Pengunaan Struktur Data Disjoint-set dan Sirkuit Euler untuk Menentukan Kebenaran Matriks Jarak pada Pohon

Muhammad Kamal Shafi - 13518113
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13518113@std.stei.itb.ac.id

***Abstract*—Data atau informasi dapat disimpan dalam berbagai macam bentuk. Bentuk-bentuk penyimpanan informasi bisa saja berada dalam suatu representasi array kontinu, representasi diskrit, atau bahkan di dalam bentuk matriks. Salah satu informasi yang dapat disimpan didalam bentuk matriks adalah informasi mengenai suatu struktur data pohon. Antara setiap simpul yang berada didalam sebuah struktur data pohon, hanya terdapat satu lintasan di antara mereka. Jarak yang harus ditempuh dalam melewati lintasan antar simpul bisa disimpan didalam bentuk matriks. Jika kita ingin menganalisis apakah suatu matriks berisi informasi menganai jarak antara simpul pada pohon, kita dapat melakukan pengecekan ini dengan menggunakan struktur data disjoint-set dan dengan kemudian melakukan pengecekan menggunakan *lowest common ancestor* yang didapat dengan sirkuit euler.**

***Kata kunci*—*Ancestor*, Disjoint-set, Matriks, Pohon, Sirkuit Euler.**

I. PENDAHULUAN

Struktur data pohon selalu memiliki simpul-simpul yang saling terhubung baik itu secara langsung oleh satu sisi ataupun secara tidak langsung yaitu dari beberapa sisi yang dapat membentuk lintasan. Karena inilah pada struktur data pohon terdapat sebuah keunikan yaitu diantara dua buah simpul di dalam pohon, akan selalu terdapat tepat satu lintasan yang menghubungkan keduanya.

Untuk menyimpan informasi mengenai hubungan berupa jarak antara simpul pada suatu pohon, terkadang digunakan sebuah matriks yang memetakan jarak antar simpul. Dengan mengetahui matriks yang memetakan jarak antar simpul pada pohon, maka pohon yang direpresentasikan oleh matriks tersebut dapat dibentuk.

Dalam melakukan pembentukan sebuah pohon dari matriks jarak antar simpul yang dimiliki pohon, maka kita perlu berasumsi bahwa matriks yang kita miliki sudah benar. Tentu saja bila matriks yang dimiliki memiliki kesalahan, maka representasi pohon yang didapat akan memiliki ketidakcocokan dengan matriks yang dimiliki. Untuk mengetahui apakah sebuah matriks dapat dibentuk menjadi sebuah representasi pohon, maka kita perlu melakukan pengecekan apakah matriks tersebut benar atau tidak.

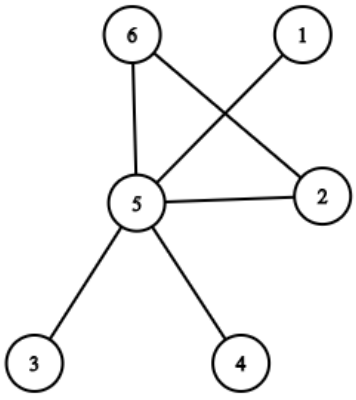
Salah satu cara yang cukup efektif untuk melakukan pengecekan kebenaran suatu matriks jarak pada pohon adalah dengan membangun representasi pohon dari matriks itu sendiri dengan menggunakan struktur data disjoint-set. Setelah pohon selesai dibentuk, maka dapat dilakukan pengecekan dengan mencocokan antara jarak setiap simpul pada pohon dan pada matriks secara langsung.

II. LANDASAN TEORI

A. Definisi Graf

Graf adalah himpunan dari pasangan (*V*, *E*) dengan V adalah himpunan tidak kosong yang berisi simpul dan E adalah himpunan-ganda yang berisi sisi dimana setiap sisi menghubungkan antara dua buah simpul[1].

Suatu graf G dapat dilambangkan dengan notasi $G = (V, E)$. Sebuah graf $G = (V, E)$ yang memiliki $V = \{1, 2, 3, 4, 5, 6\}$ dan $E = \{(1, 5), (4, 5), (2, 5), (2, 6), (3, 5), (5, 6)\}$ akan memiliki representasi gambar seperti berikut.



