MODUL VII

**IMPLEMENTASI GRAPH**

* 1. **TUJUAN**

Tujuan dari praktikum ini adalah:

1. Mahasiswa dapat mengilustrasikan suatu permasalahan ke dalam struktur data *graph*.
2. Mahasiswa dapat mengimplementasikan *graph coloring* pada permasalahan.
3. Mahasiswa dapat menyelesaikan permasalahan jalur tercepat dengan algoritma Dijkstra.
   1. **DASAR TEORI**
      1. **Algoritma Dijkstra**

Edsger Wybe Dijkstra, menemukan suatu algoritma untuk mencari lintasan terpendek pada suatu graf. Algoritma Dijkstra pada awalnya diterapkan pada graf berarah, tetapi ternyata algoritma ini juga benar untuk algoritma graf tak-berarah. Algoritma ini menggunakan prinsip *greedy* yang digunakan untuk menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke dalam himpunan solusi. Akan tetapi bobot dari graf tersebut haruslah bernilai bilangan positif (bobot>=0) [5].

Properti algoritma Dijkstra:

1. Matriks ketetanggaan M [mij]

mij = bobot sisi (i, j) (pada graf tak-berarah mij = mji )

mii = 0 mij = ∞, jika tidak ada sisi dari simpul i ke simpul j

2. Larik S = [si] yang dalam hal ini,

si = 1, jika simpul i termasuk ke dalam lintasan terpendek

si = 0, jika simpul i tidak termasuk ke dalam lintasan terpendek

3. Larik/tabel D = [di] yang dalam hal ini,

di = panjang lintasan dari simpul awal s ke simpul i

Algoritma Lintasan Terpendek Dijkstra :

(Mencari lintasan terpendek dari simpul a ke semua simpul lain}

1. Langkah 0 (inisialisasi):

Inisialisasi = 0 dan di = mai untuk i = 1, 2, ..., n

1. Langkah 1:
2. isi sa dengan 1 (karena simpul a adalah simpul asal lintasan terpendek, jadi sudah pasti terpilih)
3. isi da dengan ∞ (tidak ada lintasan terpendek dari simpul a ke a)
4. Langkah 2, 3, ... , n-1:
5. Cari j sedemikian sehingga sj = 0 dan dj = min{d1, d2, ..., dn} – isi sj dengan 1
6. Perbarui di, untuk i = 1, 2, 3, …, n

dengan: di (baru) = min{di (lama), dj + mji }.

*Input* algoritma ini adalah sebuah *weighted directed graph* G dan sebuah *source vertex* s dalam G. V adalah himpunan semua *vertices* dalam *graph* G. Setiap *edge* dari *graph* ini adalah pasangan *vertices* (u,v) yang melambangkan hubungan dari *vertex* u ke *vertex* v. Himpunan semua *edge* disebut E. *Weights* dari *edges* dihitung dengan fungsi w: E → [0, ∞); jadi w (u,v) adalah jarak non-negatif dari *vertex* u ke *vertex* v. *Cost* dari sebuah *edge* dapat dianggap sebagai jarak antara dua *vertex*, yaitu jumlah jarak semua *edge* dalam *path* tersebut. Untuk sepasang *vertex* s dan t dalam V, algoritma ini menghitung jarak terpendek dari s ke t [6].

Berikut *pseudocode* algoritma Dijkstra:

|  |
| --- |
| function Dijkstra(G, w, s)  for each vertex v in V[G]  d[v] := infinity  previous[v] := undefined  d[s] := 0  S := empty set  Q := V[G]  while Q is not an empty set  u := Extract\_Min(Q)  S := S union {u}  for each edge(u,v) outgoing from u  if d[u] + w(u,v) < d[v]  d[v] := d[u] + w(u,v)  previous[v] := u |

Contoh implementasi *graph coloring* dalam kehidupan sehari-hari yaitu:

1. Kebun raya adalah insstitusi yang memegang dokumentasi mengenai lokasi tumbuhan. Kebun raya Purwodadi terletak di Jalan Raya Surabaya Malang, KM. 65, Desa Purwodadi, Kecamatan Purwodadi, Kabupaten Pasuruan. Kebun Raya Purwodadi memiliki luas mencapai 85 hektar. Kebun Raya Purwodadi memiliki koleksi tanaman 2002 jenis/spesies, 178 suku/family, 962 marga/genus dan 11.669 *specimen*. Dengan jumlah tanaman yang begitu banyak, dibutuhkan aplikasi yang dapat menunjukkan jalan dari lokasi pengguna ke lokasi tanaman yang dituju. Dalam pembuatan aplikasi, dibutuhkan suatu metode/algoritma untuk melakukan perhitungan guna mendapatkan jarak terdekat. Algoritma yang digunakan yaitu algoritma dijkstra yang dipilih karena memiliki waktu *running time* lebih cepat dibandingkan algoritma *Bellman-Ford.* Untuk merancang aplikasi yang dibutuhkan, tahap identifikasi kebutuhan fungsional berdasarkna kebutuhan dari pengunjung kebun raya. Sedangkan untuk kebutuhan non-fungsional adalah tentang *usability* dan *compatibility*. Implementasi yang dibuat berdasarkan perancangan yang telah dibuat sebelumnya. *Web server* dibangun menggunakan bahasa PHP, sedangkan aplikasi *android* menggunakan bahasa Java dengan *tools android* studio. Pada pengujiannya dilakukan secara *black-box* untuk menguji algoritma yang digunakan. Selain itu dilakukan pengujian *usability* dan menunjukkan hasil yang memuaskan dengan presentase sebesar 70,916% dengan jumlah responden sebanyak 30 orang.
2. Dalam kehidupan sehari-hari unuk menuju ke suatu tempat kita sering mengalami kebingungan, bingung untuk memilih jalan mana yang lebih cepat sampai ke tempat tersebut. Untuk menuju ke suatu tempat yang belum pernah dikunjungi, dapat menggunakan *map* kota *tersebut.* Tetapi dengan menggunakan *map* sering kebingungan menentukan jalan mana yang tercepat menuju ketempat tersebut. Biasanya alat bantu pencarian suatu tempat hanya berupa garis-garis yang sering membingungkan karena garis tersebut seringkali berbeda dengan bentuk jalan yang sebenarnya dan juga kita sering dibingungkan dengan rute jalan yang tidak kita inginkan, yang seharusnya kita ingin melewati jalan raya tetapi ditunjukkan dengan jalan sempit yang tidak bisa dilewati oleh kendaraan roda empat. Semua itu terjadi dikarenakan alat penunjuk jalan tersebut hanya menampilkan garis-garis saja yang tidak menampilkan keadaan jalan yang sebenarnya. Pencarian rute terpendek dalam ilmu matematika adalah jarak terpendek atau *shortest path*. Pencarian jarak terdekat dengan menggunakan notasi dan simpul, notasi digambarkan sebagai tempat atau tujuan dan simpul digambarkan sebagai jalannya. Didalam ilmu pengetahuan ada banyak algoritma untuk melakukan pencarian lintasan terpendek ini seperti algoritma Dijkstra.
   * 1. **Pewarnaan *Graph***

