이진탐색트리

2018818716 컴퓨터공학부 김용현

isbalanced()

```
1   bool isbalanced() {
2       return isbalanced(root);
3   }
4   bool isbalanced(binarynode* node) {
5       /* Base Case */
6       if (node == NULL)
7           return true;
8       /* Inductive Step */
9       else
10          return
11          abs(getheight(node->getleft()) -
12              getheight(node->getright())) < 2 &&
13          isbalanced(node->getleft()) &&
14          isbalanced(node->getright());
15  }
```

[코드 : isbalanced()]

1. return isempty() ? true : isbalanced(root); 에서 isempty() 메소드는 결국 root == NULL 여부를 확인하는 것이므로, return isbalanced(root); 와 그 구조가 동일하다.
2. 노드의 개수가 0개일때도 balanced가 true가 된다 생각하고 코드를 작성하였다.

[코드 : isbalanced(binarynode* node)]

1. 재귀 방식으로 문제를 해결하여, 재귀의 탈출 조건이 되는 Base Case와, 재귀의 귀납 단계가 되는 Inductive Step으로 구분하였다.
2. [Base Case] : node == NULL 인 경우, true 값을 반환한다.
3. [Inductive Step] : ① 좌 · 우측 서브트리의 높이차가 2미만인지 확인한 후, ② 좌측 서브트리에 대해 isbalanced여부를 확인하고, ③ 우측 서브트리에 대해서도 isbalanced여부를 확인한다. ④ 마지막으로 &&연산을 통해 true · false 여부를 반환한다.

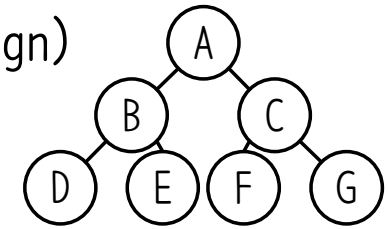
```
1   bool isbalanced() {
2       return isbalanced(root);
3   }
4   bool isbalanced(binarynode* node) {
5       /* Base Case */
6       if (node == NULL)
7           return true;
8       /* Inductive Step */
9       else
10          return
11          abs(getheight(node->getleft()) -
12              getheight(node->getright())) < 2 &&
13          isbalanced(node->getleft()) &&
14          isbalanced(node->getright());
15  }
```

[단점]

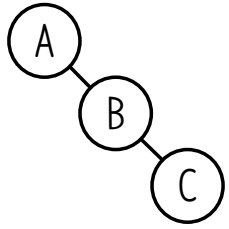
1. Tree 상의 Node의 개수를 n이라 하였을 때, getheight() 함수의 시간복잡도를 알아보자.

```
1   int getheight() {
2       return isempty() ? 0 : getheight(root);
3   }
4   int getheight(binarynode* node) {
5       if (node == NULL) return 0;
6       int hLeft = getheight(node->getleft());
7       int hRight = getheight(node->getright());
8       return (hLeft > hRight) ? hLeft + 1 : hRight + 1;
9   }
```

#. Balanced(Average) Case : O(logn)



#. Unbalanced(Worst) Case : O(n)



```
1   bool isbalanced() {
2       return isbalanced(root);
3   }
4   bool isbalanced(binarynode* node) {
5       /* Base Case */
6       if (node == NULL)
7           return true;
8       /* Inductive Step */
9       else
10          return
11          abs(getheight(node->getleft()) -
12              getheight(node->getright())) < 2 &&
13          isbalanced(node->getleft()) &&
14          isbalanced(node->getright());
15  }
```

[단점]

1. 이때, 왼쪽의 isbalanced() 함수의 경우, 루트를 제외한 모든 Node에 대해 getheight() 함수를 반복 호출하게 된다. 상위 노드들이 하위 노드들을 전부 한번씩 호출하기 때문에 O(n)의 시간복잡도가 곱해지게 된다.

[isbalanced() 함수의 시간복잡도]

1. Average Case : O(nlogn)
2. Worst Case : O(n^2)

[더 나은 방법?]

