**Laporan**

**Tugas Kecil IF2211 Strategi Algoritma**

**Lonely Island**

****

Oleh:

M. Rifky I. Bariansyah

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2019

1. **Algoritma *Decrease and Conquer***

Program ini diselesaikan menggunakan implementasi *decrease and conquer* yaitu *depth-first search*. *Depth first search* menggunakan strategi dengan mencari dalam sebuah graf sedalam mungkin. *Depth first search* menjelajah sisi dari simpul *v* terbaru ditemukan yang masih memiliki sisi yang belum terjelajah. Setelah semua simpul *v* terjelajah, dilakukan *backtrack* untuk menjelajahi tiap sisi yang meninggalkan simpul dari mana simpul *v* ditemukan.[1]

* + 1. Algoritma DFS rekursif dijalankan mulai dari *r*, pulau awal, dengan array boolean, *flag*, yang bernilai true bila sebuah pulau sudah dikunjungi.
    2. Elemen *r* pada *flag* ditandai sebagai true.
    3. Pulau disimpan pada list enumerasi langkah, *path*.
    4. Dilakukan pemanggilan fungsi yang mengecek apakah masih terdapat jembatan yang masih bisa digunakan untuk menyebrang ke pulau lain dengan mengecek semua jembatan pada pulau saat ini memiliki pulau yang bernilai false pada *flag.*
    5. Bila tidak ada jembatan yang bisa digunakan maka pulau itu akan dimasukkan kedalam list *lonely island*. *Path* akan diletakkan pada elemen list yang menyimpan enumerasi – enumerasi langkah, *Paths*.
    6. Dilakukan DFS pada simpul yang bersinggungan dengan simpul saat ini.
    7. Setelah semua simpul telah dijelajahi maka Elemen *r* pada *flag* akan dikembalikan ke false dan pulau dihapus dari *path* untuk *backtracking*.

1. **Source Program**

**//By M. Rifky I. Bariansyah**

**//13517081**

**import java.util.\*;**

**import java.util.concurrent.TimeUnit;**

**import java.io.\*;**

**public** **class** Island {

**public** **int** N; //number of islands

**public** LinkedList<Integer> adj[]; //Array of Adjacency Linked List

// adj[u]: array of adjacency from u

// [i1, i2, ...] to i1, i2, ...

**public** LinkedList<Integer> LonelyIsland;//List of Lonely Island

**public** LinkedList<Integer> path;//Temporary list for path enumeration

**public** LinkedList<LinkedList<Integer>> allPaths;//List of path enumeration

//Island Constructor

Island(**int** n){

N = n; //m islands

adj = **new** LinkedList[N+1]; //N LinkedLists

adj[0] = **null**;

LonelyIsland = **new** LinkedList();

path = **new** LinkedList();

allPaths = **new** LinkedList<>();

**for** (**int** i = 1 ; i <= n ; i++) {

adj[i] = **new** LinkedList();

}

System.out.println("Constructor Activated "+ N +" Island(s)");

}

//add a directed Bridge from u to i

**void** addBridge(**int** u, **int** i) {

**try** {

adj[u].add(i);

System.out.println("Added an Bridge from "+u+" to "+i);

}

**catch** (NullPointerException error){

System.out.println("Island is out of bound");

}

**catch** (ArrayIndexOutOfBoundsException error){

System.out.println("Island is out of bound");

}

}

**Island.java**

//Check if there's no more available bridge for connecting to other island(s)

**boolean** NoAvailBridge(**int** i, **boolean** flag[]) {

**boolean** available;

available = **true**;

Iterator<Integer> n = adj[i].listIterator();

**while** (n.hasNext() && available)

{

**int** m = n.next();

**if** (flag[m] == **false**) {

available = **false**;

}

}

**return** available;

}

**void** DFS(**int** i, **boolean** flag[]) {

LinkedList tempList = **new** LinkedList();

//Flag island as visited

flag[i] = **true**;

path.add(i);

**if** (NoAvailBridge(i,flag)) {

//Add to List of Lonely Island(s)

**if**(!LonelyIsland.contains(i)){

LonelyIsland.add(i);}

//Assigning and then Moving to the next list of path

tempList = (LinkedList) path.clone();

allPaths.add(tempList);

}

//Recursive for all bridges

Iterator<Integer> n = adj[i].listIterator();

**while** (n.hasNext())

{

**int** m = n.next();

**if** (!flag[m]) {

DFS(m, flag);}

}

//Flag island as unvisited for backtracking

flag[i] = **false**;

path.removeLast();

