CS 2123 Data Structures

Spring 2016 – Final Exam – May 9, 2016 Monday 03:15 pm - 05:45 pm

You have 150 min. + 5 min for the survey. Good luck.

This exam has 10 questions in 12 pages and an exit survey in page 13

You can use the 2-page C reference card posted in the class web page.

Name:…….. Score: ……./100 +(2pt bonus credit for the survey)

1. (10pt) Fill the diagram to show the contents of memory after each line of the following code:

```
int v1[] = {10,15}, v2 = 25;
int *p1 = v1 + 1, **p2 = &p1;
*p1-- = 60;
**p2 = 70;
v2 = *p1 + **p2;
```

Var	Addr	content
v1 [0]	1004	
v1[1]	1008	
v2	1012	
p1	1016	
p2	1020	

2. (10pt) Implement **`char *ExpandedString(char *s, char ch);`** which **creates a new string** by inserting the given character **ch** after each char in the given string **s**. If there is no memory, it returns NULL.
   For example, `ExpandedString("ABC", '-');` returns a new string as `"A-B-C-"`

```
char *ExpandedString(char *s, char ch);
{       /* Assume standard libraries stdlib.h, stdio.h, string.h are included */
```

Name:.....

3. (10pt) A color image is a 2D array of pixels where each pixel is represented by three integers showing Red, Green, Blue (RGB) components. Suppose we store a color image in a text file using a very simple format as follows: First two integers in the file represent the numbers of rows and columns. Then the file contains that many rows and columns of pixels, where each pixel has three integers to represent RGB components. For example a 3x4 color image would be saved in a file as follows:

```
3   4
1   2   3   1   2   3   1   2   3   1   2   3
3   6   5   3   34  43  51  43  5   4   8   4
6   45  34  53  5   4   8   4   66  86  12  43
```

Complete the following program which takes the file name from the command line and reads the pixel values into a dynamically create 2D array of **`pixelT`** structure defined below. So we can access the colors of each pixel using `img[i][j].r, img[i][j].g, img[i][j].b`

/* suppose all standard libraries and our book libs are included here */

```
typedef struct pixel {
    int r, g, b;
} pixelT;

int main(int argc, int *argv[])
{
  FILE *fp;
  int row, col, i, j;
  pixelT **img;

  if (argc<2) {
     printf("Usage: prog filename\n");
     exit(-1);
  }

  if((fp=fopen(argv[1], "r"))==NULL){
     printf("File cannot be opened\n");
     exit(-1);
  }

 fscanf(fp,"%d %d", &row, &col);
```

```
 /* dynamically create 2D array of pixelT  (5pt) */
```

```
/* read the RGB values of each pixel into the 2D array (3pt) */
```

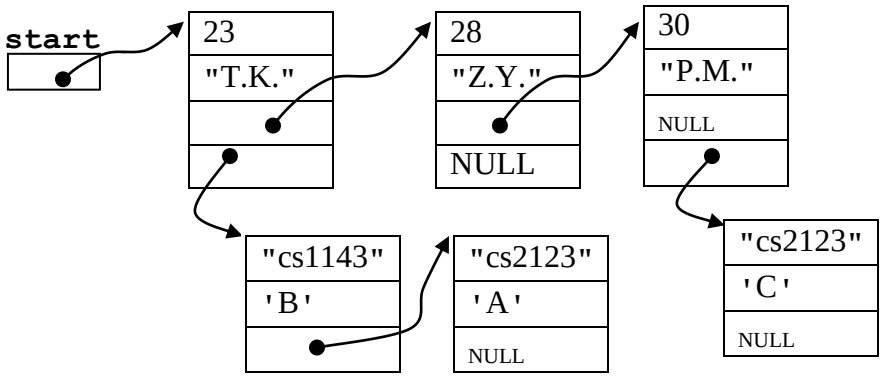
```
/* Free the dynamically allocated memory  (2pt)*/
```

4. (10pt) Suppose we store student IDs and their names in a single linked list. In addition to ID, name and next fields, each cell in this list contains a pointer (say courses) to access another single linked list where we store some information about the courses taken by each student. Suppose we store only course code (cs2123), letter grade, and next fields. Specifically, we use the following structures to store students and their courses.

```
typedef struct course_cell {
   char code[7];  // "cs2123"
   char letter_grade;
   struct course_cell *next;
} courseT;

typedef struct student_cell {
   int ID;
   char name[20];
   struct student_cell *next;
   courseT *courses;
} studentT;
```

Suppose somehow we created the below data structure using the above structures.



