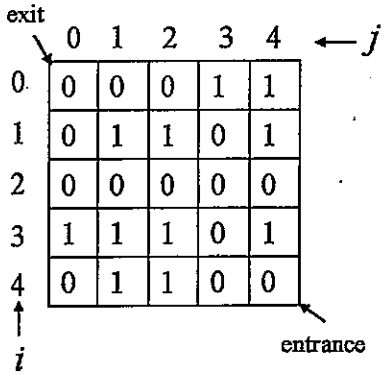
For example, if we are at *maze*[*i*][*j*] and we wish to find the new position *maze* [*g*][*h*] that is southwest of us, then we set *g* = *i* + move[SW].a and h = *j*+move[SW].b (see Fig. (a) and (b)).



(a)

q	move[q] .a	move[q] .b
N	-1	0
NE	-1	1
E	0	1
SE	1	1
S	1	0
SW	1	-1
W	0	-1
NW	-1	-1

(b)



(c)

The pseudo code is shown in the following:

```
1.  initialize list to the maze entrance coordinates and direction north;
2.  while (list is not empty)
3.  {
4.    (i, j, dir) = coordinates and direction from end of list;
5.    delete last element of list;
6.    while (there are more moves from (i, j)) // i.e., dir < 8
7.    {
8.          (g, h) = coordinates of next move;
9.          if ((g == m) && (h == p)) success;
10.         if ((!maze[g][h] && (!mark[g][h]))
11.         {
12.               mark[g][h] = 1;
13.               dir = next direction to try;
14.               add (i, j, dir) to end of list;
15.               (i, j, dir) = (g, h, N);    //move to new location
16.         }
17.   }
18. }
19. cout << "No path in maze." << endl;
```

a. (2 **pts**) What kind of the data structure that you can use it to implement the *list* in the above pseudo code? Why?

b. (2 **pts**) How do you handle the problem when a rat touches the boundary of a maze?

c. (2 **pts**) What is the purpose of the matrix *mark* which appears in Line 10 and Line 12?

d. (5 **pts**) Please show the path that is found by the program in the map of Fig. (c) in term of the index of the matrix.

e. (2 **pts**) Can you find the path if there exists at least one route from entrance to exit according to the algorithm?

f. (3 **pts**) Assume the cost will be increased by one if you go through a grid in the maze, and the cost of a path is the number of grids in the path. Dose the proposed algorithm find the path with the smallest cost? If the answer is No, what kind of algorithm and the data structure that you should use in order to find the path with the smallest cost?

4. Tree (14 **pts**)

Given the following pseudo code whose input includes two trees which are pointed by *s* and *t*, respectively, the algorithm checks equality of the two trees.

```
1.  bool operator== (const Tree& s, const Tree& t) { // Driver
2.          return Equal(s.root, t.root);
3. }

4. bool Tree <T>::Equal(TreeNode <T> *a, TreeNode <T> *b) { // Workhorse
5.      Terminal Condition
6.      else return   ( a && b
7.                      && (a->data == b->data)
8.                      && Equal(a->leftChild, b->leftChild)
9.                      && Equal(a->rightChild, b->rightChild));      }
```

a. (2 **pts**) Please complete the terminal condition in **Line** 5.

b. (2 **pts**) What is the purpose of **Line** 6?

c. (3 **pts**) Can we obtain a correct answer if we change the sequence of Lines 7 to 9? Why?

The **satisfiability (SAT)** is an important problem to answer the question if there exists an assignment to make a boolean expression true.

```
9.  enum Operator {Not, And, Or, True, False};
10. class SatTree; // forward declaration
11. class SatNode {
12.                 friend class SatTree;
13.                 Operator data;
14.                 bool value;
15.                 SatNode *leftChild;
16                  SatNode *rightChild;        }
```

```
17. void SatTree::PostOrderEval() {PostOrderEval(root);}

18. void SatTree::PostOrderEval(SatNode *s) {
19.     if (s) { // not null
            ??????????????????
20.   }
```

d. (2 **pts**) Why do we need to use the key word **friend** in **Line** 12?

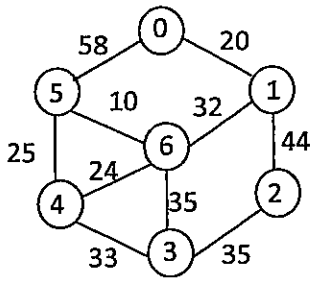
e. (2 **pts**) What is the correct sequence inside **Lines** 19-20 according to the following codes? (A) 甲 乙 丙 (B) 甲 丙 乙 (C) 乙 甲 丙 (D) None of above

```
甲、   PostOrderEval(s->leftChild);
乙、   switch(s->data) {
b.1            case Not:   s->value = !s->rightChild->value; break;
b.2            case And:   s->value = s->leftChild->value && s->rightChild->value; break;
b.3            case Or:    s->value = s->leftChild->value || s->rightChild->value;; break;
b.4            case True:  s->value = true; break; // terminal node
b.5            case False: s->value = false; // terminal node
               }
丙、   PostOrderEval(s->rightChild);
```

f. (3 **pts**) Please give the reason for your choice in the last question.

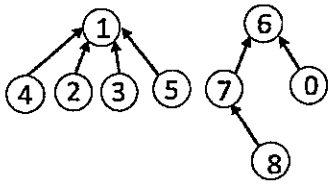
5. (14 **pts**) **Spanning Tree:**

Find the minimal cost spanning tree according to (a) Kruskal's algorithm and (b) the Sollin's Algorithm, respectively. Please show the result in each iteration of the algorithm.



6. (9 **pts**) **Set**

There exist two sets which have five members and four members, respectively, which are recorded by the tree structure. Let the number in the root as ID of a set.



Set {1, 4, 2, 3, 5}　　Set {6, 7, 0, 8}

a. (6 **pts**) Union is an operation to merge two sets. Please respectively show the resulting data structures after the function WeightUnion1(1, 6) is called when you use number of nodes as the weight and when you use height as the weight.

b. (3 **pts**) Find is an operation to find an ID which contains a member. Please show the resulting data structure if the function *CollaspingFind*(8) is called in the second set, where collapse aims to reduce the height of a tree.