Jurnal Natural
Vol. 14, No. 1, 30-33, Maret 2014
ISSN 1141-8513

Connectivity Algorithm in Simple Graphs

Reza Wafdan, Mahyus Ihsan, Rahma Zuhra

Multimedia Research Group, Jurusan Matematika, FMIPA Universitas Syiah Kuala
Darussalam 23111, Banda Aceh.
redjawavdhan11@gmail.com

Abstract. This paper discusses about connectivity algorithm in simple graphs. The algorithm has three stages, namely, adjacent vertices search, adjacent vertices investigation and component investigation. The aim of connectivity algorithm is to check whether a graph is connected or disconnected with the scope set. There are three sets which are used in this algorithm, namely, vertex, edge and adjacent vertices sets. The simple graph is connected if the graph has only one component or if the number of adjacent vertices elements is equal to the number of vertices. Otherwise, if the simple graph has more than one component or if the number of adjacent vertices elements is not equal to the number of vertices then the algorithm will conclude that the graph is disconnected.

Keywords: graph, connectivity, algorithm, adjacent vertices, component.

Pendahuluan

Graf adalah himpunan pasangan (V, E), V tidak kosong dan setiap unsur dari E merupakan pasangan dari dua unsur yang berbeda pada V. Unsur-unsur pada V disebut kumpulan *vertex*, unsur-unsur dari E disebut kumpulan *edge*. Kemudian, jika e adalah suatu *edge*, maka $e = (u, v)$, dimana u dan v adalah unsur-unsur berbeda pada V yang disebut sebagai kumpulan *vertex* akhir atau ujung dari e. *Edge* $e = (u, v)$ adalah pasangan dari *vertex* u dan v, dapat juga dinotasikan sebagai *edge* uv atau *edge* vu. *Vertex* u dan v dikatakan *incident* pada *edge* uv. *Edge* uv adalah *incident* dengan setiap *vertex*-nya. Dua *vertex* dikatakan *adjacent* jika keduanya adalah *vertex* terakhir dari suatu kumpulan *edge*. Dua *edge* dikatakan *adjacent* jika keduanya mempunyai suatu *vertex* bersama [1]. *Path* merupakan suatu perjalanan dimana tidak ada *vertex* yang berulang [2].

Suatu graf dikatakan terhubung (*connected*) jika dan hanya jika ada suatu *path* yang menghubungkan masing-masing *vertex* [3]. Sebuah graf disebut tidak terhubung (*disconnected*) jika himpunan *vertex*-nya dapat dipartisi menjadi dua himpunan bagian yang tidak kosong, V_1 dan V_2, yang tidak memiliki unsur bersama, sedemikian sehingga tidak ada *edge* dengan satu titik akhir dalam V_1 dan yang lainnya di V_2 [4]. Untuk memeriksa keterhubungan suatu graf seperti kasus di atas akan digunakan konektivitas graf (*graph connectivity*). Sebuah graf tidak terhubung terdiri atas sejumlah subgraf yang terpisah, subgraf terhubung maksimal disebut *component* [2].

Konektivitas graf merupakan hal yang sangat penting yang harus terpenuhi dalam berbagai kasus graf. Jika syarat keterhubungan pada kasus-kasus tersebut tidak terpenuhi maka semua kasus graf tersebut akan sulit untuk diselesaikan. Beberapa kasus graf yang mensyaratkan keterhubungan yaitu graf Hamilton, graf Euler, pohon (*tree*), masalah pohon pembentang minimum (MST), masalah lintasan terpendek, dan graf planar. Penerapan konektifitas graf dalam kehidupan nyata bisa ditemukan di berbagai bidang diantaranya penerapan di bidang telekomunikasi, bidang jaringan komputer, bidang transportasi, jaringan listrik dan saluran air.

Penelitian ini ditujukan untuk merancang suatu algoritma konektivitas graf yang bertujuan untuk memeriksa apakah sebuah graf terhubung atau tidak terhubung pada *simple graph* dengan ruang lingkup himpunan.

Metodologi

Pada algoritma konektivitas graf ini, input yang diterima adalah jumlah *vertex*, jumlah *edge*, himpunan *vertex* dan himpunan *edge* atau himpunan pasangan *vertex*.

Output yang dihasilkan dalam algoritma konektivitas graf ini adalah menyimpulkan apakah suatu graf terhubung (*connected*) ataupun tidak terhubung (*disconnected*). Ada tiga tahapan dari algoritma konektivitas graf ini yaitu pencarian *adjacent vertices*, pemeriksaan

himpunan *adjacent vertices* dan pemeriksaan *component*.