Now **implement** `void DisplayClass(studentT *start, char *code);` which lists the students' ID, name and grade in the given course. For example, it generates the below table when we call it as `DisplayClass(start, "cs2123");`

```
Course name: cs2123
ID   Name  Grade
--   ----  ------
23   T.K.     A
30   P.M.     C
```

Note: Don't worry about the format and spaces too much, just generate the list and make sure that you have the header as shown and one student in each line!

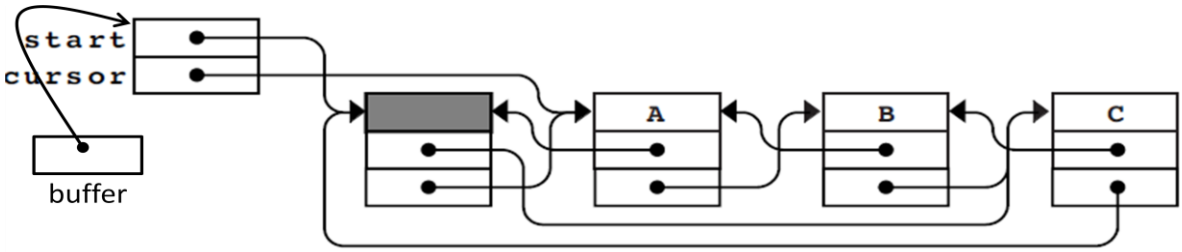
Also standard libraries are included!

Name:…….

```
void DisplayClass(studentT *start, char *code)

{
```

5. (10pt) Recall the buffer ADT, `buffer.h` and suppose we consider its implementation based on the **circular double link list (CDLL)** representation. For **A | B C**, it can be visualized as follows



You are asked to implement the following function which removes the character before the cursor if the cursor is not at the beginning. After calling it for the above case, the buffer will have: **| B C.**

/* CDLL-imp-buf.c */

```
#include "buffer.h"

typedef struct Dcell {
    char ch;
    struct Dcell *prev;
    struct Dcell *next;
} DcellT;

struct bufferCDT {
    DcellT *start;
    DcellT *cursor;
};

bufferADT NewBuffer(void)
{
 bufferADT buffer;
 DcellT *dummy;
 buffer = New(bufferADT);
 dummy = New(DcellT *);

 buffer->start =
 buffer->cursor =
 buffer->start->prev =
 buffer->start->next = dummy;
 return (buffer);
}

/* implementations of  other functions */
```

```
void DeleteCharBeforeCursor(bufferADT buffer)
{
  DcellT *cp;
```

6. (10pt) Span of a binary search tree (BST) is defined as the difference between the **largest** and **smallest** elements in the BST. Write a function `int BST_span()` which returns the span of a binary search tree. Basically find the maximum and minimum in BST and return their difference. You don't need RECURSION! Node declaration of the tree is given below.

```
typedef struct node  {
   int key;
   struct node *left,  *right;
}  nodeT, *treeT;
```