}

**import** java.io.\*;

**public** **class** Main **implements** Runnable{

**static** **void** title() {

#IMPLEMENTASI TITLE

}

**public** **static** **void** main(String args[]) {

**new** Thread(**null**, **new** Main(), "thread1", 1<<60).start();//setting stack size for depth first search

}

**public** **void** run() {

//title

title();

String fileName;

Scanner sc = **new** Scanner(System.in);

//File Input

System.out.println("");

System.out.print("Please enter your file name: ");

fileName = sc.nextLine();

System.out.println("Reading "+fileName);

System.out.println("");

Scanner fsc = **null**;

**try** {

fsc = **new** Scanner(**new** File(fileName));

} **catch** (FileNotFoundException e) {

System.out.println("File not found!");

}

//Find all lonely island(s) and enumeration to the island(s)

**void** LastResort(**int** i) {

//array of visited island(s)

**boolean** flag[] = **new** **boolean**[N+1];

//Depth first search implementation

System.out.println("Starting DFS..");

System.out.println("");

DFS(i,flag);

}

}

**Main.java**

**int** n, m, r;

n = fsc.nextInt();

m = fsc.nextInt();

r = fsc.nextInt();

**long** startTime = System.nanoTime();

//Island initialisation

Island A = **new** Island(n);

//Bridge Construction

**while**(fsc.hasNextInt()){

**int** u;

**int** i;

u = fsc.nextInt();

i = fsc.nextInt();

A.addBridge(u, i);

}

System.out.println("");

System.out.println("Island construction completed");

System.out.println("");

//Printing Bridges

**for** (**int** i = 1; i <=n; i++) {

System.out.print("Island "+i+" directs to: ");

System.out.println(A.adj[i]);

}

//Find all lonely island(s) and enumeration

A.LastResort(r);

**long** endTime = System.nanoTime();

System.out.println("==================================");

System.out.println("");

System.out.println("Lonely Island(s): ");

Iterator<Integer> i = A.LonelyIsland.iterator();

**while**(i.hasNext()){

System.out.print(i.next()+" ");

}

System.out.println("\n");

System.out.println("Path(s) enumeration:");

Iterator<LinkedList<Integer>> j = A.allPaths.iterator();

**while**(j.hasNext()){

Iterator<Integer> k = j.next().iterator();

System.out.print("[");

**while**(k.hasNext()) {

System.out.print(k.next());

**if**(k.hasNext()) {

System.out.print("-> ");

}

}

System.out.println("]");

}

//Execution time

System.out.println("");

System.out.println("Executed in " + (endTime - startTime)/1000000 + " milliseconds");

}

}

System.out.println("]");

}

//Execution time

System.out.println("");

System.out.println("Executed in " + (endTime - startTime)/1000000 + " milliseconds");

}

}

1. **Contoh *Input* dan *Output***

|  |  |  |
| --- | --- | --- |
| Input | Output | Waktu Eksekusi |
| 5 7 1  1 2  1 3  1 4  1 5  2 4  2 5  3 4 | C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 17-27-35.png | 10 ms |
| 4 4 1  1 2  1 4  2 3  3 1 | C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 17-31-33.png | 9 ms |
| 6 7 1  1 2  1 4  1 5  2 3  3 1  5 3  3 6 | C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 18-07-33.png | 10 ms |
| 1000 1000 1  1 2  2 3  3 4  4 5  5 6  6 7  7 8  .  .  .  996 997  997 998  998 999  999 1000  1 950 | C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 18-23-37.png  ... lanjutan enumerasi  C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 18-24-02.png  …lanjutan enumerasi  C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 18-24-26.png | 610 ms |
| 50000 50000 45800  1 2  2 3  3 4  4 5  5 6  .  .  .  49997 49998  49998 49999  49999 50000  45990 50000 | C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 18-15-10.png  …lanjutan enumerasi  C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 18-15-29.png  …lanjutan enumerasi  C:\Users\Bariansyah\Desktop\Stima-Tucil-2-master\test cases\Screenshot from 2019-02-24 18-15-55.png | 3163 ms |

*Checklist* poin:

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | ✓ |  |
| 2. Program berhasil dieksekusi | ✓ |  |
| 3. Program dapat menerima input dan menuliskan output | ✓ |  |
| 4. Luaran sudah benar untuk semua n | ✓ |  |

**REFERENSI**

[1] Cormen, T. H., Leiserson, C. E., *Introduction to Algorithms 3rd ed.*, The MIT Press, Massachusetts