Laporan

Tugas Kecil 2 IF2211 Strategi Algoritma

Penyelesaian *Lonely Island* dengan Algoritma *Decrease and Conquer*

oleh

Abel Stanley / 13517068

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2019

BAB I

ALGORITMA *Decrease and Conquer*

1.1 Lonely Island

Terdapat sebuah negara yang terdiri atas banyak pulau. Pulau-pulau tersebut, terhubung dengan jembatan-jembatan yang hanya bisa digunakan untuk satu arah. Jika sebuah jembatan menghubungkan pulau a dan pulau b, maka jembatan tersebut hanya bisa digunakan untuk menyeberang dari pulau a ke pulau b, dan tidak sebaliknya. Jika sedang berada di pulau a, maka bisa dipilih sebuah pulau lain secara random yang terhubung langsung dengan pulau a melalui jembatan satu arah, dan melakukan gerakan berpindah ke pulau tersebut. Pemain akan terjebak pada satu pulau jika tidak bisa berpindah ke pulau lain, karena dalam persoalan ini, setelah meninggalkan pulau maka tidak bisa kembali ke pulau yang ditinggalkan. Berdasarkan ketentuan adanya pulau dan jembatan yang menghubungkannya, maka buatlah suatu program dengan pendekatan Decrease and Conquer untuk menentukan di pulau mana pemain bisa terjebak.

1.2 Algoritma *Decrease and Conquer*

Algoritma *Decrease and Conquer* yang saya terapkan adalah bertipe *decrease by constant value* yaitu '1'. Algoritma yang diterapkan adalah *Depth-First Search* (DFS). DFS dipilih di kasus ini karena lebih optimal dibandingkan *Breadth-First Search* (BFS). Hal ini dikarenakan target yang dicari program adalah *Lonely Islands* atau pulau buntu yang bisa dikatakan *deadends* atau simpul tak bertetangga. Simpul-simpul yang tak bertetangga atau tak memiliki tetangga yang belum ditelusuri akan sangat lebih cenderung berada di posisi penelusuran yang lebih dalam pada graf sehingga algoritma DFS yang dipilih.

Berikut tahap-tahap proses algoritma yang dijalankan oleh program:

a. Pembacaan Input dari File Eksternal

Teks input dari file eksternal akan dibaca oleh program. Nama file eksternal dapat di-input oleh pengguna. Pembacaan dari file eksternal akan dilakukan oleh *scanner* yang di *import* dari *library* Java. Dari line pertama akan didapatkan banyaknya simpul, busur, dan simpul *starting point*. Setelah pembacaan line pertama, akan dilaksanakan pembacaan berturut-turut per line hingga seluruh file telah dibaca.

b. Pembangunan List of Adjacency Lists

Dari hasil pembacaan line per line dari scanner, akan dibentuk List of Adjacency Lists yang akan menunjukkan simpul apa saja yang terhubung atau bertetangga dengan suatu simpul. List ini akan memiliki nilai efektif sebanyak banyaknya pulau yang ada. Index dari list akan menunjukan kepemilikan dari Adjacency List yang tersimpan. Sebagai contoh, indeks 0 pada list jika dilewatkan ke *getter* maka akan didapatkan List of Adjacency dari pulau ke-1. List di-implementasi dengan menggunakan ArrayList yang bersifat dinamik karena banyaknya tetangga dari suatu pulau berbeda-beda dan list dinamik akan lebih menghemat *space* memory. Jika suatu simpul tidak memiliki tetangga atau suatu pulau yang merupakan lonely island, maka akan diisi dengan list kosong.

