Q12

E14086020 洪緯宸

講解min heap 實作，題目要求初始化 插入 刪除 修改的功能

heap 為complete binary tree，因此會省很多空間，也有較佳的時間複雜度

min heap 的規則是parent不能大於children

Define node

data是資料儲存處

capacity 是heap的最大容量

size 是heap目前的element數量

```
typedef struct Heap
{
    int *data;    // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int size;     // Current number of elements in min heap
} Heap;
```

建立parent與child的連結函式

parent 為child index的一半 取floor

left child 為parent的兩倍

right child 為parent的兩倍+1

```
int parent(int i) { return i / 2; }
int left(int i) { return i * 2; }
int right(int i) { return i * 2 + 1; }
```

初始化heap

先malloc給它應有的空間

開給他data的array 大小為MAX_N

capacity 也就是MAX_N

一開始size 為0

heap→data[0] = -1 data array

第一個陣列當作data取完的標示 也讓root在data[1]的位子

```
Heap *create()
{
    Heap *heap = (Heap *)malloc(sizeof(Heap));
    heap->data = (int *)malloc(sizeof(int) * MAX_N);
    heap->capacity = MAX_N;
    heap->size = 0;
    heap->data[0] = -1; //to know when the array used up
    return heap;
}
```

插入heap

當size == capacity -1 時代表heap已經滿了 沒辦法再放 所以直接return

當size < capacity -1 時就可以插入heap

index 代表new node要被放的位置

index = ++size 表示每次插入新的node 都先放在最後一個 並把size+1

再來就是檢查新插進來的node 有沒有比parent node小 有的話就把parent 往下換

然後把index換成parent的index

最後換完再把data[index]換上val

插入一個node的時間複雜度度O(log_2n)

因為tree的高度為h node數為$2^h - 1$

最多swap h次

```
void insert(Heap *heap, int val)
{
    if (heap->size == heap->capacity - 1)
        return; //out of capacity

    int index = ++heap->size; //the position to be place
    while (val < heap->data[parent(index)])
    {
        heap->data[index] = heap->data[parent(index)]; //swap down the parent
        index = parent(index);                         //make the index be the parent index
    }
    heap->data[index] = val;
}
```

change 改變node

先在array 中用iterative 方式找到要更改的數字

assign 給index

如果找不到就return

找到的話分為兩種情況

一種是modified 比origin 大

就要檢查有沒有比children大

有的話就把比較小的children換上來

這樣就確保parent會是最小的node

反之如果modified比origin 小

就要檢查有沒有比parent小

有的話就往上換 直到parent比較小

時間複雜度為(n+$log_2 n$)

n為一開始找origin的時間複雜度

$log_2 n$為上下換的時間複雜度

```
void change(Heap *heap, int origin, int modified)
{
    //change the data in the array
    int index = 0;

    for (int i = 1; i <= heap->size; ++i)
    {
        if (heap->data[i] == origin)
        {
            index = i;
            break;
        }
    }
    if (!index)
        return; //can't find the element

    //change the modified position
    if (modified < origin) //change the index up
    {
        while (modified < heap->data[parent(index)])
        {
            heap->data[index] = heap->data[parent(index)];
            index = parent(index);
        }
    }
    else //change the index down
    {
        while (left(index) <= heap->size)
        {
            if (right(index <= heap->size) && (modified > heap->data[right(index)] || modified > heap->data[left(index)]))
            {
                if (heap->data[right(index)] < heap->data[left(index)]) //deter which child to change
                {
                    heap->data[index] = heap->data[right(index)];
                    index = right(index);
                }
                else
                {
                    heap->data[index] = heap->data[left(index)];
                    index = left(index);
                }
            }
            else if (modified > heap->data[left(index)])
            {
                heap->data[index] = heap->data[left(index)];
                index = left(index);
            }
            else
                break;
```

```
        }
    }

    heap->data[index] = modified;
}
```

pop_min

取出index為1的node 也就是最小的node

pop_min 也就是min heap的精髓

把最小的pop出來

size -1

pop之後root就空掉了

我們就拿最後一個上來補

再用上面提到過的檢查children有沒有比較小 再換下去

時間複雜度O($log_2 n$)

```
int pop_min(Heap *heap)
{
    int index = 1;
    int top = heap->data[1];
    int last = heap->data[heap->size--];
    while (left(index) <= heap->size)
    {
        if (right(index <= heap->size) && (last > heap->data[right(index)] || last > heap->data[left(index)]))
        {
            if (heap->data[right(index)] < heap->data[left(index)]) //deter which child to change
            {
                heap->data[index] = heap->data[right(index)];
                index = right(index);
            }
            else
            {
                heap->data[index] = heap->data[left(index)];
                index = left(index);
            }
        }
        else if (last > heap->data[left(index)])
        {
            heap->data[index] = heap->data[left(index)];
            index = left(index);
        }
        else
            break;
    }
    heap->data[index] = last;
    return top;
}
```

實作demo

建立一個heap

將array亂數放進heap

在change 把1換成50

最後輸出結果

```
int main()
{
    Heap *heap = create();
    int n = 10;
    int arr[] = {6, 2, 4, 8, 7, 9, 1, 3, 5, 10};
    for (int i = 0; i < n; ++i)
    {
        insert(heap, arr[i]);
    }
    change(heap, 1, 50);
    for (int i = 0; i < n; ++i)
    {
        printf("pop:%d \n", pop_min(heap));
    }
    return 0;
}
```