DSA FINALS REVIEWER

Total points 44/83

Hope this helps!! ^-^

22 of 35 points

✓ When the series of stack operations below is performed on an empty stack, what is the data that is read out by the last READ operation? *2/2

Here, "PUSH x" is the operation to put data x in the stack, "POP" is used to retrieve data from the stack, and "READ" is used to read data from the top of the stack without removing the original data.

PUSH 2 → READ → PUSH 3 → PUSH 6 → POP → READ → PUSH 4 → READ → PUSH 7 → PUSH 5 → POP → POP → READ

4 ✓

✓ What is the running time in deleting the first element in the list of N elements and represented in memory using cursor-based implementation. *2/2

- O(N)
- [✓ O(1)] ✓
- O(N * N)
- O(log N)

✓ What is the worst case complexity of binary search using recursion? * 2/2

- ○ O(N)
- ○ O(N log N)
- ○ O(N * N)
- [✓ O(log N)] ✓

✓ Which of the following will cause a collision in a closed-hash table? * 2/2

- ○ Two (key, hashValue) pairs that have equal keys and different hash values.
- [✓ Two (key, hashValue) pairs that have different keys and equal hash values.] ✓
- ○ Two (key, hashValue) pairs that have equal keys and equal hash values.
- ○ Two (key, hashValue) pairs that have different keys and different hash values.

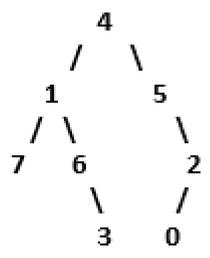
✓ For a graph G(V, E) of V vertices and E edges, which of the following is true if G is connected and has no cycles? * 2/2

- ○ # of vertices is greater than # of edges
- ○ # of vertices is equal to # of edges
- [✓ # of vertices is equal to # of edges + 1] ✓
- ○ # of vertices is equal to # of edges - 1

Determine the Inorder Listing of the nodes of the tree. *

0/2

Separate each node by comma ONLY.



7,1,3,6,4,0,2,5

Correct answer

7,1,6,3,4,5,0,2

✓ When the procedure described below is executed in sequence for an empty stack and empty queue, what is the value assigned to variable x? *2/2

Here, the functions used in the procedure are defined as follows:

[Function Definitions]

- **push(y):** pushes data **y** onto the top of the stack.
- **pop():** removes the data from the top of the stack and returns it as the function value.
- **enq(y):** inserts data **y** at the tail of the queue.
- **deq():** removes the data from the head of the queue and returns it as the function value.

Operations: push(a), push(b), enq(pop()), enq(c), push(d), push(deq()), x ← pop()

○ Garbage because stack is empty

○ a

[✓ b] ✓

○ d

○ c

✓ Given the set A with N elements, what is the running time in deleting an element in the set implemented using bit vector of size M? *2/2

Answer format: O(x)   //Replace x with a value

○ O(N*N)

[✓ O(1)] ✓

○ O(N * M)

○ O(M)

○ O(N)

✗ Which of the following describes a characteristic of array-implementation of ADT list? * 0/2

○ Moving to the middle of a list takes an amount of time proportional to the number of elements, as it requires visiting all the entries from the beginning to the middle.

○ In addition to space for each element in a list, space for a pointer to the next element is also required.

[✓ Regardless of the actual number of elements in a list, insertion and removal of an element can be performed in a fixed time.] ✗

○ The space required for an empty and a full list is the same.

Correct answer

[✓ The space required for an empty and a full list is the same.]

✓ Which of these statements is true about linked list and array list? * 2/2

○ The time to insert the first element of the list is the same for linked list and array list.

[✓ The time to update the last element of the list is shorter in an array list than in linked list.] ✓

○ The time to delete the first element of the array list is shorter compared to linked list.

○ Elements for array list and linked list are stored in contiguous memory locations.

○ Array list can only be accessed sequentially while linked list can only be accessed randomly.

✓ Given 2 sets: *3/3

A = {5, 2, 11, 16, 17, 22, 13}

B = {1, 5, 12, 11, 19, 17, 22, 13}

If the 2 sets are to be represented in memory using bit-vector, what is the size of the smallest possible size of the bit-vector?

23 ✓

✗ Given the array containing the following elements: *0/3

6, 8, 4, 7, 2, 3, 9, 1, 5

If the array is made into a max heap using maxheapify, what is the new arrangement of the array?

