14주차 2차시 이진 검색과 이진 트리 검색

【학습목표】

1. 이진 검색의 개념과 알고리즘을 설명할 수 있다.
2. 이진 탐색 트리를 이용한 검색 방법을 설명할 수 있다.

학습내용1 : 이진 검색 방법과 알고리즘

1. 이진 검색(binary search, 이분 검색, 보간 검색(interpolation search))

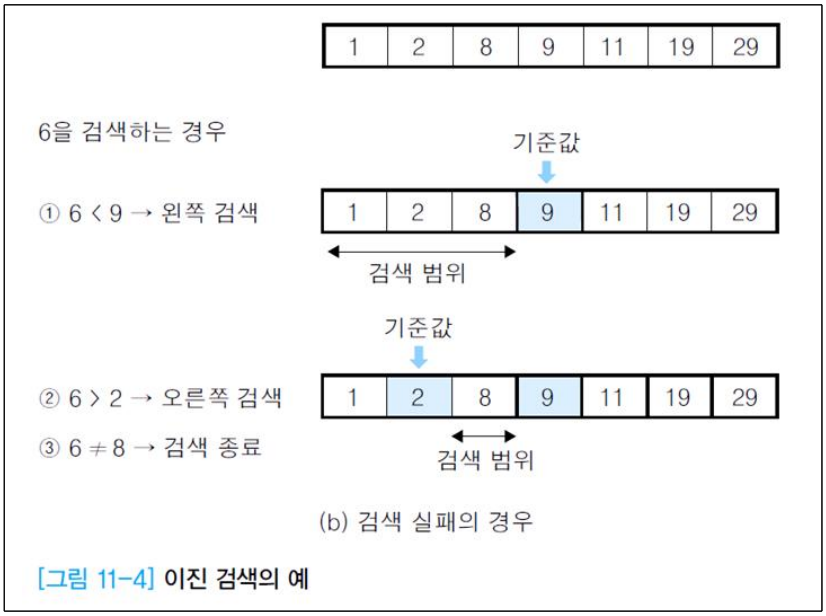
* 자료의 가운데에 있는 항목을 키 값과 비교하여 다음 검색 위치를 결정하여 검색을 계속하는 방법
- 찾는 키 값 > 원소의 키 값 : 오른쪽 부분에 대해서 검색 실행
- 찾는 키 값 < 원소의 키 값 : 왼쪽 부분에 대해서 검색 실행

* 키를 찾을 때까지 이진 검색을 순환적으로 반복 수행함으로써 검색, 범위를 반으로 줄여가면서 더 빠르게 검색

* 정복 기법을 이용한 검색 방법
- 검색 범위를 반으로 분할하는 작업과 검색 작업을 반복 수행

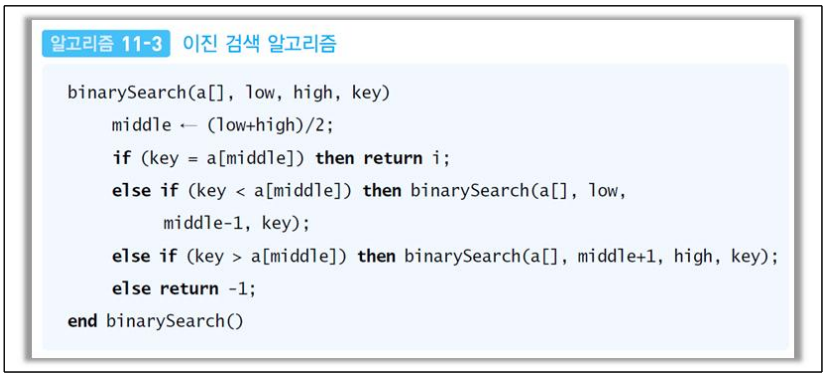
* 반드시 정렬되어있는 자료에 대해서 수행하는 검색 방법

* 이진 검색의 예제



[그림 11-4] 이진 검색의 예

2. 이진 검색 알고리즘



알고리즘 11-3 이진 검색 알고리즘

```
binarySearch(a[], low, high, key)
    middle ← (low+high)/2;
    if (key = a[middle]) then return i;
    else if (key < a[middle]) then binarySearch(a[], low,
        middle-1, key);
    else if (key > a[middle]) then binarySearch(a[], middle+1, high, key);
    else return -1;
end binarySearch()
```