Pewarnaan graf adalah metode pemberian warna pada *vertex*, *edges*, maupun wilayah dalam suatu graf. Tujuan pemberian warna untuk mencari wilayah tetangga yang ada pada graf. Pewarnaan graf terbagi menjadi 3 macam, yaitu Pewarnaan sisi (*edge colouring*) [4], pewarnaan wilayah (*region colouring*), pewarnaan simpul (*vertex coloring*). Pewarnaan sisi (*edge colouring*) merupakan pemberian warna pada setiap sisi pada graf sampai sisi-sisi yang saling berhubungan tidak memiliki warna yang sama. Pewarnaan wilayah (*region colouring*) adalah pemberian warna pada setiap wilayah di graf sehingga tidak ada wilayah yang bersebelahan memiliki warna yang sama. Pewarnan simpul (*vertex colouring*) adalah pemberian warna pada setiap simpul (*vertex*) dimana warna yang sama akan diberikan pada *vertex* yang saling bertetangga [4].

|  |
| --- |
| a) b)  c) |

Gambar 7.1 a) Pewarnaan Titik, b) Pewarnaan Wilayah, c) Pewarnaan sisi

Contoh implementasi *graph coloring* dalam kehidupan sehari-hari yaitu :

1. Pewarnaan graf khususnya pewarnaan simpul banyak diterapkan dalam menyelesaikan permasalahan. Salah satunya yaitu menyelesaikan permasalahan permainan sudoku. Permainan teka-teki angka berbasis logika ini bertujuan untuk mengisikan angka dengan aturan dalam setiap baris, kolom dan subgrid tidak ada angka yang sama. Penelitian ini menentukan solusi permainan sudoku internasional yang berbentuk bujur sangkar (𝑛2×𝑛2) dengan 𝑛=3 atau ukuran *grid*9×9. Diawali dengan merepresentasikan sel menjadi simpul dan relasi ketetanggaan sel-sel menjadi sisi sehingga terbentuk graf sudoku(𝑋𝑛). Graf tersebut berhubungan dengan pewarnaan parsial dari graf. Simpul-simpul pada graf Sudoku diwarnai sesuai dengan kaidah bilangan kromatik. Dalam mewarnai graf sudokuter dapat banyak kemungkinan solusi penyelesaian sehingga digunakan algoritma runut balik (*backtracking*). Algoritma ini berfungsi untuk memecahkan suatu permasalahan dengan banyak kemungkinan solusi secara bertahap. Tahap pencarian solusi dibuat dalam pohon berakar yang berbasis pada DFS (*Dept First Search*). Cara kerja algoritma *backtracking* dimulai dengan menentukan fungsi pembatas kemudian membangkitkan solusi. *Backtracking* dilakukan jika terjadi *dead node,* sehingga proses pencarian akan kembali ke simpul hidup terdekat. Kemudian simpul tersebut membangkitkan simpul anak dengan solusi yang baru. Proses pencarian solusi dilakukan secara manual dan rancang bangun menggunakan bahasa pemrograman VB.Net 2015. Berdasarkan hasil penelitian, diperoleh representasi permainan sudoku yaitu graf sudoku dengan *rank* 3 (𝑋3). Graf sudoku tersebut diselesaikan dengan pewarnaan graf yang memiliki bilangan kromatik yaitu 𝑛2 serta menerapkan algoritma backtracking yang dilakukan secara manual sehingga diperoleh solusi penyelesaian tunggal. Graf sudoku ini juga diselesaikan melalui rancang bangun berupa aplikasi pemecahan permainan sudoku dengan 2≤𝑛≤5 dan diperoleh solusi penyelesaian tunggal.
2. Penjadwalan kuliah merupakan hal terpenting dalam kegiatan pembelajaran. Memiliki jadwal kuliah yang baik, berarti memiliki distribusi mata kuliah yang merata setiap hari tanpa mengalami kendala proses perkuliahan. Dalam membuat jadwal kuliah tidak mudah. Beberapa aspek yang mempengaruhi yaitu penyusunan jadwal perkuliahan, mata kuliah, dan slot waktu. Setiap aspek memiliki kelemahan dalam penyusunan jadwal. Untuk mengatasi masalah tersebut, menggunakan sistem terkomputerisasi untuk simulasi jadwal kuliah. Sistem menerima semua permasalahan untuk menghasilkan solusi terbaik dalam menetapkan jadwal kuliah. Dalam penjadwalan kuliah, membutuhkan model untuk memecahkan masalah. Menetukan model yang tepat, dapat dilakukan dalam bentuk identifikasi masalah, analisis ruang lingkup masalah, dan identifikasi variabel yang terlibat dalam pengambilan keputusan. Masalah yang dihadapi dalam penjadwalan kuliah adalah masalah kompleks yang harus dihadapi secara teratur. Aspek penyusunan jadwal kuliah, terhubung dengan aspek yang lain. Penjadwalan perkuliahan dilakukan setiap semester dengan lingkup berbeda dari masalah dalam setiap semester. Penyusunan *slSPot* waktu dilalui berdasarkan penjadwalan mata kuliah, sehingga mendapatkan jalur terpendek yang optimal. Solusi yang dibutuhkan adalah penggabungan *Vertex* *Graph* *Coloring* dan *Simulated* *Annealing*. Tujuannya adalah untuk mendapatkan penjadwalan yang terstruktur dan optimal.
   1. **PERMASALAHAN**
3. Buatlah sebuah *graph* berdasarkan gambar berikut :