(Start heapifying at the lowest level parent)

Note: Separate the numbers with a comma ONLY

9,8,5,7,2,3,4,1,5 ✗

Correct answer

9,8,6,7,2,3,4,1,5

✗ Insert the following elements in sequence into an initially empty max heap:
6, 8, 4, 7, 2, 3, 9, 1, 5

Write the contents of the heap. *0/3

Note: Separate the numbers with a comma ONLY

9,8,6,7,5,3,4,1,2 ✗

Correct answer

9,7,8,6,2,3,4,1,5

✗ After seven keys **15, 11, 16, 2, 17, 22, 13** are sequentially stored in an empty hash table where the chain method is used for resolving collisions, how many comparisons are needed to search for the key **22** in the table? *0/3

Here, the hash table size is 5, and the hash function "x mod 5" is used for the key x. In the chain method, each table entry has a linked list, and synonymous keys are stored at the tail of each linked list.

2 ✗

Correct answer

3

✓ The following numbers are inserted into an empty binary search tree in the given order: **10, 1, 3, 5, 15, 12, 16** *3/3

What is the height of the binary search tree?

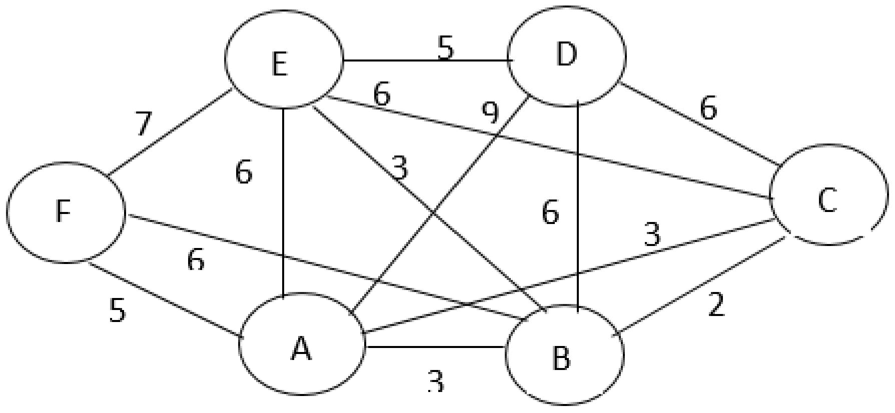
(The height is the maximum distance of a leaf node from the root)

3 ✓

DSA FINALS REVIEWER - PART 2 4 of 10 points

Keep it up!! ^-^

Given a directed graph:



Determine the minimum cost of the spanning tree (MST) using Prim's algorithm starting at vertex E. List the edges and their corresponding weights according to the order they are entered in the MST.

List of arcs in the BFS spanning forest:

✓ **1st Arc** * 1/1

Answer format: (x,y) //representing the arc x->y

(E,B) ✓

✓ **2nd Arc** * 1/1

Answer format: (x,y) //representing the arc x->y

(B,C) ✓

✗ **3rd Arc** * 0/1

Answer format: (x,y) //representing the arc x->y

(C,D) ✗

Correct answers

(C,A)

(A,C)

(B,A)

(A,B)

4th Arc *

0/1

Answer format: (x,y) //representing the arc x->y

(D,A)

Correct answers

(A,F)

(F,A)

5th Arc *

0/1

Answer format: (x,y) //representing the arc x->y

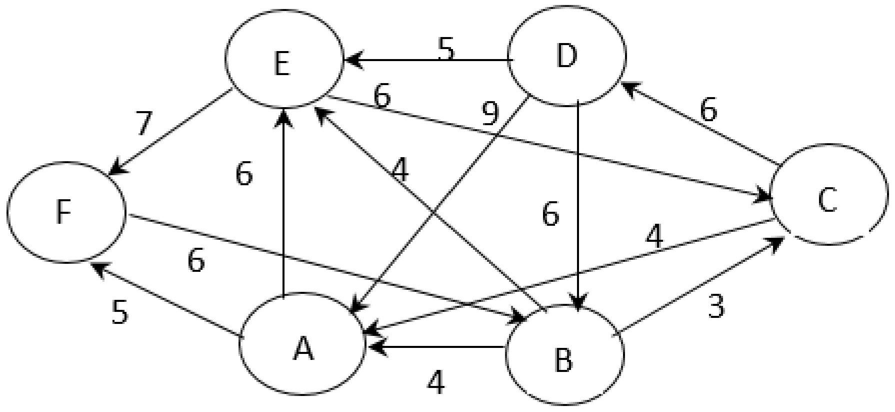
(A,F)

Correct answers

(E,D)

(D,E)

Given a directed graph:



Determine the BFS spanning forest starting at vertex F by listing the arcs according to the order they are entered into spanning forest. Adjacent vertices are sorted in ascending order according to the weights. If equal weights, arrange in ascending order according to vertex. If succeeding calls to BFS are necessary, make calls starting with the remaining vertex with the smallest value.

