1291036 이정욱

//1번, 2번

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_ELEMENT 200

typedef struct

{

char key[20];

int id;

}element;

typedef struct

{

element heap[MAX\_ELEMENT];

int heap\_size;

} HeapType;

void init(HeapType \*h)

{

h->heap\_size = 0;

}

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

{

int i;

i = ++(h->heap\_size);

// 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정

while ((i != 1) && (strcmp((item.key), h->heap[i / 2].key) == 1)) //strcmp함수 앞이 크면 1 뒤가 크면 -1 같으면 0

{

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item; // 새로운 노드를 삽입

}

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

{

int i;

i = ++(h->heap\_size);

// 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정

while ((i != 1) && (strcmp((item.key), h->heap[i / 2].key) == -1))

{

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item; // 새로운 노드를 삽입

}

element delete\_max\_heap(HeapType \*h)

{

int parent, child;

element item, temp;

item = h->heap[1];

temp = h->heap[(h->heap\_size)--];

parent = 1;

child = 2;

while (child <= h->heap\_size)

{

// 현재 노드의 자식노드중 더 큰 자식노드를 찾는다.

if ((child < h->heap\_size) && (strcmp(h->heap[child].key, h->heap[child + 1].key) == -1))

child++;

if (strcmp(temp.key, h->heap[child].key) >= 0) //if (temp.key >= h->heap[child].key)

break;

// 한단계 아래로 이동

h->heap[parent] = h->heap[child];

parent = child;

child \*= 2;

}

h->heap[parent] = temp;

return item;

}

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

{

int parent, child;

element item, temp;

item = h->heap[1];

temp = h->heap[(h->heap\_size)--];

parent = 1;

child = 2;

while (child <= h->heap\_size)

{

// 현재 노드의 자식노드중 더 작은 자식노드를 찾는다.

if ((child < h->heap\_size) && (strcmp(h->heap[child].key, h->heap[child + 1].key) == 1))

child++;

if (strcmp(temp.key, h->heap[child].key) <= 0)

break;

// 한단계 아래로 이동

h->heap[parent] = h->heap[child];

parent = child;

child \*= 2;

}

h->heap[parent] = temp;

return item;

}

void print\_heap(HeapType \*heap)

{

for (int i = 1; i <= heap->heap\_size; i++)

printf("<\"%s\",%d> ", heap->heap[i].key, heap->heap[i].id);

printf("\n");

}

void sort\_heap(element a[], int n)

{

int i;

HeapType h;

init(&h);

for (i = n - 1; i >= 0; i--)

insert\_min\_heap(&h, a[i]);

for (i = 0; i < n; i++)

a[i] = delete\_min\_heap(&h);

}

void main()

{

element e[3] = { { "aegg", 1234 }, { "bstructure", 321 }, { "cepic", 4885 } };

HeapType heap; // 히프 생성

init(&heap); // 초기화

insert\_max\_heap(&heap, e[0]);

print\_heap(&heap);

insert\_max\_heap(&heap, e[1]);

print\_heap(&heap);

insert\_max\_heap(&heap, e[2]);

print\_heap(&heap);

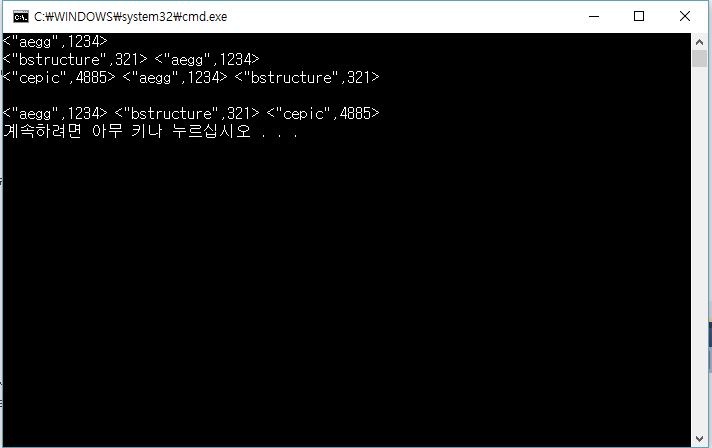
printf("\n");

sort\_heap(e, 3);

for (int i = 0; i < 3; i++)

printf("<\"%s\",%d> ", e[i].key, e[i].id);

printf("\n");

}

수업시간에 배운대로 히프의 기본 알고리즘에 따라 구성했습니다. 2번문제 또한 배운내용을 조금 응용하여 구현했습니다. 사실 2번문제의 의도를 파악하는데 조금 오래걸렸습니다.

