國立陽明交通大學
115學年度碩士班考試入學招生試題

科目：資料結構與演算法(8101)

系所班別：資訊聯招

考試日期：115年2月4日 第 1 節

【不可使用計算機】

＊作答前請先核對試題、答案卷(試卷)與准考證之所組別與考科是否相符！！

國立陽明交通大學 115 學年度碩士班考試入學招生試題

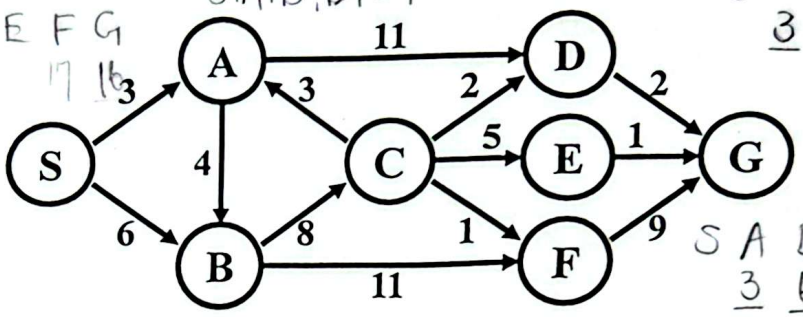
科目：資料結構與演算法(8101)　　考試日期：115 年 2 月 4 日　第 1 節

系所班別：資訊聯招　　第 1 頁，共 9 頁

【不可使用計算機】＊作答前請先核對試題、答案卷(試卷)與准考證之所組別與考科是否相符！！

一、單選題（共 24 分）：該題答對得 3 分，答錯倒扣 1 分，未作答不給分。
選擇題請使用答案卡作答

1. Given the following graph where vertex **S** is the source and the weight of each edge is specified alongside the edge, please apply the ***Bellman-Ford*** algorithm. During each pass/iteration of edge-by-edge relaxation, the order of edge to be processed is: **AB, AD, BC, BF, CA, CD, CE, CF, DG, EG, FG, SA, SB** (i.e., lexicographic order, assuming that an edge from vertex **X** to vertex **Y** is named **XY**).



For this particular graph and based on the aforementioned order of edge relaxation, how many passes/iterations are required for the ***Bellman-Ford*** algorithm to converge (i.e., to correctly find shortest paths from **S** to each of other vertices), and what is the shortest path distance from **S** to **G**? **(NOTE: The *Bellman-Ford* algorithm requires *P* passes to converge if, *for every vertex*, the $(P+1)^{th}$ pass makes no improvement in shortest path distance.)**

(a) 2 passes; shortest path distance from **S** to **G** = 16
(b) 2 passes; shortest path distance from **S** to **G** = 18
(c) 3 passes; shortest path distance from **S** to **G** = 16
(d) 3 passes; shortest path distance from **S** to **G** = 18
(e) 3 passes; shortest path distance from **S** to **G** = 19

2. Consider the same graph as in Question 7. If you can determine the order of edge relaxation, what are the minimum and maximum numbers of passes for the ***Bellman-Ford*** algorithm to converge (i.e., to correctly find shortest paths from **S** to each of other vertices)?
   (a) minimum: 1; maximum: $N-1$ (where N is the number of vertices in the graph)
   (b) minimum: 1; maximum: N
   (c) minimum: 2; maximum: $N-1$
   (d) minimum: 2; maximum: N
   (e) minimum: 3; maximum: $N-1$