List of arcs in the BFS spanning forest:

✓ **1st Arc** * 1/1

Answer format: (x,y) //representing the arc x->y

(F,B) ✓

✓ **2nd Arc** * 1/1

Answer format: (x,y) //representing the arc x->y

(B,C) ✓

✗ **3rd Arc** * 0/1

Answer format: (x,y) //representing the arc x->y

(C,D) ✗

Correct answer

(B,A)

✗ **4th Arc** * 0/1

Answer format: (x,y) //representing the arc x->y

(D,A) ✗

Correct answer

(B,E)

✗ **5th Arc** * 0/1

Answer format: (x,y) //representing the arc x->y

(A,E) ✗

Correct answer

(C,D)

DSA FINALS REVIEWER - PART 3 4 of 14 points

You got this!! ^-^

A dictionary of student records, each of which is uniquely identified through the ID, is represented in internal memory using an open hash table. Each group in the open hash table is a list of student records having the same hash value and is sorted in ascending order according to ID. The hash value is determined by adding all the bits with value 1 in the ID number, and the result is reduced to a value which is within the range of the open hash table. Given below is the prototype which will be used in insert operation.

Prototype: int openHash(unsigned long IDen);

//Data type definition

```
#define SIZE 0XD

typedef struct {
  char LN[16], FN[24], MI;
  /* Last name, First name, and Mid Initial */
}nametype;
typedef struct {
  unsigned long IDen;
  nametype name;
  char course[8];
  int year;
}studRec;
typedef struct node {
  studRec stud;
  struct node *next;
}*List;

typedef struct {
  List Head[SIZE];
  int count;    // # of elements in the dictionary
}Dictionary;
```

Complete the code below by filling-in the blanks:

```
void insertDic(Blank_01 D, studRec S)
{
   int hash;
   List *ptr, temp;
   hash = Blank_02    //determines the hash value

   /* Determines the position of the element to be inserted */
   for(ptr = Blank_03; Blank_04 && Blank_05; ptr = &(*ptr)->next) {}

   /* Conditions for a possible insertion */
   if (Blank_06 || Blank_07) {   //Black_06 and Blank_07 are conditions
      temp = (List) malloc(sizeof(struct node));
      if (temp != NULL) {
         // Statements to insert the element
      }
   }
}
```

✓ **Blank_01** * 2/2

Do not put UNNECESSARY space. Be extra careful with semicolons.

Dictionary * ✓

✓ **Blank_02** * 2/2

Do not put UNNECESSARY space. Be extra careful with semicolons.

openHash(S.IDen); ✓

✗ **Blank_03** * 0/2

Do not put UNNECESSARY space. Be extra careful with semicolons.

D->Head[hash] ✗

Correct answer

&D->head[hash]

✗ **Blank_04** * 0/2

Do not put UNNECESSARY space. Be extra careful with semicolons.

ptr!=NULL ✗

Correct answers

*ptr!=NULL

*ptr != NULL

✗ **Blank_05** * 0/2

Do not put UNNECESSARY space. Be extra careful with semicolons.

ptr->stud.IDen<S.IDen ✗

Correct answers

S.IDen>(*ptr)->stud.IDen

(*ptr)->stud.IDen<S.IDen

✕ **Blank_06** * 0/2

Do not put UNNECESSARY space. Be extra careful with semicolons.

```
ptr==NULL
```
✕

Correct answers

```
*ptr==NULL
*ptr == NULL
```

✕ **Blank_07** * 0/2

Do not put UNNECESSARY space. Be extra careful with semicolons.

```
ptr->stud.IDen!=S.IDen
```
✕

