자료구조응용

17. 최대 히프, 이진 탐색 트리 (10점)

2025.11.10

1. **다음 입력파일의 데이터를 사용하여 최대 히프(Max Heap)에 대한 실습을 수행한다.(4점)**

input.txt : 10 40 30 5 12 6 15 9 60

[실행 순서]

① 파일 입력을 받으면서 최대 히프를 구성한다.

매 입력마다, 구성된 최대 히프의 배열 원소를 인덱스 순서대로 출력한다.

② 최대 히프의 최대값을 연속으로 원소 개수 만큼 삭제한다.

매 삭제마다, 재구성된 최대 히프의 배열 원소를 인덱스 순서대로 출력한다.

[구현 전, 노트에 연습하기]

① 위 실행 순서 ①의 최대 히프가 만들어지는 과정을 **최대 히프 그림**으로 보여라.

② 위 실행 순서 ②의 삭제 연산 과정을 **최대 히프 그림**으로 보여라.

③ ①②의 각 트리 결과에 대해 **배열로 표시**해 보라.

※ 각각의 Key 추가/삭제에 대해 결과 트리만 표시하면 됨

※ 노트에 직접 그려서 사진을 찍거나 파일로 작성하여 보고서에 포함

[구현 세부 사항]

```
#define MAX_ELEMENTS 200 /* maximum heap size+1 */
#define HEAP_FULL(n) (n == MAX_ELEMENTS-1)
#define HEAP_EMPTY(n) (!n)
typedef struct {
        int key;
        /* other fields */
        } element;
element heap[MAX_ELEMENTS];
int n = 0;
```

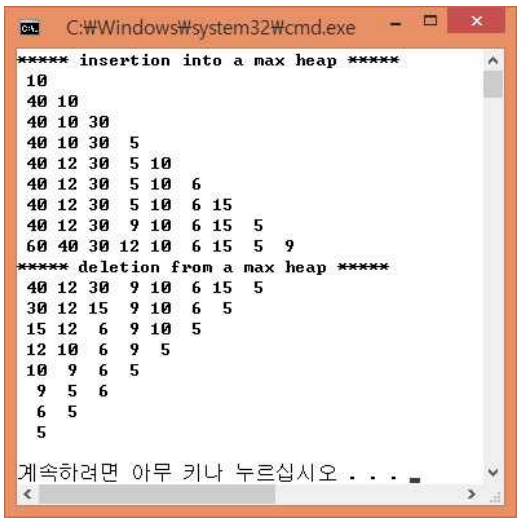
```
void push(element item, int *n)
{/* insert item into a max heap of current size *n */
   int i;
   if (HEAP_FULL(*n)){
      fprintf(stderr, "The heap is full. \n");
      exit(EXIT_FAILURE);
   }
   i = ++(*n);
   while ((i != 1) && (item.key > heap[i/2].key)) {
      heap[i] = heap[i/2];
      i /= 2;
   }
   heap[i] = item;
}
```

Program 5.13: Insertion into a max heap

```
element pop(int *n)
{/* delete element with the highest key from the heap */
   int parent, child;
   element item, temp;
   if (HEAP_EMPTY(*n)) {
      fprintf(stderr, "The heap is empty\n");
      exit(EXIT_FAILURE);
   }
   /* save value of the element with the highest key */
   item = heap[1];
   /* use last element in heap to adjust heap */
   temp = heap[(*n)--];
   parent = 1;
   child = 2;
   while (child <= *n) {
      /* find the larger child of the current parent */
      if((child < *n) && (heap[child].key < heap[child+1].key))
         child++;
      if (temp.key >= heap[child].key) break;
      /* move to the next lower level */
      heap[parent] = heap[child];
      parent = child;
      child *= 2;
   }
   heap[parent] = temp;
   return item;
}
```

Program 5.14: Deletion from a max heap

[실행예]



2. 다음과 같이 임의의 노드 n개로 구성된 이진 탐색 트리(binary search tree)를 생성하는 프로그램을 작성하라. (6점)

[실행순서]

① 난수생성을 위한 **seed**와 **이진 탐색 트리의 노드 개수(n)**를 입력받음

※ scanf

② 1~500 범위의 난수를 생성하여 노드의 key와 item 필드 값으로 동일하게 사용

※ 이진 탐색 트리의 key 값은 중복이 허용되지 않음을 주의

③ ②의 key, item을 사용하여 이진 탐색 트리에 노드를 하나 추가함

※ **Program 5.17. insert**

※ Program 5.17에서 사용된 **modifiedSearch**함수는 Program 5.16을 수정함

④ ②~③ 과정을 n번 수행하여 이진 탐색 트리를 구성

※ 난수 발생 순서대로 노드를 추가해야 함

⑤ 탐색할 key를 입력받아서 이진 탐색하여 그 결과를 출력한다.

