PRAKTIKUM 32
GRAPH
ALGORITMA DIJKSTRA

A. TUJUAN PEMBELAJARAN

1. Memahami konsep dari Algoritma Dijkstra
2. Memahami cara mengimplementasikan algoritma Dijkstra ke dalam bahasa pemrograman Java.

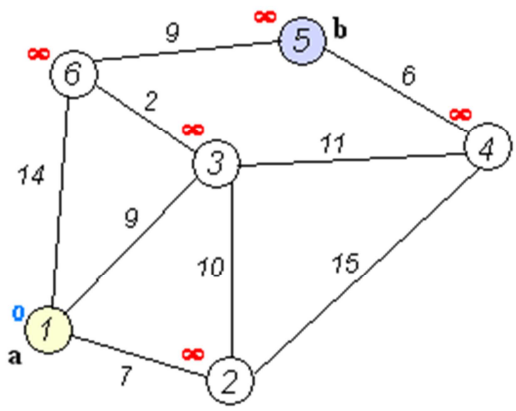
B. DASAR TEORI

Lintasan Terpendek

Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot (weighted graph), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya. Asumsi yang kita gunakan di sini adalah bahwa semua bobot bernilai positif. Kata terpendek berbeda-beda maknanya bergantung pada tipikal persoalan yang akan diselesaikan. Namun, secara umum terpendek berarti meminimisasi bobot pada suatu lintasan dalam graf.

ALGORITMA DIJKSTRA

Algoritma ini bertujuan untuk menemukan jalur terpendek berdasarkan bobot terkecil dari satu titik ke titik lainnya. Misalkan titik mengambarkan gedung dan garis menggambarkan jalan, maka algoritma Dijkstra melakukan kalkulasi terhadap semua kemungkinan bobot terkecil dari setiap titik.



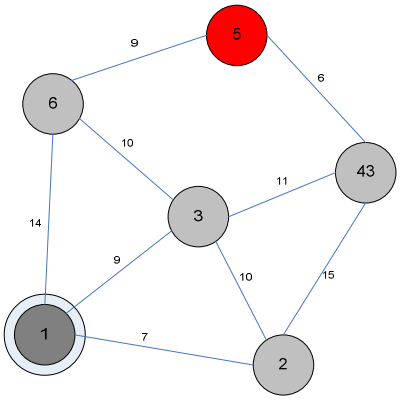
Gambar 32.1 Contoh keterhubungan antar titik dalam algoritma Dijkstra

Pertama-tama tentukan titik mana yang akan menjadi node awal, lalu beri bobot jarak pada node pertama ke node terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap. Inilah urutan logika dari algoritma Dijkstra:

1. Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (belum terisi)
2. Set semua node “Belum terjamah” dan set node awal sebagai “Node keberangkatan”
3. Dari node keberangkatan, pertimbangkan node tetangga yang belum terjamah dan hitung jaraknya dari titik keberangkatan. Sebagai contoh, jika titik keberangkatan A ke B memiliki bobot jarak 6 dan dari B ke node C berjarak 2, maka jarak ke C melewati B menjadi 6+2=8. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru.
4. Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah terjamah sebagai “Node terjamah”. Node terjamah tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
5. Set “Node belum terjamah” dengan jarak terkecil (dari node keberangkatan) sebagai “Node Keberangkatan” selanjutnya dan lanjutkan dengan kembali ke step 3

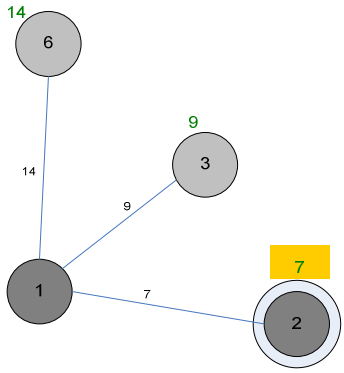
Dibawah ini penjelasan langkah per langkah pencarian jalur terpendek secara rinci dimulai dari node awal sampai node tujuan dengan nilai jarak terkecil.