1. getheight()에서 트리의 높이를 구할 때, 우리는 트리의 Balanced 여부를 생각해 볼 수 있다.
2. 만약, 왼쪽 서브트리의 높이가 3, 오른쪽 서브트리의 높이가 1이라면, 높이차가 2가 되어 Unbalanced Tree라는 사실을 바로 알 수 있다.
3. 위 방법을 이용한 함수의 시간복잡도는 O(n)이다.

```
1   bool isbalanced_fast() {
2       bool isbalanced = true;
3       isbalanced_fast(root, isbalanced);
4
5       return isbalanced;
6   }
7   int isbalanced_fast(binarynode* node, bool& isbalanced) {
8       /* Base Case */
9       if (node == NULL || !isbalanced)
10          return 0;
11      /* Inductive Step */
12      int left_height = isbalanced_fast(node->getleft(), isbalanced);
13      int right_height = isbalanced_fast(node->getright(), isbalanced);
14
15      if (abs(left_height - right_height) > 1)
16          isbalanced = false;
17
18      return max(left_height, right_height) + 1;
19  }
```

[코드 : isbalanced_fast()]

1. 주어진 이진 트리가 높이 균형이 맞는지 확인하는 주요 함수이다.
2. Bool형 변수 isbalanced에 함수의 균형여부에 대한 정보를 저장한다.

[코드 : isbalanced_fast(binarynode*, bool&)]

1. 주어진 이진 트리가 높이 균형이 맞는지 확인하는 재귀함수 이다.
2. getheight() 함수의 동작방식과 거의 유사하다. (15행, 16행만 추가되었다.)
3. Base Case : node가 NULL이거나, 트리의 균형이 맞지 않으면 0을 반환한다.
4. Inductive Step : 왼쪽 서브트리와 오른쪽 서브트리의 높이를 재귀적으로 구한다. 이때 만약 왼쪽과 오른쪽 서브트리의 높이차가 1을 초과하는 경우, isbalanced를 false로 변경시켜 준다.

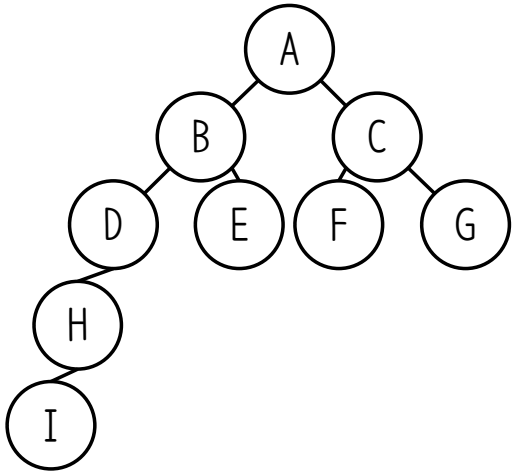
```cpp
bool isbalanced_fast() {
    bool isbalanced = true;
    isbalanced_fast(root, isbalanced);

    return isbalanced;
}
int isbalanced_fast(binarynode* node, bool& isbalanced) {
    /* Base Case */
    if (node == NULL || !isbalanced)
        return 0;
    /* Inductive Step */
    int left_height = isbalanced_fast(node->getleft(), isbalanced);
    int right_height = isbalanced_fast(node->getright(), isbalanced);

    if (abs(left_height - right_height) > 1)
        isbalanced = false;

    return max(left_height, right_height) + 1;
}
```

[예제1]



[I기준]
좌측 서브트리 높이 : 0
우측 서브트리 높이 : 0
0 - 0 = 0 이므로, true
[H기준]
좌측 서브트리 높이 : 1
우측 서브트리 높이 : 0
1 - 0 = 1 이므로, true
[D기준]
좌측 서브트리 높이 : 2
우측 서브트리 높이 : 0
2 - 0 = 2 이므로, false
…….
이후 탐색 종료.