|  |
| --- |
|  |

Gambar 7.2 Permasalahan modul 7

Sekolompok musisi yang semakin melejit kini kian tersohor sampai ke pulau Lombok. Tak dapat dipungkiri semenjak perilisan *single* terbaru mereka yang berjudul “Berak Tak Cebok”, Penggemar KUFAKU BAND semakin membeludak khususnya dari mahasiswa angkatan 2018 Informatika, Universitas Mataram. Tak tanggung-tanggung, para calon sarjana computer ini hendak menggelar *show tour* untuk KUFAKU BAND di berbagai kota di Lombok.

* + - 1. Namun, mereka cukup kebingungan dalam menentukan lokasi-lokasi tepat untuk digelarnya *show* dan memilih kota paling tepat untuk *Grand Opening*. Agar KUFAKU BAND dapat mengelilingi pulau dengan cepat, *show* selanjutnya akan di gelar melewati tepat satu kota dari *show* sebelumnya (lalang satu). Sehingga mereka membuat sebuah program yang mendata 14 kota di pulau Lombok lalu dapat mencari kota terbaik untuk *Grand Opening* (berdasarkan bagaimana kota tersebut dapat mencapai kota-kota lain paling banyak dalam rutenya namun jarak terpendek jika harus lompat/lalang satu kota untuk berpindah). Lalu menentukan rutenya dengan memilih kota-kota (lalang satu) terdekat dari *show* sebelumnya.

Batasan :

1. Rute memutar terpendek.
2. *Show* sebelum ke *show* selanjutnya harus melompati satu kota.
3. Kota *Show* sebelum ke kota show selanjutnya tidak berjarak 100 km keatas.
4. Setidaknya *show* diadakan di 5 kota.
   1. **HASIL DAN PEMBAHASAN**
      1. **Program *Show Tour***
5. Algoritma
6. Menambahkan daftar *vertex* yang terdapat dalam *graph*.
7. Menginisialisasikan *edge* atau garis yang menghubungi antara suatu *vertex* dengan *vertex* lainnya dan memberikan jaraknya.
8. Menentukan derajat dari setiap *vertex* pada *graph* yang telah dimasukkan sebelumnya dengan tujuan menentukan *vertex* pertama yang akan dilakukan *graph coloring*.
9. Melakukan *graph coloring* pada *graph* yang sudah terdaftar sesuai dengan banyaknya derajat dari setiap *graph*.
10. Menentukan *vertex* yang menjadi tempat *grand opening* KUFAKU BAND dengan memeriksa *vertex* dengan derajat terbanyak dan memiliki jarak terdekat untuk mengadakan konser dengan menggunakan algoritma Dijkstra.
11. Menampilkan setiap kota beserta jarak yang akan ditempuh selama proses *tour* KUFAKU BAND.
12. Menghitung dan menampilkan total jarak yang akan ditempuh.
13. *Source Code*