//3번

#include <stdio.h>

#define MAX\_ELEMENT 100

typedef struct

{

int key;

}element;

typedef struct

{

element heap[MAX\_ELEMENT];

int heap\_size;

}HeapType;

void init(HeapType \*h)

{

h->heap\_size = 0;

}

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

{

int i;

i = ++(h->heap\_size);

// 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정

while ((i != 1) && (item.key > h->heap[i / 2].key)) {

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item; // 새로운 노드를 삽입

}

element delete\_max\_heap(HeapType \*h)

{

int parent, child;

element item, temp;

item = h->heap[1];

temp = h->heap[(h->heap\_size)--];

parent = 1;

child = 2;

while (child <= h->heap\_size) {

// 현재 노드의 자식노드중 더 큰 자식노드를 찾는다.

if ((child < h->heap\_size) &&

(h->heap[child].key) < h->heap[child + 1].key)

child++;

if (temp.key >= h->heap[child].key) break;

// 한단계 아래로 이동

h->heap[parent] = h->heap[child];

parent = child;

child \*= 2;

}

h->heap[parent] = temp;

return item;

}

void print\_heap(HeapType \*h)

{

for (int i = 1; i <= h->heap\_size; i++)

printf("<%d >", h->heap[i].key);

printf("\n");

}

void delete\_element(HeapType \*h, int number)

{ // number를 key로 갖는 element를 삭제한다. 삭제 후에 max heap이 유지되어야 한다

int iIndex;

for(int i = 1; i<= h->heap\_size; i++)

if(h->heap[i].key == number)

iIndex = i;

element item, temp;

item = h->heap[iIndex];

temp = h->heap[(h->heap\_size)--];

int parent = iIndex;

int child = iIndex \* 2;

while (child <= h->heap\_size) {

// 현재 노드의 자식노드중 더 큰 자식노드를 찾는다.

if ((child < h->heap\_size) &&

(h->heap[child].key) < h->heap[child + 1].key)

child++;

if (temp.key >= h->heap[child].key) break;

// 한단계 아래로 이동

h->heap[parent] = h->heap[child];

parent = child;

child \*= 2;

}

h->heap[parent] = temp;

}

void main()

{

element e[9] = { { 10 },{ 40 },{ 30 },{ 5 },{ 12 },{ 6 },{ 15 },{ 9 },{ 60 } };

HeapType heap1;

int i;

element a;

init(&heap1);

for (i = 0; i < 9; i++)

insert\_max\_heap(&heap1, e[i]);

print\_heap(&heap1);

int num = 40;

delete\_element(&heap1, num);

print\_heap(&heap1);

//max heap인지 체크하는 과정 값이 큰 순서대로 출력되어야 한다

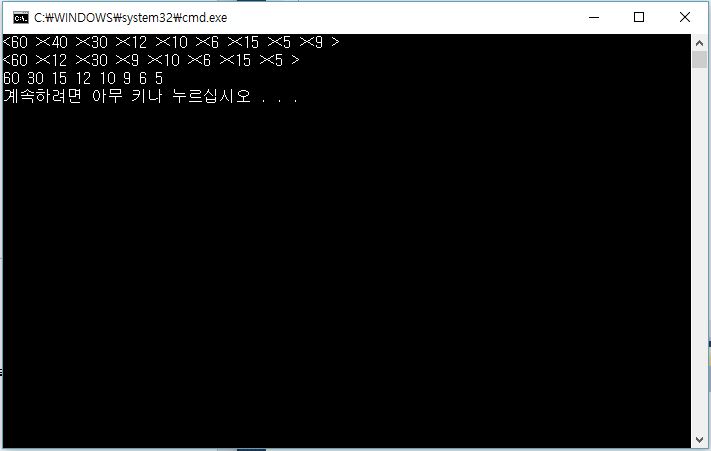
for (i = 0; i < 8; i++)

{

a = delete\_max\_heap(&heap1);

printf("%d ", a.key);

}

}

굉장히 애를 먹은 문제였습니다. 하지만 중간에 있는 임의의 원소를 제거하는 것도 어떻게 보면 그 노드부터 시작하는 heap를 delete\_max\_heap하는 알고리즘과 같을 거라는 생각으로 구현했습니다.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_ELEMENT 200

#define K\_NARY 3

typedef struct {

int key;

}element;

typedef struct {

element heap[MAX\_ELEMENT];

int heap\_size;

}HeapType;

void init(HeapType \*h)

{

h->heap\_size = 0;

}

//left\_child of i=i\*k-(k-2)

//parent index of l=(l+k-2)/k

void insert\_max\_heap(HeapType \*h, element item, int k)