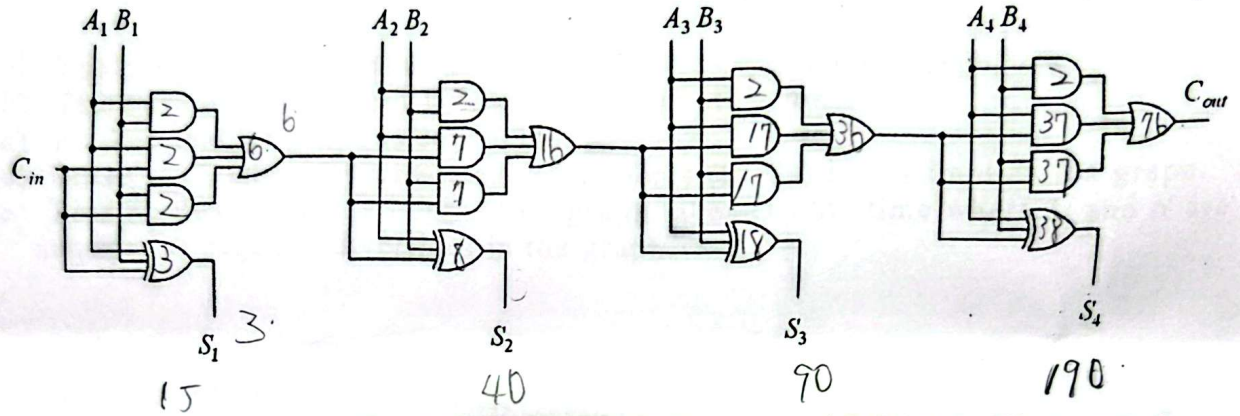
3. Consider the same graph as in Question 7. Which one of the following orders of edge relaxation results in the minimum number of passes for the ***Bellman-Ford*** algorithm to converge?
   (a) **AB, AD, BC, BF, CA, CD, CE, CF, DG, EG, FG, SA, SB**
   (same as in Question 7, i.e., lexicographic order)
   (b) **AB, AD, BC, BF, CA, CD, CE, CF, DG, EG, FG, SB, SA**
   (reverse the last 2 edges, **SA** and **SB**, from (a))

(c) **FG, EG, DG, CF, CE, CD, CA, BF, BC, AD, AB, SB, SA**
(reverse the first 11 edges, from (b))

(d) **SB, SA, FG, EG, DG, CF, CE, CD, CA, BF, BC, AD, AB**
(move the last 2 edges to the top, from (c))

(e) **SB, SA, AB, AD, BC, BF, CA, CD, CE, CF, DG, EG, FG**
(reverse the last 11 edges, from (d))

4. Consider the following gate-level combinational logic network of a 4-bit ripple carry adder. $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$ are two 4-bit input operands, C_{in} is the carry-in (input), $S_4S_3S_2S_1$ is 4-bit sum (output), and C_{out} is the carry-out (output).



What is the number of *distinct* paths in this logic network? **(HINT: There are 3 distinct paths to output S_1, and you have to consider all of the outputs.)**

(a) 78
(b) 103
(c) 118
(d) 143
(e) 158

5. Consider the same logic network as in Question 10. Among all of the possible paths (from any input, to any output), the one passing through the largest number of gates (vertices) is defined as the critical path. In this particular logic network, how many gates does the critical path pass through?

(a) 4
(b) 6
(c) 8
(d) 10
(e) 12

6. Continue the previous question. What is the time complexity of the best/fastest algorithm for solving Question 11?

(a) **O**(M+N) where M and N are the numbers of edges (wires) and vertices (gates)
(b) **O**(Mlg$N$)
(c) **O**(Mlg$M$)
(d) **O**(MN)
(e) **O**(M^2)

7. Which one of the following 2-SAT problems (in conjunctive normal form, CNF) is satisfiable?
   (a) $F = (a + b)\,(a + \neg b)\,(\neg a + b)\,(\neg a + \neg b)$
   (b) $F = (a + b)\,(a + \neg b)\,(\neg a + c)\,(\neg a + \neg c)$
   (c) $F = (a + b)\,(\neg a + b)\,(\neg b + c)\,(\neg b + d)\,(\neg c + \neg d)$
   (d) $F = (a + b)\,(\neg a + b)\,(\neg b + \neg c)\,(\neg b + \neg d)\,(c + d)$
   (e) $F = (a + b)\,(\neg a + b)\,(\neg b + \neg c)\,(\neg b + \neg d)\,(c + \neg d)$