Gambar 1. Graf G

B. Jenis-jenis Graf

Dengan melihat sisi yang dimiliki, Graf dapat dikelompokan menjadi dua yaitu berdasarkan ada atau tidaknya gelang/kalang/*loop* (sisi yang menghubungkan dua sisi yang sama) dan ada atau tidaknya sisi ganda. Pengelompokan graf berdasarkan ada atau tidaknya sisi ganda atau sisi gelang adalah

seperti berikut:

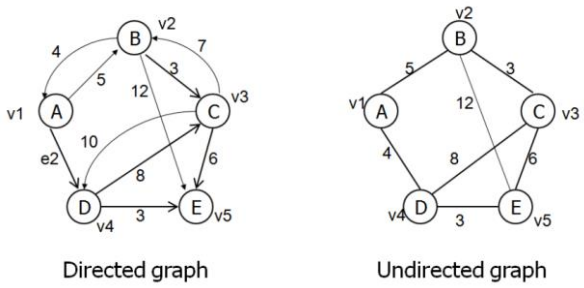
1. Graf Sederhana
   Graf sederhana adalah sebuah graf yang tidak memiliki gelang maupun sisi ganda.
2. Graf tidak sederhana
   Sesuai dengan namanya, pada graf tidak sederhana boleh mengandung sisi ganda maupun gelang. Graf tidak sederhana juga dapat dikelompokan lagi menjadi graf ganda (graf yang boleh memiliki sisi ganda) dan graf semu (graf yang boleh memiliki gelang).

Selain dari pengelompokan seperti tadi, graf juga dapat dikelopokan berdasarkan ada atau tidaknya arah dari sisi yang dimiliki graf. Pengelompokan berdasarkan arah sisi pada graf adalah seperti berikut:

1. Graf tidak berarah (*undirected graph*)
   Graf tidak berarah adalah graf yang sisinya tidak memiliki arah yang menggambarkan hubungan antar simpulnya. Pada graf berarah, jika terdapat sisi yang menghubungkan dari simpul *a* dan *b*, maka bisa juga dikatakan bahwa sisi tersebut menghubungkan simpul *b* dan *a*.
2. Graf berarah (*directed graph*)
   Graf berarah adalah sebuah graf yang memiliki arah yang menggambarkan hubungan antar simpulnya. Pada graf berarah, jika terdapat sisi yang menghubungkan simpul *a* dan *b*, maka akan terdapat hubungan dari *a* ke *b* tetapi belum tentu terdapat hubungan dari *b* ke *a*.

C. Graf Berbobot

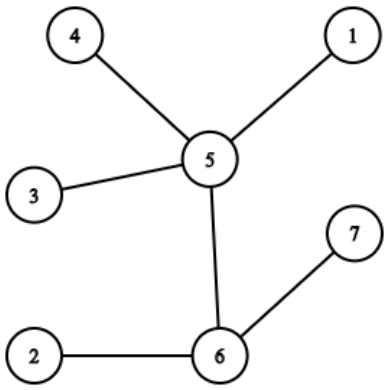
Sebuah graf yang sisi-sisinya memiliki suatu nilai bilangan dapat disebut sebagai graf berbobot. Bobot yang dimiliki oleh sisi-sisi di dalam graf berbobot dapat merepresentasikan banyak hal. Hal umum yang biasanya direpresentasikan dari bobot pada sisi adalah jarak dari dua buah simpul yang dihubungkan.



Gambar 2. Contoh graf berbobot. Sumber : http://apriliyatiwen.blogspot.com/2013/04/graph-berbobot.html

D. Definisi Pohon

Pohon adalah suatu graf terhubung yang tidak mengandung sirkuit [1]. Suatu graf $G = (V, E)$ dapat dikatakan sebagai pohon apabila graf G merupakan graf sederhana dan tidak memiliki sirkuit didalamnya.