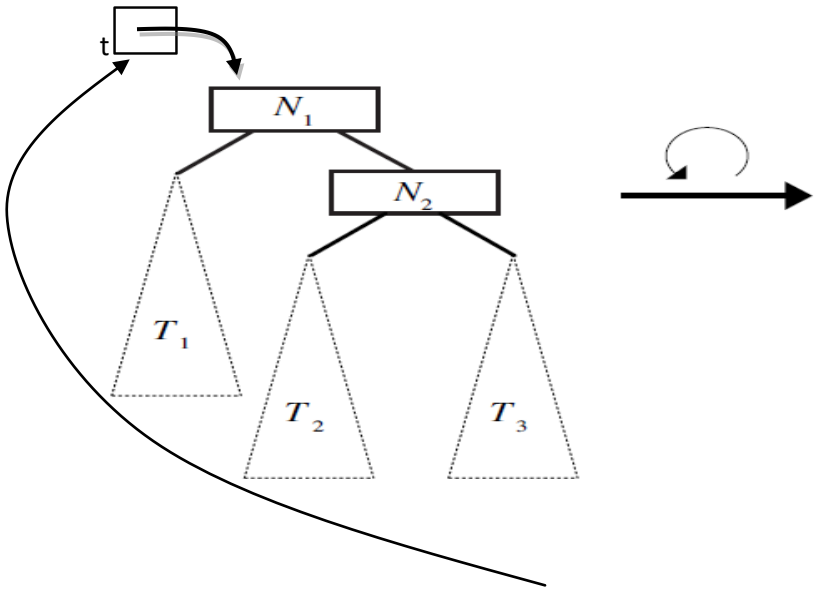
```
int BST_span(nodeT *t)
{
```

Name:.....

7. (10pt) Consider the below binary tree. Suppose we would like to perform left rotation as in AVL insert algorithm. First draw the tree after left rotation on N1-N2 edge, then complete the function to accomplish the left rotation. It will be called as `LeftRotation(&t);`

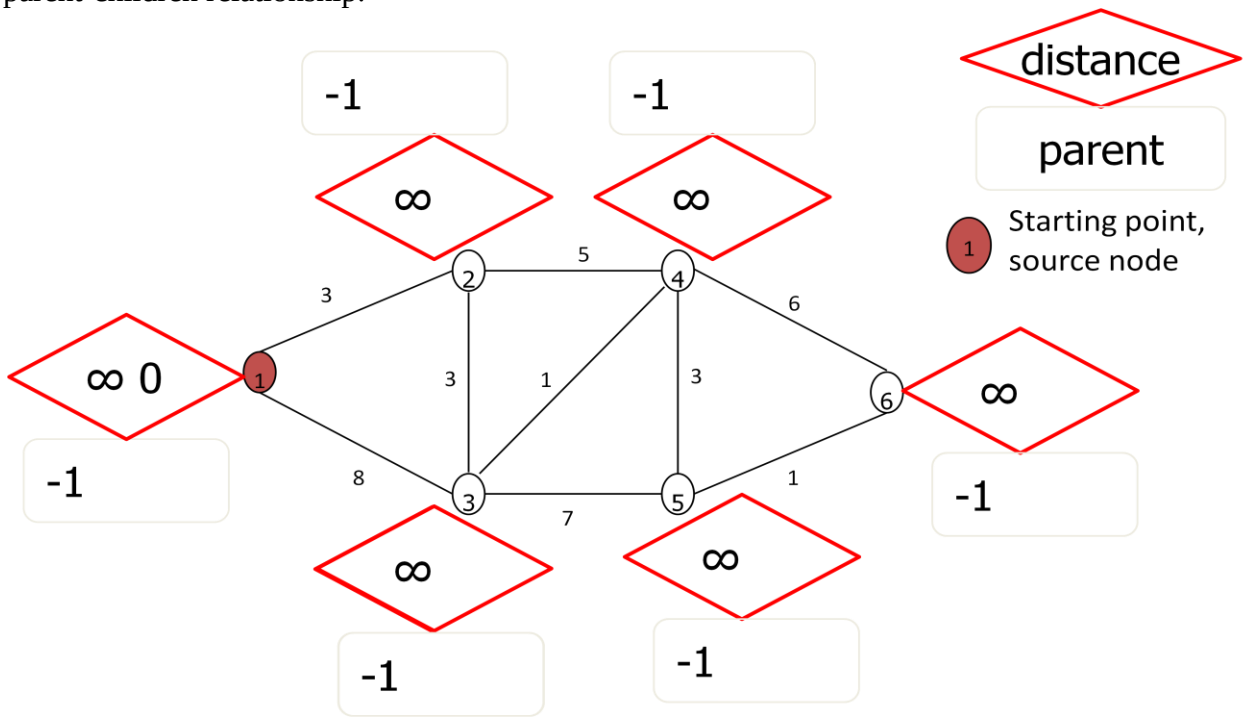
First draw the tree after LEFT rotation on

N1--N2 edge? (4pt.)

