รายงาน

เรื่อง โปรแกรม TREE STRUCTURE BY NODE POINTER METHOD

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

รายงาน

เรื่อง โปรแกรม TREE STRUCTURE BY NODE POINTER METHOD

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา ENGCE124 โครงสร้างข้อมูลและขั้นตอนวิธี (Data Structures and Algorithms) หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์ สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่ ในระดับปริญญาตรีปีที่ 2 โดยมีจุดประสงค์ในการอธิบายโค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD รวมถึงอธิบายหลักการทำงานของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD และอธิบายผลลัพธ์การใช้งานโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD

ผู้จัดทำรายงานหวังว่า รายงานฉบับนี้จะเป็นประโยชน์กับผู้ที่สนใจ หรือนักศึกษาทุกท่าน ที่กำลังหาศึกษาในหัวข้อของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD หากมีข้อแนะนำหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้ และขออภัยมา ณ ที่นี้

ผู้จัดทำ

นายพงษ์พันธุ์ เลาวพงศ์

วันที่ 28/08/2567

สารบัญ

โค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD พร้อมคำอธิบาย

```
#include <stdio.h> // ใช้ printf
#include <conio.h> // ใช้ getch
#include <stdlib.h> // ใช้ random, malloc
#include <math.h> // ใช้ pow
#define MaxNode 40
struct Node { // ประกาศโครงสร้างของโหนดต้นไม้
    int info;
    struct Node *lson;
    struct Node *rson;
};
struct Node *T, *address[MaxNode]; // ประกาศตัวชี้ T ของโหนดต้นไม้
int i,N,info[MaxNode];
char ch;
Node *Allocate() { // จัดสรรหน่วยความจำสำหรับโหนด 1 โหนดจากพูลเก็บข้อมูลและคืนค่าตัวชี้ของโหนด
    struct Node *temp;
    temp=(Node*)malloc(sizeof(Node)); // จัดสรรหน่วยความจำสำหรับโหนดตามขนาดที่กำหนด
    return(temp);
}
void CreateTreeNP(int n) {
    int i,temp,Father;
```

โค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD พร้อมคำอธิบาย (ต่อ)

```
struct Node *p, *FatherPT;
 T=NULL; // ตั้งค่าเริ่มต้นให้ตัวชี้ T เป็น NULL
 for (i=1;i<=n;i++){
     p=Allocate(); // จัดสรรหน่วยความจำให้กับโหนด p
     temp=1+rand() % 99; // สุ่มหมายเลขระหว่าง 1 ถึง 99
     info[i]=temp; // เก็บข้อมูลเพื่อใช้ตรวจสอบความถูกต้องของลำดับ
     address[i]=p; // เก็บที่อยู่ของโหนดตามลำดับ
     p->info=temp; // เก็บข้อมูลลงใน INFO ของโหนด
     p->lson=NULL; // ตั้งค่าเริ่มต้นให้ LSON=NULL
     p->rson=NULL; // ตั้งค่าเริ่มต้นให้ RSON=NULL
     if (T==NULL) { // ตรวจสอบว่า T เป็น NULL หรือไม่?
         T=p; // ตั้งค่า T ชี้ไปที่โหนดแรกเพื่อเริ่มการสร้างต้นไม้
     }
     else
     {
         Father=(i/2); // คำนวณหมายเลขของพ่อ
         FatherPT=address[Father]; // รับพ้อยเตอร์ของโหนดพ่อ
         if(FatherPT->lson == NULL)
             FatherPT->lson=p; // เชื่อม LSON ไปยังโหนดใหม่
         else
             FatherPT->rson=p; // เชื่อม RSON ไปยังโหนดใหม่
```