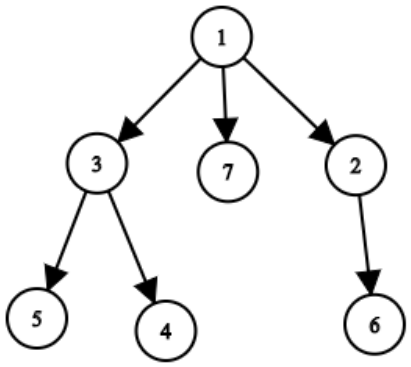
Gambar 3. Contoh pohon

E. Pohon Berakar

Pohon berakar merupakan pohon yang sebuah simpulnya dijadikan sebagai akar dan sisi-sisinya diberikan arah menjauh dari akar [1]. Simpul akar pada pohon berakar memiliki derajat masuk nol dan simpul-simpul lainnya pada pohon memiliki derajat masuk tepat satu. Penggambaran pohon berakar boleh tidak berisi arah dari pada sisi-sisinya karena setiap sisi sudah jelas arahnya.

F. Terminologi Pohon Berakar

Pada bahasan selanjutnya akan digunakan beberapa terminologi yang ada pada pohon berakar. Dibawah ini adalah terminology-terminologi yang ada pada pohon berakar untuk pohon pada gambar 4.



Gambar 4. Contoh pohon berakar

Terminologi pohon berakar mengacu pada gambar 4 adalah seperti berikut:

1. Lintasan
   Lintasan dari simpul 1 ke simpul 6 adalah runtutan simpul-simpul 1, 2, 6.
2. Anak (*child*) dan orang tua (*parent*)
   Pada gambar, simpul 3 memiliki anak simpul 4 dan simpul 5, sementara simpul 7 memiliki orang tua simpul 1.
3. Keturunan *(descendant*) dan leluhur (*ancestor*)
   Simpul *a* dikatakan sebagai keturunan simpul *b* apabila terdapat lintasan dari simpul *b* ke simpul *a*, pada kasus

ini simpul *b* juga dapat dikatakan sebagai leluhur dari simpul *a*. Pada gambar, simpul 3, 4, dan 5 merupakan keturunan dari simpul 1, simpul 3 dan simpul 1 merupakan leluhur dari simpul 4 dan 5.

4. Saudara kandung
   Simpul yang berorang tua sama adalah saudara kandung. Pada gambar, simpul 4 dan 5 adalah saudara kandung, begitu juga dengan simpul 2, 3, dan 7.
5. Upapohon (*subtree*)
   Pohon berakar yang memiliki akar yang berasal dari pohon sebenarnya dapat membentuk upapohon. Pada gambar, pohon berakar dengan akar simpul 3 yang berisi himpunan simpul {3, 4, 5} adalah merupakan upapohon dari pohon aslinya.
6. Derajat
   Derajat sebuah simpul pada pohon berarah didefinisikan sebagai banyak anak dari simpul yang dimaksud. Pada gambar, simpul 3 berderajat 2.
7. Daun
   Simpul yang berderajat nol. Pada gambar, simpul 4, 5, dan 6 adalah daun.
8. Simpul Dalam
   Simpul dalam adalah simpul yang mempunyai anak dan orang tua. Pada gambar, simpul 3 dan 2 merupakan simpul dalam.
9. Tingkat (*level*)
   Tingkat sebuah simpul pada pohon berakar adalah jarak dari akar ke simpul tersebut. Pada gambar, simpul 1 memiliki tingkat nol.
10. Tinggi atau Kedalaman
    Tinggi atau kedalaman dari sebuah pohon berakar adalah tingkat maksimum dari seluruh simpul-simpulnya. Pada gambar, tinggi dari pohon adalah 2.

G. Matriks Jarak pada Graf

Matriks jarak pada graf adalah matriks persegi $[m_{ij}]$ yang menyimpan jarak terdekat antara simpul *i* dan *j* yang terdapat pada graf [3]. Matriks jarak juga dapat digunakan untuk menyimpan jarak antara simpul pada pohon. Matriks jarak dari suatu pohon berisi panjang lintasan antara simpul-simpul yang berhubungan.

H. Common Ancestor dan Lowest Common Ancestor

Common ancestor adalah leluhur yang sama-sama dimiliki oleh sebuah himpunan simpul. Pada gambar 4, *common ancestor* dari simpul 5 dan simpul 4 adalah simpul 1 dan simpul 3.

