

Laporan Final Praktikum  
Struktur Data



Nama : Midyanisa Yuniar

NRP : 5027211025

Institut Teknologi Sepuluh Nopember  
Tahun 2022

# Bantu Urutin Dong!

## Analisis Soal

### Input:

- Baris pertama: angka untuk banyak query
- Baris kedua dst: nilai masing-masing node

### Ouput:

- Bilangan dari tiap node. Ganjil terlebih dahulu, baru genap

### Penjelasan:

- Input untuk jumlah query
- Input nilai node sebanyak jumlah query dengan looping
- if  $((i \% 2) == 0)$  atau node genap, maka masukkan ke array
- else langsung cetak
- looping untuk mencetak array

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    int jum_query, i, node, arr[5000];

    // banyak data
    cin >> jum_query;

    for (i = 1; i <= jum_query; i++)
    {
        // input value
        cin >> node;

        // node genap: ditambahkan ke array
        if ((i % 2) == 0)
        {
            arr[i] = node;
        }

        // node ganjil dan tidak sama dengan genap: cetak
        else
        {
            cout << node << " ";
        }
    }
}
```

```
}

for (i = 1; i <= jum_query; i++)
{
    // jika arr[i] != 0, maka cetak
    if (arr[i] != 0)
    {
        cout << arr[i] << " ";
    }
}

return 0;
```

```
}
```

# Daily Temperatures

## Analisis Soal

### Input:

- Baris pertama: jumlah cuaca
- Baris kedua dst: data cuaca

### Output:

- Lama hari menunggu dan data cuaca
- Jika tidak ditemukan data, cetak "lets go!!"

### Penyelesaian:

- Input jumlah cuaca
- Input data cuaca sebanyak jumlah cuaca dan masukkan ke queue temp
- Loop selama queue temp tidak kosong
- Ambil nilai pertama dari queue temp dan hapus nilai pertama
- Mencopy nilai dari queue temp ke temp\_check
- Loop selama temp\_check tidak kosong
- Masukkan nilai temp\_check ke dalam temp\_out
- Jika data cuaca selanjutnya lebih besar daripada data pertama, set flag = true
- Hapus nilai temp\_check → keluar loop
- Jika !flag, cetak "lets go!!"
- Cetak panjang queue temp\_out → lama hari menunggu
- Cetak isi temp\_out → data cuaca

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;

int main()
{
    queue<int> temp;
    int jum_temp;
    int value;

    // input jumlah cuaca
    cin >> jum_temp;

    while (jum_temp--)
    {
        // input cuaca / temperatures
        cin >> value;
        temp.push(value);
    }

    // loop selama queue temp tidak kosong
    while (!temp.empty())
```

```

{
    // ambil nilai pertama
    int f = temp.front();

    // hapus nilai pertama
    temp.pop();

    // copy queue
    queue<int> temp_check = temp;
    queue<int> temp_out;

    bool flag = false;

    // loop selama queue temp_check tidak kosong
    while (!temp_check.empty())
    {
        // masukkan nilai ke queue temp_out
        temp_out.push(temp_check.front());

        // jika nilai lebih besar dari nilai pertama
        if (temp_check.front() > f)
        {
            // set flag = true
            flag = true;
            break;
        }

        // hapus nilai pertama
        temp_check.pop();
    }

    // jika !flag , cetak "lets go!!"
    if (!flag)
    {
        cout << "lets go!!" << endl;
        continue;
    }

    // cetak panjang queue temp_out
    cout << temp_out.size() << " ";

    // cetak isi queue temp_out
    while (!temp_out.empty())
    {
        cout << temp_out.front() << " ";
        temp_out.pop();
    }
}

```

```
        cout << endl;  
    }  
  
    return 0;  
}
```

# Cek Komposisi Lagi Skuy!

Analisis Soal:

Input:

- Baris pertama: banyak input test case dan komposisi sehat
- Baris kedua dst: makanan dan komposisi makanan tersebut, terdapat command "GASS"

Output:

- Rata – rata komposisi saat ini
- Rata – rata yang memenuhi rentang sehat, jika ada cetak "AMAN", tidak ada cetak "LOH"

Penyelesaian:

- Input jumlah test case
- Input command dengan total sebanyak jumlah test case
- Selama command != "GASS" inputkan juga komposisi makanan
- Jika command == "GASS", maka akan menghitung rata – rata makanan sehat dan mencetaknya serta menambahkan nya ke total sementara
- Menghitung jumlah komposisi rata – rata keseluruhan dari makanan sehat dan mencetaknya
- Mencetak status jika memenuhi kondisi

```
#include <iomanip>
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    int jum_test, num_sehat, count = 0, jum_ren_sehat = 0;
    double tot = 0, ren_sehat = 0, tot_sementara = 0, komposisi, tot_akhir;
    string makanan;

    // input jumlah test, jumlah sehat
    cin >> jum_test;
    cin >> num_sehat;

    for (int I = 0; I < jum_test; i++)
    {
        // input makanan
        cin >> makanan;

        // jika makanan != "GASS", input komposisi makanan
```

```

    if (makanan != "GASS")
    {
        cin >> komposisi;

        // hitung total komposisi
        tot = tot + komposisi;

        // hitung jumlah makanan sehat
        count++;
    }

    // jika makanan == "GASS", hitung rata-rata komposisi makanan sehat
    else
    {
        ren_sehat = tot / count;
        cout << fixed << setprecision(2) << ren_sehat << endl;
        tot_sementara = tot_sementara + ren_sehat;
        jum_ren_sehat++;
    }
}

// hitung rata-rata komposisi makanan sehat
tot_akhir = tot_sementara / jum_ren_sehat;

// output rata-rata komposisi makanan sehat
cout << tot_akhir << " ";

// output status
if (0.5 * num_sehat <= tot_akhir && 1.5 * num_sehat >= tot_akhir)
{
    cout << "AMAN";
}
else
{
    cout << "LOH";
}
}

```



# Cinta Segitiga Lagi Nich

Analisis Soal:

Input:

- Baris pertama: jumlah komputer
- Baris kedua: komputer ke  $i$  menyukai komputer  $f_i$

Output:

- Cetak YES jika cinta segitiga
- Cetak NO jika tidak

Penyelesaian:

- Input jumlah komputer, kemudian input nomor komputer ke dalam array com
- Setiap angka dikurangi 1, sehingga index com sesuai nomor komputer
- Loop cek komputer, dimana jika com ke  $i ==$  komputer ke 4, maka cetak YES
- Jika tidak maka NO

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    int jum_komputer, com[3001];

    // input jumlah komputer
    cin >> jum_komputer;

    // input komputer
    for (int i = 0; i < jum_komputer; ++i)
    {
        cin >> com[i];

        // decrement komputer
        --com[i];
    }

    // cek komputer
    for (int i = 0; i < jum_komputer; ++i)
    {
        // jika komputer sama dengan komputer ke-4
        if (com[i] == com[com[com[com[i]]]])
        {
            // cetak YES
        }
    }
}
```

```
        cout << "YES" << endl;
        return 0;
    }
}

// cetak NO
cout << "NO" << endl;

return 0;
}
```

# Program Canggih

Analisis Soal:

Input:

- Baris pertama: jumlah query
- Baris kedua: command (insert, delete), value, dan order (inOrder, postOrder, preOrder)

Output:

- Log rotasi yang dilakukan pada saat insert atau delete node
- Hasil data dari perintah order

Penyelesaian:

- Penjelasan dari tiap program sesuai dengan keterangan pada setiap fungsi rotasi
- Dimana penjelasan diberikan sebelum fungsi rotasi dilakukan

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

//
int count = 0;

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

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

    // Fungsi untuk membuat node baru
    AVLNode *_avl_createNode(int value)
    {
        AVLNode *newNode = (AVLNode *)malloc(sizeof(AVLNode));
        newNode->data = value;
        newNode->height = 1;
        newNode->left = newNode->right = NULL;
        return newNode;
    }
}
```

```

// Fungsi untuk mencari node
AVLNode *_search(AVLNode *root, int value)
{
    // Jika root != kosong
    while (root != NULL)
    {
        // Jika value < root->data, maka cari di kiri
        if (value < root->data)
            root = root->left;
        // Jika value > root->data, maka cari di kanan
        else if (value > root->data)
            root = root->right;
        else
            return root;
    }
    return root;
}

// Fungsi untuk mencari tinggi node
int _getHeight(AVLNode *node)
{
    // Jika node kosong, maka tinggi = 0
    if (node == NULL)
        return 0;
    // Jika tidak kosong, maka tinggi = node->height
    return node->height;
}

// Fungsi untuk mencari nilai maksimum
int _max(int a, int b)
{
    return (a > b) ? a : b;
}

// Fungsi untuk melakukan rotasi kanan
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;
}

// Fungsi untuk melakukan rotasi kiri
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);
}

// Fungsi untuk melakukan rotasi kiri-kanan
AVLNode *_LeftRightCaseRotate(AVLNode *node)
{
    node->left = _LeftRotate(node->left);
    return _rightRotate(node);
}

// Fungsi untuk melakukan rotasi kanan-kiri
AVLNode *_rightLeftCaseRotate(AVLNode *node)
{
    node->right = _rightRotate(node->right);
    return _LeftRotate(node);
}

