Bantu Urutin Dong

Input Format

Baris pertama berisi sebuah bilangan N yang merupakan banyak query.
Baris selanjutnya berisi N buah bilangan dimana Ai merupakan nilai dari masing-masing node.
Node pertama dianggap ganjil, node kedua adalah genap, dan seterusnya.
Pada soal ini berfokus pada nomor node bukan value dari node-nya.
Urutan pada pengelompokkan ganjil genap harus tetap seperti pada input yang diberikan.

Output Format

N bilangan dan Ai dimana Ai merupakan nilai dari tiap node (banyaknya node ditentukan oleh N).

```
#include <iostream>
using namespace std;
int main() {
   int n, i, v, genap[5000];
   cin >> n;
   for(i = 1; i <= n; i++){
      cin >> v;
      if( (i % 2) == 0 ){
         genap[i] = v;
      }
      else{
         cout << v << " ";
      }
   }
   for(i = 1; i<= n; i++){
      if(genap[i] != 0){
         cout << genap[i] << " ";
      }
      else{
         continue;
      }
   }
   return 0;
}
```

Penjelasan

n = banyak querry (baris pertama), v & i = nilai

deklarasi nilai varibel yang diinputkan, apabila suatu nilai (i) dalam array berada pada posisi awal dengan dipersenkan dengan 2 bernilai nol atau genap maka akan dicetak awal dan dianggap ganjil dan diruntut kembail untuk mencari posisi genap lainnya, apabila node bernilai sama dengan 1 (0++) maka nilai akan dianggap genap dan diruntut selanjutnya, posisi nilai akan berupa barisan ganjil lalu genap

Urutin oi

Input Format

Diberikan sebuah array yang tidak urut. Tugas Anda menghapus duplikat isi array dan urutkan dari yang terkecil. Pada soal ini input akan berhenti apabila menemui angka nol (0).

Constraints

Bilangan <= 100 Jumlah data tidak terbatas

Output Format

Cetak array yang sudah urut dan tidak ada duplikasi.

```cpp
#include <iostream>
#include <list>
using namespace std;


int main() {
    int x, y, start = 0, flag = 0;
    list<int> arr[100];
    list<int>::iterator it;

    for(x = 0; x > -1; x++){
        cin >> y;
        if(y == 0){
            break;
        }
        for(it = arr[start].begin(); it != arr[start].end(); it++){
            if(y == *it){
                flag = 1;
            }
        }
        if(flag == 0){
            arr[start].push_back(y);
        }
        flag = 0;
    }

    arr[start].sort();

    for(it = arr[start].begin(); it != arr[start].end(); it++){
        cout << *it << endl;
    }


    return 0;
}
```

Penjelasan

x,y = besaran nilai

besaaran nilai dalam array maksimal 100 dengan awalan 0 dan tanpa Batasan Panjang dari array. Array 0 sebagai awalan akan diisi nilai yang paling kecil menuju nilai yang paling besar, redundansi nilai yang terjadi dalam array akan dieliminasi. Baris hasil akan berupa urutan nilai terkecil menuju terbesar tanpa adanya pengulangan nilai yang sama

Cek Komposisi Lagi Skuy

Program ini akan menerima inputan sebagai berikut :

1. Baris pertama adalah A B. Untuk A adalah menyatakan banyak input test case dan B adalah angka yang menyatakan komposisi sehat.
2. Untuk baris selanjutnya terdiri dari X Y. Untuk nilai X sendiri adalah sebuah string yang menyatakan sebuah makanan sedangkan Y adalah angka yang menyatakan komposisi dari makanan tersebut.
3. Setiap kali menerima input "GASS" maka program akan mencetak sebuah angka rata-rata yang melambangkan komposisi saat ini.
4. Saat test case telah habis, maka kamu diminta untuk menyatakan apakah jumlah rata-rata yang ada memenuhi rentang sehat yang ada. Rentang sehat tersebut adalah B - 50% B hingga B + 50% B termasuk (inclusive). Saat rata-ratanya memenuhi maka print “AMAN”, jika tidak maka print “LOH”
5. Nilai rata-rata yang ditampilkan adalah sebanyak 2 angka dibelakang koma.