Dalam pemrosesan sebuah pohon berakar, terdapat satu hal menyangkut *common ancestor* yang sering digunakan, yaitu *Lowest Common Ancestor* atau biasanya disingkat menjadi LCA. LCA adalah *common ancestor* yang memiliki tingkat terbesar diantara *common ancestor* yang lain. LCA dari dua buah simpul *a* dan *b* dituliskan dengan notasi $LCA(a,\ b)$. Pada gambar 4, $LCA(4,5) = 3$, dan $LCA(4,6) = 1$.

I. Struktur Data Disjoint-set

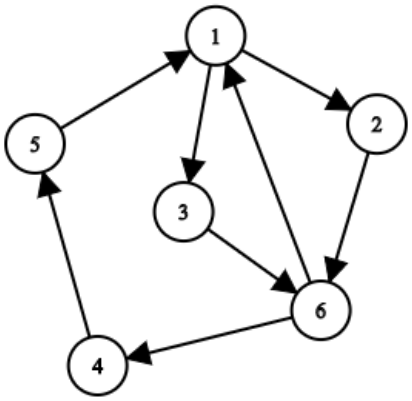
Struktur data disjoint-set adalah struktur data yang dapat digunakan untuk mengelompokan beberapa elemen berbeda kedalam beberapa *disjoint sets* atau himpunan yang terpisah-pisah [2]. Struktur data disjoint-set memiliki dua operasi utama yaitu menyatukan (*uniting*) antara dua himpunan dan mencari identitas unik himpunan yang memiliki suatu elemen tertentu (*finding*).

Sebuah disjoint-set memiliki koleksi himpunan yang berada di dalamnya. Operasi yang dimiliki dari data struktur disjoint-set adalah sebagai berikut:

1. MAKE-SET(*x*)
   Membuat himpunan baru yang berisi hanya *x*.
2. UNION(*x*, *y*)
   Menyatukan himpunan yang di dalamnya berisi *x* dan *y*.
3. FIND-SET(*x*)
   Mengembalikan identitas unik himpunan yang didalamnya berisi *x*.

I. Definisi Sirkuit Euler

Sirkuit euler dalam graf adalah lintasan yang melewati masing-masing sisi pada graf masing-masing satu kali dan lintasan ini harus kembali ke titik(simpul) asal [1]. Graf yang memiliki sirkuit euler disebut graf euler. Contoh sirkuit euler pada gambar berikut: 1, 2, 6, 1, 3, 6, 4, 5, 1.



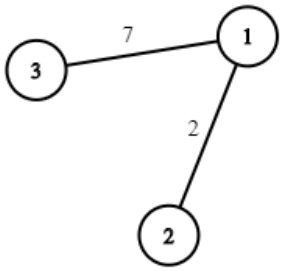
Gambar 5. Contoh graf euler

III. PEMBENTUKAN POHON DARI MATRIKS JARAK DENGAN MENGGUNAKAN DISJOINT-SET

A. Pernyataan Ulang dan Pendefinisian Masalah

Karena perlu dilakukan pengecekan apakah sebuah matriks benar merupakan sebuah matriks jarak pada pohon, maka kita dapat menyimpulkan bahwa permasalahan yang dimiliki berada dalam bentuk matriks $[m_{ij}]$ dimana pada kolom ke-i dan baris ke-j matriks seharusnya berisi jarak antara simpul ke-i dan simpul ke-j dengan besarnya adalah m_{ij} $(m_{ij} > 0)$. Untuk memudahkan pembahasan, maka simpul ke-i akan diberi nama sebagai simpul i dan simpul ke-j akan diberi nama sebagai smpul j. Contoh dari matriks jarak dan representasi pohonnya adalah sebagai berikut.

Matriks $M = \begin{bmatrix} 0 & 2 & 7 \\ 2 & 0 & 9 \\ 7 & 9 & 0 \end{bmatrix}$ akan merepresentasikan jarak-jarak antara simpul i dan simpul j pada pohon berikut,



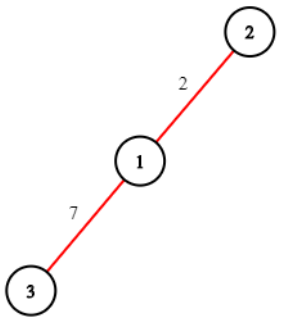
Gambar 6. Contoh representasi pohon

B. Pembentukan Pohon dari Matriks

Tahap pertama yang harus dilakukan adalah untuk membentuk pohon dari matriks. Pada pembentukan pohon, kita akan menggunakan paradigma *greedy* dan juga dengan menggunakan struktur data disjoint-set.

