



รายงาน

เรื่อง โปรแกรม BINARY SEARCH TREE (BST)

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 BINARY SEARCH TREE (BST)

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 BINARY SEARCH TREE (BST) รวมถึงอธิบายหลักการทำงานของโปรแกรม BINARY SEARCH TREE (BST) และอธิบายผลลัพธ์การใช้งานโปรแกรม BINARY SEARCH TREE (BST)

ผู้จัดทำรายงานหวังว่า รายงานฉบับนี้จะเป็นประโยชน์กับผู้สนใจ หรือนักศึกษาทุกท่านที่กำลังหาศึกษาในหัวข้อของโปรแกรม BINARY SEARCH TREE (BST) หากมีข้อเสนอแนะหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้ และขออภัยมา ณ ที่นี้

ผู้จัดทำ

นายพงษ์พันธุ์ เลาวพงศ์

วันที่ 04/10/2567

สารบัญ

	หน้า
คำนำ	ก
สารบัญ	๗
โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย	1
หลักการทำงานของโปรแกรม BINARY SEARCH TREE (BST)	9
ผลลัพธ์การใช้งานโปรแกรม BINARY SEARCH TREE (BST)	20
บรรณานุกรม	23

โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย

```
#include <stdio.h> //ใช้ printf

#include <conio.h> //ใช้ getch

#include <stdlib.h> //ใช้ rand (สุ่ม)

#include <time.h> //ใช้เวลา

#include <windows.h> //ใช้เสียง (Sound)

#define MaxData 100 // กำหนดจำนวนข้อมูลสูงสุด

int Data[MaxData];

int N,key,Times;

bool result;

struct Node //ประกาศโครงสร้างของโหนดของต้นไม้ (Tree)

{

    int info;

    struct Node *lson,*rson;

};

struct Node *Start[MaxData],*Root,*p; // ประกาศตัวชี้โหนด (Pointer node)

Node *Allocate() //จัดสรรโหนด 1 โหนดจากที่เก็บข้อมูล

{

    struct Node *temp;

    temp=(Node*)malloc(sizeof(Node)); //จัดสรรโหนดตามขนาดที่ประกาศ

    return(temp);

}
```

โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย (ต่อ)

```

bool Duplicate(int i,int Data1) //ตรวจสอบว่าข้อมูลซ้ำหรือไม่
{
    int j;

    for(j=1;j<=i;j++)

        if(Data1==Data[j])

            return(true); //ซ้ำกัน

        return(false); //ไม่ซ้ำกัน
}

void PrepareRawKey(int N)
{
    int i,temp;

    srand(time(NULL)); //เพื่อสุ่มหมายเลขที่ต่างกันใน rand()

    for (i=0;i<N;i++)

    {

        temp=(rand() % 89)+10; //สุ่มหมายเลขต่างกันในช่วง 10 ถึง 99

        while(Duplicate(i-1,temp)) //วนซ้ำหากยังซ้ำกัน

            temp=(rand() % 89)+10; //สุ่มใหม่

        Data[i]=temp; //เก็บหมายเลขใหม่

    } //สิ้นสุด for

} //สิ้นสุดฟังก์ชัน

```

โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย (ต่อ)

```

void DispKey(int N)
{
    int i;

    for(i=0;i<N;i++)

        printf("(%2d)",i); //แสดงลำดับดัชนี i

    printf("\n");

    for(i=0;i<N;i++)

        printf(" %2d ",Data[i]); //แสดงข้อมูลใน Data[]

    printf("\n");
}

void CreateBST(int N)
{
    int i;

    bool Finish;

    struct Node *T1,*p;

    p=Allocate(); //ตั้งโหนดราก (Root Node)

    p->info=Data[0];

    p->lson=NULL;

    p->rson=NULL;

    Root=p; //ตั้งตัวชี้โหนดราก

    for (i=1;i<N;i++) //นับจำนวนตามข้อมูลที่มี

```

โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย (ต่อ)

```

T1=Root; //ให้ T1 ชี้ไปที่โหนดราก

p=Allocate();

p->info=Data[i];

p->lson=NULL;

p->rson=NULL;

Finish=false;

while(!Finish)

{

    if(Data[i]<T1->info)

        if(T1->lson==NULL)

        {

            //เพิ่มโหนดทางซ้าย

            T1->lson=p; //ให้โหนดทางซ้ายชี้ไปยังโหนดใหม่

            Finish=true; //เสร็จสิ้น

        }

    else

        T1=T1->lson; //ข้ามไปยัง SON ด้านซ้าย

    else

        if(T1->rson==NULL)

        {

            //เพิ่มโหนดทางขวา