|  |
| --- |
| import java.util.Arrays;  class Vertex{  int label;  String kota;  Vertex next;  Edge edge;  public Vertex(int label, String kota) {  this.label = label;  this.kota = kota;  }  }  class Edge{  int weight;  Vertex end;  Edge nextEdge;  public Edge (Vertex end, int weight){  this.end = end;  this.weight = weight;  }  }  class node{  int number;  Vertex data;  node next;  node prev;  public node (Vertex data){  this.data = data;  }  public node(int number) {  this.number = number;  }  }  class linklist{  node head;  node tail;  boolean isEmpty(){  return head == null;  }  void addfirst(Vertex data){  node baru = new node(data);  if(head == null){  head = baru;  }else{  baru.next = head;  head = baru;  }  }  void add(Vertex data){  node baru = new node(data);  if(head == null) {  head = tail =baru;  }else{  tail.next = baru;  tail = baru;  }  }  public void remove(int currentdata){  node current = head;  node prev = null;  if(current.number == currentdata){  head = head.next;  return;  }  while(current != null && current.number != currentdata){  prev = current;  current = current.next;  }  if(current == null){  return;  }  prev.next = current.next;  }  void addpriority(int data){  node baru = new node(data);  if (head == null) {  head = tail = baru;  return;  }  node bantu = head;  while(bantu != null){  if(head.number > baru.number){  baru.next = head;  head.prev = baru;  head = baru;  return;  }else if(tail.number <= baru.number){  tail.next = baru;  baru.prev = tail;  tail = baru;  return;  }else if(baru.number >= bantu.number && baru.number < bantu.next.number ){  baru.next = bantu.next;  baru.prev = bantu;  bantu.next.prev = baru;  bantu.next = baru;  return;  }  bantu = bantu.next;  }  }  void addpriority(Vertex data){  node baru = new node(data);  if (head == null) {  head = tail = baru;  return;  }  node bantu = head;  while(bantu != null){  if(head.data.label > baru.data.label){  baru.next = head;  head.prev = baru;  head = baru;  return;  }else if(tail.data.label <= baru.data.label){  tail.next = baru;  baru.prev = tail;  tail = baru;  return;  }else if(baru.data.label >= bantu.data.label && baru.data.label < bantu.next.data.label ){  baru.next = bantu.next;  baru.prev = bantu;  bantu.next.prev = baru;  bantu.next = baru;  return;  }  bantu = bantu.next;  }  }  int removeint(){  int simpan = head.number;  if(head == null){  return -1;  }else if(head == tail){  head = null;  }else{  head = head.next;  }  return simpan;  }  Vertex remove(){  node simpan = head;  if(head == null){  return null;  }else if(head == tail){  head = null;  }else{  head = head.next;  }  return simpan.data;  }  }  class GraphJurnal{  Vertex head;  Vertex tail;  int v = 0;  void addVertex(int label,String kota){  Vertex baru = new Vertex(label,kota);  if(head == null){  head = tail = baru;  }else{  tail.next = baru;  tail = baru;  }  v++;  }  Vertex findver(int src){  Vertex cur = head;  while(cur != null){  if(cur.label == src){  break;  }  cur = cur.next;  }  return cur;  }    void addEdge(int src, int dest, int weight){  Vertex asal = findver(src);  Vertex tujuan = findver(dest);  Edge bantu = asal.edge;  if(bantu == null) {  asal.edge = new Edge(tujuan, weight);  }else{  while(bantu.nextEdge != null) {  bantu = bantu.nextEdge;  }  bantu.nextEdge = new Edge(tujuan, weight);  }  Edge bantu2 = tujuan.edge;  if(bantu2 == null){  tujuan.edge= new Edge(asal,weight);  }else {  while (bantu2.nextEdge != null) {  bantu2 = bantu2.nextEdge;  }  bantu2.nextEdge = new Edge(asal, weight);  }  }  int [] degree(){  int [] derajat = new int[v];  for(int i = 0; i < v; i++){  Vertex cur = findver(i);  Edge bantu = cur.edge;  while(bantu != null){  derajat[i] +=1;  bantu = bantu.nextEdge;  }  }  return derajat;  }  public int [] Graphcoloring() {  int[] result = new int[v];  Arrays.fill(result, -1);  boolean [] available = new boolean[v];  Arrays.fill(available, true);  result[0] = 0;  for(int i = 1; i<v; i++){  Vertex cur = findver(i);  Edge bantu = cur.edge;  while(bantu != null){  if(result[bantu.end.label] != -1){  available[result[bantu.end.label]] = false;  }  bantu = bantu.nextEdge;  }  int color = 0;  for(; color < v; color++){  if(available[color]){  break;  }  }  result[cur.label] = color;  Arrays.fill(available, true);  }  return result;  }  int [] djikstra(int src){  int [] jarak = new int[v];  int [] prev = new int[v];  linklist helper = new linklist();  boolean [] visited = new boolean[v];  for(int i = 0; i<v; i++){  if(i == src){  jarak [i] = 0;  }else{  jarak[i] = Integer.MAX\_VALUE;  }  prev[i] = -1;  }  int UkeV = 0; //jarak antar 2 node yang dicek (u - v)  Vertex asal = findver(src);  helper.addpriority(asal);  while(!helper.isEmpty()){  Vertex cur = helper.remove();  Edge bantu = cur.edge;  while(bantu != null){  UkeV = jarak[cur.label] + bantu.weight;  if(UkeV < jarak[bantu.end.label]){  jarak[bantu.end.label] = UkeV;  }  if(!visited[bantu.end.label]){  helper.addpriority(bantu.end);  prev[bantu.end.label] = cur.label;  }  bantu = bantu.nextEdge;  }  visited[cur.label] = true;  }  return jarak;  }  }  public class implementasigraph {  public static void main(String[] args) {  String[] namakota = {"Mataram", "Gerung", "Sekotong", "Praya", "Kopang", "Kuta",  "Selong", "Jerowaru", "Aik Mel", "Pringgabaya",  "Sambelia", "Sembalun", "Tanjung", "Bayan", "Gangga"};  GraphJurnal lombok = new GraphJurnal();  int[] jarak = new int[15];  int[] derajat;  int[] graphcolor;  linklist colorpick = new linklist();  for (int i = 0; i < 15; i++) {  lombok.addVertex(i, namakota[i]);  }  lombok.addEdge(2, 1, 51);  lombok.addEdge(0, 1, 18);  lombok.addEdge(12, 0, 41);  lombok.addEdge(12, 14, 24);  lombok.addEdge(14, 13, 45);  lombok.addEdge(11, 13, 37);  lombok.addEdge(13, 10, 43);  lombok.addEdge(10, 9, 38);  lombok.addEdge(9, 8, 23);  lombok.addEdge(8, 11, 42);  lombok.addEdge(9, 6, 27);  lombok.addEdge(8, 6, 21);  lombok.addEdge(6, 4, 30);  lombok.addEdge(6, 7, 35);  lombok.addEdge(7, 5, 43);  lombok.addEdge(4, 3, 20);  lombok.addEdge(4, 0, 33);  lombok.addEdge(3, 5, 26);  lombok.addEdge(3, 1, 30);  lombok.addEdge(3, 7, 51);  derajat = lombok.degree();  graphcolor = lombok.Graphcoloring();  int max = 0;  int indexmax = 0;  for (int i = 0; i < derajat.length; i++) {  if (derajat[i] > max) {  max = derajat[i];  indexmax = i;  }  }  for (int i = 0; i < lombok.v; i++) {  if (i != indexmax && graphcolor[i] == graphcolor[indexmax]) {  colorpick.addpriority(i);  }  }  jarak = lombok.djikstra(indexmax);  int indexmin = 0;  int totaltempuh = 0;  int jarakkota = 0;  Vertex cur = lombok.findver(indexmax);  System.out.println("---------Jalur Show Tour KUFAKU BAND --------");  System.out.println(cur.kota + " (" + jarak[indexmax] + " km) Grand Opeening ->");  while(true){  int min = Integer.MAX\_VALUE;  for(int i = 0; i<jarak.length; i++){  node bantu = colorpick.head;  while(bantu != null){  if(i == bantu.number){  if(jarak[i] < min){  min = jarak[i];  indexmin = i;  }  break;  }  bantu = bantu.next;  }  }  jarakkota = 0;  jarakkota += jarak[indexmin];  if(jarakkota> 100) break;  totaltempuh+=jarakkota;  cur = lombok.findver(indexmin);  System.out.print(cur.kota + " (" +jarakkota + " km) ->");  colorpick.remove(indexmin);  jarak = lombok.djikstra(indexmin);  }  System.out.println("Jarak yang ditempuh : " +totaltempuh);  System.out.println("FINISH");  }  } |

