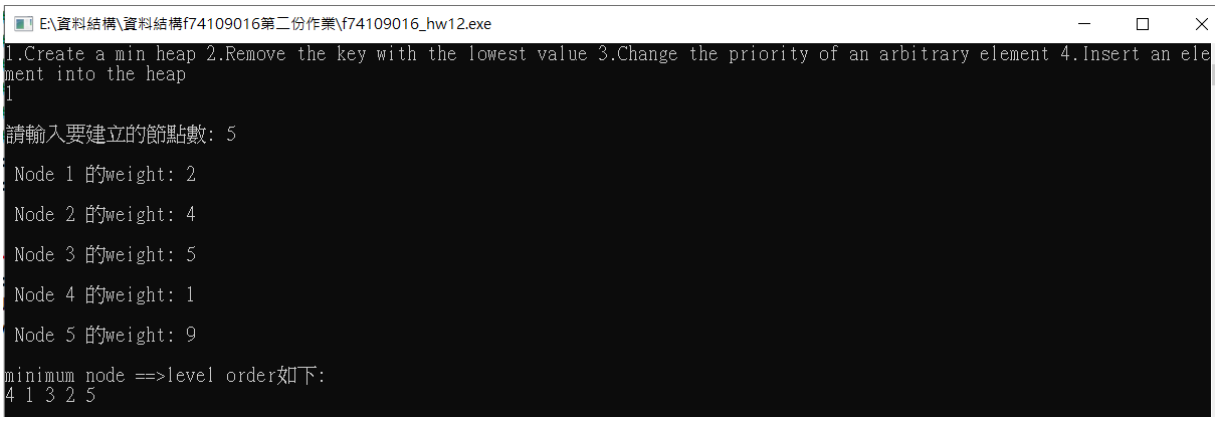
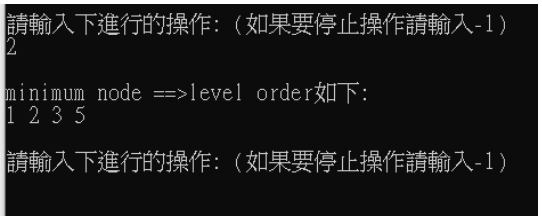
操作說明

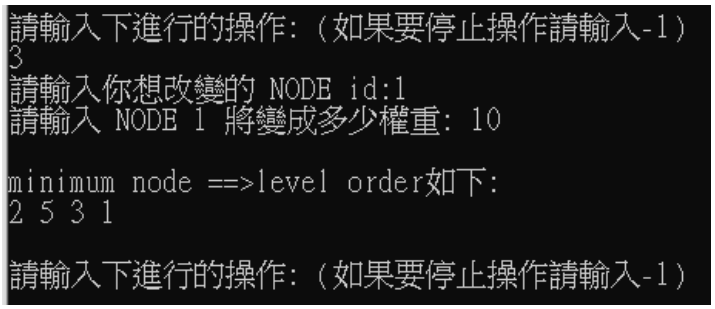
首先建立 min_heap 的樹



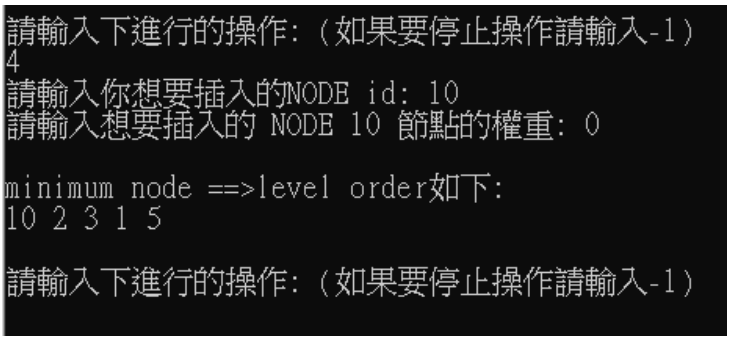
刪除 minheap 的最小值



改變權重



插入一個值



```
105 void create(int *top){
106     int node_number,temp,i;
107     do{
108         printf("\n請輸入要建立的節點數: ");
109         scanf("%d",&node_number);       //輸入要建構的節點數
110     }while(!(node_number>0));
111     for(i=1;i<=node_number;i++){
112         printf("\n Node %d 的weight: ",i);
113         scanf("%d",&temp);
114         if(i==1){
115             heap[i].weight = temp;
116             heap[i].id = i;
117         }
118         else{
119             NODE *NEW_NODE = (NODE *) malloc(sizeof(NODE));
120             NEW_NODE->weight = temp;
121             NEW_NODE->id = i;
122             push(NEW_NODE,top);
123         }
124     }
125 }
```