c. Decrease

Setelah program telah mempersiapkan List of Adjacency lists, akan dilaksanakan algoritma Depth-First Search yang bersifat rekursif. Program akan melakukan fungsi rekursif yang akan menelusuri dari suatu simpul awal menuju ke simpul terakhir yaitu lonely islands terlebih dahulu, baru mencoba kemungkinan-kemungkinan lainnya untuk enumerasikan kemungkinan-kemungkinan jalan yang mungkin diambil dan mencari lonely islands. Program akan membuat suatu Arraylist of integet global yang memiliki elemen sebanyak banyaknya simpul untuk menandakan pulau-pulau mana saja yang telah dikunjungi. Hal ini bertujuan agar tidak terjadi infinite loop pada graf yang siklik dan merupakan saat proses pencarian dikenakan *decrease* by *constant value* 1. Constant value 1 karena setiap pengunjungan pulau, akan ditambahkan angka pulau ke Arraylist of Integer e yang berarti pulau sudah ditandai bahwa telah dikunjungi, sehingga pencarian tidak dilakukan pada pulau itu lagi. Ini akan mengurangi *search space* sebanyak 1 elemen sehingga *decrease by constant value 1*.

d. Conquer

Akan dilakukan pengecekan pada fungsi rekursif sebagai basis dari rekursif tersebut, yaitu jika pulau tidak memiliki simpul yang bisa ditelusuri lagi (search space sudah habis dari list of adjacency) atau tidak ada simpul tetangga yang belum ditelusuri (belum ada di Arraylist pathChosen), maka akan menandakan bahwa pulau tersebut adalah lonely island. Jika didapatkan pulau yang bersifat lonely island, maka angka dari pulau tersebut (pulau ke-berapa) akan disimpan pada suatu list solusi dan fungsi rekursif akan berhenti.

e. Enumeration

Enumerasi untuk menampilkan semua jalur yang mungkin diambil dari *starting point* dilakukan bersamaan ketika algoritma decrease and conquer dilaksanakan. Penulisan akan dilakukan ketika ditemukan lonely islands dengan referensi pulau yang pernah ditelusuri dari Arraylist of integer pathChosen dan adjacency list dari simpul tersebut. Program akan melakukan penelusuran dari suatu simpul dengan menerapkan iterasi dari setiap simpul yang berada di adjacency list dari simpul yang sedang diamati. Dengan demikian, setiap jalur yang mungkin akan ditelusuri dan akan di print ke layar sebagai output.

f. **Output**

Penampilan output akan dilakukan yaitu enumerasi dari setiap langkah yang mungkin, solusi pulau-pulau yang lonely dengan pembacaan dari list solusi yang telah di sort, jumlah waktu yang dibutuhkan untuk mengeksekusi program yang dihitung menggunakan *library nanoTime* dari Java.

g. Threading

Stack size di java secara default bernilai kecil. Karena program menggunakan fungsi rekursif, maka akan disimpan digunakan banyak memori pada stack. Untuk menangani hal ini akan digunakan multithreading. Dibuat suatu thread baru bernama "Threading!" yang memiliki stacksize yang saya define sendiri, yaitu 8 MB. Dengan demikian, kasus simpul sangat banyak dapat ditangani.