โค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD พร้อมคำอธิบาย (ต่อ)

```
        }
    }
}
void ShowTree(){
    int j,level,start,ends;
    j=1;
    level=1; // เริ่มต้นที่ระดับ 1
    printf("\n");
    while (info[j] != NULL) {
    start=pow(2,level)/2; // คำนวณจุดเริ่มต้นของโหนดในระดับนี้
    ends=pow(2,level)-1; // คำนวณจุดสิ้นสุดของโหนดในระดับนี้
    for (j=start;j<=ends;j++)
        if(info[j] != NULL) {
            switch (level) {
                case 1 : printf("%40d",info[j]);
                        break;
                case 2 : if (j==2)
                            printf("%20d",info[j]);
                        else
                            printf("%40d",info[j]);
                        break;
```

โค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD พร้อมคำอธิบาย (ต่อ)

```
                case 3 : if(j==4)
                            printf("%10d",info[j]);
                         else
                            printf("%20d",info[j]);
                             break;
                case 4 : if(j==8)
                            printf("%5d",info[j]);
                         else
                            printf("%10d",info[j]);
                         break;
                case 5 : if(j==16)
                            printf("%d",info[j]);
                         else
                            printf("%5d",info[j]);
                         break;
            }
        }
        printf("\n"); // ลงบรรทัดใหม่
        level++;
    }
}
```

โค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD พร้อมคำอธิบาย (ต่อ)

```
void PreOrder(struct Node *i)
{
    if (i != NULL) { // ถ้า i ไม่เป็น NULL
        printf(" %d",i->info); // แสดงข้อมูล INFO
        PreOrder(i->lson); // เรียกใช้ PreOrder กับลูกซ้าย
        PreOrder(i->rson); // เรียกใช้ PreOrder กับลูกขวา
    }
}
void InOrder(struct Node *i)
{
    if (i != NULL) { // ถ้า i ไม่เป็น NULL
        InOrder(i->lson); // เรียกใช้ InOrder กับลูกซ้าย
        printf(" %d",i->info); // แสดงข้อมูล INFO
        InOrder(i->rson); // เรียกใช้ InOrder กับลูกขวา
    }
}
void PostOrder(struct Node *i)
{
    if (i != NULL) { // ถ้า i ไม่เป็น NULL
        PostOrder(i->lson); // เรียกใช้ PostOrder กับลูกซ้าย
        PostOrder(i->rson); // เรียกใช้ PostOrder กับลูกขวา
```

โค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD พร้อมคำอธิบาย (ต่อ)

```
        printf(" %d",i->info); // แสดงข้อมูล INFO
    }
}
int main()
{
    N=31;
    CreateTreeNP(N);
    while (ch != 'E') // วนลูปจนกว่าจะกด 'E'
    {
        printf("\nPROGRAM TREE(Node Pointer) \n");
        printf("================================== \n");
        printf("N : %d\n",N);
        printf("Sequence of data : ");
        for (i=1;i<=N;i++) // แสดงลำดับข้อมูล
            printf("%d ",info[i]);
        ShowTree(); // แสดงโครงสร้างต้นไม้
        printf("\nMENU => P:PreOrder I:InOrder O:PostOrder E:Exit");
        printf("\n--------------------------------------------------------------------------\n");
        ch=getch();
        switch (ch)
        {
```

โค้ดของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD พร้อมคำอธิบาย (ต่อ)

```
            case 'P' :  printf("PRE ORDER TRAVERSAL : ");
                        PreOrder(T);
                        printf("\n--------------------------------------------------------------------------\n");
                        break;
            case 'I' :  printf("IN ORDER TRAVERSAL : ");
                        InOrder(T);
                        printf("\n--------------------------------------------------------------------------\n");
                        break;
            case 'O' :  printf("POST ORDER TRAVERSAL : ");
                        PostOrder(T);
                        printf("\n--------------------------------------------------------------------------\n");
                        break;
        } // สิ้นสุด Switch...case
    } // สิ้นสุด While
    return(0);
} // สิ้นสุด MAIN
```

หลักการทำงานของโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD

โปรแกรม TREE STRUCTURE BY NODE POINTER METHOD สร้างต้นไม้และให้การเดินทางผ่านโหนดต้นไม้ในรูปแบบต่างๆ เช่น Pre-order, In-order, และ Post-order โปรแกรมนี้ใช้งานได้สูงสุดถึง 31 โหนด (สำหรับต้นไม้ที่มี 5 ระดับ) ซึ่งเป็นโครงสร้างต้นไม้ที่มีความสมบูรณ์ (complete binary tree) เมื่อพูดถึงการทำงานของแต่ละฟังก์ชันภายในโปรแกรม

