Aplikasi Algoritma *Greedy* yang Dimodifikasi dalam Pencarian Lintasan Terpendek

Ari Wardana
135 06 065

Mahasiswa Teknik Informatika, Institut Teknologi Bandung
Jln.Ganesha No.10 Bandung
e-mail: a121w8488@yahoo.co.id

ABSTRAK

Masalah pencarian lintasan terpendek dapat diselesaikan dengan bebagai metode, salah satunya menerapkan Algoritma *Greedy*. Algoritma *Greedy* memang memiliki kemampuan untuk menyelesaikan masalah dengan lebih cepat, namun dalam penerapannya sering kali kita menemui solusi yang tidak mangkus.
Akan tetapi dengan sedikit modifikasi, algoritma *Greedy* dapat menghasilkan solusi yang lebih mangkus, walaupun secara umum konsep yang digunakan masih berpatokan dengan algoritma *Greedy* sehingga masih memungkinkan menemui solusi yang tidak mangkus, namun konsep algoritma *Greedy* yang tetap ada membuat masalah dapat diselesaikan lebih cepat.

Kata kunci: *Greedy*, *Shortset Path* (Lintasan Terpendek).

1. PENDAHULUAN

Pada makalah ini akan diterangkan mengenai aplikasi algoritma *greedy* dalam pencarian lintasan terpendek *(Shortest Path)*. Akan tetapi algoritma *greedy* yang dipakai di sini telah mengalami penyempurnaan sehingga pencarian lintasan terpendek yang dilakukan akan menjadi lebih mangkus. Namun, sebelumnya menjelaskan lebih dalam penulis akan menjelaskan makna dari pencarian lintasan terpendek dan Algoritma *greedy.*

1.1 Pencarian Lintasan Terpendek *(Shortest Path)*

Pencarian lintasan Terpendek *(Shortest Path)* adalah pencarian suatu jalan/lintasan dari suatu tempat ke tempat lain yang diberikan dalam suatu masalah. Biasanya, banyak kemungkinan jalan yang bisa ditempuh dan dalam perjalanan kita juga akan menemui tempat lain. Dalam suatu solusi mungkin saja kita melakukan perjalanan yang memutar, namun pada solusi lain mungkin kita menemukan jalan yang terpendek. Oleh karena itu, dibutuhkan suatu metode untuk menemukan jalan/lintasan yang terpendek, sehingga perjalanan antara dua tempat tersebut dapat ditempuh lebih cepat.

1.2 Algoritma *Greedy*

Algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Algoritma ini sederhana dan lempang. Secara harafiah *greedy* artinya rakus atau tamak, yaitu sifat yang berkonotasi negatif. Prinsip *greedy* adalah *"take what you can get now!"*. Ambil apa yang dapat Anda peroleh sekarang! Prinsip ini juga diadopsi dalam pemecehan masalah optimasi, tetapi tentunya dalam konteks positif. Oleh karena itu, algoritma *greedy* ini juga dapat digunakan dalam pencarian lintasan terpendek (shortest path).

Algoritma *greedy* membentuk solusi langkah per langkah. Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan yang "tampaknya" memberikan perolehan terbaik, yaitu dengan membuat pilihan **optimum lokal** (*local optimum*) pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi **optimum global** (*global optimum*).

2. Pencarian Lintasan Terpendek Menggunakan Konsep Algoritma *Greedy* yang Dimodifikasi

Pada bagian ini akan diterangkan mengenai metode yang digunakan dalam pencarian lintasan terpendek. Sebelumnya, penulis akan menjelaskan struktur data yang penulis gunakan dalam algoritma yang diterapkan. Masalah pencarian lintasan terpendek ini akan digambarkan dalam sebuah graf berarah. Graf di sini akan berisi nama setiap simpul yang menggambarkan setiap tempat, baik titik awal, titik ahir, dan tempat-tempat lain

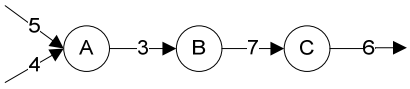
yang dilalui; dan sisi yaitu jarak setiap simpul yang menggambarkan jarak setiap 2 tempat, tentu saja jika kedua tempat tersebut memiliki lintasan/jalan di antaranya.