{

//k-max-heap(자식이 k개인 완전 k-nary tree를 이용한 heap))이 되도록 자료를 입력

int i;

i = ++(h->heap\_size);

while ((i != 1) && (item.key > h->heap[(i + k - 2) / k].key))

{

h->heap[i] = h->heap[(i + k - 2) / k];

i = (i + k - 2) / k;

}

h->heap[i] = item; // 새로운 노드를 삽입

}

int max(element heap[], int start, int number)

{

//heap의 start에서 number 개 만큼의 원소 중 가장 key가 큰 원소의 index를 반환하다.

int temp = 0;

for (int i = 0; i < number; i++)

temp = (heap[start + i].key > temp) ? heap[start + i].key : temp;

return temp;

}

element delete\_max\_heap(HeapType \*h, int k)

{

//k-max-heap에서 가장 key가 큰 원소를 반환한다. 반환 후 k-max-heap을 유지한다. 위의 max함수를 이용

int parent, child;

element item, temp;

item = h->heap[1];

temp = h->heap[(h->heap\_size)--];

parent = 1;

child = 2;

while (child <= h->heap\_size)

{

// 현재 노드의 자식노드중 더 큰 자식노드를 찾는다.

for (int i = 0; i < k; i++)

if ((child < h->heap\_size) && (h->heap[child].key) < h->heap[child + 1].key)

child++;

if (temp.key >= h->heap[child].key)

break;

// 한단계 아래로 이동

h->heap[parent] = h->heap[child];

parent = child;

child = child \* k - (k - 2);

}

h->heap[parent] = temp;

return item;

}

void heap\_sort(element a[], int n, int k)

{ //k-max-heap을 이용하여 정렬한다.

int i;

HeapType h;

init(&h);

for (i = 0; i < n; i++)

insert\_max\_heap(&h, a[i], k);

for (i = n - 1; i >= 0; i--)

a[i] = delete\_max\_heap(&h, k);

}

void print\_heap(HeapType \*h)

{

for (int i = 1; i <= h->heap\_size; i++)

printf("< %d > ", h->heap[i].key);

printf("\n");

}

void main()

{

//사이즈 10이상인 정렬 되지 않은 정수형 배열을 위의 heap\_sort를 이용하여

//정렬하시오. k값을 여러 가지 경우로 해서 test해 보시오.

element e[13] = { { 10 },{ 40 },{ 30 },{ 5 },{ 12 },{ 6 },{ 15 },{ 9 },{ 60 },{ 99 },{ 75 },{ 89 },{ 48 } };

HeapType heap; // 히프 생성

init(&heap); // 초기화

for (int i = 0; i < 13; i++)

insert\_max\_heap(&heap, e[i], K\_NARY);

print\_heap(&heap);

printf("\n");

delete\_max\_heap(&heap, K\_NARY);

print\_heap(&heap);

printf("\n");

printf("%d\n", max(e, 3, 6));

printf("\n");

heap\_sort(e, 13, K\_NARY);

for (int i = 0; i < 13; i++)

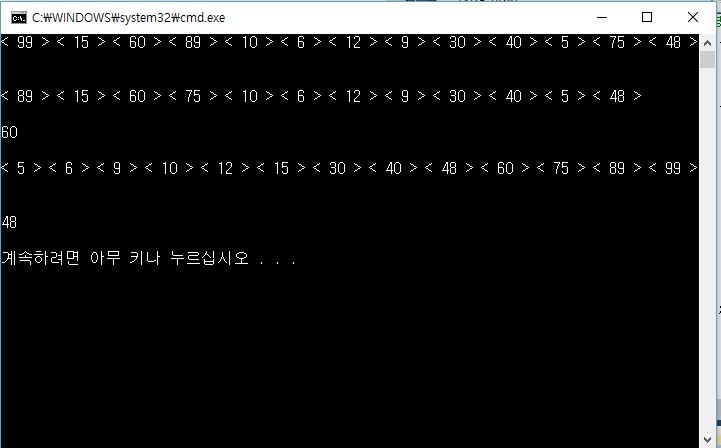
printf("< %d > ", e[i].key);

printf("\n");

printf("\n");

printf("%d\n", max(e, 3, 6));

printf("\n");

}

굉장히 어려웠지만 왼쪽자식 = 자신의 인덱스\*k – (k-2) 를 잘 사용해서 구현했습니다. 바꿔야할 부분의 특성을 알아낸 이후부터는 일사천리였습니다. 2진트리일 때 i/2는 k진트리일 때 (i+k-2)/k를 알아내어 풀었습니다.