1. การประกาศไลบรารีและค่าคงที่ของโปรแกรม

```
#include <stdio.h> // ใช้ printf

#include <conio.h> // ใช้ getch

#include <stdlib.h> // ใช้ random, malloc

#include <math.h> // ใช้ pow

#define MaxNode 40
```

ในส่วนของการประกาศไลบรารีและค่าคงที่ของโปรแกรม มีรายละเอียดดังต่อไปนี้

- #include <stdio.h> : ใช้สำหรับการทำงานกับการป้อนและการแสดงผลข้อมูล เช่น ฟังก์ชัน printf ที่ใช้ในการพิมพ์ข้อมูลไปยังหน้าจอ
- #include <conio.h> : ใช้สำหรับฟังก์ชันที่เกี่ยวข้องกับการป้อนข้อมูลจากแป้นพิมพ์ เช่น getch ที่ใช้ในการอ่านค่าแบบไม่ต้องกด Enter
- #include <stdlib.h> : ใช้สำหรับฟังก์ชันที่เกี่ยวข้องกับการจัดการหน่วยความจำและการสุ่ม เช่น malloc ที่ใช้ในการจัดสรรหน่วยความจำ และ rand ที่ใช้ในการสุ่มค่า
- #include <math.h> : ใช้สำหรับฟังก์ชันทางคณิตศาสตร์ เช่น pow ที่ใช้ในการคำนวณยกกำลัง
- #define MaxNode 40 : กำหนดค่าคงที่ MaxNode ให้มีค่าเป็น 40 ซึ่งจะใช้เป็นขนาดสูงสุดของอาร์เรย์ address และ info เพื่อจัดเก็บโหนดต้นไม้และข้อมูลของโหนด

2. การประกาศโครงสร้างโหนดต้นไม้

```
struct Node { // ประกาศโครงสร้างของโหนดต้นไม้

    int info;
```

2. การประกาศโครงสร้างโหนดต้นไม้ (ต่อ)

```
    struct Node *lson;

    struct Node *rson;

};
```

struct Node ประกาศโครงสร้างข้อมูลของโหนดต้นไม้ ซึ่งประกอบด้วย

- int info; : ตัวแปรเก็บข้อมูลของโหนดต้นไม้.
- struct Node *lson; : ตัวชี้ไปยังลูกซ้ายของโหนด.
- struct Node *rson; : ตัวชี้ไปยังลูกขวาของโหนด.

3. การประกาศพอยน์เตอร์โหนดและตัวแปรต่าง ๆ

```
struct Node *T, *address[MaxNode]; // ประกาศตัวชี้ T ของโหนดต้นไม้

int i,N,info[MaxNode];

char ch;
```

ในส่วนของการประกาศพอยน์เตอร์โหนดและตัวแปรต่าง ๆ มีรายละเอียดดังต่อไปนี้

- struct Node *T : ตัวชี้ที่ใช้ในการเก็บที่อยู่ของโหนดรากของต้นไม้ (root)
- struct Node *address[MaxNode] : อาร์เรย์ของตัวชี้ที่ใช้ในการเก็บที่อยู่ของโหนดแต่ละโหนดในต้นไม้
- int i, N, info[MaxNode]; :
    - i: ตัวแปรตัวนับที่ใช้ในลูปต่างๆ
    - N: จำนวนโหนดในต้นไม้ (สูงสุด 31 สำหรับต้นไม้ที่มี 5 ระดับ)
    - info[MaxNode]: อาร์เรย์ที่ใช้เก็บข้อมูลของโหนดตามลำดับ
- char ch; : ตัวแปรประเภท char ใช้ในการเก็บค่าที่ป้อนเข้ามาจากผู้ใช้ เช่น การเลือกการเดินทางของต้นไม้หรือการออกจากโปรแกรม