```


โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย (ต่อ)

```

        T1->rson=p; //ให้โหนดทางขวาชี้ไปยังโหนดใหม่

        Finish=true; //เสร็จสิ้น

    }

    else

        T1=T1->rson; //ข้ามไปยัง SON ด้านขวา

    } //สิ้นสุด while

    } //สิ้นสุด for

} //สิ้นสุดฟังก์ชัน

void InOrder(struct Node *i)

{

    if (i != NULL) //ถ้า i ไม่ใช่ NULL

    {

        InOrder(i->lson); //เรียก SON ด้านซ้ายด้วย InOrder

        printf(" %2d",i->info); //แสดงข้อมูล INFO

        InOrder(i->rson); //เรียก SON ด้านขวาด้วย InOrder

    }

}

bool SearchBST(int key)

{

    struct Node *T1;

    Times=0;

```

โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย (ต่อ)

```

T1=Root;

while(T1!=NULL)

{

Times++; //นับจำนวนครั้งที่ค้นหา

if(key==T1->info)

    return(true); //พบข้อมูล

else

    if(key<T1->info)

        T1=T1->lson; //ข้ามไปยังโหนดทางซ้าย

    else

        T1=T1->rson; //ข้ามไปยังโหนดทางขวา

} //สิ้นสุด While

return(false); //ไม่พบข้อมูล

} //สิ้นสุดฟังก์ชัน

int main()

{

    printf("BINARY SEARCH TREE\n");

    printf("=====\n");

    N=16;

    PrepareRawKey(N);

    CreateBST(N);

```

โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย (ต่อ)

```

while(key!=-999)

{

printf("Raw key :\n");

DispKey(N); //แสดงคีย์ดิบ

printf("-----\n");

printf("In Order :\n");

InOrder(Root);

printf("\n-----\n");

printf("\nEnter Key for Search(-999 for EXIT) : ");

scanf("%d",&key); //อ่านคีย์จากแป้นพิมพ์

if(key!=-999)

{

    result=SearchBST(key);

    printf("Searching Time : %d\n",Times);

    printf("Result...");

    if(result)

        printf("FOUND\n"); //ถ้าพบข้อมูล

    else

    {

        Beep(600,600);

        printf("NOT FOUND!!\n"); //ถ้าไม่พบข้อมูล
    }
}
}

```

โค้ดของโปรแกรม BINARY SEARCH TREE (BST) พร้อมคำอธิบาย (ต่อ)

```
    }

    printf("-----Searching Finished\n");

    } //สิ้นสุด if

    } //สิ้นสุด While

    return(0);

} //สิ้นสุด Main
```

หลักการทํางานของโปรแกรม BINARY SEARCH TREE (BST)

โปรแกรมนี้มีหน้าที่สร้างและจัดการ Binary Search Tree (BST) จากข้อมูลที่สุ่มได้ รวมถึงการค้นหาและแสดงผลข้อมูลในรูปแบบต่าง ๆ ในโปรแกรมนี้อาจมีหลายฟังก์ชันที่ทำงานร่วมกันอย่างเป็นระบบเพื่อจัดการข้อมูลอย่างมีประสิทธิภาพ

1. การนำเข้าไลบรารี

```
#include <stdio.h> //ใช้ printf
#include <conio.h> //ใช้ getch
#include <stdlib.h> //ใช้ rand (สุ่ม)
#include <time.h> //ใช้เวลา
#include <windows.h> //ใช้เสียง (Sound)
```

ในส่วนของการนำเข้าไลบรารี มีรายละเอียดดังนี้

- stdio.h: ไลบรารีมาตรฐานที่ใช้สำหรับการทำงานเกี่ยวกับการป้อนและแสดงผล โดยเฉพาะการใช้ printf เพื่อแสดงข้อมูล
- conio.h: ไลบรารีสำหรับการจัดการอินพุตและเอาต์พุตบนคอนโซล โดยใช้ฟังก์ชัน getch() เพื่อรอการกดแป้นพิมพ์จากผู้ใช้
- stdlib.h: ไลบรารีสำหรับฟังก์ชันทั่วไป เช่น การจัดการหน่วยความจำด้วย malloc() และการสุ่มตัวเลขด้วย rand()
- time.h: ไลบรารีที่ใช้ในการจัดการเวลา โดยใช้ในการกำหนดค่า seed สำหรับการสุ่มด้วย srand(time(NULL)) เพื่อให้ผลลัพธ์การสุ่มแตกต่างกันในแต่ละครั้ง
- windows.h: ไลบรารีเฉพาะสำหรับระบบ Windows ที่ใช้สำหรับฟังก์ชันการควบคุมฮาร์ดแวร์ เช่น การสร้างเสียงเตือนด้วยฟังก์ชัน Beep()

