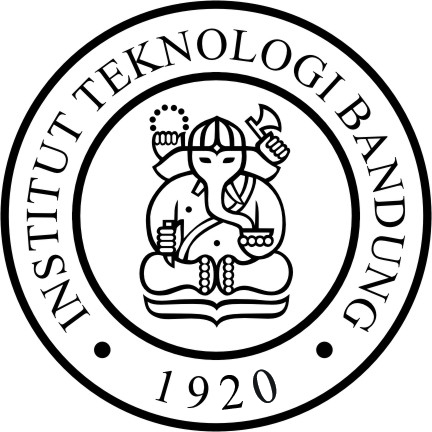
**Laporan Tugas Kecil II**

**Algoritma *Decrease and Conquer* pada Persoalan *Lonely Island***

**Disusun untuk memenuhi tugas mata kuliah**

**IF2211 Strategi Algoritma pada Semester II 2018/2019**



**Disusun oleh :**

Nama : Steve Andreas Immanuel

NIM : 13517039

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2019**

**Algoritma *Decrease and Conquer***

Algoritma *decrease and conquer* pada persoalan *Lonely Island* yang digunakan adalah sebagai berikut:

1. Ubah status semua pulau menjadi “belum dikunjungi”.
2. Ubah status pulau sekarang menjadi “sedang dikunjungi”.
3. Periksa banyaknya jembatan yang terhubung di pulau sekarang.

* Jika banyaknya jembatan yang terhubung di pulau sekarang lebih dari satu, maka untuk setiap pulau tetangga dengan status “belum dikunjungi” atau “bagian dari loop”, ulangi algoritma dari langkah-2 hingga langkah-4. Jika semua pulau tetangga memiliki status “sedang dikunjungi”, maka pulau sekarang merupakan pulau yang membuat terjebak. Ubah status pulau sekarang menjadi “bagian dari loop”.
* Jika banyaknya jembatan yang terhubung di pulau sekarang adalah satu, maka periksa status pulau berikutnya.
* Jika status pulau berikutnya adalah “belum dikunjungi” atau “bagian dari loop”, maka ulangi algoritma dari langkah-2 hingga langkah-4 untuk pulau berikutnya.
* Jika status pulau berikutnya adalah “sedang dikunjungi”, maka pulau sekarang adalah pulau yang membuat terjebak. Ubah status pulau sekarang menjadi “bagian dari loop”.
* Jika banyaknya jembatan yang terhubung di pulau sekarang adalah nol, maka pulau sekarang adalah pulau yang membuat terjebak. Ubah status pulau sekarang menjadi “sudah dikunjungi”.

1. Jika status pulau sekarang masih belum berubah karena langkah-3, maka periksa status dari pulau tetangganya.

* Jika hanya ada satu pulau tetangga, ubah status pulau sekarang seperti status pulau tetangga tersebut.
* Jika terdapat banyak pulau tertangga, maka
* Jika ada satu pulau tetangga yang statusnya “bagian dari loop”, maka ubah status pulau sekarang menjadi “bagian dari loop”.
* Jika tidak, ubah status pulau sekarang menjadi “sudah dikunjungi”.

Algoritma *decrease and conquer* yang diterapkan adalah *decrease and conquer* dengan tipe *decrease by variable size*. Hal ini berarti pada setiap saat algoritma ini dijalankan pada suatu pulau tertentu, algoritma ini tidak akan mengunjungi pulau-pulau tetangga yang sudah dikunjungi. Dengan kata lain mengurangi banyaknya persoalan sebanyak pulau tetangga yang sudah dikunjungi tergantung posisi jembatan yang ada.

***Source Code* Program**

Untuk mengimplementasikan algoritma *decrease and conquer* pada persoalan *Lonely Island*, digunakan bahasa Java. Berikut adalah *source code* dari program:

import java.util.Scanner;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashSet;

import java.util.Set;

import java.io.File;

import java.io.FileNotFoundException;

import java.lang.Thread;

public class LonelyIsland implements Runnable{

private String filename;

public LonelyIsland(String[] args){//ambil argumen

if(args.length!=1){

System.out.println("Cara run : java LonelyIsland <namafile>\n"+

"File input diletakkan di folder bernama inputfiles");

System.exit(0);

}else{

this.filename=args[0];

System.out.println("Membaca file "+filename);

}

}

public void findSink(Set<Integer> sinkIsland,island currentIsland){//mencari trap island

currentIsland.setVisitStatus(1);//set status menjadi sedang dikunjungi

int size=currentIsland.getSize();

int childStatus=2;//default=2

if (size==0){

sinkIsland.add(currentIsland.getID());