建立 minheap 的函數:

如果是第一個節點就直接放在陣列為 1 的位置。從 1 開始是方便直接找 parent node 與 child node。之後如果還有節點就直接用 push 就好。

```
15 void push(NODE *item,int *n){ //insert item into a maz heap of current size *n
16     int i;
17     if(HEAP_FULL(*n)){
18         printf("The heap is full. \n");
19     }
20     else{
21         i = ++(*n);
22         while((i!=1)&&(item->weight < heap[i/2].weight)){
23             heap[i].weight = heap[i/2].weight;
24             heap[i].id = heap[i/2].id;
25             i/=2;
26         }
27         heap[i].weight = item->weight;
28         heap[i].id = item->id;
29     }
30 }
```

Push 函數，先將 heap 陣列的最尾端加一(21 行)。當要插入的權重小於插入位置的父節點大小時。直接將父節點往下移從陣列指標[i/2]移到[i]➔父節點與小孩的關係。同時將 i/=2，這象徵了“欲插入節點”的位置移動到原先父節點的位置了。如果不再比新位置的父節點權重小時，就結束 while 迴圈。將新節點插在當下的位置。(無論起初插入的有沒有調整位置都是如此。)

```
31 NODE pop(int *n){
32     int parent,child;
33     NODE item,temp;
34 
35     if(HEAP_EMPTY(*n)){
36         printf("The heap is empty\n");
37     }
38     else{
39         item = heap[1];
40         temp = heap[(*n)--];
41         parent = 1;
42         child =2;
43         while(child<=*n){
44             //find the smallest key from the heap
45             if((child<*n)&&(heap[child].weight>heap[child+1].weight)){
46                 child++;
47             }
48             if(temp.weight <= heap[child].weight){
49                 break;
50             }
51             heap[parent] = heap[child];
52             parent= child;
53             child*=2;
54         }
55         heap[parent] = temp;
56         return item;
57     }
58 }
```

刪除 min_heap 函數。

將 heap[1]的值先被“最後一個節點”取代掉。在開始將該節點往下沉。先找左小孩與右小孩誰比較小。如果比較小的小孩小於 該節點，則換位置。沒有就直接結束迴圈。換完位置繼續與新的左右小孩比大小，所以 child*2，如果都持續換位置就比到小孩的 index 超過現在擁有的節點數量。child <=*n。

```
59 void change(int num,int change,int *n){
60     int i;
61     for(i=1;i<=*n;i++){
62         if(heap[i].id == num){
63             heap[i].weight = change;
64             break;
65         }
66     }
67     int change_num = i;
68     NODE * temp = (NODE *) malloc(sizeof(NODE));
69     temp->weight = heap[i].weight;
70     temp->id = heap[i].id;
71     if(temp->weight<heap[i/2].weight&&i!=1){ //比父節點還小
72         while((temp->weight < heap[i/2].weight)&&i!=1){ //比父節點還小
73             heap[i].weight = heap[i/2].weight;
74             heap[i].id = heap[i/2].id;
75             i=i/2;
76         }
77         heap[i].weight = temp->weight;
78         heap[i].id = temp->id;
79     }
```

```
80      else{
81          i = change_num ;
82          while(((temp->weight>heap[i*2+1].weight)&&i*2+1<=*(n))||((temp->weight>heap[i*2].weight)&&i*2<=*(n))){
83              if((temp->weight>heap[i*2+1].weight)&&i*2+1<=*(n)){
84                  if((heap[i/2+1].weight>heap[i/2].weight)&&(i*2+1<=*(n))){ //右小孩在節點裡 有兩種case:1.右節點
85                      heap[i].weight = heap[i*2].weight;
86                      heap[i].id = heap[i*2].id;
87                      i = i*2;
88                  }
89                  else{
90                      heap[i].weight = heap[i*2+1].weight;
91                      heap[i].id = heap[i*2+1].id;
92                      i = i*2+1;
93                  }
94              }
95              else{
96                  heap[i].weight = heap[i*2].weight;
97                  heap[i].id = heap[i*2].id;
98                  i = i*2;
99              }
100         }
101         heap[i].weight = temp->weight;
102         heap[i].id = temp->id;
103     }
104 }
```