2. การประกาศค่าคงที่

```
#define MaxData 100 // กำหนดจำนวนข้อมูลสูงสุด
```

2. การประกาศค่าคงที่ (ต่อ)

ในส่วนของการประกาศค่าคงที่ มีรายละเอียดดังนี้

- MaxData: ค่าคงที่นี้กำหนดให้จำนวนข้อมูลสูงสุดที่โปรแกรมสามารถจัดการได้คือ 100 ค่า ตัวแปรนี้ถูกใช้เพื่อกำหนดขนาดของอาร์เรย์ Data[] และโครงสร้างต้นไม้ Start[]

3. การประกาศตัวแปร

```
int Data[MaxData];

int N,key,Times;

bool result;
```

ในส่วนของการประกาศตัวแปร มีรายละเอียดดังนี้

- Data[MaxData]: อาร์เรย์ที่เก็บข้อมูลตัวเลขที่ถูกสุ่มขึ้นมาและใช้ในการสร้างต้นไม้
- N: จำนวนข้อมูลที่ต้องการสร้างในแต่ละครั้ง
- key: ค่าในการค้นหาข้อมูลในต้นไม้ (BST)
- Times: จำนวนครั้งในการค้นหาข้อมูลใน Binary Search Tree
- result: ตัวแปร boolean ที่ใช้เก็บผลลัพธ์จากการค้นหาในต้นไม้ ซึ่งจะเป็น true หากพบข้อมูล และ false หากไม่พบข้อมูล

4. การประกาศโครงสร้างของโหนด (Node Structure)

```
struct Node //ประกาศโครงสร้างของโหนดของต้นไม้ (Tree)

{

    int info;

    struct Node *lson,*rson;

};
```

ในส่วนของการประกาศโครงสร้างของโหนด มีรายละเอียดดังนี้

4. การประกาศโครงสร้างของโหนด (Node Structure) (ต่อ)

- info: เป็นตัวแปรที่เก็บข้อมูลของแต่ละโหนดในต้นไม้
- lson: ตัวชี้ไปยังโหนดลูกด้านซ้าย ซึ่งจะเก็บข้อมูลที่มีค่าน้อยกว่าโหนดปัจจุบันตามกฎหมายของ Binary Search Tree
- rson: ตัวชี้ไปยังโหนดลูกด้านขวา ซึ่งจะเก็บข้อมูลที่มีค่ามากกว่าโหนดปัจจุบันตามกฎหมายของ Binary Search Tree

5. การประกาศตัวชี้โหนด (Pointer Node)

```
struct Node *Start[MaxData],*Root,*p; // ประกาศตัวชี้โหนด (Pointer node)
```

ในส่วนของการประกาศตัวชี้โหนด มีรายละเอียดดังนี้

- Start[MaxData]: อาร์เรย์ของตัวชี้ที่ใช้สำหรับเก็บโหนดต้นไม้เริ่มต้น
- Root: ตัวชี้ไปยังโหนดรากของต้นไม้ (ต้นไม้จะเริ่มจากโหนดนี้)
- p: ตัวชี้ที่ใช้สำหรับสร้างโหนดใหม่

6. ฟังก์ชัน Allocate

```
Node *Allocate() //จัดสรรโหนด 1 โหนดจากที่เก็บข้อมูล
{
    struct Node *temp;

    temp=(Node*)malloc(sizeof(Node)); //จัดสรรโหนดตามขนาดที่ประกาศ

    return(temp);
}
```