1. Node awal 1, Node tujuan 5. Setiap edge yang terhubung antar node telah diberi nilai



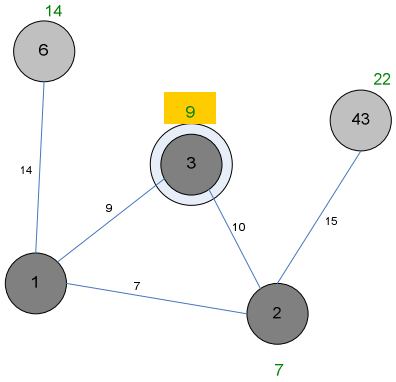
Gambar 32.2 Contoh kasus Djikstra - Langkah 1

2. Dijkstra melakukan kalkulasi terhadap node tetangga yang terhubung langsung dengan node keberangkatan (node 1), dan hasil yang didapat adalah node 2 karena bobot nilai node 2 paling kecil dibandingkan nilai pada node lain, nilai = 7 (0+7).



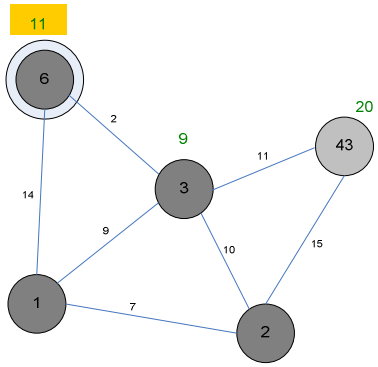
Gambar 32.3 Contoh kasus Djikstra - Langkah 2

3. Node 2 diset menjadi node keberangkatan dan ditandai sebagi node yang telah terjamah. Dijkstra melakukan kalkulasi kembali terhadap node-node tetangga yang terhubung langsung dengan node yang telah terjamah. Dan kalkulasi dijkstra menunjukan bahwa node 3 yang menjadi node keberangkatan selanjutnya karena bobotnya yang paling kecil dari hasil kalkulasi terakhir, nilai 9 (0+9).



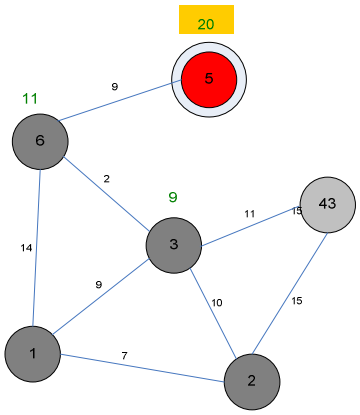
Gambar 32.4 Contoh kasus Djikstra - Langkah 3

4. Perhitungan berlanjut dengan node 3 ditandai menjadi node yang telah terjamah. Dari semua node tetangga belum terjamah yang terhubung langsung dengan node terjamah, node selanjutnya yang ditandai menjadi node terjamah adalah node 6 karena nilai bobot yang terkecil, nilai 11 (9+2).



Gambar 32.5 Contoh kasus Djikstra - Langkah 4

5. Node 6 menjadi node terjamah, dijkstra melakukan kalkulasi kembali, dan menemukan bahwa node 5 (node tujuan) telah tercapai lewat node 6. Jalur terpendeknya adalah 1-3-6-5, dan niilai bobot yang didapat adalah 20 (11+9). Bila node tujuan telah tercapai maka kalkulasi dijkstra dinyatakan selesai.



Gambar 32.6 Contoh kasus Djikstra - Langkah 5

C. TUGAS PENDAHULUAN

Buatlah review mengenai :

- Konsep dari Algoritma Dijkstra
- Cobalah sendiri studi kasus graph yang terdapat di dasar teori, selesaikan dengan algoritma Dijkstra. Jelaskan menggunakan gambar.

D. PERCOBAAN

Percobaan 1 : Class MinInfo adalah data yang digunakan pada saat mencari minimum Path

```
private static class MinInfo<T> implements
Comparable<MinInfo<T>>
{
   public T endV;
   public int pathWeight;
   public int compareTo(MinInfo<T> item)
   {
      if (pathWeight < item.pathWeight)
         return -1;
      else if (pathWeight == item.pathWeight)
         return 0;
      else
         return 1;
   }
}
```

Percobaan 2 : method minimumPath() mencari jarak terpendek dari Graph dengan langkah-langkah sebagai berikut :

- Hapus objek MinInfo dari priority queue, tandai node tersebut karena sudah dikunjungi dengan warna BLACK, cari node lain yang belum dikunjungi dengan jarak yang terdekat. Selanjutnya cek apakah penambahan jarak dari vertex awal ke vertex sekarang memiliki path yang lebih baik. Jika ya maka buat object MinInfo dan masukkan ke priority queue. Ubah data dan alamat dengan node tetangganya.
- Algoritma ini berhenti jika priority queue sudah kosong atau jumlah dari node yang sudah dikunjungi sudah sama dengan jumlah node pada graph tersebut.