Sebelum melangkah ke algoritma pemeriksaan graf, langkah awal dalam penyusunan algoritma ini adalah pembentukan himpunan. Ada tiga himpunan yang harus dibentuk yaitu himpunan *vertex*, *edge* dan *adjacent vertices*. Himpunan *V* adalah himpunan *vertex* pada sebuah *simple graph* dengan *m* anggota. Bila diberikan suatu *simple graph* dengan *m vertex* dan *n edge*, dapat dibentuk sebuah himpunan *V* dengan *m* anggota.

$$V = \{v_1, v_2, v_3, \ldots, v_{m-1}, v_m\}$$

Himpunan *E* berupa himpunan pasangan *vertex* dengan *n* anggota yang memuat *vertex* pertama dan *vertex* kedua yang saling berpasangan. Bila diberikan suatu *simple graph* dengan *m vertex* dan *n edge*, dapat dibentuk sebuah himpunan *E* dengan *n* anggota.

$$E = \{e_1, e_2, e_3, \ldots, e_{n-1}, e_n\}$$

Himpunan *adjacent vertices* (*adjV*) berupa himpunan *vertex* dimana setiap *vertex*-nya saling terhubung. Himpunan *adjV* tidak kosong dan memiliki *k* anggota dengan $k \leq m$, dimana *m* adalah jumlah *vertex*.

$$adjV = \{w_1, w_2, w_3, \ldots, w_{k-1}, w_k\}$$

Hasil dan Pembahasan

Pada penelitian ini, algoritma pemeriksaan konektivitas graf pada *simple graph* dibagi dalam tiga tahap yaitu pencarian *adjacent vertices*, pemeriksaan himpunan *adjacent vertices* dan pemeriksaan *component*. Berikut algoritma konektivitas graf pada *simple graph* dengan *m vertex* dan *n edge*, *u* adalah *vertex* dengan *label* terkecil pada himpunan *V*.

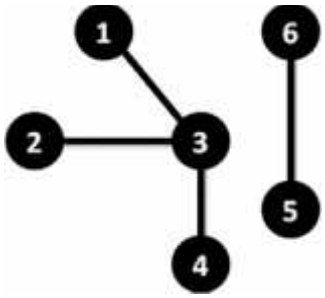
Step 1. *adjV* = {*u*}.
Step 2. *i* =1.
Step 3. **while** $i \leq |adjV|$
for each *vertex* $v \in V$,
if $|adjV| = |V|$, **stop**;
else if $(w_i, v) \in E$ **and** $v \notin adjV$,
$adjV = adjV \cup \{v\}$.
$i = i + 1$.
end while
Step 4. **if** $|adjV| = |V|$,
output "*Connected*".
else
output "*Disconnected*".

Langkah-langkah pada algoritma di atas akan diuraikan dalam tiga tahapan berikut:

Pencarian *Adjacent Vertices*
Pencarian *adjacent vertices* dimulai dari *vertex* dengan *label* terkecil misalkan *u* dan himpunan *adjV* diisi dengan *vertex u*. Kemudian akan dicari *vertex-vertex* yang berpasangan dengan *u* pada himpunan *E*. Misalkan $e_1 = (u, v)$ dan $e_2 = (u, w)$, *vertex v* dan *w* adalah *vertex* yang berpasangan dengan *u*, maka himpunan *adjV* akan ditambahkan dengan *vertex v* dan *w*.

$$adjV = \{u, v, w\}$$

Selanjutnya akan dicari *adjacent vertices* dari *vertex v* pada himpunan *E* dan ditambahkan ke himpunan *adjV*. Untuk *vertex-vertex* berikutnya, pencarian dilanjutkan sesuai urutan *vertex* pada himpunan *adjV*. Pencarian akan berhenti saat jumlah anggota himpunan *adjV* sama dengan jumlah anggota himpunan *V* atau akan berhenti sampai tidak ada lagi *vertex* yang terhubung dengan *vertex-vertex* pada himpunan *adjV*. Berikut adalah sebuah contoh *simple graph* dengan enam *vertex* dan empat *edge*.



Gambar 1. Contoh *simple graph*

Dari graf pada Gambar 1, dapat dibentuk himpunan *V* dengan enam anggota, yaitu:

$$V = \{1, 2, 3, 4, 5, 6\}$$

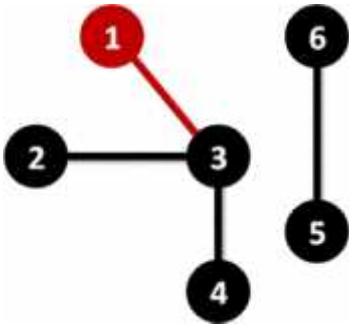
Kemudian dari himpunan *V* tersebut akan dibentuk himpunan *E* dengan empat anggota, yaitu:

$$E = \{13, 23, 34, 56\}$$

Pencarian *adjacent vertices* dimulai dari *vertex* dengan *label* terkecil yaitu *vertex* 1 dan himpunan *adjV* diisi dengan *vertex* 1. Kemudian dari *vertex* 1 akan dicari *vertex-vertex* yang berpasangan dengan *vertex* 1. Pada himpunan *E*, *vertex* yang berpasangan dengan *vertex* 1 adalah *vertex* 3 yaitu pada *edge* 13 seperti diperlihatkan pada Gambar 2. Selanjutnya himpunan *adjV* ditambahkan dengan *vertex* 3.

$$adjV = \{1, 3\}$$

Pencarian *adjacent vertices* terus berlanjut untuk mencari pasangan *vertex* dari *vertex* 3 pada himpunan *E* dan ditambahkan ke himpunan *adjV*. Untuk *vertex-vertex* berikutnya, pencarian dilakukan sesuai dengan urutan *vertex* pada himpunan *adjV*.