Correct answers

```
(*ptr)->stud.IDen>S.IDen
(*ptr)->stud.IDen!=S.IDen
S.IDen<(*ptr)->stud.IDen
S.IDen!=(*ptr)->stud.IDen
```

DSA FINALS REVIEWER - PART 4

14 of 24 points

Last push!! ^-^

Given the definition of an labeled adjacency matrix and an edge:

```
#define SENTINEL   999      //  for infinity ∞
#define MAX_VERTEX   10     // vertices 0, 1, 2,…, 9
#define ARRAY_SIZE  0XF

/*** Data Structure Definition ****/
typedef int adjMatrix[MAX_VERTEX][MAX_VERTEX];
typedef struct {
   int u,v;              // vertices representing the edge
   int weight;
}edgetype;

typedef struct {
   edgetype edges[ARRAY_SIZE];
   int count;     // actual # of edges in the array
}edgelist;
```

Function Specification: Function createEdgeList() creates and returns a list of edges arranged in ascending order according to weights, given a labeled adjacency matrix M of an undirected graph. Assume that in the adjacency matrix, a sentinel value represents no edge which includes the diagonal of the matrix. Also note that in an undirected graph the edge (u,v) = (v,u).

Complete the function by filling-in the blanks.

```
                                 adjMatrix
edgelist createEdgeList(Blank_01 M)
{
   int x, y, pos;  {0};
   edgelist  E = Blank_02        // Declares and initializes the edge list E to be empty
                                 // Note: for the array, just initialize one component

   for(x = 0; x < MAX_VERTEX; x++){
                   x+1       y < MAX_VERTEX
      for(y = Blank_03; Blank_04; y++){            // Reminder: (u,v) == (v,u)
              M[x][y] != SENTINEL
         if (Blank_05){                            // Edge is found

            /* finds the proper position of the edge in the list */
                          pos < E.count       E.edges[pos].weight  <  M[x][y]
            for(pos = 0; Blank_06 && Blank_07; pos++){}

            /* Shifts succeeding elements to make room for the new edge */
                           & E.edges[pos+1]          sizeof(edgetype) * E.count - pos
            memcpy(Blank_08, Blank_09, Blank_10);
                                   & E.edges[pos]

            /* Insertion Proper */
            edgelist edge = Blank_11         // declares and initialize edge to contain
                                             // the edge data to be inserted in the list

            E.edges[Blank_12] = edge;        // inserts the edge in the list
            E.count++;
         }
      }
   }
   return E;
}
```

✗ **Blank_01** * 0/2

Do not put UNNECESSARY spaces and be careful with semicolons.

adjMatrix * ✗

Correct answer

adjMatrix

✗ **Blank_02** * 0/2

Do not put UNNECESSARY spaces and be careful with semicolons.

malloc(sizeof(edgelist)); ✗

Correct answers

{0};

{{0},0};

✗ **Blank_03** * 0/2

Do not put UNNECESSARY spaces and be careful with semicolons.

x ✗

Correct answer

x+1

✓ **Blank_04** * 2/2

Do not put UNNECESSARY spaces and be careful with semicolons.

y<MAX_VERTEX ✓

✓ **Blank_05** *
2/2

Do not put UNNECESSARY spaces and be careful with semicolons.

M[x][y]!=SENTINEL ✓

✓ **Blank_06** *
2/2

Do not put UNNECESSARY spaces and be careful with semicolons.

pos<E.count ✓

✓ **Blank_07** *
2/2

Do not put UNNECESSARY spaces and be careful with semicolons.

E.edges[pos].weight<M[x][y] ✓

✓ **Blank_08** *
2/2

Do not put UNNECESSARY spaces and be careful with semicolons.

&E.edges[pos+1] ✓

✓ **Blank_09** *
2/2

Do not put UNNECESSARY spaces and be careful with semicolons.

&E.edges[pos] ✓

✗ **Blank_10** * 0/2

Do not put UNNECESSARY spaces and be careful with semicolons.

sizeof(edge)*(E.count-pos) ✗

Correct answers

sizeof(edgetype)*(E.count-pos)

(E.count-pos)*sizeof(edgetype)

✗ **Blank_11** * 0/2

Do not put UNNECESSARY spaces and be careful with semicolons.

sizeof(edge)*(E.count-pos) ✗

Correct answer

{x,y,M[x][y]};

✓ **Blank_12** * 2/2

Do not put UNNECESSARY spaces and be careful with semicolons.

pos ✓