ฟังก์ชัน Allocate มีหน้าที่ในการจัดสรรหน่วยความจำสำหรับโหนด (Node) ใหม่ที่จะถูกสร้างขึ้นใน BST ทุกครั้งที่ต้องการเพิ่มโหนดในต้นไม้ ฟังก์ชันจะจัดเตรียมหน่วยความจำที่เพียงพอเพื่อรองรับโครงสร้างข้อมูลของโหนด โดยหลักการทำงานจะเริ่มจาก ใช้คำสั่ง malloc() เพื่อจองหน่วยความจำขนาดที่เท่ากับโครงสร้างของโหนดที่กำหนด (Node) หลังจากจองหน่วยความจำเรียบร้อยแล้ว จะคืนค่าตัวชี้ (pointer) ที่ชี้ไปยังหน่วยความจำนั้น เพื่อนำไปใช้ในการสร้างโหนดใหม่ในต้นไม้

7. ฟังก์ชัน Duplicate

```

bool Duplicate(int i,int Data1) //ตรวจสอบว่าข้อมูลซ้ำหรือไม่
{
    int j;

    for(j=1;j<=i;j++)

        if(Data1==Data[j])

            return(true); //ซ้ำกัน

    return(false); //ไม่ซ้ำกัน
}

```

ฟังก์ชัน Duplicate ทำหน้าที่ตรวจสอบว่าค่าที่จะถูกสุ่มเพิ่มลงในอาร์เรย์ Data[] ซ้ำกับค่าที่มีอยู่แล้วในอาร์เรย์หรือไม่ เพื่อให้มั่นใจว่าค่าข้อมูลในอาร์เรย์จะไม่ซ้ำกัน โดยหลักการทำงานเริ่มจากฟังก์ชันจะวนลูปตรวจสอบค่าที่ถูกเก็บในอาร์เรย์ Data[] จนถึงตำแหน่งปัจจุบัน หากพบค่าที่ซ้ำกัน จะคืนค่า true ซึ่งบ่งบอกว่าค่าที่สุ่มได้ซ้ำกับค่าที่มีอยู่ แต่ถ้าหากไม่พบค่าซ้ำ จะคืนค่า false เพื่อบอกว่าค่าที่สุ่มได้นั้นไม่ซ้ำ

8. ฟังก์ชัน PrepareRawKey

```

void PrepareRawKey(int N)
{
    int i,temp;

    srand(time(NULL)); //เพื่อสุ่มหมายเลขที่ต่างกันใน rand()

    for (i=0;i<N;i++)
    {
        temp=(rand() % 89)+10; //สุ่มหมายเลขต่างกันในช่วง 10 ถึง 99

        while(Duplicate(i-1,temp)) //วนซ้ำหากยังซ้ำกัน
    }
}

```


8. ฟังก์ชัน PrepareRawKey (ต่อ)

```

        temp=(rand() % 89)+10; //สุ่มใหม่

        Data[i]=temp; //เก็บหมายเลขใหม่

    } //สิ้นสุด for
} //สิ้นสุดฟังก์ชัน

```

ฟังก์ชัน PrepareRawKey มีหน้าที่ในการสุ่มข้อมูลจำนวน N ค่า เพื่อนำไปใช้ในการสร้างต้นไม้ โดยค่าที่สุ่มได้จะต้องไม่ซ้ำกัน และมีค่าระหว่าง 10 ถึง 99 ซึ่งจะถูกเก็บไว้ในอาร์เรย์ Data[] โดยหลักการทำงานเริ่มจาก ใช้ฟังก์ชัน rand() เพื่อสุ่มค่าตัวเลขในช่วง 10 ถึง 99 จากนั้นใช้ srand(time(NULL)) เพื่อให้การสุ่มแต่ละครั้งให้ผลลัพธ์ที่แตกต่างกันในแต่ละครั้งที่โปรแกรมถูกเรียกใช้ โดยฟังก์ชันจะเรียก Duplicate เพื่อตรวจสอบว่าค่าที่สุ่มได้ซ้ำกับค่าที่มีอยู่ในอาร์เรย์หรือไม่ ถ้าซ้ำจะทำการสุ่มใหม่จนกว่าจะได้ค่าที่ไม่ซ้ำ และเมื่อได้ค่าที่ไม่ซ้ำแล้ว จะเก็บค่าในอาร์เรย์ Data[]

9. ฟังก์ชัน DispKey

```

void DispKey(int N)
{
    int i;

    for(i=0;i<N;i++)

        printf("(%2d)",i); //แสดงลำดับดัชนี i

    printf("\n");

    for(i=0;i<N;i++)

        printf(" %2d ",Data[i]); //แสดงข้อมูลใน Data[]

    printf("\n");
}

```

9. ฟังก์ชัน DispKey (ต่อ)