2.1 Komponen-komponen dalam Algoritma

2.1.1 Pengabungan simpul dari beberapa simpul

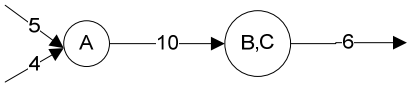
Suatu simpul dapat digabungkan dengan simpul lain dengan syarat-syarat tertentu. Berikut ini penjelasan mengenai syarat-syaratnya.

1. Misalkan ada simpul A, B dan C. Simpul B hanya bisa dicapai dari simpul A, dan dari simpul C hanya bisa dicapai simpul B. Simpul A memiliki sisi masukkan atau simpul C memiliki satu sisi yang keluar, kecuali jika simpul C adalah simpul akhir, yang tidak memiliki sisi yang keluar.



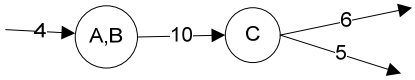
Gambar 1 Contoh 1 Penggabungan Simpul

Dapat diubah menjadi:



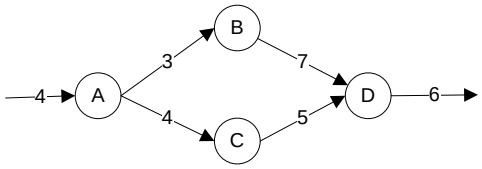
Gambar 2 Contoh 2 Penggabungan Simpul

Sebaliknya, jika simpul C memiliki satu sisi yang keluar, maka:



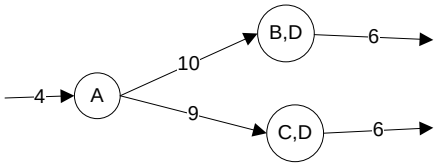
Gambar 3 Contoh 3 Penggabungan Simpul

2. Misalkan ada simpul A, B, C, dan D. seperti gambar berikut



Gambar 4 Contoh 4 Penggabungan Simpul

Dapat digabungkan menjadi:



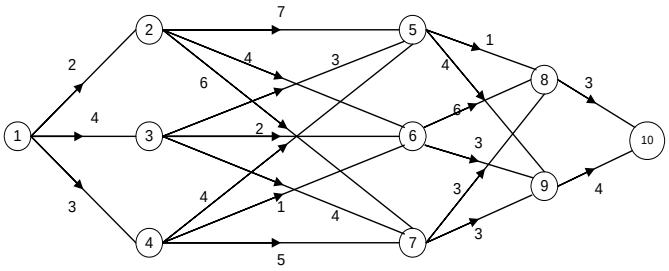
Gambar 5 Contoh 5 Penggabungan Simpul

3. Karena sisi antara 2 buah simpul merupakan sisi terpendek. Hal ini akan dijelaskan lebih lanjut nantinya.

2.1.2 Pemilihan Sisi Terpendek secara *Greedy*

Bagian ini merupakan bagian yang paling penting dalam algoritma pencarian ini. Algoritma yang digunakan akan mencari sisi yang terpendek dari setiap sisi yang ada dalam graf.

Contohnya:



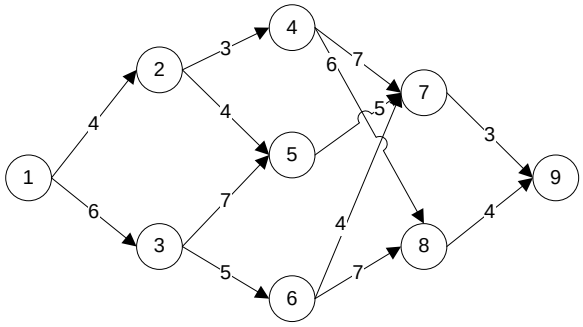
Gambar 6 Contoh Pencarian Sisi Terpendek

Pada gambar di atas, dapat dilihat bahwa sisi terpendek bernilai 1, dan sisi yang bernilai satu ada 2. Keempat simpul diatas akan disimpan dalam suatu *array*, untuk digunakan algoritma utama.