改變權重的函數

改變後比父節點還小，就條件判斷為 true，進行裡面的 body。直接將父節點往下移從陣列指標[i/2]移到[i]➔父節點與小孩的關係。同時將 i/=2，這象徵了“欲插入節點”的位置移動到原先父節點的位置了。如果不再比新位置的父節點權重小時，就結束 while 迴圈，把改變權重的節點直接給當下位置。

改變後比子節點還大於等於就慢慢的跟小孩做比較進行 else 的 body。

有 3 種 case

While(比左小孩或右小孩大。

1. 權重比右小孩大時(也有可能比左小大)所以還有下面的判斷式。
   甲、判斷右小孩與左小孩誰大(左小孩小➔跟左小孩換)
   乙、(右小孩小➔跟右小孩換)
2. 權重比右小孩還小時，那一定是比左小孩大。
   跟左小孩換。

```
126 void display(int *n){
127     int i;
128     printf("\nminimum node ==>level order如下:\n");
129     for(i=1;i<=*n;i++){
130         printf("%d ",heap[i].id);
131     }
132     printf("\n\n");
133 }
```

因為 minheap 是 complete binary tree，所以用 level order 呈現，不會有位置上的問題與誤會。

check 函數，在改變節點與插入節點時判斷節點編號有無重複用的。

```
bool check(int temp,int * n){
    int i;
    bool exist = false;
    for(i=1;i<=*n;i++){
        if(heap[i].id==temp){
            exist = true;
            return exist;
        }
    }
    return exist;
}
```

主函數

```
void main(){
    printf("1.Create a min heap 2.Remove the key with the lowest value 3.Change the priority of an arbitrary element 4.Insert an element into 
    int operation=1;
    int current_num=1;
    int *top = &current_num;
    int i;
    do{
        do{
            if(operation!=1){
                printf("請先建立min heap\n");
            }
            scanf("%d",&operation);
        }while(operation!=1);
        create(top); //先建立heap
    }while(HEAP_EMPTY(top));
    display(top);
    do{
        printf("請輸入下進行的操作：(如果要停止操作請輸入-1)\n");
        scanf("%d",&operation);

        switch(operation){
            case 1:{
                for(i=1;i<=*top;i++){
                    heap[i].weight = -1;
                    heap[i].id = -1;
                }
                current_num = 1;
                create(top); //先建立heap
                break;
            }
```

```
            case 2:{
                pop(top);
                break;
            }
            case 3:{
                int num,bechange;
                bool exist = false;
                do{
                    printf("請輸入你想改變的 NODE id:");
                    scanf("%d",&num);
                    exist = check(num,top); //節點存在就回傳true
                    if(!exist){
                        printf("節點不存在\n");
                    }
                }while(!exist);
                printf("請輸入 NODE %d 將變成多少權重: ",num);
                scanf("%d",&bechange);
                change(num,bechange,top);
                break;
            }
            case 4:{
                int new_id,new_weight;
                bool exist;
                do{
                    exist = false;
                    printf("請輸入你想要插入的NODE id: ");
                    scanf("%d",&new_id);
                    exist = check(new_id,top);
                    if(exist == true){
                        printf("NODE %d 已經存在了，請勿再次插入\n",new_id);
                    }
                }while(exist==true);

                printf("請輸入想要插入的 NODE %d 節點的權重: ",new_id);
                scanf("%d",&new_weight);
                NODE * temp = (NODE *) malloc(sizeof(NODE));
                temp->id = new_id;
                temp->weight = new_weight;
                push(temp,top);
                break;
            }
        }
        display(top);
    }while(operation!=-1);
}
```