1. Hasil *Run*

|  |
| --- |
|  |

Gambar 7.3Tampilan program permasalahan modul 7

Pada **Gambar 7.3** membuktikan bahwa pada kasus diatas, merupakan implementasi *graph* yaitu *graph coloring* dan algoritma Dijkstra.

* 1. **ANALISA**
     1. **Program *Show Tour* KUFAKU BAND**

|  |
| --- |
| class Vertex{  int label;  String kota;  Vertex next;  Edge edge;  public Vertex(int label, String kota) {  this.label = label;  this.kota = kota;  }  } |

*Script* “class Vertex{” merupakan sebuah *method* yang digunakan untuk membuat kelas *vertex* yang menampung data berupa “int label; String kota; Vertex next; Edge edge;”. *Code* “int label;” digunakan untuk mendeklarasikan variabel “label” yang memiliki tipe data *integer.* Penggunaan tipe data *integer* dimaksudkan untuk menampung data yang berupa nilai pada program yang dibuat, “public Vertex(int label, String kota) {” berfungsi sebagai *method* yang akan dijalankan pada saat pembuatan objek yang kemudian dilakukan pemanggilan *method* ini melalui *object* yang telah dibuat dan membuat parameter yaitu “(int label, String kota)” yang dapat digunakan untuk memasukkan data pada objek yang memiliki tipe data “integer” dan “String”.

|  |
| --- |
| class Edge{  int weight;  Vertex end;  Edge nextEdge;  public Edge (Vertex end, int weight){  this.end = end;  this.weight = weight;  }  } |

*Script* “class Edge{” merupakan sebuah *method* yang digunakan untuk membuat kelas *edge* yang menampung data “int weight; Vertex end; Edge nextedge;”. “public Edge(vertex end, int weight)” berfungsi sebagai *constructor* atau *method* yang akan dieksekusi pada saat dilakukan sebuah pembuatan objek yang kemudia proses pemanggilan *method* akan dilakukan melalui *object* yang sudah dibuat atau dideklarasikan, dan pada *method* yang telah dibuat, diisikan parameter yaitu “vertex end, int weight” yang dapat digunakan untuk memasukkan data pada objek yang memiliki tipe data “vertex” dan “integer”.

|  |
| --- |
| class node{  int number;  Vertex data;  node next;  node prev;  public node (Vertex data){  this.data = data;  }  public node(int number) {  this.number = number;  }  } |

*Script* “class Node{” merupakan sebuah kelas *node* yang dapat menyimpan sebuah nilai dengan bentuk bilangan bulat dan terdapat sebuah *pointer* penunjukknya. “int data;” merupakan variabel untuk menyimpan data yang akan dimasukkan. “Node next” merupakan *pointer* yang berfungsi untuk menunjuk data selanjutnya dalam sebuah *linked list*, “Node prev” merupakan *pointer* yang berfungsi untuk menunjuk data sebelumnya dalam sebuah *linked list*.

|  |
| --- |
| class linklist{  node head;  node tail;  boolean isEmpty(){  return head == null;  } |

*Script* “class LinkList {” sebuah kelas yang nanti akan menyimpan *method*-*method* *link list* dan untuk mengakses setiap *method* yang berada di dalam kelas ini, maka cukup dilakukan penginisialisasian nama kelas pada fungsi utama. “node head,tail;” berguna untuk memberikan nilai *head* dan *tail* pada *node* yang yang sudah dibuat. “boolean isEmpty” difungsikan sebagai *method* yang digunakan untuk melakukan pengecekan kondisi, apabila *method* tersebut dipanggil.

|  |
| --- |
| public void remove(int currentdata){  node current = head;  node prev = null;  if(current.number == currentdata){  head = head.next;  return;  }  while(current != null && current.number != currentdata){  prev = current;  current = current.next;  }  if(current == null){  return;  }  prev.next = current.next;  } |