4. การจองพื้นที่หน่วยความจำด้วยฟังก์ชัน Allocate

```
Node *Allocate() { // จัดสรรหน่วยความจำสำหรับโหนด 1 โหนดจากพูลเก็บข้อมูลและคืนค่าตัวชี้ของโหนด
    struct Node *temp;
    temp=(Node*)malloc(sizeof(Node)); // จัดสรรหน่วยความจำสำหรับโหนดตามขนาดที่กำหนด
    return(temp);
}
```

ฟังก์ชันนี้ทำหน้าที่จัดสรรหน่วยความจำสำหรับโหนดต้นไม้ใหม่หนึ่งโหนดจากพูลเก็บข้อมูล และคืนค่าตัวชี้ไปยังโหนดที่จัดสรร โดยการทำงานของฟังก์ชันนี้คือการใช้ฟังก์ชัน malloc เพื่อจัดสรรหน่วยความจำสำหรับโครงสร้างของโหนด ซึ่งจะเก็บข้อมูลของโหนดต้นไม้และพ้อยเตอร์ไปยังลูกซ้าย (lson) และลูกขวา (rson) เมื่อจัดสรรหน่วยความจำเสร็จสิ้น ฟังก์ชันจะคืนค่าตัวชี้ไปยังโหนดที่จัดสรร

5. การสร้างต้นไม้ด้วยฟังก์ชัน CreateTreeNP

```
void CreateTreeNP(int n) {
    int i,temp,Father;
   struct Node *p, *FatherPT;
    T=NULL; // ตั้งค่าเริ่มต้นให้ตัวชี้ T เป็น NULL
    for (i=1;i<=n;i++){
        p=Allocate(); // จัดสรรหน่วยความจำให้กับโหนด p
        temp=1+rand() % 99; // สุ่มหมายเลขระหว่าง 1 ถึง 99
        info[i]=temp; // เก็บข้อมูลเพื่อใช้ตรวจสอบความถูกต้องของลำดับ
        address[i]=p; // เก็บที่อยู่ของโหนดตามลำดับ
        p->info=temp; // เก็บข้อมูลลงใน INFO ของโหนด
        p->lson=NULL; // ตั้งค่าเริ่มต้นให้ LSON=NULL
```

5. การสร้างต้นไม้ด้วยฟังก์ชัน CreateTreeNP (ต่อ)

```
        p->rson=NULL; // ตั้งค่าเริ่มต้นให้ RSON=NULL
        if (T==NULL) { // ตรวจสอบว่า T เป็น NULL หรือไม่?
            T=p; // ตั้งค่า T ชี้ไปที่โหนดแรกเพื่อเริ่มการสร้างต้นไม้
        }
        else
        {
            Father=(i/2); // คำนวณหมายเลขของพ่อ
            FatherPT=address[Father]; // รับพ้อยเตอร์ของโหนดพ่อ
            if(FatherPT->lson == NULL)
                FatherPT->lson=p; // เชื่อม LSON ไปยังโหนดใหม่
            else
                FatherPT->rson=p; // เชื่อม RSON ไปยังโหนดใหม่
        }
    }
}
```

ฟังก์ชันนี้สร้างต้นไม้จากจำนวนโหนดที่กำหนด โดยการจัดสรรโหนดใหม่และเชื่อมโยงโหนดเหล่านั้นในโครงสร้างต้นไม้ที่เหมาะสม โดยฟังก์ชันนี้เริ่มจากการตั้งค่าให้ตัวชี้ต้นไม้ T เป็น NULL จากนั้นสร้างโหนดใหม่ด้วยการเรียก Allocate() และสุ่มค่าข้อมูลสำหรับโหนดนั้น การเชื่อมโยงโหนดทำโดยการคำนวณหมายเลขของพ่อและเชื่อมโยงลูกซ้ายหรือลูกขวาตามลำดับ

