Cewe

Mata Kuliah	Struktur Data
Kelas	C
Tahun Ajaran	2021/2022
Praktikum Modul	2

Oleh: Luthfiyyah Hanifah Amari (5025201090)
Username Hackerrank: Luthfiyyah_hani1

Description

ada seorang cewe yang meminta kamu untuk mengerjakan hal yang dia suruh. dia memintamu untuk membuat sebuah program memakai binary tree. dia menyebutkan beberapa angka. dia mengatakan “masukin” kemudian menyebutkan angkanya. Lalu, kamu diminta untuk memasukkan angka tersebut ke dalam binary tree (sesuai dengan aturan binary tree).

Terkadang, dia ingin mengetahui angka berapa yang ada di index tertentu berdasarkan urutan ascending (dari kecil ke besar). kalau dia ingin menanyakan hal itu, dia mengatakan “berapa” kemudian menyebutkan suatu index. index dimulai dari 1. tugasmu mencari tau angka apa yang ada di index tersebut. dia ingin kamu mencari indexnya menggunakan binary search tree.

Terkadang juga, dia ingin mengetahui parent dari suatu angka pada node di binary tree tersebut. kalau dia ingin menanyatakn hal itu, dia mengatakan “anaksiapa” lalu diikuti oleh angka yang ingin ditanyakan siapa parent-nya.

Turutilah kemauan cewe ini. terima kasih

Input Format

baris pertama adalah N, yang jumlah perintah dari si cewe.

N baris selanjutnya ialah omongan (perintah) si cewe.

Apabila `masukin` , maka input selajutnya berupa x yang harus insert ke binary tree

Apabila `berapa` , maka input selanjutnya adalah i yang merupakan index.

Apabila `anaksiapa` , maka input selanjutnya adalah y, yakni angka yang ingin dicari tau siapa parent-nya

Constraints

- $N < 100$
- $1 < x, y < 10000$
- $1 < i < 100$

Output Format

keluarkan output setiap perintah `berapa` dan `anaksiapa` .

apabila inputnya `berapa i` , maka outpunya ialah angka yang berada di index i

apabila inputnya `anaksiapa y` , maka outputnya adalah angka dari node parent y. apabila y tidak ditemukan di binary tree atau y merupakan root, maka keluarkan `ini anak siapa dah`

output tersebut dipisahkan dengan newline.

Sample Input 0

```
7
masukin 100
masukin 90
anaksiapa 90
anaksiapa 80
masukin 120
masukin 110
berapa 100
```

Sample Output 0

```
100
ini anak siapa dah
2
```

Sample Input 1

```
9
masukin 100
masukin 90
masukin 120
anaksiapa 85
anaksiapa 120
anaksiapa 100
masukin 10
berapa 10
berapa 100
```