ฟังก์ชัน DispKey มีหน้าที่ในการแสดงผลข้อมูลในอาร์เรย์ Data[] บนหน้าจอในรูปแบบที่เข้าใจง่าย โดยจะแสดงทั้งดัชนี (index) และค่าที่เก็บในแต่ละตำแหน่งของอาร์เรย์ โดยหลักการทำงานเริ่มจาก ฟังก์ชันจะวนลูปผ่านข้อมูลในอาร์เรย์ และแสดงดัชนีของข้อมูลในบรรทัดแรก ในส่วนบรรทัดที่สองจะแสดงค่าข้อมูลในแต่ละตำแหน่งของอาร์เรย์ตามลำดับ และค่าจะถูกจัดให้อยู่ในรูปแบบที่อ่านได้ง่าย

10. ฟังก์ชัน CreateBST

```
void CreateBST(int N)
{
    int i;

    bool Finish;

    struct Node *T1,*p;

    p=Allocate(); //ตั้งโหนดราก (Root Node)

    p->info=Data[0];

    p->lson=NULL;

    p->rson=NULL;

    Root=p; //ตั้งตัวชี้โหนดราก

    for (i=1;i<N;i++) //นับจำนวนตามข้อมูลที่มี

    T1=Root; //ให้ T1 ชี้ไปที่โหนดราก

    p=Allocate();

    p->info=Data[i];

    p->lson=NULL;

    p->rson=NULL;

    Finish=false;
```

10. ฟังก์ชัน CreateBST (ต่อ)

```

while(!Finish)

{

    if(Data[i]<T1->info)

        if(T1->lson==NULL)

        {

            //เพิ่มโหนดทางซ้าย

            T1->lson=p; //ให้โหนดทางซ้ายชี้ไปยังโหนดใหม่

            Finish=true; //เสร็จสิ้น

        }

        else

            T1=T1->lson; //ข้ามไปยัง SON ด้านซ้าย

        else

            if(T1->rson==NULL)

            {

                //เพิ่มโหนดทางขวา

                T1->rson=p; //ให้โหนดทางขวาชี้ไปยังโหนดใหม่

                Finish=true; //เสร็จสิ้น

            }

            else

                T1=T1->rson; //ข้ามไปยัง SON ด้านขวา

        } //สิ้นสุด while

```

10. ฟังก์ชัน CreateBST (ต่อ)

```

        } //สิ้นสุด for
    } //สิ้นสุดฟังก์ชัน

```

ฟังก์ชัน CreateBST มีหน้าที่ในการสร้างต้นไม้ Binary Search Tree (BST) จากข้อมูลในอาร์เรย์ Data[] โดยเรียงข้อมูลตามกฎของ BST ซึ่งข้อมูลที่น้อยกว่าจะอยู่ด้านซ้ายของโหนด และข้อมูลที่ยิ่งกว่าจะอยู่ด้านขวาของโหนด โดยหลักการทำงาน เริ่มต้นด้วยการสร้างโหนดราก (root) โดยใช้ข้อมูลตัวแรกในอาร์เรย์ จากนั้นจะวนลูปเพิ่มโหนดใหม่ลงในต้นไม้สำหรับข้อมูลตัวที่เหลือ โดยสำหรับโหนดใหม่ จะตรวจสอบว่าค่าของโหนดนั้น น้อยกว่าหรือมากกว่าโหนดปัจจุบัน ถ้าน้อยกว่าจะไปทางซ้าย ถ้ามากกว่าจะไปทางขวา ซึ่งสุดท้ายจะทำการวนลูปเช่นนี้จนกว่าโหนดใหม่จะถูกเพิ่มในตำแหน่งที่เหมาะสมในต้นไม้

11. ฟังก์ชัน InOrder

```

void InOrder(struct Node *i)
{
    if (i != NULL) //ถ้า i ไม่ใช่ NULL
    {
        InOrder(i->lson); //เรียก SON ด้านซ้ายด้วย InOrder
        printf(" %2d",i->info); //แสดงข้อมูล INFO
        InOrder(i->rson); //เรียก SON ด้านขวาด้วย InOrder
    }
}