6. การแสดงรูปแบบของต้นไม้ด้วยฟังก์ชัน ShowTree

```
void ShowTree(){
    int j,level,start,ends;
    j=1;
    level=1; // เริ่มต้นที่ระดับ 1
    printf("\n");
    while (info[j] != NULL) {
    start=pow(2,level)/2; // คำนวณจุดเริ่มต้นของโหนดในระดับนี้
    ends=pow(2,level)-1; // คำนวณจุดสิ้นสุดของโหนดในระดับนี้
    for (j=start;j<=ends;j++)
        if(info[j] != NULL) {
            switch (level) {
                case 1 : printf("%40d",info[j]);
                        break;
                case 2 : if (j==2)
                            printf("%20d",info[j]);
                        else
                            printf("%40d",info[j]);
                        break;
                case 3 : if(j==4)
                            printf("%10d",info[j]);
                        else
```

6. การแสดงรูปแบบของต้นไม้ด้วยฟังก์ชัน ShowTree (ต่อ)

```
                    printf("%20d",info[j]);
                     break;
            case 4 : if(j==8)
                    printf("%5d",info[j]);
                else
                    printf("%10d",info[j]);
                break;
            case 5 : if(j==16)
                    printf("%d",info[j]);
                else
                    printf("%5d",info[j]);
                break;
          }
        }
        printf("\n"); // ลงบรรทัดใหม่
        level++;
    }
}
```

ฟังก์ชันนี้แสดงต้นไม้ในรูปแบบที่จัดรูปแบบตามระดับของต้นไม้ โดยฟังก์ชันนี้แสดงโครงสร้างของต้นไม้ในรูปแบบที่เรียงลำดับตามระดับของต้นไม้ โดยการคำนวณจุดเริ่มต้นและสิ้นสุดของแต่ละระดับและแสดงโหนดแต่ละโหนดให้พอดีกับระดับของต้นไม้

7. การท่องต้นไม้ในลำดับ Pre Order ด้วยฟังก์ชัน PreOrder

```
void PreOrder(struct Node *i)
{
    if (i != NULL) { // ถ้า i ไม่เป็น NULL
        printf(" %d",i->info); // แสดงข้อมูล INFO
        PreOrder(i->lson); // เรียกใช้ PreOrder กับลูกซ้าย
        PreOrder(i->rson); // เรียกใช้ PreOrder กับลูกขวา
    }
}
```

ฟังก์ชันนี้ทำการเดินทางในรูปแบบ Pre-order (เยี่ยมชมโหนดปัจจุบันก่อนลูกซ้ายและลูกขวา) โดยฟังก์ชันนี้เริ่มจากการแสดงข้อมูลของโหนดปัจจุบัน จากนั้นเรียกใช้ Pre-order กับลูกซ้ายและลูกขวาตามลำดับ

8. การท่องต้นไม้ในลำดับ InOrder ด้วยฟังก์ชัน InOrder

```
void InOrder(struct Node *i)
{
    if (i != NULL) { // ถ้า i ไม่เป็น NULL
        InOrder(i->lson); // เรียกใช้ InOrder กับลูกซ้าย
        printf(" %d",i->info); // แสดงข้อมูล INFO
        InOrder(i->rson); // เรียกใช้ InOrder กับลูกขวา
    }
}
```

ฟังก์ชันนี้ทำการเดินทางในรูปแบบ In-order (เยี่ยมชมลูกซ้ายก่อนโหนดปัจจุบันและลูกขวา) โดยฟังก์ชันนี้เริ่มจากการเรียกใช้ In-order กับลูกซ้าย จากนั้นแสดงข้อมูลของโหนดปัจจุบัน และสุดท้ายเรียกใช้ In-order กับลูกขวา

9. การท่องต้นไม้ในลำดับ PostOrder ด้วยฟังก์ชัน PostOrder

```
void PostOrder(struct Node *i)
{
    if (i != NULL) { // ถ้า i ไม่เป็น NULL
        PostOrder(i->lson); // เรียกใช้ PostOrder กับลูกซ้าย
        PostOrder(i->rson); // เรียกใช้ PostOrder กับลูกขวา
        printf(" %d",i->info); // แสดงข้อมูล INFO
    }
}
```