1번 문제

```cpp
int main(void) {
    /* Faster */
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    /* Init */
    binarysearchtree bst;

    /* Input */
    int n; cin >> n;

    /* Loop */
    while (n--) {
        /* Input */
        char c; cin >> c;

        if (c == 'I') {
            int x; cin >> x;
            bst.insert(new binarynode(x));
        }
        else if (c == 'D') {
            int x; cin >> x;
            bst.remove(x);
        }
    }

    /* Output */
    if (bst.isbalanced_fast())
        cout << "Balanced";
    else
        cout << "Unbalanced";

    /* Return */
    return 0;
}
```

[문제1]

/* Header*/

1. binarynode, binarytree, binarysearchtree에 해당하는 클래스는 교과서와 동일하게 진행하였다.

/* Faster */

1. 싱글 쓰레드 환경에서, printf와 scanf함수가 이용하는 stdio.h버퍼와, cin과 cout이 이용하는 iostream의 버퍼의 동기화를 해제함으로써, 입출력 속도를 빠르게 하였다.
2. ios::sync_with_stdio(0)은 stdio.h와 iostream의 동기화를 해제하겠다는 뜻이다.
3. cin.tie(0) 및 cout.tie(0)의 경우, cin과 cout은 서로 연결되어 있어 cin을 쓰면 출력 버퍼를 비우고 입력이 발생한다. 이러한 flush(버퍼를 비우는 과정) 과정도 시간이 소요되기 때문에, cin과 cout의 상호 연결을 끊어주기 위하여 해당 코드를 삽입하는 것이다.

/* Init */

1. Binarysearchtree를 선언한다.

/* Input */

1. n값을 입력받는다.

/* Loop */

1. Loop을 돌며, Input으로 들어오는 값들을 입력받는다. I인 경우 binarysearchtree에 해당하는 node를 추가하고, D인 경우 삭제한다.

/* Output */

1. tree가 Balanced인 경우, “Balanced”를 출력하고, 아닌 경우 "Unbalanced를 출력한다.

2번 문제

```cpp
vector<int> list;

void targetsum(binarynode* node, int sum = 0) {
    /* Base Case */
    if (node == NULL)
        return;
    /* Inductive Step */
    sum += node->getdata();
    list.push_back(sum);

    targetsum(node->getleft(), sum);
    targetsum(node->getright(), sum);
}

int main(void) {
    /* Faster */
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    /* Init */
    binarysearchtree bst;
    int count = 0;

    /* Input */
    int n, tg; cin >> n >> tg;

    /* Loop */
    while (n--) {
        /* Input */
        char c; cin >> c;

        if (c == 'I') {
            int x; cin >> x;
            bst.insert(new binarynode(x));
        }
        else if (c == 'D') {
            int x; cin >> x;
            bst.remove(x);
        }
    }

    /* Targetsum */
    targetsum(bst.getroot());

    /* Output */
    for (auto i : list)
        if (i == tg)
            count++;

    cout << count;

    /* Return */
    return 0;
}
```

[문제2]

/* Header*/

1. binarynode, binarytree, binarysearchtree에 해당하는 클래스는 교과서와 동일하게 진행하였다.

/* List */

1. 각 노드 별 누적합을 저장할 list를 vector로 선언하였다.

/* Targetsum */

1. 각 노드 별 누적합을 구한다. ∴ 각 노드의 개수만큼 누적합의 개수가 나오게 된다.

/* Faster */

1. 싱글 쓰레드 환경에서, printf와 scanf함수가 이용하는 stdio.h버퍼와, cin과 cout이 이용하는 iostream의 버퍼의 동기화를 해제함으로써, 입출력 속도를 빠르게 하였다.
2. ios::sync_with_stdio(0)은 stdio.h와 iostream의 동기화를 해제하겠다는 뜻이다.
3. cin.tie(0) 및 cout.tie(0)의 경우, cin과 cout은 서로 연결되어 있어 cin을 쓰면 출력 버퍼를 비우고 입력이 발생한다. 이러한 flush(버퍼를 비우는 과정) 과정도 시간이 소요되기 때문에, cin과 cout의 상호 연결을 끊어주기 위하여 해당 코드를 삽입하는 것이다.