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main(){
    int a, b, count = 0, dvd = 0;
    double tot = 0, res = 0, end = 0, y, sum;
    string x;
    cin >> a;
    cin >> b;
    for(int i = 0; i < a; i++){
        cin >> x;
        if(x != "GASS"){
            cin >> y;
            tot = tot + y;
            count++;
        }
        else{
            res = tot/count;
            cout << fixed << setprecision(2) << res << endl;
            end = end + res;
            dvd++;
        }
    }
    sum = end/dvd;
```

```
        cout << sum << " ";
        if(0.5*b <= sum && 1.5*b >= sum){
            cout << "AMAN";
        }
        else{
            cout << "LOH";
        }
}
```

Penjelasan

Terdapat string dari nama makanan disertai besaran nilai 2 angka dibelakang koma menggunakkan tipe data double, A merupakan banyaknya nilai input sedangkan B adalah indeks Kesehatan pangan. Program akan jalan apabila menemui string gass dalam input, apabila kata gas terdapat ditengah input maka akan terjadi pengulangan perhitungan disertai penambahan total string yang dimasukkan. Rentang makanan berdasar soal akan berkisar antara 0.5xB sampai 1.5xB akan menghasilkan analisis “aman” apabila diluar rentang akan menghasilkan “loh”

Cinta Segitiga

Input Format

- Baris pertama terdiri atas 1 integer - jumlah komputer
- Baris kedua terdiri atas integer yang berarti komputer ke- menyukai komputer .

Output Format

- Print `YES` jika terdapat cinta segitiga
- Print `NO` jika tidak ada cinta segitiga

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    int v, a[3001];
    cin >> v;
    for(int i = 0; i < v; ++i) {
        cin >> a[i];
        --a[i];
    }

    for(int i = 0; i < v; ++i) {
        if(a[i] == a[a[a[a[i]]]]) {
            cout << "YES" << endl;
            return 0;
        }
```

```
    }

    cout << "NO" << endl;

    return 0;
}
```

Penjelasan

Deklarasi baris pertama berisi integer atas banyaknya computer yang akan diinputkan, sedangkan computer (i) akan diseleksi melalui array bersusun, apabila dalamnya terdapat nilai atas identitas nilai computer lain sebanyak 3 maka terjadi cinta segitiga, apabila tidak maka akan dicetak “no”

Toko ARA

Input Format

- Baris pertama berisi sebuah integer *n* yang menyatakan durasi dari promosi (periode promosi)
- *N* baris selanjutnya berisi rangkaian integer yang dipisahkan dengan spasi. Integer pertama *a* menggambarkan jumlah receipt yang diterima pada hari tersebut. Kemudian, *a* integer berikutnya menggambarkan jumlah transaksi setiap receiptnya.

Constraints

- 1 <= N <= 1000
- 0 <= a <= 10000

Output Format

Hanya ada satu baris yang berisi satu integer yang menggambarkan total prize yang diberikan oleh Toko ARA.

```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
    int x;
    cin >> x;
    priority_queue<int> high;
    priority_queue<int, vector<int>, greater<int> > low;
```

```
    int total_prize = 0;
    for (int i = 0; i < x; i++)
    {
        int a;
        cin >> a;
        for (int b = 0; b < a; b++)
        {
            int transaction;
            cin >> transaction;
            high.push(transaction);
            low.push(transaction);
        }
        total_prize += high.top() - low.top();
        high.pop();
        low.pop();
    }
    cout << total_prize << endl;
    return 0;
}
```

Penjelasan

Program canggih

Input Format

Baris pertama adalah N, yaitu jumlah kueri yang akan dilakukan. N baris selanjutnya adalah perintah pada program