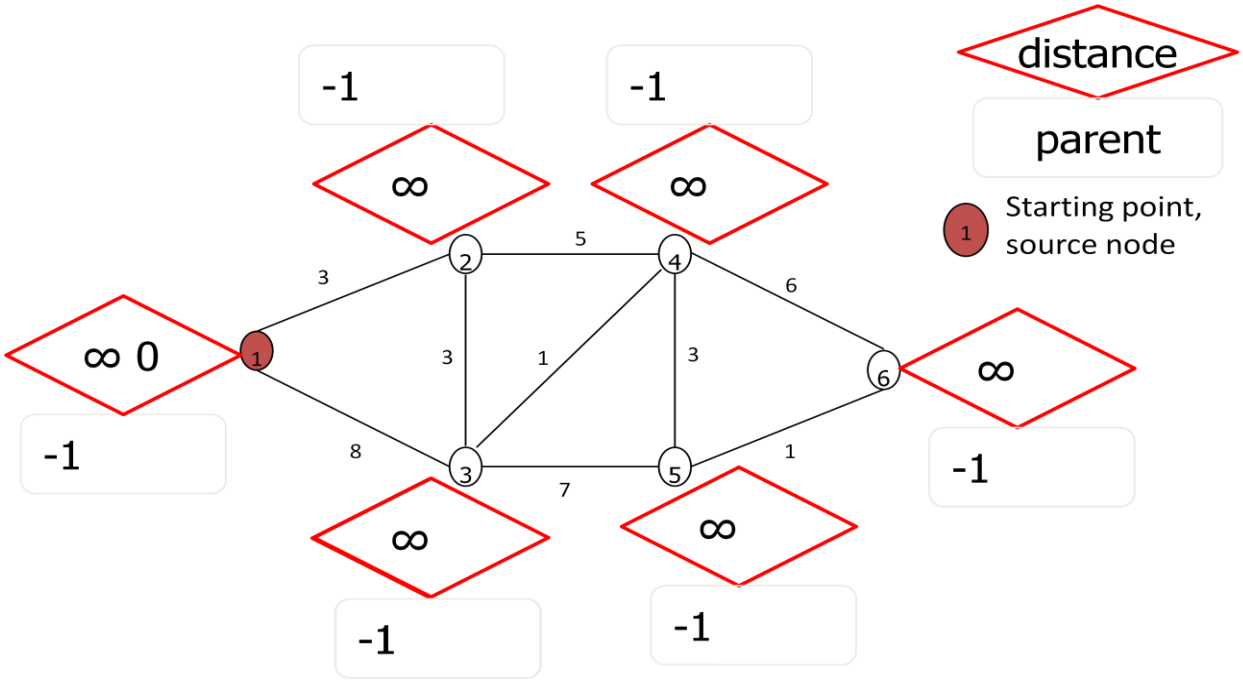



```
LeftRotation(nodeT **tptr)  {
  nodeT *parent, *child;
  parent = *tptr;          // N1
  child = parent->rigtht; // N2
  /*  Write the piece of code to the above LEFT
      rotation (you can use temporary pointers) 6pt */
```

8. (10pt) Using the below graph, show how shortest path algorithm (known as Dijkstra's algorithm) finds the shortest paths starting from **NODE 1** to every other node. Don't just inspect the graph and give solutions! Instead follow the Dijkstra's algorithm by showing all the changes in distance and parent labels, as we did in class. Then use solid arrows to show parent-children relationship.

