hw8\_1.cpp

(Visual studio 2019)

1. Code Explanation

1) typedef struct element, HeapType

:use this data structure to build max/min heap

2) void init(HeapType\* h)

: initialize the heap

3) void insert\_min\_heap(HeapType\* h, element item)

: inset 'item' into min-heap

4) void decrease\_key(HeapType\* h, int i, element item)

: decrease the value of ith element's value to the new value 'item'

-h: heaptype

-i: index of node that we will change

-item: we will change ith node into item

-implementation is similar with the insert operation of min heap

5) void increase\_key(HeapType\* h, int i, element item)

: increase the value of ith element's value to the new value 'item'

-h: heaptype

-i: index of node that we will change

-item: we will change i th node into item

-implementation is similar with the delete operation of min heap

6) int main()

-build min heap

-print elements of initial min heap in order

-implement increase\_key, decrease\_key operation

-print the changed heap in order

2. Result

텍스트이(가) 표시된 사진

자동 생성된 설명

**hw8\_2.cpp**

(Visual studio 2019)

1. Code Explanation

1) typedef struct element

-dist: instead of unsorted array 'dist'

-num: index number representing vertex(a->0, b->1, ...)

2) typedef struct Vertex

:structure in order to specify the parent-child relation

- parent: pointer of parent vertex

3) void init(HeapType\* h)

: initialize the heap

4) int find\_index(HeapType\* h, int a)

: find the index of node in heap 'h' which 'num' is a

e.g) if h->heap[x].num == a then return x

5) void insert\_min\_heap(HeapType\* h, element item)

: inset 'item' into min-heap

6) void decrease\_key\_min\_heap(HeapType\* h, int i, element item)

: decrease the value of ith element's value to the new value 'item'

-> same with function 4) int- hw8\_1.cpp

7) element delete\_min\_heap(HeapType\* h)

: delete the root node in min heap

8) void build\_min\_heap(HeapType\* h)

: sort the nodes in heap into order and build min heap

9) void prim(int s, int n)

: prim algorithm function using min heap

-s: starting vertex

-n: the number of vertices on the graph

//u.parent=NULL

ver[u]->parent = NULL;

//inset all vertiecs into the priority queue Q

build\_min\_heap(&Q);

//Extract\_Min(Q)

e2 = delete\_min\_heap(&Q);

ver[v]->parent = ver[u]; //parent of v is u

//replace dist[v] into weight[u][v]

element e3 = { weight[u][v],v };

decrease\_key\_min\_heap(&Q, find\_index(&Q, v), e3);