ฟังก์ชันนี้ทำการเดินทางในรูปแบบ Post-order (เยี่ยมชมลูกซ้ายและลูกขวาก่อนโหนดปัจจุบัน) โดยฟังก์ชันนี้เริ่มจากการเรียกใช้ Post-order กับลูกซ้ายและลูกขวาก่อนที่จะแสดงข้อมูลของโหนดปัจจุบัน

10. ฟังก์ชันหลัก (Main)

```
int main()
{
    N=31;
    CreateTreeNP(N);
    while (ch != 'E') // วนลูปจนกว่าจะกด 'E'
    {
        printf("\nPROGRAM TREE(Node Pointer) \n");
        printf("================================== \n");
        printf("N : %d\n",N);
        printf("Sequence of data : ");
```

10. ฟังก์ชันหลัก (Main) (ต่อ)

```
for (i=1;i<=N;i++) // แสดงลำดับข้อมูล
    printf("%d ",info[i]);
ShowTree(); // แสดงโครงสร้างต้นไม้
printf("\nMENU => P:PreOrder I:InOrder O:PostOrder E:Exit");
printf("\n--------------------------------------------------------------------------\n");
ch=getch();
switch (ch)
{
    case 'P' :  printf("PRE ORDER TRAVERSAL : ");
                PreOrder(T);
                printf("\n--------------------------------------------------------------------------\n");
                break;
    case 'I' :  printf("IN ORDER TRAVERSAL : ");
                InOrder(T);
                printf("\n--------------------------------------------------------------------------\n");
                break;
    case 'O' :  printf("POST ORDER TRAVERSAL : ");
                PostOrder(T);
                printf("\n--------------------------------------------------------------------------\n");
                break;
} // สิ้นสุด Switch...case
```

10. ฟังก์ชันหลัก (Main) (ต่อ)

```
    } // สิ้นสุด While

    return(0);

} // สิ้นสุด MAIN
```

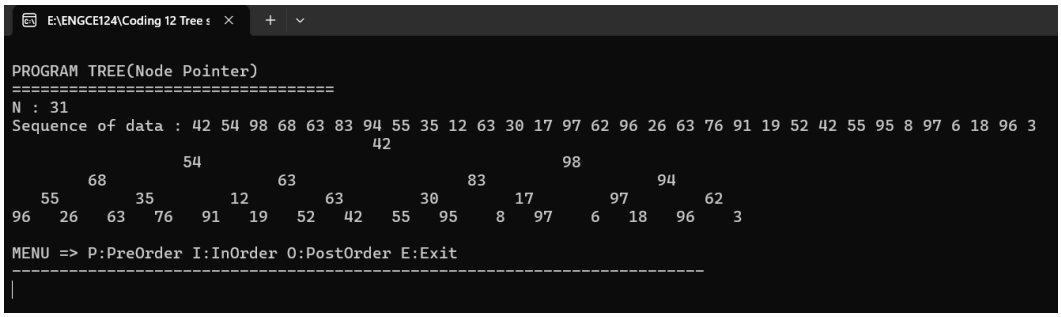
ฟังก์ชันหลักของโปรแกรมที่ทำการเริ่มต้นและจัดการการทำงานของโปรแกรม โดยฟังก์ชันนี้ทำการสร้างต้นไม้ด้วยการเรียก CreateTreeNP และแสดงผลลัพธ์ของต้นไม้ การเดินทางในรูปแบบต่างๆ จะถูกเรียกใช้ตามคำสั่งที่ป้อนเข้ามาจากผู้ใช้ จนกว่าจะกด 'E' เพื่อออกจากโปรแกรม

ผลลัพธ์การใช้งานโปรแกรม TREE STRUCTURE BY NODE POINTER METHOD

โปรแกรม TREE STRUCTURE BY NODE POINTER METHOD มีหน้าที่ในการสร้างโครงสร้างต้นไม้แบบทวิภาค (binary tree) โดยใช้วิธีการเชื่อมโยงด้วยตัวชี้ (Node Pointer) ผู้ใช้สามารถสร้างต้นไม้ที่มีความลึกได้สูงสุดถึง 5 ระดับ ซึ่งสามารถรองรับจำนวนโหนดได้มากถึง 31 โหนด จากนั้นโปรแกรมจะทำการเดินทางผ่านโหนดในต้นไม้และแสดงผลตามลำดับที่กำหนด ได้แก่ Pre-order, In-order, และ Post-order