2.1.3 Terdapat Sisi Terpendek lebih dari 1

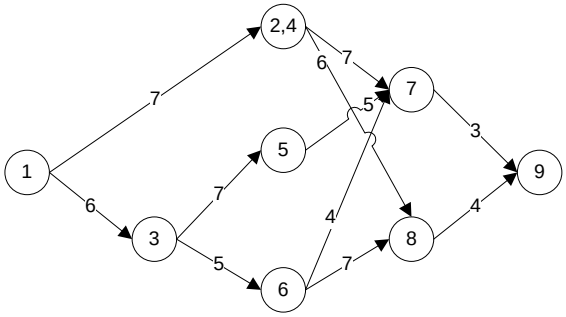
Jika terdapat beberapa sisi terpendek, saat melakukan pencarian sisi terpendek seperti yang telah dijelaskan di atas maka algoritma ini akan menangganinya dengan membentuk pohon solusi dengan cabang-cabangnya akan mengarah ke solusi masing-masing, dan setiap solusi akan dibandingkan pada akhir algoritma.

Dalam setiap cabang, dua simpul yang menghubungkan satu sisi terpendek yang dipilih (kita sebut simpul awal dan simpul akhir) akan digabungkan menjadi suatu simpul baru, setiap sisi yang keluar dari simpul awal akan dihapus dan sisi yang menuju simpul akhir juga akan dihapus. Sementara simpul yang bukan simpul akhir, jika tidak memiliki sisi yang keluar, maka simpul itu akan dihapus, atau simpul yang bukan simpul awal, jika tidak memiliki sisi yang masuk, maka simpul tersebut juaga akan dihapus.



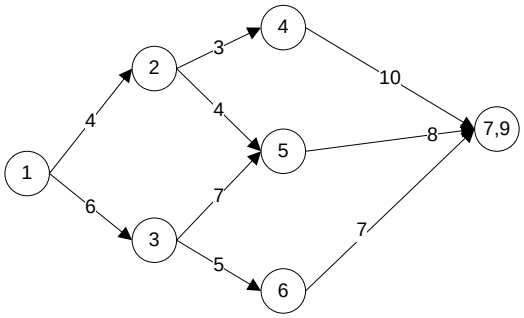
Gambar 7 Contoh1 Jika Terdapat Sisi Terpendek lebih dari 1

Cabang Pertama, dipilih simpul 2 dan 4 , sehingga graf akan menjadi:



Gambar 8 Cabang Pertama

Cabang kedua, diplih simpul 7 dan 9



Gambar 9 Cabang kedua

2.2 Algoritma

Algoritma dalam persoalan ini dalam `procedure LintasanTerpendek(input Graf_A : Graf, simpul Awal, simpul Akhir)` adalah sebagai berikut.

1. *Procedure* akan memanggil *function-function* yang akan memberikan nilai sisi yang terpendek, jumlah sisi yang terpendek, simpul mana saja yang di antaranya terdapat sisi terpendek.
2. *Procedure* akan melakukan pencabangan dengan pola *DFS.*
   a. Kemudian melakukan penggabungan simpul-simpul antara sisi terpendek.
   b. *Procedure* kemudian akan melakukan penggabungan simpul-simpul yang sesuai syarat pada bagian 2.1.1
   c. Jika solusi ditemukan pencarian berakhir untuk pencabangan ini. Solusi yang ditemukan kemudian akan dibandingkan dengan solusi yang ditemukan sebelumnya.
   d. Jika solusi belum ditemukan, maka *procedure* akan memanggil dirinya sendiri.