* 삽입이나 삭제가 발생했을 경우에 항상 배열의 상태를 정렬 상태로 유지하는 추가적인 작업 필요

* 시간 복잡도 : $O(\log_2 n)$

학습내용2 : 이진 트리 검색

1. 정의

* 11주차에서 설명한 이진 탐색 트리를 사용한 검색 방법

* 이진 탐색 트리는 루트 노드를 기준으로 왼쪽 서브 트리는 루트 노드보다 킷값이 작은 원소들로 구성하고 오른쪽 서브 트리는 루트 노드보다 킷값이 큰 원소들로 구성한 이진트리이다

* 이진 탐색 트리의 특성을 이용하여 검색은 용이하나, 원소의 삽입이나 삭제 연산에 대해서 항상 이진 탐색 트리를 재구성하는 작업 필요

2. 이진 탐색 트리의 검색 연산

* 루트에서 시작한다

* 탐색할 킷값 x를 루트 노드의 킷값과 비교한다
- (킷값 x = 루트노드의 킷값)인 경우 : 원하는 원소를 찾았으므로 탐색연산 성공
- (킷값 x < 루트노드의 킷값)인 경우 : 루트노드의 왼쪽 서브트리에서 탐색연산 수행
- (킷값 x > 루트노드의 킷값)인 경우 : 루트노드의 오른쪽 서브트리에서 탐색연산 수행

3. 탐색 연산 알고리즘

알고리즘 8-4 이진 탐색 트리에서의 탐색 연산 알고리즘

```
searchBST(bsT, x)
    p ← bsT;
    if (p=null) then
        return null;
    if (x = p.key) then
        return p;
    if (x < p.key) then
        return searchBST(p.left, x);
    else return searchBST(p.right, x);
end searchBST( )
```

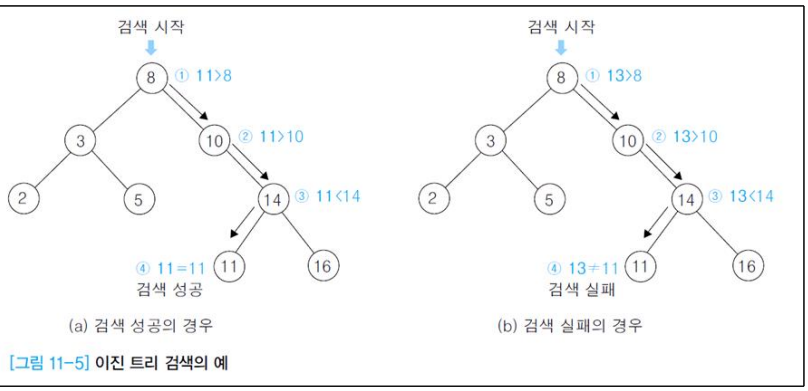
* 탐색 연산 예제 - 원소 11 탐색하기

① 찾는 킷값 11을 루트노드의 킷값 8과 비교 (찾는 킷값 11 > 노드의 킷값 8) 이므로 오른쪽 서브트리를 탐색

② (찾는 킷값 11 > 노드의 킷값 10) 이므로, 다시 오른쪽 서브트리를 탐색

③ (찾는 킷값 11 < 노드의 킷값 14) 이므로, 왼쪽 서브트리를 탐색

④ (찾는 킷값 11 = 노드의 킷값 11) 이므로, 탐색 성공! (연산 종료)