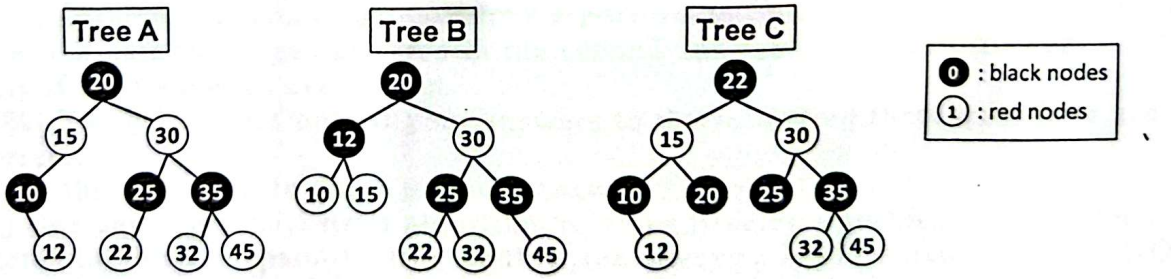
8. Which one of the following statements about the implication graph for 2-SAT is true?

   CNF: $F = (a + b)\,(\neg a + b)\,(\neg b + c)\,(\neg b + d)\,(\neg c + \neg d)$

   (a) The implication graph for a 2-SAT CNF is an undirected graph.
   (b) There are 8 vertices and 12 edges in F's implication graph.
   (c) F's implication graph is connected.
   (d) There are 2 strongly connected components (SCCs) in F's implication graph.
   (e) Finding SCC(s) in F's implication graph takes $O(MN)$ time where M and N are the numbers of edges and vertices in the graph.

二、簡答題（共 25 分）：請務必依序、並依題意簡要作答。

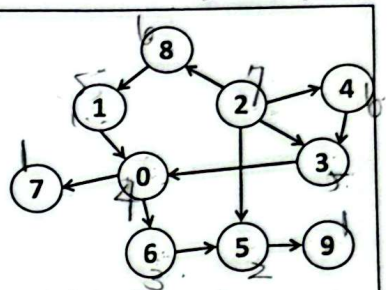
9. **[8%]** Three students tried to build red-black trees from the following numbers: 10, 20, 30, 15, 25, 35, 45,12, 22, 32. They ended up with three different trees (for clarity, the external nodes are not shown):



   (a) **[6%]** For each of these trees, indicate whether it is a valid red-black tree. If your answer is NO for any tree, you need to state the reason why it is invalid to get the credit.
   (b) **[2%]** The depth ratio between two subtrees of the same parent in a red-black tree is no more than ____ (give the closest integer).

> 43810765

10. **[10%]** For the given directed graph (the numbers inside the vertices are the vertex indices), answer the following questions:



(a) **[2%]** Give the count of "1" elements in the transitive closure of this graph.

(b) **[2%]** Give a topological order of the vertices in the graph. (The answer is not unique; you only need to give a valid one.)

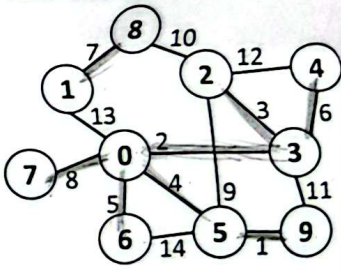
(c) **[2%]** Assume now that the direction of edge 2→5 is flipped to 5→2. What happens to the topological order? If there still exists a topological order, give one. If no topological order exists now, explain why. 2,4,3,8,1,0,7,6,5,9

(d) **[2%]** Starting from the original graph, if one more edge 1→3 is added, what happens to the topological order? If there still exists a topological order, give one. If no topological order exists now, explain why.

(e) **[2%]** Consider the following statement: "If a suitable starting vertex is selected, one can obtain the topological order of a directed graph using breadth-first search (BFS) if one exists." Indicate whether the statement is TRUE of FALSE. *If your answer is TRUE, indicate how you will select the suitable starting vertex. If you answer if FALSE, describe one condition that makes BFS fail this task.*

4, 11, 12, 13, 14

11. ***[7%]*** For the given undirected weighted graph (the numbers *inside the vertices* are the vertex indices; the numbers beside the edges are their weights), answer the following questions:



(a) **[2%]** Assume that you're using Kruskal's algorithm to build the minimum spanning tree (MST). Indicate the edges selected in the second and the fifth steps. You get credits only if both answers are correct.