Berikut ini *pseudo code* dari algoritma pencarian lintasan terpendek ini dalam `procedure LintasanTerpendek`

```
Deklarasi:/*Deklarasi Kontruktor dan KamusGlobal*/

var simpul : integer
var static simpul_awal : simpul
var static simpul_akhir : simpul

contructor Solusi()
  idx : integer
  nilaiolusi : integer = -1
  arraysolusi : array[1..simpul_akhir] of string
    /*asumsi simpul akhir selalu nilai terbesar, kasus
    terburuk semua simpul masuk dalam solusi sehingga
    array penuh, kasus lain, jika simpul_awal adalah
    simpul_akhir maka array hanya berisi 1 element*/

  for (idx = 1 sampai idx = simpul_akhir) do
     /*array solusi awalnya berisi "-1"*/
     Arraysolusi[idx] = "-1"
  /*end for*/
/*end constructor*/

constructor Graf(input Max_size : integer)
/*
Graf adalah strucktur data yang berisi nilai simpul dan
   nilai sisi
namasimpul merupakan nama yang akan diberikan jika ada 2
   atau lebih simpul yang digabungkan, namasimpul =
   namasimpul1 + namasimpul2 + .., pada saat awal dibuat
   namanya hanya satu
nilaisimpul dinyatakan dalam array 1 dimensi, maksudnya
   nilaisimpul[i], adalah nilai simpul i
Sisi(dalam hal ini nilainya) merupakan nilai jarak
   antara 2 simpul
   sisi dinyatakan dalam array 2 dimensi,maksudnya
   sisi[i][j], adalah jarak antara simpul i dan simpul j
*/
   Deklarasi:

   idx = integer
   idx2 = integer

   var static simpul_akhir : simpul
   var namasimpul : array[1..Max_size] of string
   var sisi : array[1..Max_size,1..Max_size] of integer

   Algoritma:
   /*
   nilaisimpul pada awalnya diberi nilai 0;
   sementara sisi diberi nilai -1
  */
  for (idx = 0 sampai idx = Max_size) do
     namasimpul[idx] = toString(idx)
        /*
        mengubah integer menjadi string;
        nama simpul sesuai dengan urutannya,1,2,3,...
        */
  /*end for*/
  for (idx = 1 sampai idx = Max_size)do
   for (idx2 = 1 sampai idx2 = Max_size) do
     sisi[idx,idx2] = -1
   /*end for*/
  /*end for*/
/*end constructor*/

var S : Solusi() /*membuat solusi*/
```

Algoritma Utama:

```
procedure LintasanTerpendek(input Graf_A : Graf,
Awal:simpul,Akhir : simpul)

Deklarasi:
  N : integer = Sizeof(A) /*N = Ukuran dari graf A*/
  idx : integer
  nilaisolusisementara : integer = 9999999

  X : integer
  Y : integer
  Z : array[1..100] of string
  B : Graf(N)/*inisialisasi Graf B*/

Algoritma:

  X = NilaiSisiTerpendek(B)/*mencari nilai sisi
                            terpendek*/
  Y = JumlahSisiTerpendek(B,X)/*menghitung jumlah
                            terpendek*/
  Z = GetSimpuldenganSisiTerpendek(B,X)

  for (idx = 1 sampai Y) do/*pembentukan pohon*/
     simpulyangdigabung = array[2] of string

     simpulyangdigabung[1] = Z[2*idx - 1]
     simpulyangdigabung[2] = Z[2*idx]

     B = PerbaikiPohon(A,simpulyangdigabung[1],
                              simpulyangdigabung[2])
     B = PerbaikiPohon2(B)

     if(SolusiKetemu())then

        if(nilaisolusisementara < S.nilaisolusi)then
                Solusi = GetSolusi(B)
        /*end if*/
    else
      LintasanTerpendek(B,simpul_awal,B.simpulakhir)
           /*Rekuren memanggil kembali procedure
        LintasanTerpendek*/
    /*end if*/
 /*end for*/
```