[그림 11-5] 이진 트리 검색의 예

학습내용3 : 색인 순차 검색

1. 이진 트리 검색을 이용한 영어사전 검색 프로그램

* 영어 단어와 뜻을 입력하면 알파벳순서대로 이진 탐색 트리에 삽입
* 검색할 단어를 입력하면 이진 트리 검색으로 검색하여 단어의 뜻을 출력
* 사전 출력을 선택하면 트리를 중위 순회하면서 트리에 있는 모든 단어를 출력
* 이진 검색을 이용한 영어 사전 프로그램

```
001 #include <stdio.h>
002 #include <stdlib.h>
003 #include <string.h>
004 #include <memory.h>
005
006 #define MAX_WORD_LENGTH 20
007 #define MAX_MEAN_LENGTH 200
008
009 typedef struct {           // 영어 사전 항목의 구조 정의
010     char word[MAX_WORD_LENGTH];
011     char mean[MAX_MEAN_LENGTH];
012 } element;
013
014 typedef struct treeNode{  // 영어 사전 이진 트리의 노드 구조 정의
015     element key;
016     struct treeNode *left;
017     struct treeNode *right;
018 } treeNode;
019
020 treeNode* insertKey(treeNode *p, element key)    // 예제 8-3 참고
021 {  // 포인터 p가 가리키는 노드와 비교하여 항목 key를 삽입하는 연산
022     treeNode *newNode;
023     int compare;
024     if (p == NULL) {  // 삽입할 자리에 새 노드를 구성하여 연결
025         newNode = (treeNode*)malloc(sizeof(treeNode));
026         newNode->key = key;
027         newNode->left = NULL;
028         newNode->right = NULL;
029         return newNode;
030     }
031     else {   // 이진 트리에서 삽입할 자리 탐색
```

```
032         compare = strcmp(key.word, p->key.word);
033         if(compare < 0)       p->left = insertKey(p->left, key);
034         else if(compare > 0)  p->right = insertKey(p->right, key);
035         else  printf("\n 이미 같은 단어가 있습니다! \n");
036
037         return p;  // 삽입한 자리를 반환
038     }
039 }
040
041 void insert(treeNode** root, element key)
042 {
043     *root = insertKey(*root, key);
044 }
045
046 void deleteNode(treeNode *root, element key)
047 {  // root 노드 부터 탐색하여 key와 같은 노드를 찾아 삭제하는 연산
048     treeNode *parent, *p, *succ, *succ_parent;
049     treeNode *child;
050
051     parent=NULL;
052     p=root;
053     while((p != NULL) && (strcmp(p->key.word, key.word)!=0)){
054         parent=p;
055         if(strcmp(key.word, p->key.word)<0)  p=p->left;
056         else  p=p->right;
057     }
058     if(p == NULL){   // 삭제할 노드가 없는 경우
059         printf("\n 삭제할 단어가 사전에 없습니다!!");
060         return;
061     }
062     // 삭제할 노드가 단말 노드인 경우
063     if((p->left == NULL) && (p->right == NULL)){
064         if(parent != NULL){
```

```
065                 if(parent->left == p) parent->left=NULL;
066                 else parent->right=NULL;
067             }
068             else root=NULL;
069     }
070     // 삭제할 노드가 한 개의 자식 노드를 가진 경우
071     else if((p->left == NULL) || (p->right == NULL)){
072         if(p->left != NULL) child=p->left;
073         else  child=p->right;
074
075         if(parent != NULL){
076             if(parent->left == p) parent->left=child;
077             else  parent->right=child;
078         }
079         else root=child;
080     }
081     // 삭제할 노드가 두 개의 자식 노드를 가진 경우
082     else{
083         succ_parent=p;
084         succ=p->right;
085         while(succ->left != NULL){
086             succ_parent=succ;
087             succ=succ->left;
088         }
089         if(succ_parent->left == succ)
090             succ_parent->left=succ->right;
091         else   succ_parent->right=succ->right;
092
093         p->key=succ->key;
094         p=succ;
095     }
096     free(p);
097 }
```