Terdapat beberapa Kueri pada program ini, yaitu :

- insert x, dengan x adalah nilai yang ingin dimasukkan ke AVL Tree
- delete x, dengan x adalah nilai yang ingin dihapus dari AVL Tree
- inOrder, menampilkan data tree secara inOrder
- preOrder, menampilkan data tree secara preOrder
- postOrder, menampilkan data tree secara postOrder

Constraints

1 < N < 1000 1 < x < 1000

Output Format

Output dari program adalah log dari setiap rotasi yang dilakukan pada saat insert atau delete node tertentu.

Pada perintah inOrder, preOrder, dan postOrder output program adalah data tree yang ditampilkan secara inOrder, preOrder, atau postOrder

Note : Gunakan \t untuk membuat tab

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
using namespace std;

int count = 0;

struct AVLNode {
    int data;
    AVLNode *left, *right;
    int height;
};

struct AVL
{
private:
    AVLNode *_root;
    unsigned _size;

    AVLNode* _avl_createNode(int value) {
        AVLNode *newNode = (AVLNode*) malloc(sizeof(AVLNode));
        newNode->data = value;
        newNode->height =1;
        newNode->left = newNode->right = NULL;
        return newNode;
    }

    AVLNode* _search(AVLNode *root, int value) {
        while (root != NULL) {
            if (value < root->data)
                root = root->left;
            else if (value > root->data)
                root = root->right;
            else
                return root;
        }
        return root;
    }
```

```
    int _getHeight(AVLNode* node){
        if(node==NULL)
            return 0;
        return node->height;
    }

    int _max(int a,int b){
        return (a > b)? a : b;
    }

    AVLNode* _rightRotate(AVLNode* pivotNode){
        cout << "\tDilakukan rotasi kanan dengan pivot node " << pivotNode-
>data << ")" << endl;
        AVLNode* newParrent=pivotNode->left;
        pivotNode->left=newParrent->right;
        newParrent->right=pivotNode;

        pivotNode->height=_max(_getHeight(pivotNode->left),
                        _getHeight(pivotNode->right))+1;
        newParrent->height=_max(_getHeight(newParrent->left),
                        _getHeight(newParrent->right))+1;

        return newParrent;
    }

    AVLNode* _leftRotate(AVLNode* pivotNode) {
        cout << "\tDilakukan rotasi kiri dengan pivot node " << pivotNode-
>data << ")" << endl;
        AVLNode* newParrent=pivotNode->right;
        pivotNode->right=newParrent->left;
        newParrent->left=pivotNode;

        pivotNode->height=_max(_getHeight(pivotNode->left),
                        _getHeight(pivotNode->right))+1;
        newParrent->height=_max(_getHeight(newParrent->left),
                        _getHeight(newParrent->right))+1;

        return newParrent;
    }

    AVLNode* _leftCaseRotate(AVLNode* node){

        return _rightRotate(node);
    }

    AVLNode* _rightCaseRotate(AVLNode* node){
```

```
        return _leftRotate(node);
    }

    AVLNode* _leftRightCaseRotate(AVLNode* node){
        node->left=_leftRotate(node->left);
        return _rightRotate(node);
    }

    AVLNode* _rightLeftCaseRotate(AVLNode* node){
        node->right=_rightRotate(node->right);
        return _leftRotate(node);
    }

    int _getBalanceFactor(AVLNode* node){
        if(node==NULL)
            return 0;
        return _getHeight(node->left)-_getHeight(node->right);
    }

