ใบงานที่ 24

เรื่อง ต้นไม้เพื่อการค้นหา

เสนอ

อาจารย์ ปิยพล ยืนยงสถาวร

จัดทำโดย

นาย สารินทร์ อินต๊ะรักษา รหัส 65543206082-1

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา โครงสร้างข้อมูลและขั้นตอนวิธี

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 1 ปีการศึกษา 2566

คำสั่ง/คำชี้แจง

- สร้างโค้ดโปรแกรมตามตัวอย่างในเอกสารประกอบการสอน
- แสดงโค้ดโปรแกรมเป็นส่วนๆ พร้อมอธิบาย
- แสดงผลการรันโปรแกรม พร้อมอธิบายการทำงาน
- สรุปผลการทดลอง

ลำดับขั้นการทดลอง

```
int Data[MaxData];
int N,M,key,Addr,Times;
bool result;
struct Node //Declare structure of node
{
int info;
struct Node *link;
};
struct Node *Start[MaxData],*H1,*p; // Declare pointer node
Node *Allocate() //Allocate 1 node from storage pool
{
struct Node *temp;
temp=(Node*)malloc(sizeof(Node)); //Allocate node by size declare
return(temp);
}
bool Duplicate(int i,int Data1) //Check Duplication Data
{
int j;
for(j=1;j<=i;j++)
{
if(Data1==Data[j])
return(true);
}
return(false);
}
```

- ประกาศอาร์เรย์ชื่อ Data ที่ใช้เก็บข้อมูลจำนวนเต็ม โดย MaxData คือขนาดสูงสุดของอาร์เรย์
- ประกาศตัวแปร N, M, key, Addr, Times โดยทั้งหมดเป็นตัวแปรประเภทจำนวนเต็ม (int)
- ประกาศตัวแปร result ที่ใช้เก็บค่าผลลัพธ์ของการทำงาน เป็นตัวแปรประเภท boolean เก็บค่า true หรือ false
- ประกาศโครงสร้าง (struct) ชื่อ Node ซึ่งเป็นโครงสร้างข้อมูลที่มีสองสมาชิก คือ info (จำนวนเต็ม) และ link (ตัวชี้ไปยังโหนดถัดไปใน linked list)
- ประกาศตัวแปรแบบ pointer ที่ใช้ในการจัดการกับโหนดของ linked list โดย Start เป็นอาร์เรย์ของ pointer ที่ใช้เก็บตำแหน่งเริ่มต้นของแต่ละรายการข้อมูลใน linked list
- Node *Allocate เป็นฟังก์ชันที่ใช้ในการจัดสรรหน่วยความจำสำหรับโหนดใน linked list
- bool Duplicate เป็นฟังก์ชันที่ใช้ในการตรวจสอบว่าข้อมูล Data1 ซ้ำกับข้อมูลที่อยู่ในอาร์เรย์ Data หรือไม่ โดยจะค้นหาในอาร์เรย์ Data จาก index 1 ถึง i เพื่อตรวจสอบความซ้ำซ้อนของข้อมูล

```
void PrepareRawKey(int N)
{
int i,temp;
srand(time(NULL)); //for difference random number in rand()
for (i=0;i<N;i++)
{
temp=(rand() % 989)+10; //random difference number 10..999
while(Duplicate(i-1,temp)) //Loop if Still Duplicate
temp=(rand() % 989)+10; //random again
Data[i]=temp; //Keep new Number
} //End for
} //End Fn.
void DispKey(int N)
{
int i;
for(i=0;i<N;i++)
printf(" %2d ",Data[i]); //Show Data[]
printf("\n");
}
void CreateHead(int Head)
{
int i;
struct Node *p;
for (i=1;i<=Head;i++) //Count by Number of Head
{
p=Allocate();
p->info=NULL;
p->link=NULL; //Let NEXT = NULL
Start[i]=p; //Let Start of each node = Address of first Node
} //End for
}
```