}else if (size==1){//tetangga hanya satu

if(currentIsland.getNextIsland(0).getVisitStatus()==0||currentIsland.getNextIsland(0).getVisitStatus()==3){//boleh divisit

findSink(sinkIsland, currentIsland.getNextIsland(0));

childStatus=currentIsland.getNextIsland(0).getVisitStatus();

}else if(currentIsland.getNextIsland(0).getVisitStatus()==1){//parent blm selesai divisit

sinkIsland.add(currentIsland.getID());

childStatus=3;

}

}else{

boolean canTravel=false;

for(int i=0;i<currentIsland.getSize();i++){

if(currentIsland.getNextIsland(i).getVisitStatus()==0||currentIsland.getNextIsland(i).getVisitStatus()==3){//kunjungi yang blm dikunjungi

canTravel=true;

findSink(sinkIsland, currentIsland.getNextIsland(i));

}else if(currentIsland.getNextIsland(i).getVisitStatus()==2){

canTravel=true;

}

if(currentIsland.getNextIsland(i).getVisitStatus()==3){

childStatus=3;

}

}

if(!canTravel){//jika punya banyak tetangga tapi tidak bisa travel, berarti dia bagian siklik

sinkIsland.add(currentIsland.getID());

childStatus=3;

}

}

currentIsland.setVisitStatus(childStatus);//set status sesuai childnya

}

public void printPath(island source,island dest,ArrayList<Integer> path,boolean[] visited){

path.add(source.getID());

visited[source.getID()-1]=true;

int initialSize=path.size();

for(int i=0;i<source.getSize();i++){

if(initialSize<path.size()){

path.subList(initialSize, path.size()).clear();

}

if(source.getNextIsland(i)==dest){

path.add(dest.getID());

System.out.println(path);

}else if(!visited[source.getNextIsland(i).getID()-1]&&source.getNextIsland(i)!=dest){

printPath(source.getNextIsland(i), dest, path, visited);

}

}

visited[source.getID()-1]=false;

}

public static void main(String[] args) throws Exception {//buat thread baru agar tidak stackoverflow

new Thread(null, new LonelyIsland(args), "Main", 1 << 26).start();

}

public void run() {//jalankan thread baru

int n\_island=0,n\_bridge=0,cur\_island=0;

ArrayList<island> islands=new ArrayList<island>();//sebagai graf

Set<Integer> tempResult=new HashSet<Integer>();

ArrayList<Integer> result;

boolean[] visitStatus;//visitstatus untuk print semua path,

//mempercepat program agar tidak perlu mereset visitstatus setiap island

try{//baca input dari file

Scanner scanner = new Scanner(new File("inputfiles/"+filename));

if(scanner.hasNextLine()){

String[] buffer=scanner.nextLine().split(" ");

n\_island=Integer.parseInt(buffer[0]);

n\_bridge=Integer.parseInt(buffer[1]);

cur\_island=Integer.parseInt(buffer[2]);

for(int i=0;i<n\_island;i++){

islands.add(new island(i+1));//ID island dimulai dari 1

}

for(int i=0;i<n\_bridge;i++){//bentuk graf

buffer=scanner.nextLine().split(" ");

islands.get(Integer.parseInt(buffer[0])-1).addIsland(islands.get(Integer.parseInt(buffer[1])-1));

}

}

scanner.close();

}catch(FileNotFoundException|ArrayIndexOutOfBoundsException e){

if(e instanceof FileNotFoundException){

System.out.println("Error: File tidak ditemukan");

}else if(e instanceof ArrayIndexOutOfBoundsException){

System.out.println("Error: Parameter input salah");

}

System.exit(0);

}

System.out.println("Baca file selesai");

visitStatus=new boolean[n\_island];

long startTime = System.nanoTime();//catat waktu awal

findSink(tempResult,islands.get(cur\_island-1));

result=new ArrayList<Integer>(tempResult);

Collections.sort(result);

System.out.println("Pulau yang membuat terjebak :");

System.out.println(result);

System.out.println("Semua path yang mungkin :");

for(int i=0;i<result.size();i++){//

printPath(islands.get(cur\_island-1), islands.get(result.get(i)-1),new ArrayList<Integer>(),visitStatus);

}

long endTime = System.nanoTime();//catat waktu akhir

System.out.println("Total waktu eksekusi: "+(endTime-startTime)/1000000+" ms");

System.exit(0);