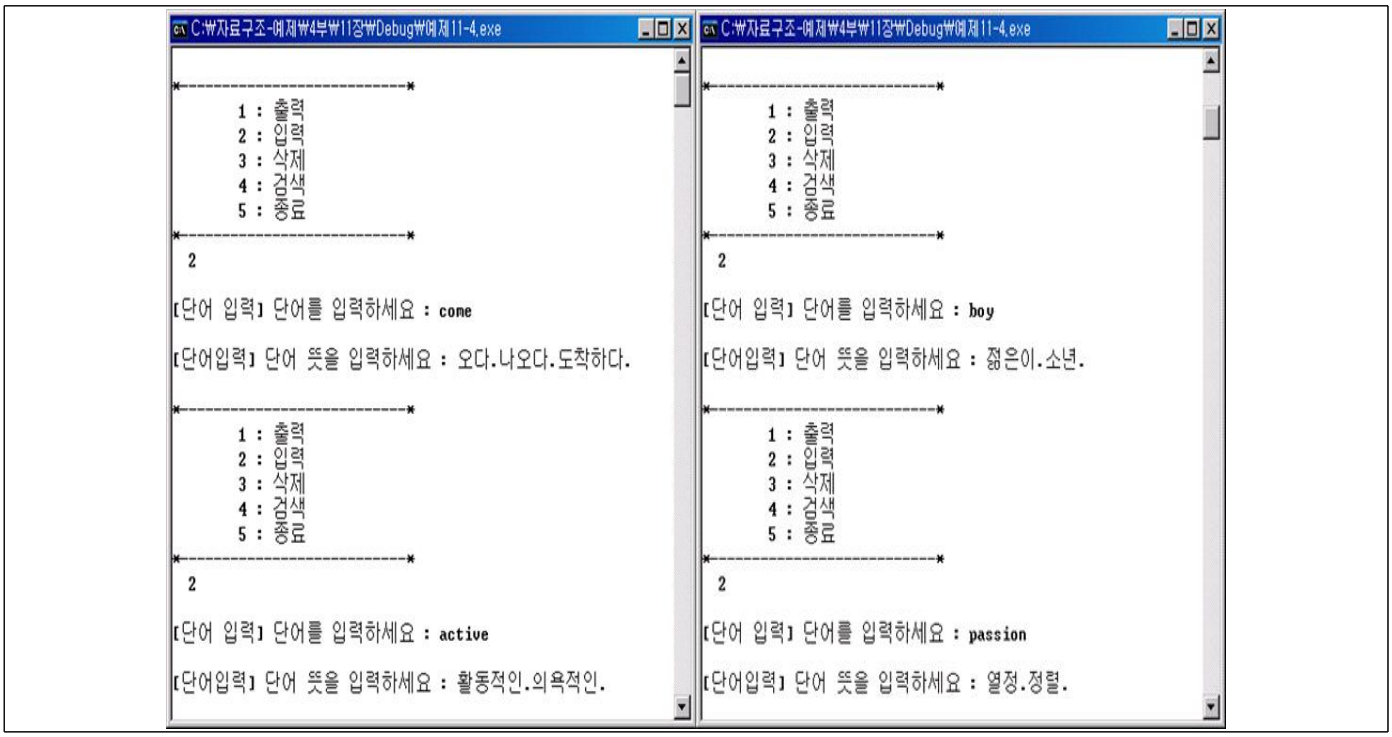
```
098
099 treeNode* searchBST(treeNode* root, element key)
100 {  //이진 탐색 트리에서 키값이 key인 노드의 위치를 탐색하는 연산
101    treeNode* p;
102    int compare;
103    p = root;
104
105    while (p != NULL){
106       compare = strcmp(key.word, p->key.word);
107       if(compare < 0)      p = p->left;
108       else if(compare > 0)  p = p->right;
109       else {
110          printf("\n찾은 단어 : %s", p->key.word);
111          return p;
112       }
113    }
114    return p;
115 }
116
117 void displayInorder(treeNode* root)
118 {   //이진 탐색 트리를 중위 순회하면서 출력하는 연산
119    if(root){
120       displayInorder(root->left);
121       printf("\n%s : %s", root->key.word, root->key.mean);
122       displayInorder(root->right);
123    }
124 }
125
126 void menu()
127 {
128    printf("\n*--------------------------*");
129    printf("\n\t1 : 출력");
130    printf("\n\t2 : 입력");
```