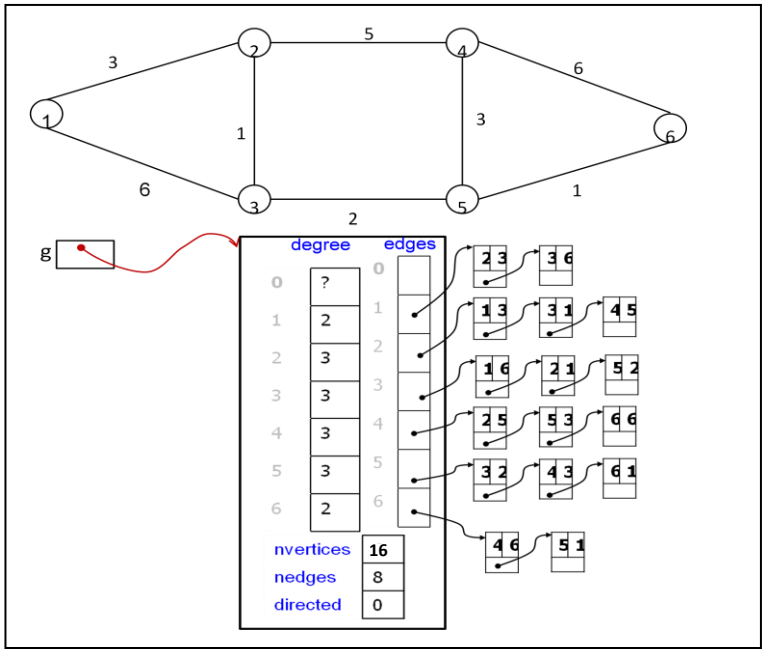


Here is another copy. If you make too many mistakes in the above one, you can use this one…..



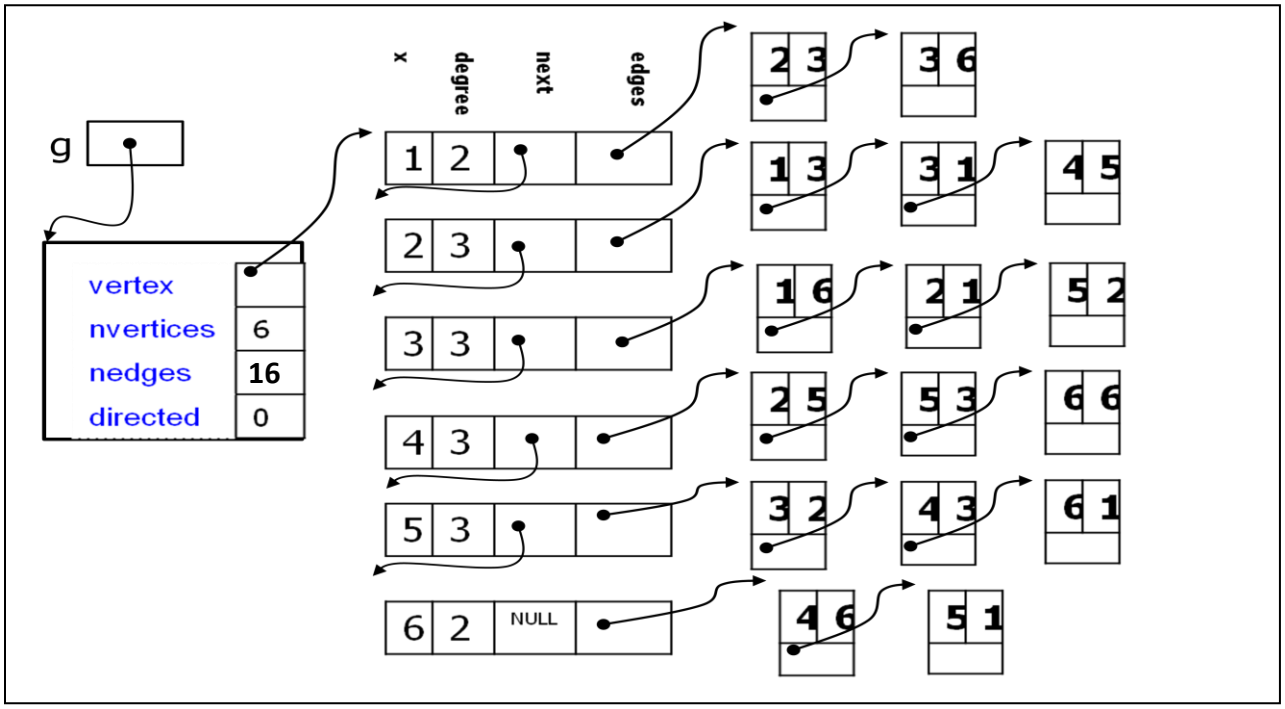
9. (10pt) Recall that we used the following structures to represent a graph using **adjacency list.**

```
#define MAXV 6
typedef struct edgenode {
    int y;
    int w;
    struct edgenode *next;
} edgenodeT;
typedef struct {
    edgenodeT *edges[MAXV+1];
    int degree[MAXV+1];
    int nvertices;
    int nedges;
    bool directed;
} graphT;
graphT *g;
```