    AVLNode* _insert_AVL(AVLNode* node,int value) {

        if(node==NULL)
            return _avl_createNode(value);
        if(value < node->data)
            node->left = _insert_AVL(node->left,value);
        else if(value > node->data)
            node->right = _insert_AVL(node->right,value);

        node->height= 1 + _max(_getHeight(node->left),_getHeight(node-
>right));

        int balanceFactor=_getBalanceFactor(node);
        // if(_size >= 2 && balanceFactor != 0){
        //     cout << balanceFactor;
        // }

        if(balanceFactor > 1 && value < node->left->data){
            cout << "Ketika insert node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF > 1 dan subtree kiri > subtree kanan)" << endl;
            return _leftCaseRotate(node);
        }
        if(balanceFactor > 1 && value > node->left->data){
            cout << "Ketika insert node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF > 1 dan subtree kiri < subtree kanan)" << endl;
            return _leftRightCaseRotate(node);
        }
```

```
        if(balanceFactor < -1 && value > node->right->data){
            cout << "Ketika insert node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF < -1 dan subtree kiri < subtree kanan)" <<
endl;
            return _rightCaseRotate(node);
        }
        if(balanceFactor < -1 && value < node->right->data){
            cout << "Ketika insert node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF < -1 dan subtree kiri > subtree kanan)" <<
endl;
            return _rightLeftCaseRotate(node);
        }

        return node;
    }

    AVLNode* _findMinNode(AVLNode *node) {
        AVLNode *currNode = node;
        while (currNode && currNode->left != NULL)
            currNode = currNode->left;
        return currNode;
    }

    AVLNode* _remove_AVL(AVLNode* node,int value){
        if(node==NULL)
            return node;
        if(value > node->data)
            node->right=_remove_AVL(node->right,value);
        else if(value < node->data)
            node->left=_remove_AVL(node->left,value);
        else{
            AVLNode *temp;
            if((node->left==NULL)||(node->right==NULL)){
                temp=NULL;
                if(node->left==NULL) temp=node->right;
                else if(node->right==NULL) temp=node->left;

                if(temp==NULL){
                    temp=node;
                    node=NULL;
                }
                else
                    *node=*temp;

                free(temp);
            }
```

```
            else{
                temp = _findMinNode(node->right);
                node->data=temp->data;
                node->right=_remove_AVL(node->right,temp->data);
            }
        }

        if(node==NULL) return node;

        node->height=_max(_getHeight(node->left),_getHeight(node->right))+1;

        int balanceFactor= _getBalanceFactor(node);

        if(balanceFactor>1 && _getBalanceFactor(node->left)>=0) {
            cout << "Ketika delete node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF > 1 dan subtree kiri > subtree kanan)" << endl;
            return _leftCaseRotate(node);
        }

        if(balanceFactor>1 && _getBalanceFactor(node->left)<0) {

            cout << "Ketika delete node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF > 1 dan subtree kiri < subtree kanan)" << endl;
            return _leftRightCaseRotate(node);
        }

        if(balanceFactor<-1 && _getBalanceFactor(node->right)<=0) {

            cout << "Ketika delete node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF < -1 dan subtree kiri < subtree kanan)" <<
endl;
            return _rightCaseRotate(node);
        }

        if(balanceFactor<-1 && _getBalanceFactor(node->right)>0) {

            cout << "Ketika delete node " << value << endl;
            cout << "\tSubtree " << node->data << " tidak seimbang BF = " <<
balanceFactor << " (Kondisi BF < -1 dan subtree kiri > subtree kanan)" <<
endl;
            return _rightLeftCaseRotate(node);
        }

        return node;
    }
```

```
    void _inorder(AVLNode *node) {
        if (node) {
            _inorder(node->left);
            cout << node->data << " ";
            _inorder(node->right);
        }
    }

    void _preorder(AVLNode *node) {
        if (node) {
            cout << node->data << " ";
            _preorder(node->left);
            _preorder(node->right);
        }
    }

    void _postorder(AVLNode *node) {
        if (node) {
            _postorder(node->left);
            _postorder(node->right);
            cout << node->data << " ";
        }
    }


public:
    void init() {
        _root = NULL;
        _size = 0U;
    }

    bool isEmpty() {
        return _root == NULL;
    }

    bool find(int value) {
        AVLNode *temp = _search(_root, value);
        if (temp == NULL)
            return false;

        if (temp->data == value) return true;
        else return false;
    }

    void insert(int value) {
        if (!find(value)) {
            _root = _insert_AVL(_root, value);
```