*Script* “void remove(int currentdata){” merupakan sebuah *method* yang digunakan untuk melakukan proses penghapusan data pada *graph*.Penghapusan dilakukan dengan cara memeriksa setiap data mulai dari data terdepan kemudian menuju data selanjutnya hingga seterusnya dan akan berhenti apabila data selanjutnya sudah bernilai *null* atau kosong. Proses penghapusan dilakukan dengan memeriksa kondisi data pertama, jika data pertama sama dengan data yang akan dihapus, maka data pertama dihapus. Jika data yang dihapus tidak sama dengan data pertama maka data diperiksa secara keseluruhan dan akan berhenti pada data yang akan dihapus.

|  |
| --- |
| void addpriority(int data){  node baru = new node(data);  if (head == null) {  head = tail = baru;  return;  }  node bantu = head;  while(bantu != null){  if(head.number > baru.number){  baru.next = head;  head.prev = baru;  head = baru;  return;  }else if(tail.number <= baru.number){  tail.next = baru;  baru.prev = tail;  tail = baru;  return;  }else if(baru.number >= bantu.number && baru.number < bantu.next.number ){  baru.next = bantu.next;  baru.prev = bantu;  bantu.next.prev = baru;  bantu.next = baru;  return;  }  bantu = bantu.next;  }  } |

*Script* “public void addPriority(int data) {” merupakan sebuah *method* yang memberikan prioritas terhadap suatu data pada program yang dimana dilakukan pengurutan data dengan prioritas, data diurutkan mulai dari yang terkecil hingga ke yang terbesar. Pada proses pengurutan suatu data maka diperlukan suatu *statement* kondisi agar penempatan data sesuai dengan besar data yang diurutkan. Jika “head.number > baru.number”, maka penempatan data akan diletakkan sebelum *head*, pada kondisi ini digunakan *method* dengan konsep penambahan data di depan seperti *addfirst*. Jika “tail.number <= baru.number”, maka data akan diletakkan setelah *tail*, pada kondisi ini digunakan *method* dengan konsep penambahan data di belakang yang menggunakan prinsip seperti *addlast*. Dan jika kondisi pada *statement* tereksekusi yaitu “if (baru.number >= bantu.number && baru.number < bantu.next.priority) {”, maka data akan diletakkan diantara dua buah data dengan memeriksa kondisi data yang ada pada sebelum dan setelahnya, konsep yang digunakan pada *statement* ini adalah penambahan data di antara. Sehingga, dengan kondisi-kondisi demikian maka data akan dipetakkan sesuai dengan urutan dari yang terkecil hingga terbesar.

|  |
| --- |
| Vertex remove(){  node simpan = head;  if(head == null){  return null;  }else if(head == tail){  head = null;  }else{  head = head.next;  }  return simpan.data;  } |

*Script* “Vertex remove {” merupakan sebuah *method* yang digunakan untuk menghapus vertex pada *graph*.Penghapusan dilakukan dengan cara memeriksa setiap *vertex* mulai dari *vertex* paling awal. Kemudian, memeriksa kondisi, jika *head* sama dengan *null* maka *return null*. Jika *head* sama dengan *tail* maka *head* sama dengan *null.* Jika tidak memenuhi kondisi tersebut, maka *head* sama dengan *head dot next.* Penggunaan *head* *dot next* sendiri berguna untuk menunjuk data selanjutnya atau dalam artian lain digunakan untuk menelusuri *node* pada suatu *linked list*.

|  |
| --- |
| void addVertex(int label,String kota){  Vertex baru = new Vertex(label,kota);  if(head == null){  head = tail = baru;  }else{  tail.next = baru;  tail = baru;  }  v++;  } |

*Script* “void addVertex(int label, String kota){” merupakan sebuah *method* yang digunakan untuk menambahkan suatu data dengan cara melakukan pendeklarasian *object* yang bernama baru, dimana nilai dari pada variabel ini akan diberikan melalui variabel “label, kota” dengan tipe *integer* yang akan digunakan sebagai parameternya. Kemudian sebelum ditambahkannya data perlu suatu kondisi yang digunakan untuk memeriksa apakah data dalam kondisi kosong atau sudah terisi. Jika data dalam kondisi kosong maka “head=tail=baru;”. Jika data sudah terisi, maka setelah “Vertex” *tail* akan diberikan “Vertex” baru bernama “ace” dan *tail* berganti menjadi “baru”. Fungsi dari “v++;” adalah untuk mengukur panjang dari total banyak “Vertex” yang telah ditambahkan pada program.

|  |
| --- |
| Vertex findver(int src){  Vertex cur = head;  while(cur != null){  if(cur.label == src){  break;  }  cur = cur.next;  }  return cur;  } |

*Script* “Vertex findver(int src){” merupakan sebuah *method* yang digunakan untuk melakukan pencarian data dengan parameter data yang dicari. Kemudian, diinisialisasikan sebuah *vertex* yang digunakaan untuk menyimpan nilai *head* dan sebuah *vertex* yang digunakan untuk menyimpan data apabila data sudah ditemukan. Pada proses pencarian data, maka dilakukan sebuah perulangan selama data tersebut tidak sama dengan *null*, selama data selanjutnya tidak bernilai *null* dan proses pencarian belum menemukan apa yang dicari maka proses *looping* akan dilakukan secara terus menerus hingga yang dicari telah ditemukan. Apabila data pada *vertex* yang menyimpan nilai *head* sama dengan data yang dicari, maka *vertex* akan menyimpan nilai cari dan kemudian nilainya akan dikembalikan pada fungsi.

|  |
| --- |
| void addEdge(int src, int dest, int weight){  Vertex asal = findver(src);  Vertex tujuan = findver(dest);  Edge bantu = asal.edge;  if(bantu == null) {  asal.edge = new Edge(tujuan, weight);  }else{  while(bantu.nextEdge != null) {  bantu = bantu.nextEdge;  }  bantu.nextEdge = new Edge(tujuan, weight);  }  Edge bantu2 = tujuan.edge;  if(bantu2 == null){  tujuan.edge= new Edge(asal,weight);  }else {  while (bantu2.nextEdge != null) {  bantu2 = bantu2.nextEdge;  }  bantu2.nextEdge = new Edge(asal, weight);  }  } |