```

ฟังก์ชัน InOrder ทำหน้าที่เรียกผ่านโหนดของต้นไม้แบบ In-Order Traversal และแสดงค่าข้อมูลของโหนดในลำดับจากน้อยไปมาก ซึ่งเป็นคุณสมบัติของการเรียกผ่านแบบ In-Order ใน Binary Search Tree โดยหลักการทำงานเริ่มจากฟังก์ชันจะตรวจสอบว่าโหนดปัจจุบันเป็น NULL หรือไม่ หากไม่ใช่จะดำเนินการเรียกผ่านต่อ ซึ่งขั้นตอนของ In-Order Traversal คือ เรียกผ่านโหนดลูกทางซ้าย จากนั้นจะแสดงค่าข้อมูลของโหนดปัจจุบัน และเรียกผ่านโหนดลูกทางขวา

12. ฟังก์ชัน SearchBST

```

bool SearchBST(int key)
{
    struct Node *T1;

    Times=0;

    T1=Root;

    while(T1!=NULL)
    {
        Times++; //นับจำนวนครั้งที่ค้นหา

        if(key==T1->info)

            return(true); //พบข้อมูล

        else

            if(key<T1->info)

                T1=T1->lson; //ข้ามไปยังโหนดทางซ้าย

            else

                T1=T1->rson; //ข้ามไปยังโหนดทางขวา

    } //สิ้นสุด While

    return(false); //ไม่พบข้อมูล

} //สิ้นสุดฟังก์ชัน

```

ฟังก์ชัน SearchBST ทำหน้าที่ในการค้นหาข้อมูลใน Binary Search Tree โดยรับค่า key ที่ต้องการค้นหา และคืนค่า true หากพบข้อมูล หรือคืนค่า false หากไม่พบ โดยหลักการทำงานเริ่มจาก ฟังก์ชันจะเริ่มต้นค้นหาจากโหนดราก (Root) ถ้าค่าของ key ตรงกับข้อมูลในโหนดปัจจุบัน จะคืนค่า true แต่ถ้าค่าของ key

น้อยกว่าข้อมูลในโหนดปัจจุบัน จะไปทางซ้าย แต่ถ้าค่าของ key มากกว่า จะไปทางขวา จากนั้นจะทำการวน
 ลูปเช่นนี้จนกว่าจะพบค่าที่ต้องการหรือไม่พบ (เมื่อโหนดถัดไปเป็น NULL)

13. ฟังก์ชัน main

```
int main()
{
    printf("BINARY SEARCH TREE\n");

    printf("=====\n");

    N=16;

    PrepareRawKey(N);

    CreateBST(N);

    while(key!=-999)
    {
        printf("Raw key :\n");

        DispKey(N); //แสดงคีย์ดิบ

        printf("-----\n");

        printf("In Order :\n");

        InOrder(Root);

        printf("\n-----\n");

        printf("\nEnter Key for Search(-999 for EXIT) : ");

        scanf("%d",&key); //อ่านคีย์จากแป้นพิมพ์

        if(key!=-999)
        {
            result=SearchBST(key);
```

13. ฟังก์ชัน main (ต่อ)

```

        printf("Searching Time : %d\n",Times);

        printf("Result...");

        if(result)

            printf("FOUND\n"); //ถ้าพบข้อมูล

        else

        {

            Beep(600,600);

            printf("NOT FOUND!!\n"); //ถ้าไม่พบข้อมูล

        }

        printf("-----Searching Finished\n");

    } //สิ้นสุด if

} //สิ้นสุด While

return(0);

} //สิ้นสุด Main

```

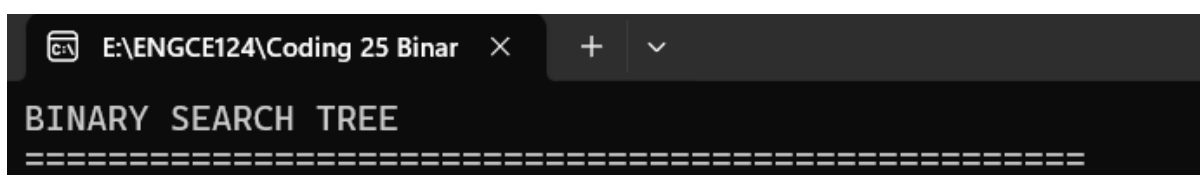
ฟังก์ชัน main เป็นฟังก์ชันหลักของโปรแกรม มีหน้าที่ควบคุมการทำงานทั้งหมด ตั้งแต่การสร้าง BST ไปจนถึงการแสดงผลและการค้นหาข้อมูล โดยหลักการทำงานเริ่มต้นด้วยการเตรียมข้อมูลสุมด้วยฟังก์ชัน PrepareRawKey และสร้าง Binary Search Tree ด้วยฟังก์ชัน CreateBST จากนั้นวนลูปรับค่าจากผู้ใช้เพื่อทำการค้นหาใน BST โดยเรียกใช้ฟังก์ชัน SearchBST หากพบข้อมูลจะแสดงผล "FOUND" และหากไม่พบจะแสดงผล "NOT FOUND" พร้อมเสียงเตือน

