Pendahuluan Struktur Data Stack

Maulana Ifandika

Pendahuluan

Stack adalah struktur data yang menyimpan data menggunakan prinsip LIFO. Stack sendiri merupakan perluasan dari Vektor.

Operasi Stack

push(item) = Operasi menambah data ke stack.

pop() = Operasi menghapus data paling atas pada stack.

peek() = Operasi mengambil data tapi tidak menghilagkan data yang diambil.

top() = Operasi mengambil data paling atas.

size() = Operasi mengambil total jumlah nilai pada stack.

Java Stack API

class: public class Stack<E> extends Vector<E>

Untuk menggunakan stack pada java sudah tersedia di package 'java.util', lebih spesifiknya

java.lang.Object

java.util.AbstractCollection<E>

java.util.AbstractList<E>

java.util.Vector<E>

java.util.Stack<E>

Karena Stack adalah ADT(Abstract Data Type) maka bisa menyimpan berbagai data tidak hanya bilangan, untuk menyimpan data dengan tipe yang tertentu bisa kita masukan tipe pada argument <E> atau elemen. Ganti simbol E dengan tipe data, semisal Integer, String, Objek, atau Class.

Stack<Buah> stackBuah = new Stack<Buah>();

Pada java versi > 8 penulisan tipe hanya di awal saja.

Stack<Buah> stackBuah = new Stack<>();

Implementasi stack.

public interface Stack<E> {

int size( );

boolean isEmpty( );

void push(E e);

E top( );

E pop( );

}

Operasi

Modifier Type Method and Description

boolean empty() Cek stack kosong tidak.

E peek() Ambil data stack tanpa menghapus.

E pop() Hapus nilai paling atas.

E push(E item) Tambah data ke stack.

int search(Object o) Cari nilai pada stack jika tidak ketemu retun -1.

Implementasi Stack Dengan Array

Mengimplementasikan stack dengan array sebagai penyimpan data, maka saat mengakses data pada stack menggunakan index. Pada operasi cukup kita panggil operasi dari single linked list.

Method Stack Method Single Linked List

push(item) .addFist(item)

pop() .removeFirst()

isEmpty() .isEmpty()

top() .first()

size() .size()

public class ArrayStack<E> impelemnts Stack<E> {

private static final int CAPACITY = 100;

private E[] data;

private int last = -1;

public ArrayStack() {

this(CAPACITY);

}

public ArrayStack(int capacity) {

data = new E[capacity];

}

public int size() {

return (last + 1);

}

public boolean isEmpty() {

return (last == -1);

}

public E top() {

if(isEmpty()) return null;

return data[last];

}

public E push(E item) {

if(size() == CAPACITY) throw new IllegalStateException("Stack is full");

data[++last] = item;

}

public E pop() {

if(isEmpty()) return null;

E removed = data[last];

data[last--] = null;

return removed;

}

}

Kompleksitas

Dari implemntasi di atas kompleksitas dari waktu yang dijalakan setiap method/fungsi

Method Kompleksitas

size() O(1) // Waktu tetap

isEmpty() O(1)

top() O(1)

push() O(1)

pop() O(1)

Validasi Parenthesis Dengan Stack

Menggunakan Stack untuk validasi apakah sebuah parenthesis benar/tidak. Parentesis sendiri adalah beberapa gabungkan simbol yang sudah ditentutkan aturannya. Setiap simbol permbuka harus diakhiri dengan simbol penutup, jika salah satu kurang maka paenthesis tidak valid.

Parenthesis ( dan )

Braces { dan }

Bracket [ dan ]

Contoh:

- ()((())) true

- ([{}]) true

- ({(})}) false

- (} false

Langkah-langkah:

1. Jika argumen kosong return true.
2. Jika panjang argument modulo 2 hasil != 0 "Berati panjang ganjil" return false.
3. Deklarasi Stack.
4. Perulangan dari 1 - panjang argumen.
5. Jika simbol 'i' == '(' atau '{' atau '[' maka push ke Stack.
6. Jika Stack kosong berati simbol depan bukan simbol pembuka maka return false.
7. Jika argumen 'i' == simbol pembuka dan nilai top stack == bukan simbol penutup yang keduanya
8. tipe simbol yang sama maka return false.

public boolean mathingParentheses(String s) {

if(s == null || s == "") return true;

if(s.length() % 2 != 0) return false;

Stack<Character> stack = new Stack<>();

for(int i = 0; i < s.length(); i++) {

if(s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[') {

stack.push(s.charAt(i));

}

else if(stack.empty()) return false;

else if((s.charAt(i) == ')' && stack.peek() != '(') ||

(s.charAt(i) == '}' && stack.peek() != '{') ||

(s.charAt(i) == ']' && stack.peek() != '[') {

return false;

}

else {

stack.pop();

}

}

return stack.empty();

}