GIẢI ĐỀ ÔN DSA

Đề 1

Câu 1:

$$S = 1 + \frac{1}{1+2} + \frac{1}{2+3} + \cdots + u(n)$$

1. Xác định $u(n)$

$$u(n) = \frac{1}{2n-1}$$

2. Viết hàm `float S(int n) {}` sử dụng kỹ thuật đệ quy để tính tổng trên

```
float S(int n) {
    if(n == 1) {
        return 1;
    }
    return 1.0/(2*n - 1) + S(n - 1);
}
```

3. Phân tích theo quy trình 4 bước:

- Base case:

```
if(n == 1) {
        return 1;
    }
```

- Kết quả base case:
  Trả về 1.
- Trước base case
  Là khi `n = 2` hàm `S(2)` trả về `1.333`
- Trước của trước base case
  Là khi `n = 3` hàm `S(3)` trả về `1.5333`

4. Cài hàm `S` thành `S1` không dùng đệ quy

```
float S1(int n) {
    float sum = 0;
    while(n != 0) {
        sum = sum + 1.0/(2*n-1);
        n -= 1;
    }
    return sum;
}
```

Câu 2:

```
#include<stdio.h>
#include<stdlib.h>

struct Node {
        char data;
        struct Node* next;
};
typedef struct Node Node;

struct LinkedList {
    Node* head;
};
typedef struct LinkedList LinkedList;

void init(LinkedList *list) {
        list->head = NULL;
}

Node* makeNode(char data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertTail(char data, LinkedList *list) {
```

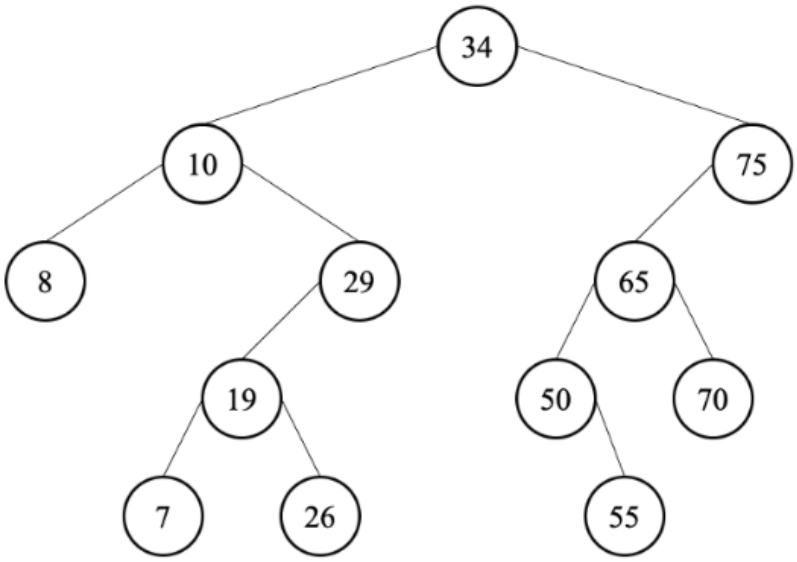
```
    Node* newNode = makeNode(data);
    if(list->head == NULL) {
        list->head = newNode;
    }else {
        Node* tmp = list->head;
        while(tmp->next != NULL) {
            tmp = tmp->next;
        }
        tmp->next = newNode;
    }
}

void print(LinkedList list) {
    if(list.head == NULL) return;
    Node* tmp = list.head;
    while(tmp) {
        printf("%c", tmp->data);
        tmp = tmp->next;
    }
}

int main() {
    LinkedList list;
    init(&list);
    char str[] = "DSATTHS12023";
    int n = sizeof(str)/sizeof(str[0]);
    for(int i = 0; i < n; i++) {
        insertTail(str[i], &list);
    }
    print(list);
    return 0;
}
```

Câu 3:

Câu 3: Cho cây nhị phân tìm kiếm sau:



1. Cây trên có lỗi không? Tại sao?
   Cây trên bị lỗi. Vì Node mang số 7 không nằm đúng vị trí. Nghĩa là `7 < 10` nhưng lại nằm phía bên phải của Node mang số 10 không đúng tính chất của cây nhị phân tìm kiếm.