ผลลัพธ์การใช้งานโปรแกรม BINARY SEARCH TREE (BST)

โปรแกรมนี้ถูกออกแบบมาเพื่อสร้าง Binary Search Tree (BST) จากข้อมูลที่สุ่มขึ้นมา และให้ผู้ใช้ทำการค้นหาค่าผ่านทางคีย์บอร์ด จากนั้นโปรแกรมจะแสดงผลว่าพบค่าที่ค้นหาหรือไม่ และนับจำนวนครั้งที่ใช้ในการค้นหา

1. การเริ่มต้นของโปรแกรม

เมื่อรันโปรแกรม ผู้ใช้จะเห็นข้อความหัวข้อของโปรแกรมที่มีชื่อว่า “BINARY SEARCH TREE” แสดงออกมาบนหน้าจอคอนโซล



```
E:\ENGCE124\Coding 25 Binar  X  +  v
BINARY SEARCH TREE
=====
```

2. การสุ่มข้อมูล

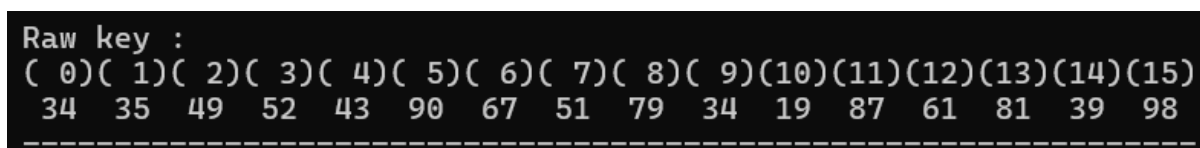
โปรแกรมจะทำการสุ่มข้อมูลจำนวน 16 ค่า โดยข้อมูลแต่ละค่าจะอยู่ในช่วงตั้งแต่ 10 ถึง 99 และรับประกันว่าค่าจะไม่ซ้ำกันในแต่ละตำแหน่งของอาร์เรย์ Data[] โดยใช้ฟังก์ชัน rand() ซึ่งการสุ่มจะเกิดขึ้นทุกครั้งที่เริ่มโปรแกรมใหม่

3. การสร้างต้นไม้ (BST)

หลังจากสุ่มข้อมูลแล้ว โปรแกรมจะสร้าง Binary Search Tree (BST) โดยเริ่มต้นจากโหนดราก (Root) และทำการเพิ่มโหนดใหม่ลงในต้นไม้ตามกฎของ BST โดยค่าที่น้อยกว่าจะถูกเพิ่มทางด้านซ้ายของโหนด และค่าที่มากกว่าจะถูกเพิ่มทางด้านขวาของโหนด การสร้างต้นไม้จะใช้ข้อมูลจากอาร์เรย์ Data[] ซึ่งข้อมูลที่อยู่ในตำแหน่งแรก (Data[0]) จะเป็นโหนดราก และโปรแกรมจะเรียงค่าต่าง ๆ ลงไปในต้นไม้อย่างต่อเนื่อง

4. การแสดงข้อมูลและต้นไม้ที่สร้างขึ้น

โปรแกรมจะแสดงข้อมูลดิบที่ถูกสุ่มขึ้นมาผ่านอาร์เรย์ Data[] โดยจะแสดงตำแหน่งในอาร์เรย์ Data[] และแสดงค่าของแต่ละตำแหน่งในอาร์เรย์



```
Raw key :
( 0)( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)(11)(12)(13)(14)(15)
 34 35 49 52 43 90 67 51 79 34 19 87 61 81 39 98
=====
```


5. การแสดงข้อมูลตามลำดับของต้นไม้ (In-Order Traversal)

โปรแกรมจะเรียงลำดับและแสดงข้อมูลในต้นไม้ด้วยวิธี In-Order Traversal ซึ่งเป็นการแสดงผลจากน้อยไปมาก โดยเริ่มต้นจากโหนดลูกด้านซ้ายสุด ขึ้นไปยังโหนดราก และไปยังโหนดลูกด้านขวาสุด

```
In Order :
19 34 34 35 39 43 49 51 52 61 67 79 81 87 90 98
-----
```