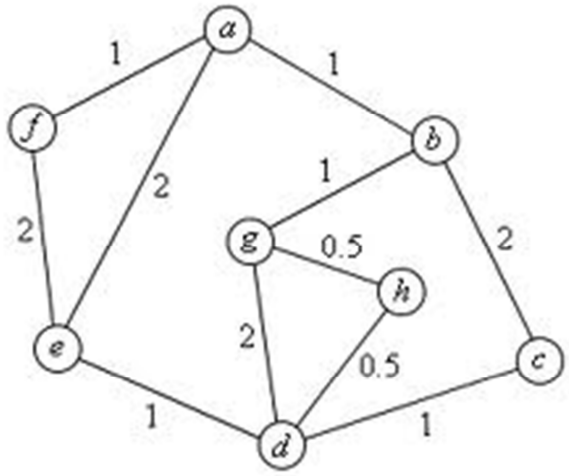
```
public static <T> void minimumPath(
DiGraph<T> g, T sVertex)
{
   T currVertex = null, neighborVertex = null;
   HeapPQueue<MinInfo<T>> minPathPQ =
      new HeapPQueue<MinInfo<T>>(new Less<MinInfo<T>>());
   MinInfo<T> vertexData = null;

   Set<T> edgeSet;
   Iterator<T> edgeIter;
   int newMinWeight, numVisited = 0, numVertices;
   ...


while (numVisited < numVertices && !minPathPQ.isEmpty())
{
   vertexData = minPathPQ.pop();
   currVertex = vertexData.endV;
   if (g.getColor(currVertex) != VertexColor.BLACK)
   {

      g.setColor(currVertex, VertexColor.BLACK);
      numVisited++;
      edgeSet = g.getNeighbors(currVertex);
      edgeIter = edgeSet.iterator();
while (edgeIter.hasNext())
      {
         neighborVertex = edgeIter.next();
         if (g.getColor(neighborVertex) == VertexColor.WHITE)
         {
            newMinWeight = g.getData(currVertex) +
                  g.getWeight(currVertex, neighborVertex);
            if (newMinWeight < g.getData(neighborVertex))
            {
               vertexData = new MinInfo<T>();
               vertexData.endV = neighborVertex;
               vertexData.pathWeight = newMinWeight;
               minPathPQ.push(vertexData);
               g.setData(neighborVertex, newMinWeight);
               g.setParent(neighborVertex, currVertex);
            }
         }
```

```
        }
    }
}
```

E. LATIHAN

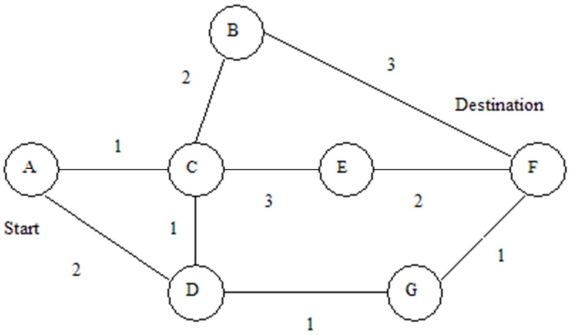
1. Selesaikan studi kasus di bawah ini menggunakan Algoritma Dijkstra !
2. Selesaikan studi kasus di bawah ini dengan program yang telah di buat !

Studi kasus 1



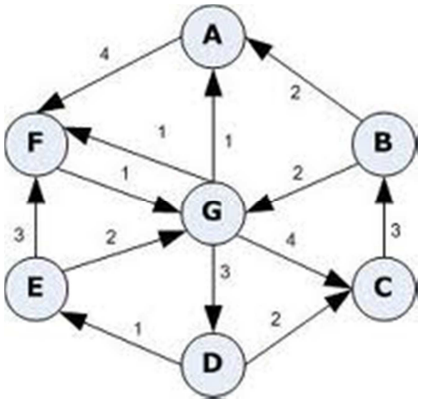
Gambar 32.7 Latihan Soal , Studi Kasus 1

Studi kasus 2



Gambar 32.8 Latihan Soal , Studi Kasus 2

Studi Kasus 3



Gambar 32.9 Latihan Soal , Studi Kasus 3

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.