Sample Output 1

```
ini anak siapa dah
100
ini anak siapa dah
1
3
```

Source Code

```
/**
 * Implementasi Binary Search Tree (ADT: BST)
 * yakni BST yang tidak menyimpan key duplikat (unique key)
 *
 * Dibuat dan ditulis oleh Bayu Laksana
 * -- tanggal 29 Februrari 2019
 * Struktur Data 2020
 *
 * Implementasi untuk Bahasa C
 *
```

```
 *
 * Dimodifikasi oleh luthfiyyah hanifah
 * -- Maret 2022
 * Untuk penyelesaian soal praktikum "cewe" pada modul 2
 * Struktur data 2022
 *
 */

#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

/**
 * Node structure and
 * uniqueBST structure
 */

typedef struct bstnode_t {
    unsigned long long key;
    struct bstnode_t \
        *left, *right;
    int anak_kiri;
} BSTNode;

typedef struct bst_t {
    BSTNode *_root;
    unsigned int _size;
} BST;


/**
 * !!! WARNING UTILITY FUNCTION !!!
 * Recognized by prefix "__bst__"
 * ----------------------------------------------
 * Note that you better never access these functions,
 * unless you need to modify or you know how these functions work.
 */

BSTNode* __bst__createNode(unsigned long long value) {
    BSTNode *newNode = (BSTNode*) malloc(sizeof(BSTNode));
    newNode->key = value;
    newNode->left = newNode->right = NULL;
    newNode->anak_kiri = 0;
    return newNode;
}

BSTNode* __bst__insert(BSTNode *root, unsigned long long value) {
    if (root == NULL)
        return __bst__createNode(value);

    if (value < root->key)
    {
      // jika newnode ke kiri, berarti jumlah anak kiri dari node yg sekarang bertambah
      root->anak_kiri++;
      root->left = __bst__insert(root->left, value);
  }
    else if (value > root->key)
    {
      root->right = __bst__insert(root->right, value);
  }

    return root;
}

BSTNode* __bst__search__parent(BSTNode *root, int value) {
  BSTNode *temp = NULL;
    while (root != NULL) {

        if (value < root->key)
        {
          temp = root;
          root = root->left;
    }
        else if (value > root->key)
        {
          temp = root;
          root = root->right;
    }
        else
            return temp;
```

```
    }
    return root;
}

int parent;
// DIMODIFIKASI DARI FUNGSI SEARCH
BSTNode* __bst__search(BSTNode *root, unsigned long long value) {

  // untuk mengetahui suatu node berada di index ke berapa (secara descending), perlu menghitung jumlah child kanan dan kalo dia merupakan
  parent = 0;
  while (root != NULL) {
        if (value < root->key)
        {
          root = root->left;
    }
        else if (value > root->key)
        {
          parent += root->anak_kiri + 1;
          root = root->right;
    }
        else
            return root;
    }
    return root;
}

BSTNode* __bst__findMinNode(BSTNode *node) {
    BSTNode *currNode = node;
    while (currNode && currNode->left != NULL)
        currNode = currNode->left;

    return currNode;
}

BSTNode* __bst__remove(BSTNode *root, unsigned long long value) {
    if (root == NULL) return NULL;

    if (value > root->key)
        root->right = __bst__remove(root->right, value);
    else if (value < root->key)
        root->left = __bst__remove(root->left, value);
    else {

        if (root->left == NULL) {
            BSTNode *rightChild = root->right;
            free(root);
            return rightChild;
        }
        else if (root->right == NULL) {
            BSTNode *leftChild = root->left;
            free(root);
            return leftChild;
        }

        BSTNode *temp = __bst__findMinNode(root->right);
        root->key     = temp->key;
        root->right   = __bst__remove(root->right, temp->key);
    }
    return root;
}


/**
 * PRIMARY FUNCTION
 * ----------------------------
 * Accessible and safe to use.
 */

void bst_init(BST *bst) {
    bst->_root = NULL;
    bst->_size = 0u;
}

bool bst_isEmpty(BST *bst) {
    return bst->_root == NULL;
}


// DIMODIFIKASI
int bst_find(BST *bst, unsigned long long value) {
    BSTNode *temp = __bst__search(bst->_root, value);
```

```
    if (temp == NULL)
        return 0;

    if (temp->key == value)
    {
//      printf("anak kanan = %d\n", temp->anak_kanan);
      return temp->anak_kiri + parent + 1;
  }
    else
        return 0;
}

void bst_insert(BST *bst, unsigned long long value) {
    if (!bst_find(bst, value)) {
        bst->_root = __bst__insert(bst->_root, value);
        bst->_size++;
    }
}

void bst_remove(BST *bst, unsigned long long value) {
    if (bst_find(bst, value)) {
        bst->_root = __bst__remove(bst->_root, value);
        bst->_size--;
    }
}

int bst_find_parent(BST *bst, int value) {
    BSTNode *temp = __bst__search__parent(bst->_root, value);
    if (temp == NULL)
        return 0;
    else return temp->key;
}


int main()
{
    BST set;
    bst_init(&set);

    unsigned long long i, q, data;
  char command[100];
  scanf("%llu", &q);
  for(i=0; i<q; i++)
  {
    scanf("%s %llu", command, &data);

    if(strcmp(command, "masukin") == 0)
    {
      bst_insert(&set, data);
    }
    else if(strcmp(command, "berapa") == 0)
    {
      unsigned long long hasil = bst_find(&set, data);
      if(hasil != 0)
      {
        printf("%llu\n", hasil);
      }
    }
    else if(strcmp(command, "anaksiapa") == 0)
    {
      int temp = bst_find_parent(&set, data);
      if(temp)
        printf("%d\n", temp);
      else
        printf("ini anak siapa dah\n");

    }

  }

    return 0;
}
```