6. การรับข้อมูลจากผู้ใช้เพื่อค้นหาในต้นไม้

หลังจากแสดงข้อมูลเรียบร้อยแล้ว โปรแกรมจะเข้าสู่ขั้นตอนการค้นหาโดยผู้ใช้งานสามารถป้อนค่าที่ต้องการค้นหาในต้นไม้ (BST) ผ่านทางคีย์บอร์ด โดยผู้ใช้งานสามารถป้อนค่าใด ๆ เพื่อค้นหาในต้นไม้ และโปรแกรมจะทำการค้นหาค่านั้นในต้นไม้โดยใช้ฟังก์ชัน SearchBST() ซึ่งจะทำการตรวจสอบและนับจำนวนครั้งในการค้นหา (จากโหนดรากไปยังโหนดลูก)

```
Enter Key for Search(-999 for EXIT) : |
```

7. การแสดงผลการค้นหา

ถ้าพบค่าที่ผู้ใช้งานป้อน โปรแกรมจะแสดงผลว่า FOUND พร้อมบอกจำนวนครั้งที่ใช้ในการค้นหา แต่ถ้าไม่พบค่าที่ผู้ใช้งานป้อน โปรแกรมจะแสดงผลว่า NOT FOUND พร้อมส่งเสียงเตือนด้วยฟังก์ชัน Beep() ซึ่งจะสร้างเสียงเตือนเป็นเวลา 600 มิลลิวินาที

```
Enter Key for Search(-999 for EXIT) : 19
Searching Time : 2
Result...FOUND
-----Searching Finished
Raw key :
( 0)( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)(11)(12)(13)(14)(15)
 34 35 49 52 43 90 67 51 79 34 19 87 61 81 39 98
-----
In Order :
19 34 34 35 39 43 49 51 52 61 67 79 81 87 90 98
-----
```

```
Enter Key for Search(-999 for EXIT) : 100
Searching Time : 6
Result...NOT FOUND!!
-----Searching Finished
Raw key :
( 0)( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)(11)(12)(13)(14)(15)
 34 35 49 52 43 90 67 51 79 34 19 87 61 81 39 98
-----
In Order :
19 34 34 35 39 43 49 51 52 61 67 79 81 87 90 98
-----
```

8. การรวมรูปการค้นหา และการสิ้นสุดโปรแกรม

โปรแกรมจะให้ผู้ใช้ค้นหาข้อมูลในต้นไม้ซ้ำได้เรื่อย ๆ โดยที่ผู้ใช้สามารถป้อนค่าตัวเลขอื่น ๆ เพื่อค้นหาได้อย่างต่อเนื่อง จนกระทั่งผู้ใช้ป้อนค่า -999 เพื่อออกจากโปรแกรม โดยถ้าผู้ใช้ป้อน -999 โปรแกรมจะแสดงข้อความและสิ้นสุดการทำงาน

```
E:\ENGCE124\Coding 25 Binar  ×  +  v

BINARY SEARCH TREE
=====
Raw key :
( 0)( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)(11)(12)(13)(14)(15)
 34  35  49  52  43  90  67  51  79  34  19  87  61  81  39  98
-----
In Order :
 19 34 34 35 39 43 49 51 52 61 67 79 81 87 90 98
-----

Enter Key for Search(-999 for EXIT) : 19
Searching Time : 2
Result...FOUND
-----Searching Finished

Raw key :
( 0)( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)(11)(12)(13)(14)(15)
 34  35  49  52  43  90  67  51  79  34  19  87  61  81  39  98
-----
In Order :
 19 34 34 35 39 43 49 51 52 61 67 79 81 87 90 98
-----

Enter Key for Search(-999 for EXIT) : 100
Searching Time : 6
Result...NOT FOUND!!
-----Searching Finished

Raw key :
( 0)( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)(11)(12)(13)(14)(15)
 34  35  49  52  43  90  67  51  79  34  19  87  61  81  39  98
-----
In Order :
 19 34 34 35 39 43 49 51 52 61 67 79 81 87 90 98
-----

Enter Key for Search(-999 for EXIT) : -999
-----
Process exited after 637.1 seconds with return value 0
Press any key to continue . . . |
```

บรรณานุกรม

ChatGPT. (-). Efficient Data Searching with Binary Search Tree (BST). สืบค้น 4 ตุลาคม 2567,
จาก <https://chatgpt.com/>