BAB II

SOURCE CODE PROGRAM

2.1 *Source Code JAVA*

```
//Preprocessing libraries:
import java.util.Scanner;
import java.io.*;
import java.util.*;
import java.util.ArrayList; //Digunakan ArrayList yang bersifat dinamik agar manajemen memori lebih akurat dan mudah.
import java.util.Arrays;
import java.util.List;
import java.lang.Thread;

public class LonelyIsland implements Runnable {
// Global Variables----------------------------------------------------------------:
  //Global variable mencatat deadends yang ditemukan:
  public static ArrayList<Integer> deadEnds = new ArrayList<Integer>();
  // Mempersiapkan Matriks adjacency sebagai representasi graf.
  public static ArrayList<ArrayList<Integer>> AdjcMat = new ArrayList<ArrayList<Integer>>();
// Helper Functions----------------------------------------------------------------:
  //H.F. melakukan print enumerasi
  public static void printArray(ArrayList<Integer> arr){
    int neff = arr.size();
    for(int i=0; i<neff;i++){
      if (i== neff-1)
        System.out.print(arr.get(i));
      else
        System.out.print(arr.get(i)+">>");
    }
    System.out.println();
  }
  //H.F. melakukan deepcopy pada array solusi:
```

```
  public static void copySols(ArrayList<Integer> arr1,ArrayList<Integer> arr2){
    arr2.clear();
    for(int i=0; i<arr1.size();i++){
      arr2.add(arr1.get(i));
    }
  }
  //H.F. melakukan pengecekan apakah array pilihan mengandung nilai integer x
  public static boolean arrayContains(ArrayList<Integer> arr, int x){
    for(int i=0; i<arr.size();i++){
      if (arr.get(i)==x)
        return true;
    }
    return false;
  }
  //H.F. Scanner function untuk melakukan pembacaan file eksternal
  private static Scanner extracted(File file) throws FileNotFoundException {
    Scanner sc = new Scanner(file);
    return sc;
  }
// Decrease and Conquer Algorithm----------------------------------------------------------------:
  public static void DnCrecursion(int vtx, ArrayList<Integer> Adjclist, ArrayList<Integer> pathChosen ){
  // Jika ditemukan deadend:
    //printArray(pathChosen);
    boolean foundonce=false; //dummmy variable untuk mengecek apakah ditemukan minimal satu simpul yang cocok dan bertetangga.
    pathChosen.add(vtx);
    // Simpul bukan deadend:
    if (!(Adjclist.size()==0)){
      //ArrayList<Integer> tempsols = new ArrayList<Integer>();
      //copySols(pathChosen,tempsols);
      //Rekursif untuk semua simpul yang bertetangga dengan simpul vtx.
      for(int i=0;i<Adjclist.size();i++){
        //Jika belum traversed, maka akan dipilih:
        if (!arrayContains(pathChosen, Adjclist.get(i))){
            //Dilempar ke rekursif DnC lagi.
```

```
        foundonce=true; // Ditemukan simpul bertetangga yang cocok!
        DnCrecursion(Adjclist.get(i),AdjcMat.get(Adjclist.get(i)-1), pathChosen);
        //Reset array traversed agar tidak terjadi pengulangan pencetakan.
        //copySols(tempsols,pathChosen);
      }
     }
    }

   if(!foundonce){
     //Menulis hasil ke layar output:
     printArray(pathChosen);
     //Mencatat deadends yang ditemukan:
     //KONDISI: simpul deadends tidak boleh duplikat!
     if (!arrayContains(deadEnds, vtx)){
       deadEnds.add(vtx);
     }
   }
   pathChosen.remove((Integer)vtx);
 }
 public static void CallMainAlgorithm(int vtx, int totalvtx){
   //Fungsi pemanggil rekursif agar pemanggilan di mainprogram lebih sederhana.
   ArrayList<Integer> pathChosen = new ArrayList<Integer>();
   //Penampilan hasil algoritma:
   System.out.println("-----------------------------");
   System.out.println("Enumerations: ");
   long startTime = System.nanoTime(); // Menghitung waktu eksekusi algorithm
   DnCrecursion(vtx, AdjcMat.get(vtx-1),pathChosen); // MAIN ALGORITHM
   long stopTime = System.nanoTime(); // Stop hitung.
   System.out.println("-----------------------------");
   System.out.println("Deadends adalah:");
   Collections.sort(deadEnds);//Sort deadEnds list.
   System.out.println(Arrays.deepToString(deadEnds.toArray()));
   System.out.println("-----------------------------");
   System.out.print("Execution Time is ...");
   System.out.print( (double)(stopTime - startTime) / 1000000000);System.out.println(" seconds!");
```

```
}

// Threading-------------------------------------------------------------:
public static void main(String[] args) throws Exception {
  new Thread (null, new LonelyIsland(), "Threading!", 1 << 26).start();
}
// Main Program----------------------------------------------------------:
public void run(){
  // Mempersiapkan file input dan scanner untuk proses pembacaan dari file.
  //Meminta input filename dari user:
  System.out.println("LONELY ISLANDS");
  Scanner uinput = new Scanner(System.in);
  System.out.println("Please enter your desired filename: (Without \" \", but include extension!)");
  String filename= uinput.nextLine();
  uinput.close();
  //ERROR CATCHING (FILE EXTERNAL):
  try{
    //Memanggil filename:
    File file = new File(filename);
    Scanner sc = extracted(file);
    String[] temp = sc.nextLine().split("\\s+"); // memisahkan whitespaces.
    int vtxnum = Integer.parseInt(temp[0]); //vtxnum -> banyaknya simpul
    //int edgnum = Integer.parseInt(temp[1]); //edgnum -> banyaknya busur
    int vstart =  Integer.parseInt(temp[2]); //vstart -> starting point


    //Pembacaan file dilakukan & Mengisi adjacency Matrix graf:
    // Mempersiapkan Matriks adjacency sebagai representasi graf.
    for(int i=0;i<vtxnum;i++){
      AdjcMat.add(new ArrayList<Integer>());
    }
    while (sc.hasNextLine()) {
      temp = sc.nextLine().split("\\s+"); // memisahkan whitespaces.
      AdjcMat.get(Integer.parseInt(temp[0])-1).add(Integer.parseInt(temp[1]));
    }
```

```
      sc.close();


      //Penampilan Adjacency Matriks:
      /*
      System.out.println("Adjacency matriks: ");
      for(int i=0;i<vtxnum;i++){
        System.out.println(Arrays.deepToString(AdjcMat[i].toArray()));
      }*/
      CallMainAlgorithm(vstart, vtxnum); // MAIN ALGORITHM!
    }
    catch (FileNotFoundException notfound){
      //notfound.printStackTrace();
      System.out.println("Invalid File Name. Please stop playing around...");
    }
  }
}
```