/* Init */

1. Binarysearchtree를 선언한다.
2. tg에 해당하는 경로의 개수를 세기 위한 변수 count를 0으로 초기화 한다.

/* Input */

1. n값과 tg값을 입력 스트림으로 부터 입력받는다.

/* Loop */

1. Loop을 돌며, Input으로 들어오는 값들을 입력받는다. I인 경우 binarysearchtree에 해당하는 node를 추가하고, D인 경우 삭제한다.

/* Targetsum */

1. targetsum() 함수를 통해 list에 값을 추가한다.

/* Output */

1. list를 순회하며 i == tg인 경우, count를 ++ 해준다.
2. count를 출력한다.

3번 문제

```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main(void) {
    /* Faster */
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    /* Init */
    map<string, string> ke;
    map<string, string> ek;

    /* Input */
    while (true) {
        /* Input */
        char c; cin >> c;

        /* I */
        if (c == 'i') {
            cin.ignore();
            string kr; getline(cin, kr);
            string eg; getline(cin, eg);

            ke.insert(make_pair(kr, eg));
            ek.insert(make_pair(eg, kr));
        }
        /* K */
        else if (c == 'k') {
            cin.ignore();
            string kr; getline(cin, kr);

            if (ke.find(kr) == ke.end())
                cout << kr << " UNKNOWN ENTRY" << '\n';
            else
                cout << ke.find(kr)->first << ' ' << ke.find(kr)->second << '\n';
        }
        /* E */
        else if (c == 'e') {
            cin.ignore();
            string eg; getline(cin, eg);

            if (ek.find(eg) == ek.end())
                cout << eg << " UNKNOWN ENTRY" << '\n';
            else
                cout << ek.find(eg)->first << ' ' << ek.find(eg)->second << '\n';
        }
        /* P */
        else if (c == 'p') {
            cout << "K-E dictionary:\n";
            for (auto i : ke)
                cout << i.first << ' ' << i.second << '\n';

            cout << "E-K dictionary:\n";
            for (auto i : ek)
                cout << i.first << ' ' << i.second << '\n';
        }
        /* Q */
        else if (c == 'q')
            return 0;
    }
}
```

[문제3]

/* Header*/

1. STL Map과 String을 사용하기 위한 헤더파일을 추가하였다.

/* Faster */

1. 싱글 쓰레드 환경에서, printf와 scanf함수가 이용하는 stdio.h버퍼와, cin과 cout이 이용하는 iostream의 버퍼의 동기화를 해제함으로써, 입출력 속도를 빠르게 하였다.
2. ios::sync_with_stdio(0)은 stdio.h와 iostream의 동기화를 해제하겠다는 뜻이다.
3. cin.tie(0) 및 cout.tie(0)의 경우, cin과 cout은 서로 연결되어 있어 cin을 쓰면 출력 버퍼를 비우고 입력이 발생한다. 이러한 flush(버퍼를 비우는 과정) 과정도 시간이 소요되기 때문에, cin과 cout의 상호 연결을 끊어주기 위하여 해당 코드를 삽입하는 것이다.

/* Init */

1. 한영사전에 해당하는 ke와 영한사전에 해당하는 ek를 선언한다.

/* I */

1. 입력값을 받아 ke와 ek에 각각 삽입한다.

/* K */

1. 입력으로 들어온 값에 해당하는 값을 한영사전에서 찾아 출력값에 맞게 출력한다.

/* E */

1. 입력으로 들어온 값에 해당하는 값을 영한사전에서 찾아 출력값에 맞게 출력한다.

/* P */

1. ke와 ek를 순회하며 모든 내용을 출력한다. 이때, map은 자동적으로 오름차순 정렬되므로, 문제의 조건에 맞게 값이 출력된다.

감사합니다!