*Script* “void addEdge (int src, int dest, int weight){” merupakan *method* yang digunakan untuk melakukan proses penghubungan sebuah data dengan parameter data asal dan data tujuan. Kemudian, melakukan pemeriksaaan terhadap data yang terhubung pada data asal. Jika tidak terdapat data yang terhubung pada data asal, maka diberikan suatu *node* bantu yang difungsikan sebagai penghubung data asal dengan data yang akan dituju. Jika data sudah terdapat data yang terhubung dengan data asal, maka akan diperiksa data tersebut secara keseluruhan hingga terdapat data yang belum terhubung dengan data asal dan kemudian dihubungkan dengan data tujuan.

|  |
| --- |
| int [] degree(){  int [] derajat = new int[v];  for(int i = 0; i < v; i++){  Vertex cur = findver(i);  Edge bantu = cur.edge;  while(bantu != null){  derajat[i] +=1;  bantu = bantu.nextEdge;  }  }  return derajat;  } |

*Script* “int [] degree () {” merupakan sebuah *method* yang digunakan untuk melakukan proses perhitungan terhadap derajat pada suatu *graph*. Perhitungan ini dilakukan untuk menentukkan suatu titik awal yang dimana titik ini adalah titik yang akan dilalui dari sebuah *graph*. Pemeriksaan derajat dilakukan dengan cara memeriksa setiap *vertex* yang ada dan menghitungsetiap tetangga dari *vertex* tersebut.

|  |
| --- |
| public int [] Graphcoloring() {  int[] result = new int[v];  Arrays.fill(result, -1);  boolean [] available = new boolean[v];  Arrays.fill(available, true);  result[0] = 0;  for(int i = 1; i<v; i++){  Vertex cur = findver(i);  Edge bantu = cur.edge;  while(bantu != null){  if(result[bantu.end.label] != -1){  available[result[bantu.end.label]] = false;  }  bantu = bantu.nextEdge;  }  int color = 0;  for(; color < v; color++){  if(available[color]){  break;  }  }  result[cur.label] = color;  Arrays.fill(available, true);  }  return result;  } |

*Script* “public int [] Graphcoloring {” merupakan sebuah *method* yang digunakan untuk melakukan proses pemberian warna pada *graph*. Pewarnaan pada *graph* memiliki tujuan untuk memberikan tanda pada *graph* dan memeriksa *vertex-vertex* yang bertetangga serta *vertex-vertex* yang tidak bertetangga. Pewarnaan pada *graph* dilakukan dengan cara setiap tetangga. Apabila *vertex* satu bertetangga dengan *vertex* dua, maka harus diberikan warna yang berbeda. Proses pewarnaan ini bertujuan dikarenakan ada suatu kondisi dimana tetangga pada suatu vertex harus dilewati, ketika program *graph* ini melakukan penelusuran, maka diberikanlah warna untuk menandai sehingga tetangga pada suatu vertex terlewati.

|  |
| --- |
| int [] djikstra(int src){  int [] jarak = new int[v];  int [] prev = new int[v];  linklist helper = new linklist();  boolean [] visited = new boolean[v];  for(int i = 0; i<v; i++){  if(i == src){  jarak [i] = 0;  }else{  jarak[i] = Integer.MAX\_VALUE;  }  prev[i] = -1;  }  int UkeV = 0; //jarak antar 2 node yang dicek (u - v)  Vertex asal = findver(src);  helper.addpriority(asal);  while(!helper.isEmpty()){  Vertex cur = helper.remove();  Edge bantu = cur.edge;  while(bantu != null){  UkeV = jarak[cur.label] + bantu.weight;  if(UkeV < jarak[bantu.end.label]){  jarak[bantu.end.label] = UkeV;  }  if(!visited[bantu.end.label]){  helper.addpriority(bantu.end);  prev[bantu.end.label] = cur.label;  }  bantu = bantu.nextEdge;  }  visited[cur.label] = true;  }  return jarak;  } |

*Script* “int [] djikstra(int src){” merupakan sebuah *method* yang digunakan untuk melakukan penentuan jalur terpendek yang akan dilalui dengan menggunakan algoritma Dijkstra. Algoritma ini memeriksa setiap *vertex* dan menghitung jarak yang dilalui, algoritma Dijkstra sendiri merupaka *greedy algorithm* atau algoritma rakus. Algoritma ini akan melakukan pengecekan data jarak pada setiap *vertex*, algoritma ini akan berusaha mencari jalur terpendek secara terus menerus hingga ditemukan jalur yang memang paling pendek yang menghubungkan *graph*. Namun, tidak semua *vertex* selalu dikunjungi setiap kali. *Vertex* yang sudah dikunjungi dan sudah memiliki jarak terpendek maka akan ditandai dan tidak akan dikunjungi kembali. Sehingga dari algoritma tersebut diperlukan suatu variabel dengan tipe data yaitu *boolean* yang berfungsi untuk memeriksa kondisi darisetiap data dan memberikan kondisi *true* atau *false* pada data tersebut.