- void PrepareRawKey ฟังก์ชันนี้ใช้ในการเตรียมข้อมูลเริ่มต้นสำหรับใช้ใน linked list โดยสร้างข้อมูลสุ่มแบบไม่ซ้ำกัน แล้วนำข้อมูลนี้ไปเก็บในอาร์เรย์ Data
- void DispKey(ฟังก์ชันนี้ใช้ในการแสดงข้อมูลที่อยู่ในอาร์เรย์ Data ที่เราเตรียมไว้ในขั้นตอนที่แล้ว. ฟังก์ชันนี้จะแสดงค่าทั้งหมดในอาร์เรย์ Data โดยใช้ลูป for
- void CreateHead ฟังก์ชันนี้ใช้ในการสร้างโหนดหลัก (Head Node) สำหรับ linked list โดยจะสร้างจำนวน Head Node ตามค่า Head ที่รับเข้ามา และกำหนดให้แต่ละ Head Node มีค่า info เป็น NULL และ link เป็น NULL เพื่อเป็น Head Node ที่ว่างเปล่า และใช้ Start เก็บตำแหน่งเริ่มต้นของแต่ละ Head Node

```
void CreateHashTable(int N)
{
int i;
struct Node *H1, *p;
for(i=0;i<N;i++)
{
Addr=Data[i]%M+Lo; //Calculate Address of Key (Addr=K mod M+l0)
H1=Start[Addr];
if(H1->info==NULL) //if Head Node
H1->info=Data[i];
else
{
while(H1->link!=NULL)
H1=H1->link;
p=Allocate(); //Add new Node
p->info=Data[i];
p->link=NULL;
H1->link=p;
} //End if
} //End for
}
void DispHashTable()
{
int i;
struct Node *H1;
for(i=1;i<=M;i++)
{
H1=Start[i];
printf("\nAddress %2d : ",i);
while(H1!=NULL)
{
printf("%3d ",H1->info);
H1=H1->link; //Skip next Node
}
} //End for
printf("\n");
}
```

- void CreateHashTable ฟังก์ชันนี้ใช้ในการสร้างตารางแฮชโดยรับจำนวนข้อมูล N และทำการจัดเก็บข้อมูลในตารางแฮชตามกฎการแฮชที่กำหนด ขั้นตอนที่สำคัญคือ
  - ในลูป for ที่วนทุกข้อมูลในอาร์เรย์ Data, ฟังก์ชันจะคำนวณ Address (Addr) ของแต่ละข้อมูล
  - จากนั้นเราจะกำหนดตัวชี้ H1 ให้ชี้ไปยังตัวชี้หลัก (Head Node) ที่อยู่ในตารางแฮช
  - ตรวจสอบว่า Head Node นี้มีค่า info เป็น NULL หรือไม่ ถ้ามีค่า info เป็น NULL แสดงว่าตัวชี้หลักนี้ว่างเปล่า จึงจะกำหนดค่า Data[i] เข้าไปใน info ของตัวชี้หลัก
  - ถ้าตัวชี้หลักไม่ว่างเปล่า แสดงว่ามีข้อมูลอยู่แล้ว จึงต้องเพิ่มข้อมูล Data[i] ลงในรายการข้อมูลที่เชื่อมเข้ากับตัวชี้หลัก โดยใช้ลูป while
- void DispHashTable ฟังก์ชันนี้ใช้ในการแสดงข้อมูลที่อยู่ในแต่ละช่องของตารางแฮช โดยแสดงที่อยู่ของแต่ละช่องและข้อมูลที่อยู่ในรายการของแต่ละช่อง โดยใช้ลูป for ในการวนทุกช่องของตารางแฮช และลูป while ในการแสดงข้อมูลที่อยู่ในรายการของแต่ละช่อง

```
bool SearchHash(int key)
{
struct Node *H1;
Addr=key%M+Lo; //Calculate Address of Key (Addr=K mod M+Lo)
H1=Start[Addr];
Times=0;
while(H1!=NULL)
{
Times++; //Add Counter Times
if(H1->info==key)
return(true); //Found
else
H1=H1->link;
}
return(false); //NOT Found
}
```