2.3 Kelebihan dan Kekurangan

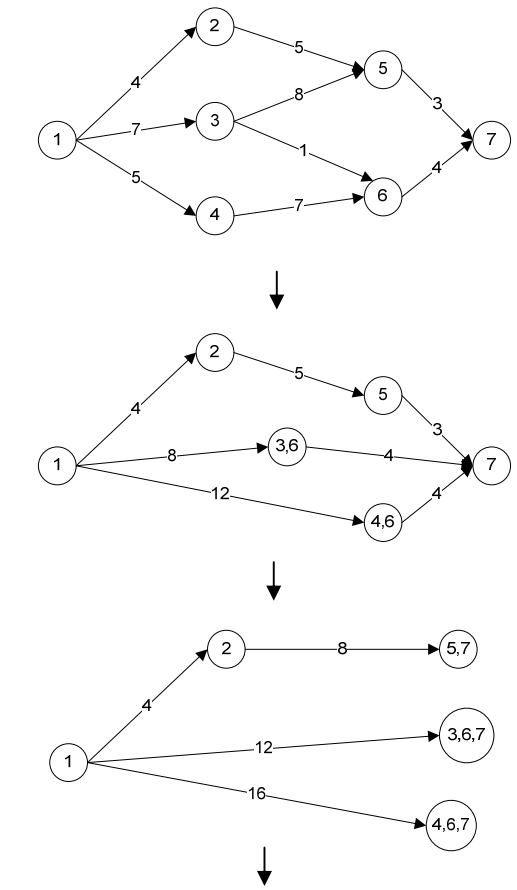
2.3.1 Kelebihan

1. Dengan mengambil sisi yang paling kecil terlebih dahulu, membuat solusi sementara yang dibuat menjadi semakin dekat dengan solusi yang paling mangkus.
2. Penerapan Algoritma *Greedy* sebagai dasar pemecahan masalah membuat penyelesaian masalah menjadi lebih cepat, karena tidak semua kemungkinan solusi diperiksa.
3. Algoritma dalam pemecahan masalah ini akan lebih mangkus jika permasalahan memiliki banyak solusi, dan satu solusi dapat berhubungan dengan solusi yang lain.
   Algoritma ini juga mangkus untuk masalah yang sangat sedikit kemungkinan solusisnya.

2.3.2 Kekurangan

1. Untuk beberapa kasus, algoritma ini tidak mangkus, namun lebih baik dari algoritma *greedy* secara umum.

3.Contoh Kasus



Gambar 10 Contoh Kasus

Penjelasan mengenai kasus di atas

1. Gambar I
   Program akan mencari sisi terpendek yaitu 1, antara simpul 3 dan 6
   Oleh karena itu simpul 3 dan 6 harus digabungkan
2. Gambar II

Setelah 3 dan 6 digabungkan maka sesuai dengan aturan ke 2 pada bagian 2.1.1 maka 4 dan 6 juga digabung, sehingga terbentuk graf baru
Program akan memulai dari awal lagi, mencari sisi terpendek, yaitu 3, antara 5 dan 7

3. Gambar III
   Setalah 5 dan 7 digabungkan berdasarkan aturan ke 2 pada bagian 2.1.1, maka simpul 3,6 dan 7 harus digabung, begitu juga simpul 4,6 dan 7
   Sehingga didapatkan graf baru, dan dicari lagi sisi terkecil yaitu 4, antara simpul 1 dan 2
   Pemngabungan simpul 1 dan 2, dilakukan dengan pengecualian simpul awal, sehingga sisi ditambahkan ke belakang.
4. Gambar IV
   Karena antara simpul 1 dan 2 digabung maka hanya ada satu sisi yang tertinggal yaitu antara simpul 1,2 dan 5,7.

Karena hanya teringgal satu sisi , maka sisi tersebutlah yang menjadi nilai terpendek dari lintasan yang ditempuh, Sedangkan lintasannya sendiri yaitu 1-2-5-7, sesuai urutan dari nama kedua simpul yang terbentuk diakhir program.

4. KESIMPULAN

1. Algoritma *Greedy* dapat dimodifikasi sehingga dalam masalah pencarian lintasan terpendek *(Shortest Path)* lebih mangkus.
2. Walaupun secara umum konsep yang digunakan masih berpatokan dengan algoritma *Greedy* sehingga masih memungkinkan menemui solusi yang tidak mangkus, namun konsep algoritma *Greedy* yang tetap ada membuat masalah dapat diselesaikan lebih cepat.

REFERENSI

[1] Munir, Rinaldi, Strategi Algoritmik,Bandung, 2007.