One of the disadvantages of the above representation is that the maximum number of nodes is fixed to MAXV. So we waste some spaces when we don't have that many nodes or we cannot use this program if we have more nodes than MAXV. Accordingly, we want to remove fixed size array and use a link list to dynamically store nodes, their IDs, degrees etc. So we need to modify `graphT` and define a new cell structure (say `nodeT`) to store nodes in a linked list. We like to keep `edgenodeT` as is.

We give the new form of `graphT` and `nodeT` structures in the next page such that we can conceptually represent the above same graph as follows:



```
typedef  struct  node {
   int x; // vertex id
   int degree;
   struct  node  *next;
   edgenodeT *edges;
} nodeT;
```

```
typedef struct {
   nodeT *vertex;
   int nvertices;
   int nedges;
   bool directed;
} graphT;
```

NOW you are asked to implement `print_total_link_w_per_node(graphT *g);` based on the new representations/structures defined above. It basically prints each node ID and the total weight of the links that are incidents to that node. For the above graph, your program should print:

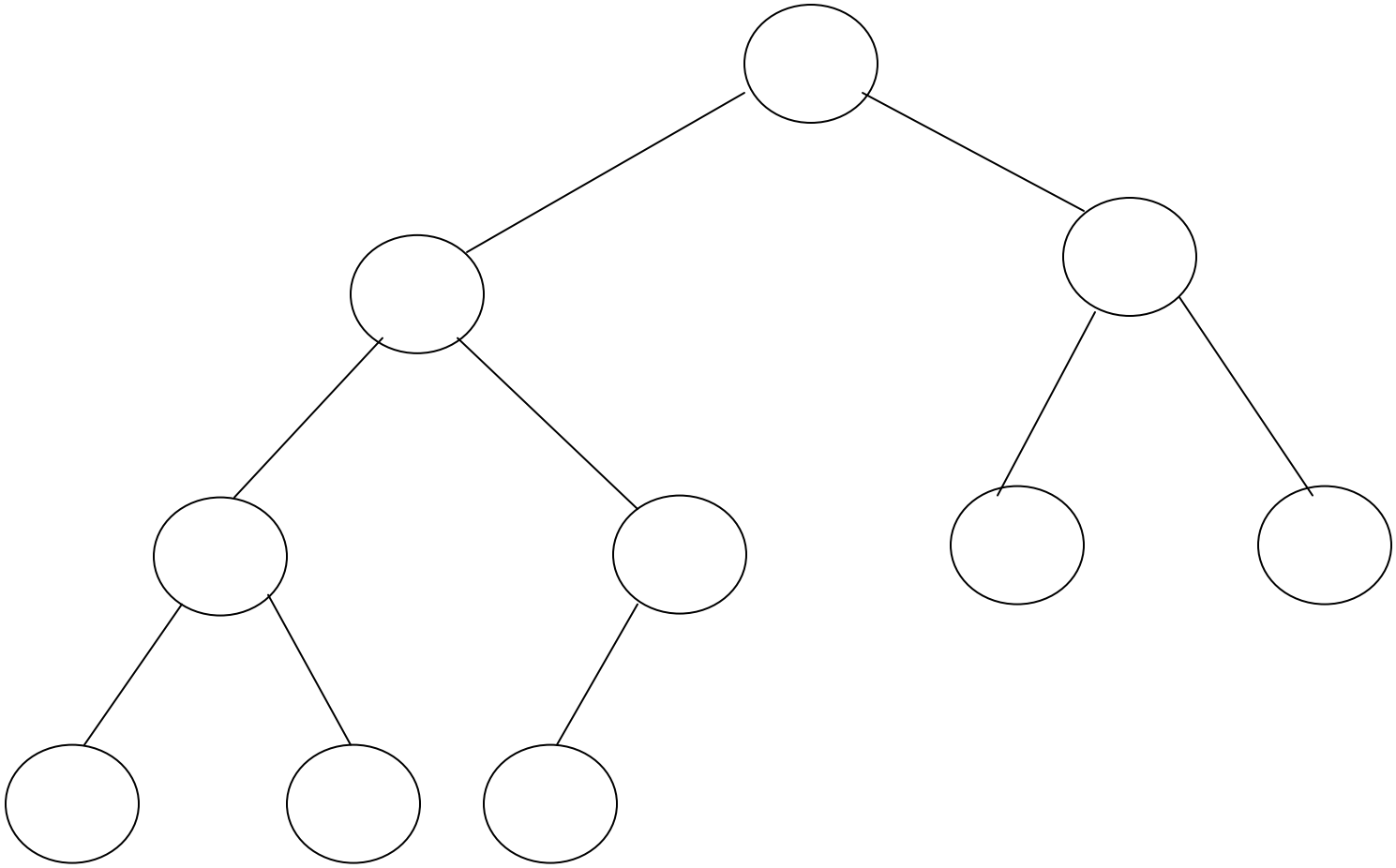
```
Node 1 --> total link W =  9
Node 2 --> total link W =  9
....
```

```
print_total_link_w_per_node(graphT *g)
{
   edgenodeT *pe;
   nodeT *pn;
```