```

```

// Fungsi untuk mencari balance factor
int _getBalanceFactor(AVLNode *node)
{
    if (node == NULL)
        return 0;
    return _getHeight(node->left) - _getHeight(node->right);
}

// Fungsi untuk melakukan insert node
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 (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;
}

// Fungsi untuk mencari node terkecil
AVLNode *_findMinNode(AVLNode *node)
{
    AVLNode *currNode = node;
    while (currNode && currNode->left != NULL)
        currNode = currNode->left;
    return currNode;
}

// Fungsi untuk melakukan remove node
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;
}

// Fungsi untuk melakukan traversal inorder
void _inorder(AVLNode *node)
{
    if (node)
    {
        _inorder(node->left);
        cout << node->data << " ";
        _inorder(node->right);
    }
}

// Fungsi untuk melakukan traversal preorder
void _preorder(AVLNode *node)
{
    if (node)
    {
        cout << node->data << " ";
        _preorder(node->left);
        _preorder(node->right);
    }
}

// Fungsi untuk melakukan traversal postorder
void _postorder(AVLNode *node)
{
    if (node)
    {
        _postorder(node->left);
        _postorder(node->right);
        cout << node->data << " ";
    }
}

public:
    // Constructor
    void init()
    {
        _root = NULL;
    }

```

```

        _size = 0U;
    }

    // Fungsi untuk mengecek apakah tree kosong
    bool isEmpty()
    {
        return _root == NULL;
    }

    // Fungsi untuk mencari node
    bool find(int value)
    {
        AVLNode *temp = _search(_root, value);
        if (temp == NULL)
            return false;

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

    // Fungsi untuk melakukan insert node
    void insert(int value)
    {
        if (!find(value))
        {
            _root = _insert_AVL(_root, value);
            _size++;
        }
    }

    // Fungsi untuk melakukan remove node
    void remove(int value)
    {
        if (find(value))
        {
            _root = _remove_AVL(_root, value);
            _size--;
        }
    }

    // Fungsi untuk melakukan traversal inorder
    void inorder()
    {
        this->_inorder(_root);
    }

```

```

// Fungsi untuk melakukan traversal preorder
void preorder()
{
    this->_preorder(_root);
}

// Fungsi untuk melakukan traversal postorder
void postorder()
{
    this->_postorder(_root);
}
};

int main()
{
    // Buat objek AVL
    AVL avl_tree;

    // Panggil fungsi init
    avl_tree.init();

    int jum_query, value;
    string comm;

    // Input jumlah query
    cin >> jum_query;

    // Looping sebanyak jumlah query
    while (jum_query--)
    {
        // Input command
        cin >> comm;
        if (comm == "insert")
        {
            cin >> value;
            avl_tree.insert(value);
        }
        else if (comm == "delete")
        {
            cin >> value;
            avl_tree.remove(value);
        }
        else if (comm == "inOrder")
        {
            avl_tree.inorder();
        }
        else if (comm == "preOrder")
        {

```

```
        avl_tree.preorder();
    }
    else if (comm == "postOrder")
    {
        avl_tree.postorder();
    }
}

return 0;
}
```

# Urutin oi

Analisis Soal:

Input:

- Input value yang tidak urut ke dalam array

Output:

- Urutkan value dan hapus duplikasi

Penyelesaian:

- Membuat list 'arr[100]' dan inisialisasi iterator, sehingga akan memasuki loop tidak terbatas
- Jika num == 0, loop berhenti. Jika != 0, maka berlanjut
- List arr akan diurutkan kemudian dihapus duplikasinya
- Mencetak semua isi list

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <list>
#include <iterator>
using namespace std;

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

    for (int i = 0; i > -1; i++)
    {
        // input number
        cin >> num;

        // jika num = 0, break
        if (num == 0)
        {
            break;
        }

        // push number ke array
        arr[start].push_back(num);
    }

    // sort dan unique: urutkan dan hapus duplikat
```

```
arr[start].sort();
arr[start].unique();

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

return 0;
}
```

# Doorprize Toko ARA

Analisis Soal:

Input:

- Baris pertama: durasi
- Baris kedua: jumlah receipt dan jumlah transaksi tiap receipt

Output:

- Total prize

Penyelesaian:

- Input durasi
- Membuat dua priority\_queue low dan high untuk menyimpan transaksi tertinggi dan terendah
- Loop untuk menerima masukan receipt dan loop untuk menerima masukan transaksi. Setiap transaksi ditambahkan ke low dan high
- Nilai tertinggi dari 'high' dan nilai terendah dari 'low' dikurangi dan ditambahkan ke total prize
- Mencetak total prize

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;

int main()
{
    int durasi, total_prize = 0, receipt, transaksi;

    priority_queue<int> high;
    priority_queue<int, vector<int>, greater<int>> low;

    // input durasi
    cin >> durasi;

    for (int i = 0; i < durasi; i++)
    {
        // input receipt
        cin >> receipt;

        // input transaksi
        for (int j = 0; j < receipt; j++)
        {
            cin >> transaksi;
```

```
        // masukkan ke priority queue
        high.push(transaksi);
        low.push(transaksi);
    }

    // hitung total prize
    total_prize += high.top() - low.top();

    // hapus data
    high.pop();
    low.pop();
}

// cetak total prize
cout << total_prize << endl;
return 0;
}
```