

Assignment 7 Technical Report

2171039 Chaewon Lee

Code 1

This is the implementation of decrease and increase key function in the min heap. This code additionally contains deletion code to show that all keys are stored well in the heap.

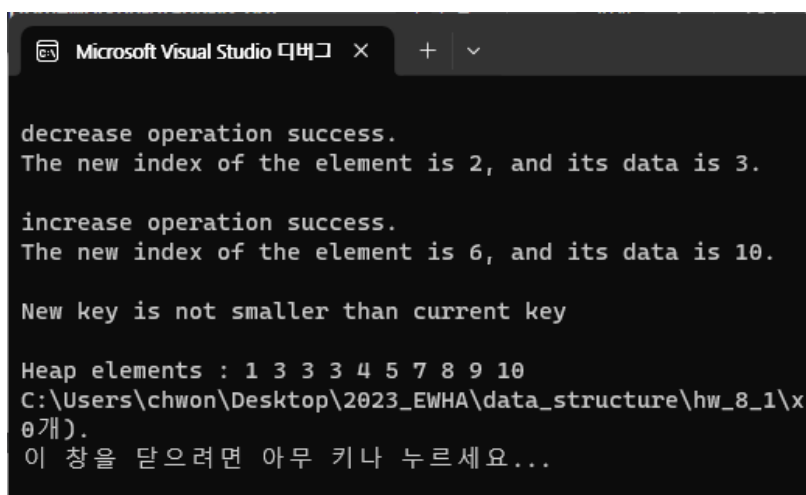
First, decrease_key_min_heap function checks the value of index i is smaller than the given key. And if it is smaller, it checks its parent whether it is smaller or not than the new key, since smaller element goes up in the min heap. Then, until there is no more smaller element left at the top, the function stops and returns the new value and index of the element.

Increase_key_min_heap works in the similar way, but the main difference is this function increases the value of the element. This also checks whether the original value of that element is larger than the key or not. And if it is larger, it checks its child whether it is bigger or not than the new key. This is opposite with decrease function. Until there is no more larger element left at the bottom of the heap, the function stops and returns the result.

Delete_min_heap function is actually not necessary for this implementation, but I added to show that my two functions mentioned above are working well, and saves the key in the heap well. This function actually works almost same as increase_key_min_heap function does. The difference is, in this function, there is always a process that the most last element of the heap becomes the first element, and comes down until there is no larger element below.

In main function, it first initializes heap, and it operates decrease and increase function. Finally, until the heap is empty, it keeps delete the top element from the heap.

- Result



```
Microsoft Visual Studio 디버그 콘솔
decrease operation success.
The new index of the element is 2, and its data is 3.

increase operation success.
The new index of the element is 6, and its data is 10.

New key is not smaller than current key

Heap elements : 1 3 3 3 4 5 7 8 9 10
C:\Users\chwon\Desktop\2023_EWHA\data_structure\hw_8_1\실행결과.txt
이 창을 닫으려면 아무 키나 누르세요...
```

You can see decrease and increase operation of the heap is successfully operated, and the result is correct. Also, you can check all the elements in the heap is stored well with the last line of the output.

Code 2

This is the implementation of prim algorithm with min heap. This also includes print_prim function, which traverses the generated mst in an inorder matter.

First, there are build_min_heap, decrease_key_min_heap, and delete_min_heap function which is need in min heap operation. Build_min_heap function makes the initial heap into min heap. Decrease and delete function is same as the function which is implemented above.

Build_min_heap function only checks the half of the elements, since another elements already meet the condition of heap. This function checkes whether there is smaller element for its child, since larger elements go down in the min heap. Until there is no smaller element left, the operation iterates the process.

Then, after the setting of the heap, prim algorithm starts to make mst. First, vertices with the smallest dist would be popped out from min heap, and the function is going to check is there any vertices that is connected with the given vertices. Then, if there is, it checks whether that vertices is still in heap, by checking selected array, and if it does not, than it also checks that the distance between two vertices is smaller than the distance saved in that new vertices. When it is, it updates the distance saved as the distance between two vertices. Then it finds that element from heap, and updates the element's distance with new value. And lastly, the new vertices' parent is saved as the vertices that is from the heap. The code keeps this process 8 times, which is the count of vertices in this graph.

Finally, print_prim function prints the completed mst in inorder traversal. To do this, we use parent array, which saves the parent of each vertices. This is implemented in an recursive manner. First, it traverses all elements of parent array and finds out the given element's child. If it is found, the function saves that child in idx array, and prints the relationship between parent vertices and child vertices. The distance between two vertices would be the value of dist[child]. After checking all elements in parent array, it runs print_prim function for all the child that is found. Those elements are stored in idx array. This brings the result of inorder traversal of mst.

- Result



```
Microsoft Visual Studio 디버그 × + ~
v : 0, dist : 0
v : 1, dist : 3
v : 2, dist : 8
v : 4, dist : 2
v : 6, dist : 4
v : 7, dist : 5
v : 5, dist : 9
v : 3, dist : 15
Vertex 0 -> 1      edge: 3
Vertex 1 -> 2      edge: 8
Vertex 2 -> 3      edge: 15
Vertex 2 -> 4      edge: 2
Vertex 4 -> 5      edge: 9
Vertex 4 -> 6      edge: 4
Vertex 4 -> 7      edge: 5
```

You can see each elements are popped well from min heap, and the graph traversal also shows the right result.