|  |
| --- |
| public static void main(String[] args) {  String[] namakota = {"Mataram", "Gerung", "Sekotong", "Praya", "Kopang", "Kuta",  "Selong", "Jerowaru", "Aik Mel", "Pringgabaya",  "Sambelia", "Sembalun", "Tanjung", "Bayan", "Gangga"};  GraphJurnal lombok = new GraphJurnal();  int[] jarak = new int[15];  int[] derajat;  int[] graphcolor;  linklist colorpick = new linklist();  for (int i = 0; i < 15; i++) {  lombok.addVertex(i, namakota[i]);  }  lombok.addEdge(2, 1, 51);  lombok.addEdge(0, 1, 18);  lombok.addEdge(12, 0, 41);  lombok.addEdge(12, 14, 24);  lombok.addEdge(14, 13, 45);  lombok.addEdge(11, 13, 37);  lombok.addEdge(13, 10, 43);  lombok.addEdge(10, 9, 38);  lombok.addEdge(9, 8, 23);  lombok.addEdge(8, 11, 42);  lombok.addEdge(9, 6, 27);  lombok.addEdge(8, 6, 21);  lombok.addEdge(6, 4, 30);  lombok.addEdge(6, 7, 35);  lombok.addEdge(7, 5, 43);  lombok.addEdge(4, 3, 20);  lombok.addEdge(4, 0, 33);  lombok.addEdge(3, 5, 26);  lombok.addEdge(3, 1, 30);  lombok.addEdge(3, 7, 51); |

*Script* “String[] namakota” merupakan sebuah *array* yang duganakan untuk menyimpan nama kota-kota yang dapat dipilih untuk melakukan *tour*. Kemudian “lombok.addVertex(i, namakota[i]);”digunakan untuk membuat sebuah *vertex* dimana berisini nama dari kota-kota yang sudah diinput sebelumnya. Pada fungsi *main* ini dilakukan pemanggilan *method-method* yang telah dibuat, dan dilakukan pendeklarasian *object* untuk memanggil *method-method* tersebut.

|  |
| --- |
| derajat = lombok.degree();  graphcolor = lombok.Graphcoloring();  int max = 0;  int indexmax = 0;  for (int i = 0; i < derajat.length; i++) {  if (derajat[i] > max) {  max = derajat[i];  indexmax = i;  }  }  for (int i = 0; i < lombok.v; i++) {  if (i != indexmax && graphcolor[i] == graphcolor[indexmax]) {  colorpick.addpriority(i);  }  } |

*Script* “derajat = lombok.degree();” merupakan *script* yang digunakan untuk mencari *vertex* yang mempunyai derajat paling banyak diantara *vertex* yang ada. “graphcolor = lombok.Graphcoloring();” merupakan pemanggilan *method* untuk melakukan pewarnaan pada *vertex*. Kemudian melakukan pewarnaan pada tetangga-tetangga dari *vertex* dengan warna yang berbeda dan sesuai dengan kaidah pada *coloring graph*.

|  |
| --- |
| jarak = lombok.djikstra(indexmax);  int indexmin = 0;  int totaltempuh = 0;  int jarakkota = 0;  Vertex cur = lombok.findver(indexmax);  System.out.println("---------Jalur Show Tour KUFAKU BAND --------");  System.out.println(cur.kota + " (" + jarak[indexmax] + " km) Grand Opeening ->");  while(true){  int min = Integer.MAX\_VALUE;  for(int i = 0; i<jarak.length; i++){  node bantu = colorpick.head;  while(bantu != null){  if(i == bantu.number){  if(jarak[i] < min){  min = jarak[i];  indexmin = i;  }  break;  }  bantu = bantu.next;  }}  jarakkota = 0;  jarakkota += jarak[indexmin];  if(jarakkota> 100) break;  totaltempuh+=jarakkota;  cur = lombok.findver(indexmin);  System.out.print(cur.kota + " (" +jarakkota + " km) ->");  colorpick.remove(indexmin);  jarak = lombok.djikstra(indexmin);  }  System.out.println("Jarak yang ditempuh : " +totaltempuh);  System.out.println("FINISH");  }} |

*Script* “jarak = lombok.djikstra(indexmax);” digunakan untuk mencari jarak terpendek dari rute-rute yang ada pada setiap kota yang dilakukan *tour*. Pada proses pencarian rute terpendek dilakukanlah pengecekan terhadap kota berikutnya yang akan dikunjungi dimana jaraknya tidak lebih dari 100km. *Script* “System.out.print (cur.kota + " (" +jarakkota + " km) ->");” digunakan untuk menampilkan kota yang dikunjungi selanjutnya beserta jaraknya dan kemudian diakhiri oleh program yang akan menampilkan jarak dari total keseluruhan dari kota yang telah dikunjungi.

* 1. **KESIMPULAN**

Berdasarkan praktikum yang sudah dilaksanakan, dapat diambil beberapa kesimpulan, yaitu sebagai berikut:

* + - * 1. Permasalahan pada suatu *graph* dapat diilustrasikan dengan cara melihat setiap hubungan dari setiap *vertex*. Hubungan pada *vertex-vertex* tersebut dapat dijadikan sebagai patokan untuk menyelesaikan permasalahan yang ada pada sebuah *graph* di dalam sebuah program.
        2. *Graph coloring* diimplementasikan untuk memberikan warna pada setiap *vertex*. Warna yang diberikan pada suatu *vertex* harus berbeda dengan warna *vertex* tetangga. Penggunaan *graph coloring* sendiri sangat bermanfaat, apabila program meminta bahwa *vertex* tetangga harus dilewati oleh tetangganya dan tidak dapat dikunjungi.
        3. Suatu permasalahan *graph* dalam menentukan suatu jalur tercepat, dapat diselesaikan dengan menggunakan algoritma Dijkstra. Algoritma Dijkstra adalah sebuah algoritma yang dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah [*graph*](https://id.wikipedia.org/wiki/Graf) berarah (*directed graph).* Algoritma Dijkstra ini bekerja dengan cara memeriksa setiap jarak pada *vertex* dan memperbarui jarak tersebut apabila menemukan jarak yang lebih cepat.