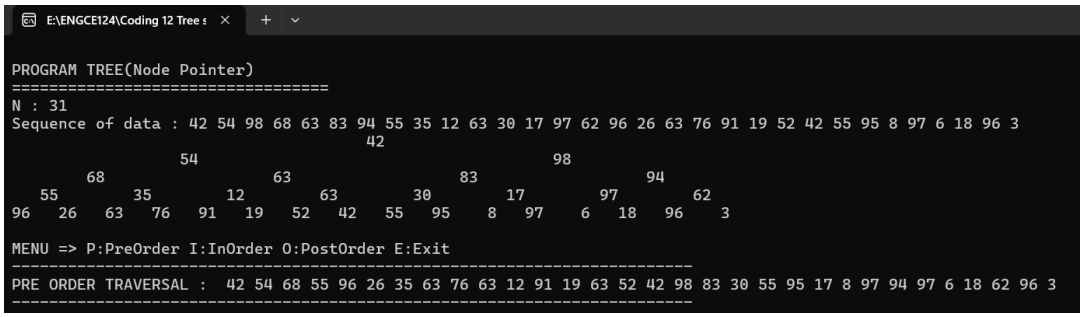
1. การสร้างต้นไม้และการแสดงโครงสร้างต้นไม้

เมื่อต้นไม้ถูกสร้างขึ้น โปรแกรมจะสุ่มข้อมูลสำหรับแต่ละโหนด จากนั้นจะจัดเรียงโหนดเหล่านั้นในรูปแบบต้นไม้ทวิภาคแบบสมบูรณ์ (Complete Binary Tree) โดยโหนดแรกจะเป็นราก (root) และโหนดต่อมาจะเป็นลูกซ้ายและลูกขวาของโหนดพ่อในระดับที่สูงกว่า โดยเมื่อหลังจากสร้างต้นไม้แล้ว โปรแกรมจะแสดงโครงสร้างของต้นไม้บนหน้าจอ เพื่อให้ผู้ใช้เห็นการจัดเรียงโหนดในแต่ละระดับ



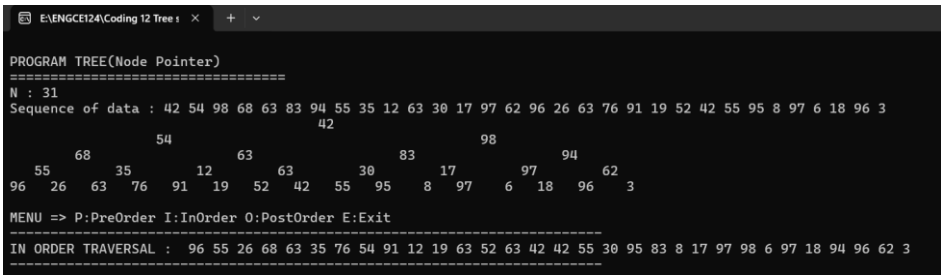
ผลลัพธ์นี้แสดงจำนวนโหนดทั้งหมดในต้นไม้ (31 โหนด) และข้อมูลที่ถูกสุ่มสำหรับแต่ละโหนดตามลำดับ