`Sửa lại(nếu có)` Mang Node số 7 gắn và Node con bên trái của Node số 8.

2. Viết chương trình C nhập cây và xuất ra dạng dãy số
3. Bổ sung hàm `deleteNode(BinaryTree *tree, int val)`. Hãy xóa 7, 50, 65

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

typedef struct TreeNode{
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
}TreeNode;

typedef struct BinaryTreeType {
    struct TreeNode* root;
} BinaryTree;

TreeNode* makeNode(int data) {
```

```
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void insert(BinaryTree* tree, int data) {
    TreeNode** node = &(tree->root);
    while (*node) {
        if (data < (*node)->data) node = &((*node)->left);

        else node = &((*node)->right);
    }
    *node = makeNode(data);
}

int isLeaf(TreeNode* root) {
    if(!root) return 0;
    if(!root->left && !root->right) {
        return 1;
    } else return 0;
}

void deleteNode(BinaryTree* tree, int val) {
    // search the val
    TreeNode* curr = tree->root;
    TreeNode* prev = NULL;
    while (1) {
        if (curr == NULL)    // not found
            return;
        if (curr->data == val) {    // found
            // delete the curr node

            // case 1: no child / leaf
            if (curr->left == NULL && curr->right == NULL) {
                free(curr);
                if (prev == NULL) {
                    tree->root = NULL;
                }
                else {
```

```
                if (prev->data > val) {
                    prev->left = NULL;
                }
                else {
                    prev->right = NULL;
                }
            }
            return;
        }
        // case 2: one child
        if (curr->left == NULL) {
            if (prev == NULL) {
                tree->root = curr->right;
            }
            else {
                if (prev->data > val) {
                    prev->left = curr->right;
                }
                else {
                    prev->right = curr->right;
                }
            }
            free(curr);
            return;
        }
        if (curr->right == NULL) {
            if (prev == NULL) {
                tree->root = curr->left;
            }
            else {
                if (prev->data > val) {
                    prev->left = curr->left;
                }
                else {
                    prev->right = curr->left;
                }
            }
            free(curr);
            return;
        }
```

```
            // case 3: 2 children (leafs)
            if (isLeaf(curr->left) && isLeaf(curr->right)) {
                // copy data from the right
                curr->data = curr->right->data;
                free(curr->right);
                curr->right = NULL;
                return;
            }
            // case 4: curr->right is not a leaf
            if (!isLeaf(curr->right)) {
                TreeNode* leftMost = curr->right;
                TreeNode* parent = NULL;
                while(leftMost != NULL && leftMost->left !=
NULL) {
                    parent = leftMost;
                    leftMost = leftMost->left;
                }
                curr->data = leftMost->data;
                val = leftMost->data;
                prev = curr;
                curr = curr->right;
            }

        }
        prev = curr;
        if (curr->data > val) {
            curr = curr->left;
        }
        else {
            curr = curr->right;
        }
    }
}

void print(TreeNode* root) {
    if(!root) return;
    printf("%2d ", root->data);
    print(root->left);
    print(root->right);
}
```

```c
int main() {
    BinaryTree tree;
    tree.root = NULL;
    int arr[] = {34, 10, 8, 29, 19, 7, 26, 75, 65, 50, 70, 55};
    int n = sizeof(arr)/sizeof(arr[0]);
    for(int i = 0; i < n; i++) {
        insert(&tree, arr[i]);
    }
    printf("Initial tree:          ");
    print(tree.root); printf("\n");
    int deletearr[] = {7, 50, 65};
    int m = sizeof(deletearr)/sizeof(deletearr[0]);
    for(int i = 0; i < m ; i++) {
        deleteNode(&tree, deletearr[i]);
        printf("Tree after deleted %2d: ", deletearr[i]);
        print(tree.root);
        printf("\n");
    }
    return 0;
}
```