BAB III

EKSPERIMEN

(Keterangan: Eksperimen dilakukan dengan Java pada O.S. Linux Ubuntu)

1

```
kuma@kuma-HP-ProBook-4430s:~/Documents/LonelyIsland$ java -Xmx1024m LonelyIsland
LONELY ISLANDS
Please enter your desired filename: (Without " ", but include extension!)
input.txt
-----------------------------
Enumerations:
1>>2>>4
1>>2>>5
1>>3>>4
1>>4
1>>5
-----------------------------
Deadends adalah:
[4, 5]
-----------------------------
Execution Time is ...0.026620132 seconds!
```

Test Case seperti spek

input.txt:

```
5 7 1
1 2
1 3
1 4
1 5
2 4
2 5
3 4
```

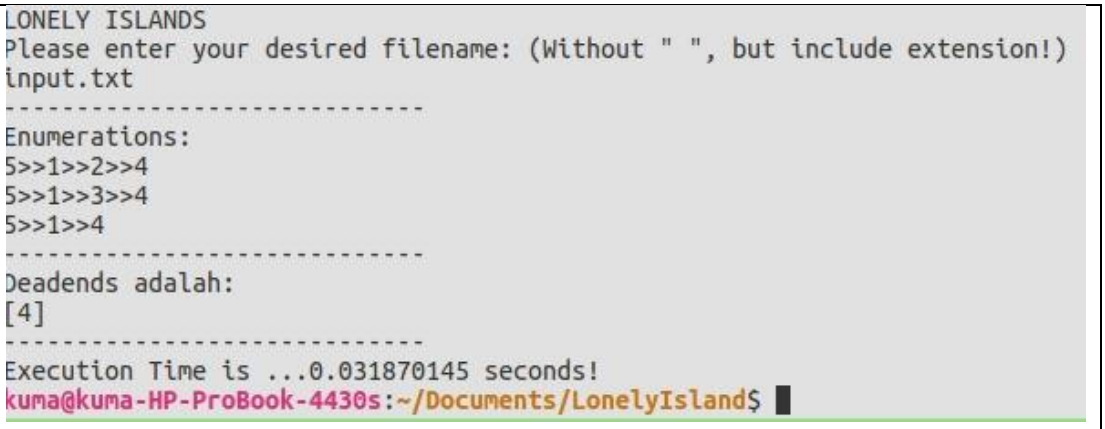
2

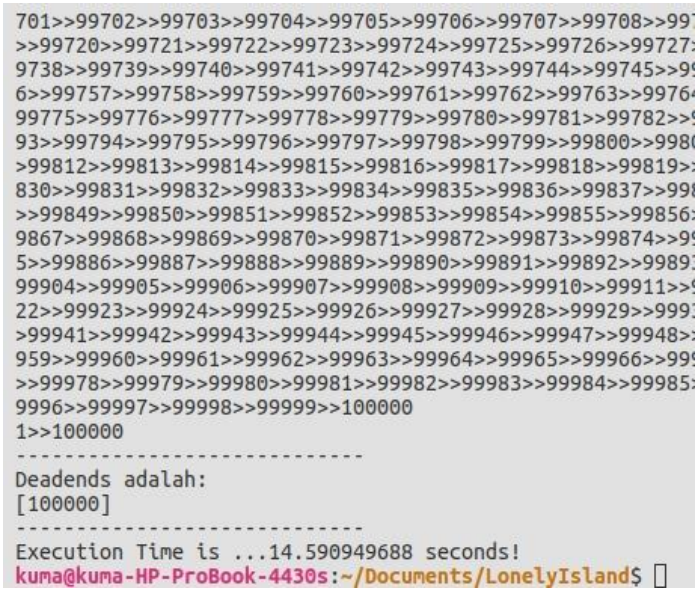
```
LONELY ISLANDS
Please enter your desired filename: (Without " ", but include extension!)
input.txt
-----------------------------
Enumerations:
1>>2>>4
1>>2>>5
1>>3>>4>>2>>5
1>>4>>2>>5
1>>5
-----------------------------
Deadends adalah:
[4, 5]
-----------------------------
Execution Time is ...0.034785961 seconds!
kuma@kuma-HP-ProBook-4430s:~/Documents/LonelyIsland$
```

Test Case graf bersifat siklik (4 bisa balik ke 2, 5 bisa balik ke 1)

input.txt:

```
5 9 5
1 2
1 3
1 4
1 5
2 4
2 5
3 4
5 1
4 2
```

3	 Test Case graf ada yang tak mungkin dicapai Input.txt berupa: 7 9 5 1 2 1 3 1 4 1 5 2 4 2 5 5 1 3 4 6 7 7 6
4	 Test Case graf ada <u>yang</u> tak mungkin dicapai & siklik input.txt: 7 9 5 1 2 1 3 1 4 1 5 2 4 2 5 3 4 5 1 4 2 6 7
5	



Test Case simpul 100k

input:
100000 100000 1
1 menuju setiap angka hingga 100000
1 100000

Keterangan Tambahan:

Untuk yang testcase yang memiliki simpul dalam jumlah besar, gunakan perintah java -Xmx1024m. Jika error heap memory, alokasikan lebih sesuai dengan RAM komputer yang menjalankan.

Test case batas atas yaitu 200k simpul dengan +/- 400k busur sudah dicoba dan tidak terjadi stack overflow error, namun memakan waktu yang sangat lama dan screenshot tidak sempat diambil.

BAB IV

REFERENSI

Sumber Tertulis:

http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/Tucil2-Stima-2019.pdf, diakses pada 23 Februari 2019 pukul 12.01 WIB.

https://en.wikipedia.org/wiki/Depth-first_search, diakses pada 24 Februari 2019 pukul 15.37 WIB.

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	✓	