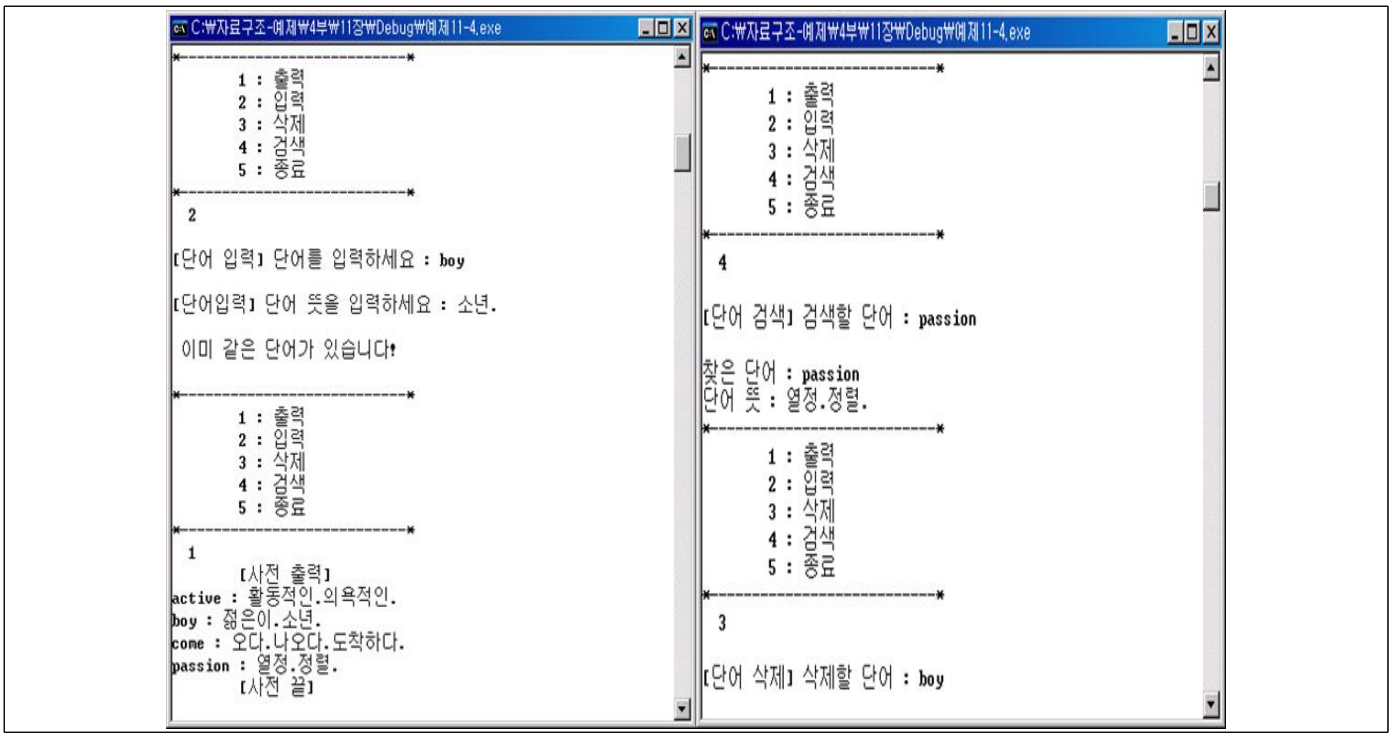
```
131     printf("\n\t3 : 삭제");
132     printf("\n\t4 : 검색");
133     printf("\n\t5 : 종료");
134     printf("\n*---------------------------*\n  ");
135 }
136
137 void main()
138 {
139     char choice;
140     element e;
141     treeNode *root=NULL, *temp=NULL;
142
143     do{
144         menu();
145         choice = getchar(); getchar();
146
147         switch(choice-'0'){
148             case 1 :
149                 printf("\t[사전 출력]");
150                 displayInorder(root); printf("\n\t[사전 끝]\n");
151                 break;
152             case 2 :
153                 printf("\n[단어 입력] 단어를 입력하세요 : "); gets(e.word);
154                 printf("\n[단어입력] 단어 뜻을 입력하세요 : "); gets(e.mean);
155                 insert(&root, e);
156                 break;
157             case 3 :
158                 printf("\n[단어 삭제] 삭제할 단어 : "); gets(e.word);
159                 deleteNode(root, e);
160                 break;
161             case 4 :
162                 printf("\n[단어 검색] 검색할 단어 : ");
163                 gets(e.word);
```