(b) **[2%]** Assume that you're using Prim's algorithm to build the minimum spanning tree. Indicate the edges selected in the second and the fifth steps. You get credits only if both answers are correct.

(c) **[3%]** You get credits only if your answers to the following three questions are all correct:
(1) Is the MST of this graph unique? Answer YES or NO.
(2) One can apply Dijkstra's algorithm to an undirected weighted graph from any vertex to obtain a spanning tree. Is this tree always a MST? Answer YES or NO.
(3) Give the number of edges in the MSTs obtained in (a) and (b).

三、程式碼除錯題（共 25 分）：請務必依序、並依題意簡要作答。

12. **[5%] Binary Search**

There is ONE BUG in the following binary search program. Please provide:
- The line number where the bug occurs, and
- The corrected solution in C.

```c
#include <stdio.h>
// Function to perform binary search
// Returns the index of the target if found, otherwise returns -1
int binarySearch(int arr[], int size, int target) {
    int left = 0;              // Left boundary of the search
    int right = size - 1;      // Right boundary of the search
    while (left <= right) {
        int mid = left + (right - left) / 2; // Calculate middle index safely
        // Check if target is at mid
        if (arr[mid] == target) {
            return mid; // Target found
        }
        if (arr[mid] < target) {
            right = mid - 1;
        }
        else {
            left = mid + 1;
        }
    }
    return -1; // Target not found
}
int main() {
    // Sorted array (binary search requires sorted data)
    int arr[] = {2, 4, 6, 8, 10, 12, 14, 16};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 10; // Value to search for
    int result = binarySearch(arr, size, target);
    return 0;
}
```

left = mid

13. **[10%] Binary Search Tree**

There are TWO BUGS in the following doubly linked list program. For each bug, please provide:

- The line number where the bug occurs, and
- The corrected solution in C.

```c
#include <stdio.h>
#include <stdlib.h>
// Define the structure for a doubly linked list node
struct Node {
    int data;              // Data stored in the node
    struct Node* prev;     // Pointer to the previous node
    struct Node* next;     // Pointer to the next node
};
```

```
// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode; // If list is empty, new node becomes head
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next; // Traverse to the last node
    }
    temp->next = newNode;
}   newNode->prev = temp;
void deleteNode(struct Node** head, int key) {
    struct Node* temp = *head;
    // Traverse the list to find the node with the given key
    while (temp != NULL && temp->data != key) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Node with value %d not found.\n", key);
        return;
    }
    free(temp);
    if (temp->prev == NULL) {
        *head = temp->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
    } else {
        temp->prev->next = temp->next;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    printf("Node with value %d deleted.\n", key);
}
// Main function to test the doubly linked list
int main() {
    struct Node* head = NULL;
    // Insert nodes
    insertEnd(&head, 10);
    insertEnd(&head, 20);
    // Delete a node
    deleteNode(&head, 20);
```

```
61      return 0;
62  }
```

14. **[10%] Min Heap**

There are TWO BUGS in the following min heap program. Note that the data stored in a min heap should not contain identical values. For each bug, please provide:

– The line number where the bug occurs, and

– The corrected solution in C.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #define MAX_SIZE 100  // Maximum size of the heap
4   // Structure to represent a Min Heap
5   struct MinHeap {
6       int arr[MAX_SIZE];  // Array to store heap elements
7       int size;           // Current number of elements in the heap
8   };
9   // Function to initialize the heap
10  void initHeap(struct MinHeap* heap) {
11      heap->size = 0;
12  }
13  // Function to get the index of the parent node
14  int parent(int i) {
15      return (i - 1) / 2;
16  }
17  // Function to get the index of the left child
18  int leftChild(int i) {
19      return 2 * i + 1;
20  }
21  // Function to get the index of the right child
22  int rightChild(int i) {
23      return 2 * i + 2;
24  }
25  // Function to swap two elements
26  void swap(int* a, int* b) {
27      int temp = *a;
28      *a = *b;
29      *b = temp;
30  }
31  // Function to insert a new element into the heap
32  void insert(struct MinHeap* heap, int key) {
33      if (heap->size == MAX_SIZE) {
34          printf("Heap overflow! Cannot insert.\n");
35          return;
36      }
37      // Insert the new key at the end
38      int i = heap->size;
```