Untuk membentuk representasi pohon dari matriks yang dimiliki, maka perlu diketahui terlebih dahulu sisi-sisi yang dimiliki pohon. Sisi-sisi yang perlu dibuat dari pohon dengan banyak simpul sebanyak n adalah n-1 buah sisi. Sisi-sisi yang ada pada pohon akan ditentukan dengan membaca jarak dari simpul-simpul yang ada. Perhatikan bahwa jarak antara dua buah simpul tidak mungkin nol, kecuali untuk jarak sebuah simpul terhadap dirinya sendiri. Dari sini kemudian dapat dirancang sebuah algoritma *greedy* untuk menentukan mana saja sisi-sisi yang ada. Sisi-sisi yang ada ditentukan dengan melihat jarak antara simpul di mulai dari yang paling kecil. Hal ini dilakukan karena jarak terkecil yang terdapat pada matriks jarak tidak mungkin berasal dari hubungan dua sisi atau lebih sisi lainnya.

Seperti pada matriks $M = \begin{bmatrix} 0 & 2 & 7 \\ 2 & 0 & 9 \\ 7 & 9 & 0 \end{bmatrix}$, jarak terkecil dari dua buah simpul adalah 2 yaitu jarak antara simpul 1 dan 2. Jarak 2 ini tidak mungkin berasal dari dua atau lebih sisi lainnya, sehingga kita dapat memastikan bahwa antara simpul 1 dan 2 terdapat sisi yang menghubungkannya dengan besar beban sebesar 2. Hal ini berbeda dengan jarak antara simpul 2 dan 3 yang merupakan jarak yang ditimbulkan karena 2 buah sisi, yaitu sisi (1, 2) dan sisi (2, 3), dengan demikian jarak antara simpul 2 dan 3 adalah 2 + 7.



Gambar 7. Pohon bahasan 1

Setelah membuat langkah-langkah seperti ini, perhatikan bahwa di dalam langkah-langkah tersebut tidak boleh kita membentuk sisi antara simpul-simpul yang sudah terhubung. Di sinilah struktur data disjoint-set digunakan. Sebelum melakukan pembentukan sisi antara dua simpul *a* dan *b*, akan dicek terlebih dahulu apakah simpul *a* dan *b* sudah berada pada himpunan yang sama dengan menggunakan FIND-SET(*a*) dan FIND-SET(*b*). Jika kedua operasi tersebut menghasilkan identitas himpunan yang sama, maka tidak perlu lagi dibentuk sisi antara simpul *a* dan *b*, sebaliknya, jika kedua operasi itu menghasilkan identitas himpunan yang berbeda, maka akan dibuat sisi yang menghubungkan antara simpul *a* dan *b*. Jika sisi (*a*, *b*) akan dibentuk maka akan dilakukan operasi penyatuan himpunan yaitu UNION(*a*, *b*).

Pseudocode dari algoritma ini adalah sebagai berikut.

```
Require: n > 0, for all M[i][j] ≥ 0
Ensure: n = NodeCount

  for all node : node ∈ Tree do
      MAKE − SET(node)
  end for

  len ← 0
  for i ← 1 to n − 1 do
      for j ← i + 1 to n do
          len ← len + 1
          Array[len] ← MakeTypeEdge(i, j, M[i][j])
              ▷ Array diisi dengan edge(i, j) dengan jarak
  sebesar M[i][j]
      end for
  end for

  Sort(Array)
      ▷ Array diurutkan berdasarkan besar jarak(terurut
  tak-turun)

  for i ← 1 to len do
      edge ← Array[i]
      if FIND − SET(edge.i) ≠ FIND − SET(edge.j)
  then
          UNION(edge.i, edge.j)
          MakeEdge(edge)
      end if
  end for
```