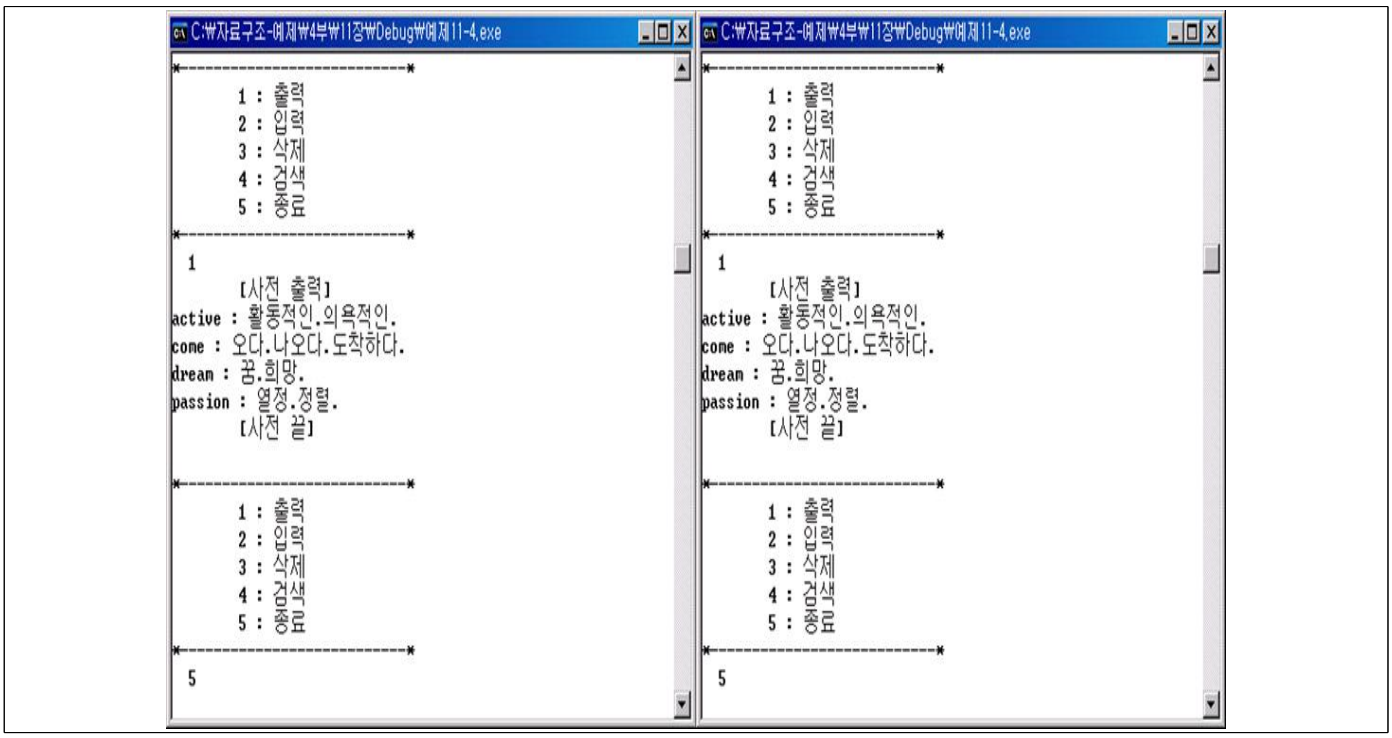
- 020~124행 이진 탐색 트리 연산 수행

```
164             temp=searchBST(root, e);
165             if(temp != NULL)
166                printf("\n단어 뜻 : %s", temp->key.mean);
167             else printf("\n사전에 없는 단어입니다.");
168             break;
169          }
170    }while((choice-'0') != 5);
171
172    getchar();
173 }
```

* 실행결과


C:₩자료구조-예제₩4부₩11장₩Debug₩예제11-4.exe
1 : 출력
2 : 입력
3 : 삭제
4 : 검색
5 : 종료
2
[단어 입력] 단어를 입력하세요 : come
[단어입력] 단어 뜻을 입력하세요 : 오다.나오다.도착하다.
[단어 입력] 단어를 입력하세요 : active
[단어입력] 단어 뜻을 입력하세요 : 활동적인.의욕적인.
C:₩자료구조-예제₩4부₩11장₩Debug₩예제11-4.exe
1 : 출력
2 : 입력
3 : 삭제
4 : 검색
5 : 종료
2
[단어 입력] 단어를 입력하세요 : boy
[단어입력] 단어 뜻을 입력하세요 : 젊은이.소년.
[단어 입력] 단어를 입력하세요 : passion
[단어입력] 단어 뜻을 입력하세요 : 열정.정렬.


C:₩자료구조-예제₩4부₩11장₩Debug₩예제11-4.exe
1 : 출력
2 : 입력
3 : 삭제
4 : 검색
5 : 종료
2
[단어 입력] 단어를 입력하세요 : boy
[단어입력] 단어 뜻을 입력하세요 : 소년.
이미 같은 단어가 있습니다!
1
[사전 출력]
active : 활동적인.의욕적인.
boy : 젊은이.소년.
come : 오다.나오다.도착하다.
passion : 열정.정렬.
[사전 끝]
C:₩자료구조-예제₩4부₩11장₩Debug₩예제11-4.exe
4
[단어 검색] 검색할 단어 : passion
찾은 단어 : passion
단어 뜻 : 열정.정렬.
3
[단어 삭제] 삭제할 단어 : boy



【학습정리】

1. 이진 검색은 가운데에 있는 항목을 킷값과 비교하여 킷값이 더 크면 오른쪽 부분을 검색하고 더 작으면 왼쪽 부분을 검색하는 방법을 반복 수행하는 분할 정복 기법을 이용한 검색 방법이다 이진 검색은 반드시 정렬되어 있는 자료에서의 검색에만 사용할 수 있다.

2. 이진 트리 검색은 루트 노드를 기준으로 왼쪽 서브 트리는 루트 노드보다 킷값이 작은 원소들로 구성하고 오른쪽 서브 트리는 루트 노드보다 킷값이 큰 원소들로 구성한 이진 탐색 트리를 이용한 검색 방법이다.