```
39      heap->arr[i] = key;
40      heap->size++;
41      while (i != 0 && heap->arr[parent(i)] < heap->arr[i]) {
42          swap(&heap->arr[i], &heap->arr[parent(i)]);
43          i = parent(i);
44      }
45  }
46  // Function to heapify (maintain min heap property) from index i
47  void heapify(struct MinHeap* heap, int i) {
48      int l = leftChild(i);
49      int r = rightChild(i);
50      int smallest = i;
51      if (l < heap->size && heap->arr[l] < heap->arr[smallest]) {
52          smallest = l;
53      }
54      if (r < heap->size && heap->arr[r] < heap->arr[smallest]) {
55          smallest = r;
56      }
57      if (smallest != i) {
58          swap(&heap->arr[i], &heap->arr[smallest]);
59          heapify(heap, smallest);
60      }
61  }
62  // Function to delete and return the minimum element (root)
63  int deleteMin(struct MinHeap* heap) {
64      if (heap->size <= 0) {
65          printf("Heap underflow! No elements to delete.\n");
66          return -1;
67      }
68      if (heap->size == 1) {
69          heap->size--;
70          return heap->arr[0];
71      }
72      // Store the minimum value and remove it
73      int root = heap->arr[0];
74      heap->arr[0] = heap->arr[1];
75      heap->size--;
76      heapify(heap, 0);
77      return root;
78  }
79  // Main function to test the Min Heap
80  int main() {
81      struct MinHeap heap;
82      initHeap(&heap);
83      insert(&heap, 20);
84      insert(&heap, 5);
85      printf("Deleted min: %d\n", deleteMin(&heap));
86      return 0;
87  }
```

四、非選擇題（共 26 分）：請詳讀下列說明。

Answer the following questions. If you do not know the answer, you have the option to answer "PASS"; in this case, you will receive exact ONE point for that question. Other irrelevant answers to the question make you lose the option. Make good use of the the option to maximize your score.

15. 4%

index	0	1	2	3	4	5	6	7	8	9	10	11
val	-	11	5	10	2	9	3	8	4	7	6	1

A binary min-heap (the one for heap-sort) is stored in an array. In a C array, we store values starting from index 1. The array in the above is not a min-heap. Build the min-heap from the array shown above using **the bottom-up linear-time algorithm**. Present your result in the same array where each value is in the correct array entry.

16. 5% Given a graph G=(V,E), to answer whether there is Hamiltonian Cycle in G is NP-Complete. Do you think the function `f`(G,u,v,k) is tractable? The function `f` returns yes if there is a path of length k from u to v in G, otherwise it returns no. If you think it is tractable, give me your reasons, or try to argue that the function `f` is not tractable.

17. (a) 4%

```
class Node {
   public int v;
   public Node lc;
   public Node rc;
   public Node p;
}
```

The above is pseudo-code defining a `Node` class used in a binary search tree T. In the `Node`, `v` stores the key value, `lc` and `rc` are the pointers to the left child and right child, respectively, and `p` is the pointer to the parent. If T contains a single node `X`, then `X.p`, `X.lc`, `X.rc` are `NULL`. There are 3 NULL pointers in T. Show that there are $n+2$ NULL pointers in T, where T is an n-node binary search tree.

(b) 6%

```
a = n1
b = n2
while a!=b:
   a=a.p if a else n2
   b=b.p if b else n1
return a
```

In the above pseudo code, `n1` and `n2` are two nodes in a binary search tree T. `a=a.p if a else n2` means if `a` is not `NULL` then set `a` to `a.p` otherwise set `a` to `n2`. What does this piece of code compute (what is "a" returned by the function)?

(c) 7% Show the correctness of the piece of code, i.e., the code correctly computes the answer you gave me in 3.(b).