Gambar 8. Pseudocode 1

IV. PEMROSESAN DAN PENGECEKAN POHON DENGAN MENGGUNAKAN LCA

A. Menyimpan Lintasan Sirkuit Euler

Berdasarkan syarat bahwa sebuah graf tidak berarah akan memiliki sirkuit euler jika dan hanya jika semua simpul pada graf memiliki derajat genap dan jika graf tersebut berarah maka graf tersebut akan memiliki sirkuit euler jika dan hanya jika semua simpul pada graf memiliki derajar keluar dan derajat

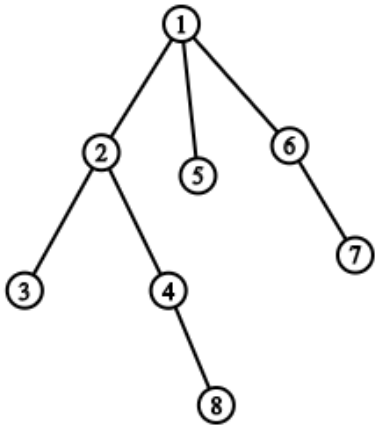
masuk yang sama. Dari sini dapat dibuktikan bahwa pada sebuah pohon tidak mungkin ada sebuah sirkuit euler karena daun pada pohon hanya memiliki derajat 1 yang merupakan bilangan ganjil. Maka dari itu, untuk membentuk sirkuit euler kita akan melakukan modifikasi terhadap pohon.

Lintasan sirkuit euler didapat dengan memodifikasi bentuk pohon yang di miliki. Langkah-langkah modifikasinya adalah sebagai berikut:

1. Berikan pohon akar dengan memilih salah satu simpul yang kemudian akan dijadikan sebagai akar.
   Karena sekarang pohon sudah berbentuk pohon berakar, maka setiap sisi pada simpul sudah memiliki arah menjauh dari akarnya.
2. Gandakan semua sisi yang ada pada pohon dan kemudian beri arah berlawanan dari sisi yang diduplikat.

Dengan melakukan langkah-langkah tersebut, dapat dipastikan bahwa simpul-simpul dari graf yang didapat memiliki derajat keluar dan derajat masuk yang sama. Dengan demikian sirkuit euler dapat dicari.

Sirkuit euler yang akan dicari memiliki awalan dan akhiran di akar yang telah ditentukan. Perhatikan contoh pencarian sirkuit euler pada pohon berikut.



Gambar 9. Pohon bahasan 2

Langkah-langkah pembentukan sirkuit euler:

1. Pilih simpul 1 sebagai akar.
2. Gandakan semua sisi dengan arah berlawanan dari sisi yang sudah ada.
3. Lakukan pembentukan sirkuit euler dengan memulai lintasan dari simpul 1.

Dengan melakukan langkah tersebut, maka akan didapatkan lintasan :1, 2, 3, 2, 4, 8, 4, 2, 1, 5, 1, 6, 7, 6, 1. Lintasan yang didapat ini kemudian akan disimpan didalam sebuah array.

B. Menyimpan Kemunculan Pertama Simpul dan Tingkat Simpul

Setelah mendapat kan lintasan sirkuit euler dari pohon, hal selanjutnya yang harus dilakukan adalah mencatat indeks kemunculan pertama simpul di dalam array lintasan sirkuit euler. Selain itu, perlu juga untuk membuat array yang menyimpan tinggi simpul dengan urutan yang sesuai dengan lintasan yang telah didapat.

Berdasarkan contoh yang telah dikerjakan tadi, maka array yang dibuat akan berisi seperti berikut.

Index	**Lintasan**	**Tingkat Simpul**
1	1	0
2	2	1
3	3	2
4	2	1
5	4	2
6	8	3
7	4	2
8	2	1
9	1	0
10	5	1
11	1	0
12	6	1
13	7	2
14	6	1
15	1	0

Tabel 1. Isi array 1 dan 2

Simpul	1	2	3	4	5	6	7	8
Kemunculan Pertama	1	2	3	5	10	12	13	6

Tabel 2. Isi array 3

Pseudocode dari algoritma penyimpanan sirkuit euler ini adalah sebagai berikut.