```
            _size++;
        }
    }

    void remove(int value) {
        if (find(value)) {
            _root = _remove_AVL(_root, value);
            _size--;
        }
    }

    void inorder() {
        this->_inorder(_root);
    }

    void preorder() {
        this->_preorder(_root);
    }
    void postorder() {
        this->_postorder(_root);
    }
};

int main()
{
    AVL avl;
    avl.init();
    int max, node;
    cin >> max;
    string command;
    while (max--)
    {
        cin >> command;
        if(command == "insert"){
            cin >> node;
            avl.insert(node);
        } else if (command == "delete"){
            cin >> node;
            avl.remove(node);
        } else if (command == "inOrder"){
            avl.inorder();

        } else if (command == "preOrder"){
            avl.preorder();

        } else if (command == "postOrder"){
            avl.postorder();
        }
```

```
    }

    return 0;
}
```

Penjelasan

Daily Temperature

Input program ini:

1. Baris pertama adalah N yang menyatakan banyaknya jumlah cuaca
2. N baris berikutnya adalah data cuaca

Kamu diminta untuk menghitung berapa lama Nandy harus menunggu hingga ia dapat keluar kos. Kamu juga diminta untuk memberikan data cuaca pada hari berikutnya hingga hari dengan cuaca yang hangat. Apabila tidak ditemukan data, maka keluarkan string "letsgo!!"

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

struct Nodes {
    int num;
    Nodes *next;
};

struct List{
    Nodes *head;
    Nodes *tail;
};
Nodes *newnode, *curr, *temp, *del, *curt;

void init(List *list){
    list->head = NULL;
    list->tail = NULL;
}

bool check(List *list){
    curr = list->head->next;
    temp = list->head;
    while(curr != NULL){
        if(temp->num > curr->num){
```

```
            curr = curr->next;
        }
        else{
            break;
        }
    }
    if(curr == NULL){
        return true;
    }
    else{
        return false;
    }
}

void AddBack(List *list, int num){
    Nodes *newnode = new Nodes;
    newnode->num = num;
    if(list->head == NULL){
        list->head = newnode;
        list->tail = newnode;
        list->head->next = list->tail;
    }
    else{
        newnode->next = NULL;
        list->tail->next = newnode;
        list->tail = newnode;
    }
}

void delFirst(List *list){
    del = list->head;
    list->head = list->head->next;
    delete del;
}


void printRes(List *list){
    curr = list->head->next;
    curt = list->head->next;
    del = curr->next;
    temp = list->head;
    int now = 1;

    if(temp->num < curr->num){
        cout << now << " " << curr->num;
        cout << endl;
    }

    else if (check(list) == false){
```

```
        curr = list->head->next;
        while(curt != NULL){
            if(temp->num > curt->num){
                now++;
            }
            else{
                break;
            }
            curt = curt->next;
        }
        cout << now << " ";
        while(curr != NULL){
            if(temp->num > curr->num){
                cout << curr->num << " ";
            }
            if(temp->num < curr->num){
                    cout << curr->num << " ";
                    break;
                }
            curr = curr->next;
        }
        cout << endl;
    }
        else if(check(list) == true){
            cout << "letsgo!!";
            cout << endl;
        }
}
int main(){
    List list;
    int max, inp;
    init(&list);
    cin >> max;
    if(max >= 1 && max <= 10){
        for(int i = 0; i < max; i++){
            cin >> inp;
            if(inp >= 30 && inp <= 100){
                AddBack(&list, inp);
            }
        }
    }
    for(int j = 0; j < max-1; j++){
        printRes(&list);
        delFirst(&list);
    }
    cout << "letsgo!!";
}
```

Penjelasan