2. การท่องต้นไม้ตามลำดับ Pre Order



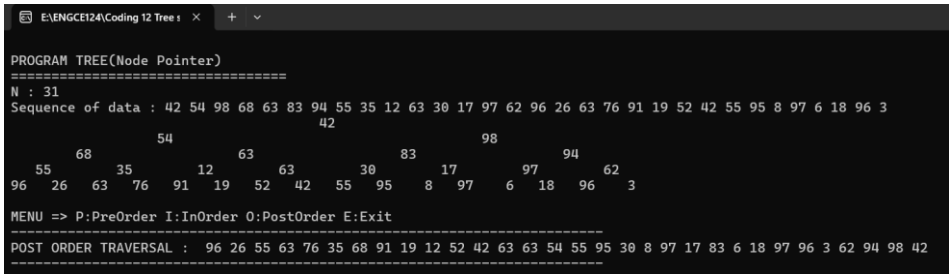
เมื่อผู้ใช้เลือก P (Pre-order Traversal) โปรแกรมจะแสดงผลลัพธ์การเดินทางผ่านโหนดต้นไม้ในลำดับ โดยเริ่มจากแสดงข้อมูลของโหนดปัจจุบัน (เริ่มจากราก) จากนั้นเดินทางไปยังลูกซ้าย และเดินทางไปยังลูกขวา ซึ่งผลลัพธ์ของ Pre order จะการท่องต้นไม้เริ่มต้นจากโหนดราก (42) และทำการเรียกซ้ำสำหรับโหนดลูกทางซ้ายและขวาตามลำดับ

3. การท่องต้นไม้ตามลำดับ In Order



เมื่อผู้ใช้เลือก I (In-order Traversal) โปรแกรมจะแสดงผลลัพธ์การเดินทางผ่านโหนดต้นไม้ในลำดับโดยเริ่มจากเดินทางไปยังลูกซ้ายสุดก่อน จากนั้นแสดงข้อมูลของโหนดปัจจุบัน และเดินทางไปยังลูกขวา ซึ่งผลลัพธ์นี้แสดงลำดับการเดินทางผ่านโหนดในรูปแบบ In-order คือ เริ่มจากลูกซ้ายสุดของต้นไม้ (96) จากนั้นเดินทางขึ้นมาแสดงข้อมูลโหนดพ่อ (55) และเดินทางไปยังลูกขวาของ 55 (26) แล้วเดินทางขึ้นไปที่โหนดราก (68) และทำซ้ำลักษณะเดียวกันในฝั่งขวาจนครบทุกโหนด

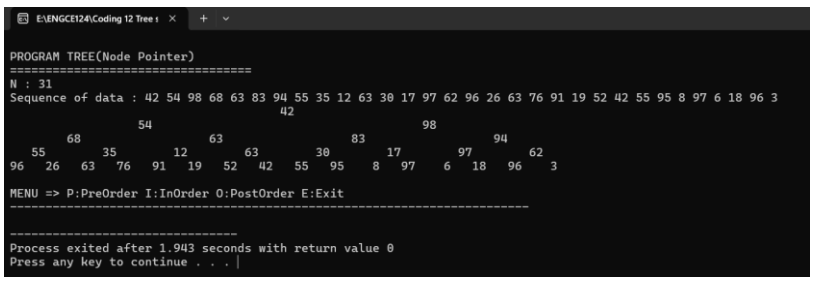
4. การท่องต้นไม้ตามลำดับ Post Order



เมื่อผู้ใช้เลือก O (Post-order Traversal) โปรแกรมจะแสดงผลลัพธ์การเดินทางผ่านโหนดต้นไม้ในลำดับ โดยเริ่มจากเดินทางไปยังลูกซ้าย จากนั้นเดินทางไปยังลูกขวา และแสดงข้อมูลของโหนดปัจจุบัน ซึ่งผลลัพธ์นี้แสดงลำดับการเดินทางผ่านโหนดในรูปแบบ Post-order คือ เริ่มจากการเดินทางไปยังลูกซ้ายสุดของต้นไม้ก่อน (96) จากนั้นไปยังลูกขวาของ 96 (26) แล้วกลับมาแสดงข้อมูลโหนดพ่อ (55) ทำซ้ำลักษณะนี้ในทุกระดับจนไปถึงโหนดรากสุดท้าย

5. การออกจากโปรแกรม (Exit)

เมื่อผู้ใช้กด E โปรแกรมจะสิ้นสุดการทำงานและปิดตัวลง



บรรณานุกรม

ChatGPT. (-). Exploring Traversal Methods with Node Pointers. สืบค้น 28 สิงหาคม 2567, จาก https://chatgpt.com/