}

}

class island{//representasi node

private ArrayList<island> nextIsland;

private int ID;

private int visited; //0=belum dikunjungi,

//1=sudah dikunjungi dan sedang mengunjungi childnya,

//2=sudah selesai dikunjungi sampai ke child2nya

//3=island merupakan bagian dari suatu loop

public island(int \_ID){

this.nextIsland = new ArrayList<island>();

this.ID=\_ID;

this.visited=0;

}

public void addIsland(island \_next){

this.nextIsland.add(\_next);

}

public void setVisitStatus(int i){

this.visited=i;

}

public int getSize(){

return this.nextIsland.size();

}

public int getID(){

return this.ID;

}

public island getNextIsland(int i){

return this.nextIsland.get(i);

}

public int getVisitStatus(){

return this.visited;

}

}

**Contoh *Input Output***

Berikut adalah contoh *input output* dari program pada berbagai kondisi beserta waktu eksekusinya.

*File input:*

8 12 1

1 5

2 1

3 2

3 4

4 8

5 2

5 6

6 2

6 7

7 3

7 8

7 6

*Output:*

Membaca file input.txt

Baca file selesai

Pulau yang membuat terjebak :

[2, 8]

Semua path yang mungkin :

[1, 5, 2]

[1, 5, 6, 2]

[1, 5, 6, 7, 3, 2]

[1, 5, 6, 7, 3, 4, 8]

[1, 5, 6, 7, 8]

Total waktu eksekusi: 9 ms

*File input*:

6 7 1

1 2

2 3

3 4

4 2

2 5

5 6

6 2

*Output*:

Membaca file input2.txt

Baca file selesai

Pulau yang membuat terjebak :

[4, 6]

Semua path yang mungkin :

[1, 2, 3, 4]

[1, 2, 5, 6]

Total waktu eksekusi: 6 ms

*File input*:

5 7 1

1 2

1 3

1 4

1 5

2 4

2 5

3 4

*Output*:

Baca file selesai

Pulau yang membuat terjebak :

[4, 5]

Semua path yang mungkin :

[1, 2, 4]

[1, 3, 4]

[1, 4]

[1, 2, 5]

[1, 5]

Total waktu eksekusi: 10 ms

*File input*:

7 7 1

1 2

1 3

3 4

2 4

4 5

4 6

4 7

*Output*:

Membaca file input4.txt

Baca file selesai

Pulau yang membuat terjebak :

[5, 6, 7]

Semua path yang mungkin :

[1, 2, 4, 5]

[1, 3, 4, 5]

[1, 2, 4, 6]

[1, 3, 4, 6]

[1, 2, 4, 7]

[1, 3, 4, 7]

Total waktu eksekusi: 12 ms

*File input*:

4 4 1

1 2

2 3

3 1

3 4

*Output*:

Membaca file input5.txt

Baca file selesai

Pulau yang membuat terjebak :

[4]

Semua path yang mungkin :

[1, 2, 3, 4]

Total waktu eksekusi: 5 ms

*File input*:

7 9 1

1 2

1 3

1 4

2 5

3 6

4 7

5 6

6 7

7 5

*Output*:

Membaca file input6.txt

Baca file selesai

Pulau yang membuat terjebak :

[5, 6, 7]

Semua path yang mungkin :

[1, 2, 5]

[1, 3, 6, 7, 5]

[1, 4, 7, 5]

[1, 2, 5, 6]

[1, 3, 6]

[1, 4, 7, 5, 6]

[1, 2, 5, 6, 7]

[1, 3, 6, 7]

[1, 4, 7]

Total waktu eksekusi: 14 ms

**Hasil Program**

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

Pada program awalnya terdapat kendala *stackoverflow* yang dikarenakan *deep* *recursion* jika terdapat sangat banyak pulau. Maka untuk mengatasinya digunakan thread baru sehingga memory stack yang dialokasikan dapat lebih besar.

Untuk menuliskan semua kemungkinan pergerakan dari awal pulau hingga terjebak diperlukan *memory* yang sangat besar untuk menyimpan pulau-pulau yang sudah dilewati jika terdapat sangat banyak pulau. Akibatnya terjadi *heap memory* *error* karena kurang *memory*. Untuk mengatasinya, cara program untuk mengingat pulau yang sudah dilewati adalah dengan menyimpan sebagian pulau terlebih dahulu. Kemudian setelah ditemukan pergerakan hingga terjebak, langsung print pergerakan tersebut ke layar dan hapus nilainya dari memory. Efek samping dari hal ini adalah waktu eksekusi yang cukup lama jika terdapat sangat banyak kombinasi pergerakan.