10. (10pt) You are given the following heap array of integers, which is not a MAX HEAP yet.

	0	1	2	3	4	5	6	7	8	9
heap	11	13	12	16	10	14	19	18	15	17

a. (7pt) You are asked to visualize it as a tree structure and make the necessary changes to build a valid MAX heap *(i.e., the value of every node is greater than or equal to the values of its children).* You don't need to re-draw the tree everytime, just cross the previous values and put the new values on the same tree as you build the heap upwards. We need to see changes!



b. (3pt) Give the new version of the heap array, after the heap is built.

	0	1	2	3	4	5	6	7	8	9
heap										

Exit Survey (2pt bonus credit)

1. Did you attend the last class on 4/28 when we did a general review for the final exam?
   [1] Yes [2] No

2. If **Yes**, do you think that the review session helped you in this final exam?
   [1] Definitely not. [2] Not really. [3] Somewhat. [4] Definitely.

---

3. Did you solve the sample exam problems provided in the class web page?
   [1] No, None. [2] Yes, Some. [3] Yes, Most. [4] Yes, All of them.

4. If **Yes**, do you think these sample problems helped you in this final exam?
   [1] Definitely not. [2] Not really. [3] Somewhat. [4] Definitely.

As mentioned in the class, The Department had Common TAs available in the Main CS lab (NPB 2.118) almost all the time (from 9am to 5pm on the week days) to help all students.

5. Throughout the semester, how many times did you get any help from **the common TAs other than** our own course TA (Mahmoud Abdelsalam)?
   [1] None. [2] 5 times or less. [3] 10 times or less. [4] More than 10 times.

Answer the following questions
if you get any help from common TAs other than our own course TA (Mahmoud Abdelsalam).

6. What **type of help** did you get from other common TAs in the main CS lab? (e.g, Help with programming assignments, solving sample exam questions, system problems such as login etc.)

   .....

7. How **easy** did you find it to meet with other common TAs in the main CS lab?
   [1] Not at all easy [2] Slightly ... [3] Moderately ... [4] Quite easy

8. How **knowledgeable** were other common TAs about the subject matter of the course?
   [1] Not at all knowledgeable [2] Slightly... [3] Moderately ... [4] Quite knowledgeable

9. How **clearly** did other common TAs present material?
   [1] Not at all clearly [2] Slightly ... [3] Moderately ... [4] Quite clearly

10. Overall, how would you rate the **usefulness** of other common TAs in your learning the subject matter and passing this course?
    [1] Not at all useful [2] Slightly ... [3] Moderately ... [4] Quite useful

Please use the back of this page if you like to provide more feedback about Common TA system (e.g., its weaknesses, strengths, how to make it more effective etc.) Thanks!