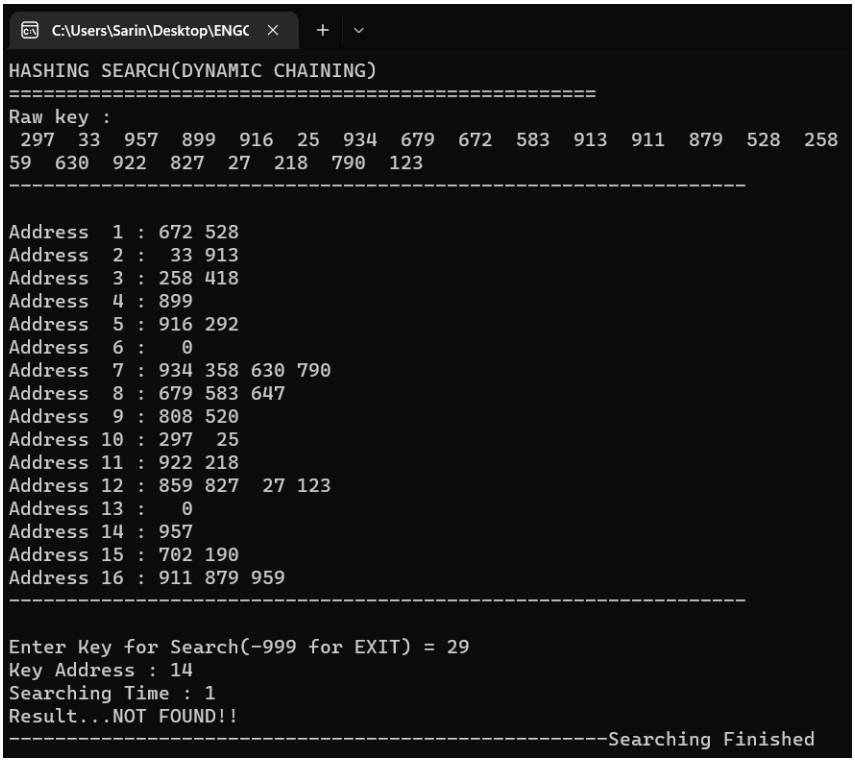
ฟังก์ชัน bool SearchHash(int key) นี้ใช้ในการค้นหาค่า key ในตารางแฮชที่ถูกสร้างขึ้น โดยทำการคำนวณ Address ของ key ตามกฎการแฮชที่กำหนด (Addr = (key % M) + Lo) แล้วใช้ตัวชี้ H1 เพื่อชี้ไปที่ตัวชี้หลักของช่องที่มี Address เท่ากับ Addr

```
int main()
{
printf("HASHING SEARCH(DYNAMIC CHAINING)\n");
printf("================================================\n");
N=32;
M=N*0.50; //Let M=50% of N
PrepareRawKey(N);
printf("Raw key :\n");
DispKey(N); //Raw key
printf("-----------------------------------------------------------------\n");
CreateHead(M); //Create Head Node
CreateHashTable(N);
while(key!=-999)
{
DispHashTable();
printf("-----------------------------------------------------------------\n");
printf("\nEnter Key for Search(-999 for EXIT) = ");
scanf("%d",&key); //Read key from KBD
if(key!=-999)
{
result=SearchHash(key);
printf("Key Address : %d\n",Addr);
printf("Searching Time : %d\n",Times);
printf("Result...");
if(result)
printf("FOUND\n"); //if found
else
{
Beep(600,600);
printf("NOT FOUND!!\n"); //if NOT found
}
printf("-------------------------------------------------Searching Finished\n");
} //End if
} //End While
return(0);
} //End Main
```

- แสดงข้อความ "HASHING SEARCH(DYNAMIC CHAINING)"
- โปรแกรมกำหนดค่า N และ M โดยใช้ M เป็น 50% ของ N (M = N * 0.50) และจากนั้นเริ่มการเตรียมข้อมูล (raw data) และแสดงข้อมูล
- โปรแกรมทำการสร้าง (Head Node) และตารางแฮชที่จัดเก็บข้อมูลโดยใช้ CreateHead และ CreateHashTable
- และทำการแสดงข้อมูลในตารางแฮชด้วยฟังก์ชัน DispHashTable

- จากนั้นโปรแกรมรอรับค่า key จากผู้ใช้และทำการค้นหาค่า key ในตารางแฮชโดยใช้ SearchHash
- โปรแกรมแสดง Address ของ key, เวลาการค้นหา (Times), และผลลัพธ์การค้นหา ("FOUND" หรือ "NOT FOUND") และทำการเปิดเสียง "Beep" ถ้าค้นหาไม่สำเร็จ
- จากนั้นโปรแกรมจะทำการแสดงข้อความ "Searching Finished" และรอรับค่า key ถัดไป
- โปรแกรมจะวนลูปทำงาน จนกว่าผู้ใช้ป้อนค่า -999 เพื่อออกจากโปรแกรม

ผลลัพธ์การทดลอง



สรุปผลการทดลอง

ในการทดลองนี้เรามีการสร้างแฮชทางไดนามิก (Dynamic Chaining) สำหรับการค้นหาข้อมูลที่มีขนาดใหญ่ โดยใช้ตารางแฮช (Hash Table) และการแบ่งกลุ่มข้อมูลที่มีค่าเท่ากันในแต่ละกลุ่มด้วยการสร้างโครงสร้างแบบลิงค์ลิสต์ (Linked List) ในแต่ละช่องของตารางแฮช

สื่อ / เอกสารอ้างอิง

ไฟล์ประกอบการสอนของ อาจารย์ ปิยพล ยืนยงสถาวร เรื่อง : ต้นไม้เพื่อการค้นหา