12 프로그래밍 프로젝트

01 **히스토그램은 n개의 키들로 이루어진 모임(collection)에서 각각의 키들이 얼마나 자주 등장하는지를 나타내는 도표이다. 히스토그램은 흔히 데이터의 분포를 결정하는데 사용된다. 예를 들면 시험에서의 점수의 분포, 영상에서의 명암값의 분포, 각 차종별 판매대수 등이다. 만약 킷값이 0에서 r사이의 정수이고 r이 비교적 작다면 쉽게 히스트그램을 얻을 수 있다. 즉 1차원 배열 h[i]를 사용하여 킷 값 i가 등장할 때마다 h[i]을 증가시켜주면 된다.**

```
for(i=0;i<n;i++) {
    key = list[i];
    h[key]++;
}
```

그러나 만약 킷 값의 범위가 매우 크거나 킷값이 정수가 아니라면(예를 들면 실수) 위의 방법은 사용하기가 불가능하다. 예를 들어 우리가 책안에 등장하는 단어들의 빈도를 측정한다고 가정하여 보자. 책안에 나타나는 단어들의 개수에 비하여 서로 다른 가능한 단어들이 매우 많다. 따라서 이러한 경우에는 등장하는 단어들을 정렬한 다음, 각 서로 다른 키들의 개수를 세어서 단어들의 등장 빈도를 계산할 수 있다. 만약 히프정렬을 이용한다면 $O(n\log_2 n)$시간안에 정렬이 가능하고 서로 다른 단어들의 개수를 세는 연산이 $O(n)$이므로 전반적인 시간 복잡도는 $O(n\log_2 n)$이 된다.
단어들의 빈도, 즉 히스토그램을 구하는 이러한 알고리즘은 만약 서로 다른 킷값들의 개수 m이 n에 비하여 비교적 작을 경우, 균형 트리를 사용한다면 더욱 향상될 수 있다. $O(n\log_2 m)$시간 안에 히스토그램을 구할 수 있다. 방법은 먼저 균형 이진 트리를 만들고 책안의 단어가 등장하면 균형 이진 트리에 삽입한다. 만약 이미 삽입된 단어이면 단어의 빈도를 나타내는 노드안의 변수를 1 증가한다. 만약 아직 삽입이 되지 않은 단어라면 단어를 균형 이진 트리안에 삽입한다. 이것을 위하여 insert_visit(char *key) 함수를 작성한다. insert_visit 함수는 삽입과 방문의 2가지 기능을 수행한다. 즉 만약 탐색 트리안에 key가 없으면 삽입 기능을 수행한다. 만약 키가 존재하면 키와 관련된 빈도를 나타내는 변수를 1 증가한다. 이것은 간단히 원래의 삽입 함수에서 중복된 키를 처리하는 다음의 코드를

```
else{
        fprintf(stderr, "중복된 키 에러\n");
        exit(1);
    }
```

에서

```
else{
        (*root)->freq++;
        return(NULL);
    }
```

로 변경하면 된다. 이상 설명한 바와 같이 책안의 단어들이 등장하는 빈도를 측정하는 프로그램을 본문의 소스를 참조하여 AVL 트리를 이용하여 구현하여 보라.

01 **히스토그램은 n개의 키들로 이루어진 모임(collection)에서 각각의 키들이 얼마나 자주 등장하는지를 나타내는 도표이다. 히스토그램은 흔히 데이터의 분포를 결정하는데 사용된다. 예를 들면 시험에서의 점수의 분포, 영상에서의 명암값의 분포, 각 차종별 판매대수 등이다. 만약 킷값이 0에서 r사이의 정수이고 r이 비교적 작다면 쉽게 히스트그램을 얻을 수 있다. 즉 1차원 배열 h[i]를 사용하여 킷값 i가 등장할 때마다 h[i]을 증가시켜주면 된다.**

```
for(i=0;i<n;i++) {
    key = list[i];
    h[key]++;
}
```

그러나 만약 킷값의 범위가 매우 크거나 킷값이 정수가 아니라면(예를 들면 실수) 위의 방법은 사용하기가 불가능하다. 예를 들어 우리가 책안에 등장하는 단어들의 빈도를 측정한다고 가정하여 보자. 책안에 나타나는 단어들의 개수에 비하여 서로 다른 가능한 단어들이 매우 많다. 따라서 이러한 경우에는 등장하는 단어들을 정렬한 다음, 각 서로 다른 키들의 개수를 세어서 단어들의 등장빈도를 계산할 수 있다. 만약 히프정렬을 이용한다면 $O(n\log_2 n)$시간안에 정렬이 가능하고 서로 다른 단어들의 개수를 세는 연산이 $O(n)$이므로 전반적인 시간복잡도는 $O(n\log_2 n)$이 된다.
단어들의 빈도, 즉 히스토그램을 구하는 이러한 알고리즘은 만약 서로 다른 킷값들의 개수 m이 n에 비하여 비교적 작을 경우, 균형트리를 사용한다면 더욱 향상될 수 있다. $O(n\log_2 m)$시간안에 히스토그램을 구할 수 있다. 방법은 먼저 균형이진트리를 만들고 책안의 단어가 등장하면 균형이진트리에 삽입한다. 만약 이미 삽입된 단어이면 단어의 빈도를 나타내는 노드안의 변수를 1 증가한다. 만약 아직 삽입이 되지 않은 단어라면 단어를 균형트리안에 삽입한다. 이것을 위하여 insert_visit(char *key) 함수를 작성한다. insert_visit 함수는 삽입과 방문의 2가지 기능을 수행한다. 즉 만약 탐색트리안에 key가 없으면 삽입 기능을 수행한다. 만약 키가 존재하면 키와 관련된 빈도를 나타내는 변수를 1 증가한다. 이것은 간단히 원래의 삽입함수에서 중복된 키를 처리하는 다음의 코드를

```
else{
        fprintf(stderr, "중복된 키 에러\n");
        exit(1);
    }
```

에서

```
else{
        (*root)->freq++;
        return(NULL);
    }
```

로 변경하면 된다. 이상 설명한 바와 같이 책안의 단어들이 등장하는 빈도를 측정하는 프로그램을 AVL 트리를 이용하여 구현하여 보라.