Ensure: Initialization $FirstApp[node] = infinite$, for all node
Ensure: Initialization global variable $curIdx = 0$

procedure MAKEEULERTOUR(curNode)
  $curIdx \leftarrow curIdx + 1$
  $Lintasan[curIdx] \leftarrow curNode$
  $FirstApp[curNode] \leftarrow min(FirstApp[curNode], curIdx)$
  $Tingkat[curIdx] \leftarrow Level(curNode)$
  for all $next : (curNode, next) \in EdgeList$ **do**
    $MakeEulerTour(next)$
    $curIdx \leftarrow curIdx + 1$
    $Lintasan[curIdx] \leftarrow curNode$
    $Tingkat[curIdx] \leftarrow Level(curNode)$
  end for
end procedure

Gambar 10. Pseudocode 2

C. Mencari LCA

Pencarian LCA dapat dengan mudah dilakukan dengan menggunakan array yang telah dibuat. Untuk mencari LCA dari dua buah simpul *a* dan *b*, yang harus diperhatikan hanyalah isi array pada indeks antara awal kemunculan *a* dan awal kemunculan *b*. Kemudian untuk menentukan LCA, cukup dengan memilih simpul yang memiliki tingkat terendah yang berada pada rentang array.

Misal akan dicari sebuah LCA dari pohon yang diberikan pada contoh sebelumnya (gambar 9).

LCA(3, 8) dapat diselesaikan dengan cara berikut,

1. Perhatikan rentang array dari awal kemunculan simpul 3 dan 8, yaitu rentang [3, 6],

Index	Lintasan	Tingkat Simpul
3	3	2
4	2	1
5	4	2
6	8	3

Tabel 3. Simulasi LCA

2. Pilih simpul dengan tingkat terendah, sehingga didapatkan bahwa simpul 2 merupakan LCA dari simpul 3 dan 8.

D. Mengecek Kebenaran Matriks dengan LCA

Algoritma pengecekan matriks akan melakukan iterasi ke setiap pasangan simpul (kecuali dengan diri sendiri) yang mungkin dan lalu untuk setiap pasang simpul *a* dan *b* akan dilakukan pengecekan berupa perbandingan berikut:

1. Jarak(*LCA*(*a*, *b*), *a*) + Jarak(*LCA*(*a*, *b*), *b*) dengan Jarak(*a*, *b*).
2. Jarak(Akar, *a*) + Jarak(Akar, *b*) – 2 (Jarak(Akar, *LCA*(*a*, *b*))) dengan Jarak(*a*, *b*).

Apabila hasil perhitungan salah satunya memiliki hasil yang berbeda, maka dapat disimpulkan bahwa matriks yang diberikan tidaklah valid. Matriks akan dianggap valid jika dan hanya jika hasil semua perbandingan adalah sama.

Pseudocode dari algoritma pengecekan adalah sebagai berikut.

```
valid ← True
for i ← 1 to n do
    for j ← 1 to n do
        if i ≠j then
            lca ← LCA(i, j)
            if M[treeRoot][i]+M[treeRoot][j]−2∗M[treeRoot][lca] ≠
M[i][j] or M[lca][i] + M[lca][j] ≠ M[i][j] then
                valid ← False
            end if
        end if
    end for
end for
```

Gambar 11. Pseudocode 3

V. KESIMPULAN

Struktur data disjoint-set sangat *powerful* dalam melakukan operasi yang berhubungan dengan pengecekan hubungan. Pada kasus di dalam makalah ini, disjoint-set membantu untuk mengecek apakah dua buah simpul sudah terhubung atau belum.

Menggunakan LCA dalam memproses pohon akan sangat membantu.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada semua pihak yang secara langsung maupun tidak langsung telah membantu kelancaran pembuatan makalah ini.

DAFTAR PUSTAKA

[1] Munir, Rinaldi, Matematika Diskrit, Bandung: Informatika Bandung, 2009.

[2] Cormen, Thomas H.; Leiserson, Charles E.; Rivest Ronald L.; Stein Clifford, Introduction to Algorithms (3rd ed.), MIT Press & McGraw Hill, 2009.

[3] http://mathworld.wolfram.com/GraphDistanceMatrix.html diakses pada 4 Desember 2019 pukul 15.00.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Desember 2019

Muhammad Kamal Shafi - 13518113