tags: 資料結構

HW 12

題目要求建立 min heap 以及提供初始化、插入、刪除、修改功能，因為 heap 為完全二叉堆，所以用數組實現除了更加容易外也能更充分利用空間，先定義以下結構體用以承載 min heap，size 代表目前元素個數，capt 代表容量，最後 data 為一個一維陣列用以儲存堆。

```
typedef struct _heap {
    int size, capt, *data;
} Heap;
```

Init

初始化建立一個 min heap，需要注意的地方是在 `data[0]` 的位置設置一個極小值方便後面跳出循環。

```
Heap* init()
{
    Heap* tmp = (Heap*)malloc(sizeof(Heap));
    tmp->data = (int*)malloc(sizeof(int) * (maxN + 1));
    tmp->size = 0;
    tmp->capt = maxN;
    tmp->data[0] = -1;
    return tmp;
}
```

Insert

插入部分首先我們將需要插入的元素放置到最末尾 (也就是 size+1 的地方)，接著開始向上移動，先將 idx 設為 size 然後開始與其父節點比較，如果值比父節點還小的話兩者就交換位置，接著一直重複上述步驟直到再也無法進行就說明已經安插到合法的位置了。

其複雜度在最糟的情況下也就是需要將其上濾到 root 的位置，這邊假設 heap 的高度為 h ，我們可以從堆的特性知道最多會有 $2^{h+1}-1$ 個節點，因此插入一個節點最多會需要 $O(\log_2 n)$，不過有趣的是在實踐得到的平均值其實插入幾乎是逼近 $O(1)$ 的。

```
1  void insert(Heap* h, int x)
2  {
3      int idx = ++h->size;
4      while (x < h->data[idx >> 1])
5          h->data[idx] = h->data[idx >> 1], idx >>= 1;
6      h->data[idx] = x;
7  }
```

這邊有另一個有趣的現象是，在常規想法中將一個長度為 n 的無序序列做 heapfy 複雜度會是 $O(n\log_2 n)$，不過其實會是 $O(n)$，這邊就不另外求證了。

Destroy

首先取出 index 為 1 的節點就是我們要 pop 出的值，接著將末尾的節點安插到 1 的位置開始進行下濾的動作，先從左右子節點中取出最小的，如果比當前節點的值還小的話就將其互換，並將 index 也下移到我們交換的位置，然後只要一直重複下濾步驟直到再也無法繼續交換位置就說明我們找到合法的位置了，而其複雜度也會是 $O(log_2 n)$，原因與上相同不再贅述。

```
int destroy(Heap* h)
{
    int idx = 1, child;
    int top = h->data[1], tmp = h->data[h->size--];
    while (idx << 1 <= h->size) {
        child = idx << 1;

        // 判斷是否有左右孩子，有的話取出最小的那一個
        if (child != h->size && h->data[child] > h->data[child | 1])
            child++;

        // 如果 tmp 比子節點還小的話代表已經下濾到合法的位置了
        if (tmp < h->data[child])
            break;
        else
            h->data[idx] = h->data[child], idx = child;
    }

    // 最後記得將 tmp 安插到我們尋找到的位置
    h->data[idx] = tmp;
    return top;
}
```

Change

更改值其實就是上濾以及下濾，只要將兩種操作都處理一遍就可以將節點上移或是下移到正確位置了。

```
void change(Heap* h, int x, int y)
{
    // x 是要被修改的值，y 是要修改成的值
    // 在這裡需要先找到 x 在堆裡的位置
    int idx = 0, child = 0;
    for (int i = 1; i <= h->size; i++)
        if (x == h->data[i]) {
            idx = i, h->data[i] = y;
            break;
        }

     if (x > y) {
        // 修改後的值小於原本的值就需要將其上移，跟插入節點時的操作是一樣的
        while (y < h->data[idx >> 1])
            h->data[idx] = h->data[idx >> 1], idx >>= 1;
    } else if (x < y) {
        // 修改後的值大於原本的值就需要將其上移，跟刪除節點時的操作是一樣的
        while (idx << 1 <= h->size) {
            child = idx << 1;
            if (child != h->size &&
                h->data[child] > h->data[child | 1])
                child++;
            if (y < h->data[child])
                break;
            else
                h->data[idx] = h->data[child], idx = child;
        }
    }
    h->data[idx] = y;
}
```

Verfy

這邊手寫一個驗證函數來測試 heap 有無錯誤，只需要遍歷一次 data 陣列然後判斷當前節點的左右子節點有沒有都大於就好了。

```c
int verfy(Heap* h)
{
    for (int i = 1; i << 1 <= h->size; i++) {
        if (h->data[i << 1] <= h->data[i])
            return 0;
        if ((i << 1 | 1) <= h->size &
            h->data[i << 1 | 1] < h->data[i])
            return 0;
    }
    return 1;
}
```

Test

最後產生一組測資來檢驗我們的 heap。

```c
Heap* h = init(); // 初始化 heap
int n = 10;
int arr[15] = { 0, 12, 43, 65, 32, 4, 46, 87, 21, 54, 66 };

// 插入節點
for (int i = 1; i <= n; i++)
    insert(h, *(arr + i));

// 修改值並驗證
change(h, 4, 100000);
printf("%d\n", verfy(h));
change(h, 87, 1);
printf("%d\n", verfy(h));

// 將所有元素 pop，確定是否有從小排序到大
for (int i = 1; i <= n; i++)
    printf("%d ", destroy(h));
```

下面為編譯運行後的結果。

```
> gcc -Wall -Wno-unused-variable -std=c11 sol.c && ./a.out
1
1
1 12 21 32 43 46 54 65 66 100000 %
```