※ Program 5.15 혹은 5.16

⑥ 이진 탐색 트리를 구성하고 있는 노드의 **key값을 오름차순으로 정렬되도록 출력**함

※ inorder traversal 사용

[구현 세부 사항]

```
typedef int iType;
typedef struct{
        int key;
        iType item;
        }element;
typedef struct node *treePointer;
typedef struct node{
        element data;
        treePointer leftChild, rightChild;
        }node;
```

```
element* search(treePointer root, int key)
{/* return a pointer to the element whose key is k, if
    there is no such element, return NULL. */
  if (!root) return NULL;
  if (k == root→data.key) return &(root→data);
  if (k < root→data.key)
    return search(root→leftChild, k);
  return search(root→rightChild, k);
}
```

Program 5.15: Recursive search of a binary search tree

```
element* iterSearch(treePointer tree, int k)
{/* return a pointer to the element whose key is k,  if
    there is no such element, return NULL. */
  while (tree) {
    if (k == tree→data.key) return &(tree→data);
    if (k < tree→data.key)
      tree = tree→leftChild;
    else
      tree = tree→rightChild;
    }
  return NULL;
}
```

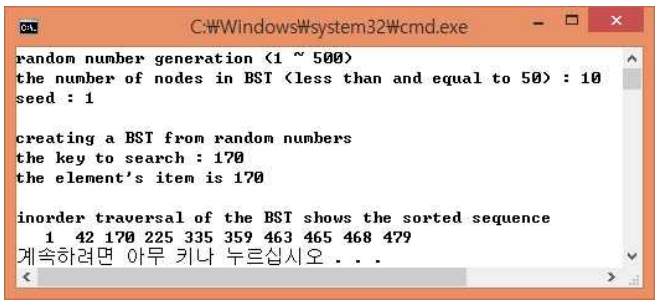
Program 5.16: Iterative search of a binary search tree

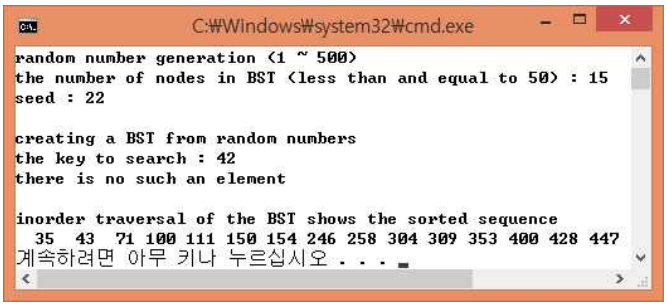
```
void insert(treePointer *node, int k, iType theItem)
{/* if k is in the tree pointed at by node do nothing;
    otherwise add a new node with data = (k, theItem) */
  treePointer ptr, temp = modifiedSearch(*node, k);
  if (temp || !(*node)) {
    /* k is not in the tree */
    MALLOC(ptr, sizeof(*ptr));
    ptr→data.key = k;
    ptr→data.item = theItem;
    ptr→leftChild = ptr→rightChild = NULL;
    if (*node) /* insert as child of temp */
      if (k < temp→data.key) temp→leftChild = ptr;
      else temp→rightChild = ptr;
    else *node = ptr;
  }
}
```

Program 5.17: Inserting a dictionary pair into a binary search tree

modifiedSearch 알고리즘 (Program 5.16 iterSearch의 수정)
1. 만약 BST가 empty라면 NULL을 반환한다. 2. empty BST가 아닌 한 다음 과정을 반복한다. ① 루트 키 값이 탐색키 k와 같으면 NULL을 반환한다. ② 만약 *k*가 루트 키 값보다 작으면, 왼쪽 부트리의 루트를 새로운 루트로 만든다. 그렇지 않으면, 루트의 오른쪽 부트리의 루트를 새로운 루트로 한다. 3. *2의 탐색 과정에서 만난 마지막 노드에 대한 포인터*를 반환한다. ※ 마지막 노드 : non-leaf 혹은 leaf node일 수 있음

[실행예]





■ 제출 형식

- 솔루션 이름 : DS 17
- 프로젝트 이름 : 1, 2
- 각 소스파일에 주석처리

  "학번 이름"

  "본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다."
- 제출 파일

  ① 소스코드와 실행 결과가 보이도록 화면을 캡쳐한 **보고서** 파일(**"학번.pdf"**)

  ※ 한글 [파일 → pdf로 저장하기...] 메뉴 사용

  ② **C 소스 파일**을 하나의 디렉터리에 모아 압축한 파일 (**"학번.zip"**)

  ※ "학번.pdf"와 "학번.zip"을 하나로 압축하지 말고 별도 파일로 제출

■ **주의**

- 소스 복사로는 실력향상을 기대할 수 없습니다!!!
- 1차 마감 : 수업일 자정
- 2차 마감 : 수업 익일 자정(만점의 60%, 반올림)
- 문항 별로 1차 2차 나눠서 제출할 수 없으며, 최종 제출 시간에 따라 1차, 2차로 구분함