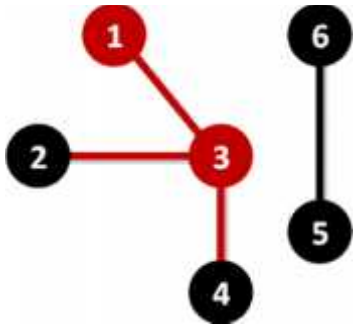


Gambar 2. Pencarian *adjacent vertices* dari *vertex* 1

Pencarian akan berhenti saat semua *adjacent vertices* pada graf di atas yaitu *vertex* 1, 3, 2 dan 4 sudah terisi ke dalam himpunan *adjV*.

Pemeriksaan Himpunan *Adjacent Vertices*

Tahap berikutnya adalah pemeriksaan himpunan *adjacent vertices*, yaitu memastikan *vertex* tersebut tidak pernah terpilih sebelumnya menjadi anggota himpunan *adjV*. Misalkan *vertex* yang berpasangan dengan *vertex u* adalah *v*, Selanjutnya akan dicari *adjacent vertices* dari *vertex v* pada himpunan *E*, *adjacent vertices* dari *vertex v* adalah *vertex u* dan *w*. Kemudian akan dilakukan pemeriksaan himpunan *adjV*, dari hasil pemeriksaan diperoleh bahwa salah satu *adjacent vertices* dari *vertex v* sudah pernah ditambahkan ke himpunan *adjV* yaitu *vertex u*, maka hanya *vertex w* yang ditambahkan ke himpunan *adjV*.



Gambar 3. Pencarian *adjacent vertices* dari *vertex* 3

Pada langkah sebelumnya, v*ertex-vertex* yang berpasangan dengan *vertex* 3 pada himpunan *E* yaitu *vertex* 1, *vertex* 2 dan *vertex* 4 seperti diperlihatkan pada Gambar 3. Dari hasil pemeriksaan diperoleh bahwa salah satu *adjacent vertices* dari *vertex* 3 sudah pernah ditambahkan ke himpunan *adjV* yaitu *vertex* 1, maka hanya *vertex* 2 dan 4 yang ditambahkan ke himpunan *adjV* dengan memperhatikan urutan *label*-nya.

$$adjV = \{1, 3, 2, 4\}$$

Begitu juga untuk *vertex* 2 dan *vertex* 4, tidak ada lagi *vertex* yang berpasangan dengannya kecuali *vertex* 3. Sehingga tidak ditambahkan lagi ke himpunan *adjV* dan diperoleh himpunan *adjV* dengan empat anggota seperti di atas.

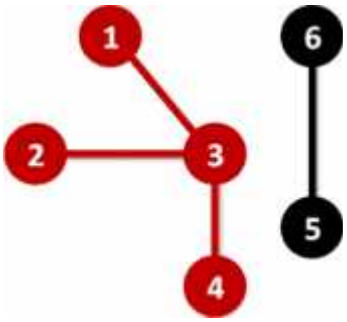
Pemeriksaan *Component*

Tahap terakhir adalah pemeriksaan *component*, yaitu memeriksa apakah *simple graph* yang diinputkan hanya memiliki satu *component* atau lebih dari satu *component*. Misalkan himpunan *V* dan himpunan *adjV* dari sebuah *simple graph* diperoleh himpunan sebagai berikut:

$$V = \{v_1, v_2, v_3, \ldots, v_{m-1}, v_m\}$$

$$adjV = \{w_1, w_2, w_3, \ldots, w_{k-1}, w_k\}$$

Dari kedua himpunan dapat dilihat bahwa jika $k = m$ atau jumlah anggota himpunan *V* sama dengan himpunan *adjV*, maka graf tersebut hanya memiliki satu *component*. Sebaliknya, jika $k \neq m$ atau jumlah anggota himpunan *V* berbeda dengan himpunan *adjV*, maka graf tersebut memiliki lebih dari satu *component*.



Gambar 4. Graf memiliki dua *component*

Pada contoh sebelumnya, dapat dilihat bahwa tidak ada *edge* yang terhubung ke *vertex* 5 dan *vertex* 6 maka diperoleh himpunan *adjV* sebagai berikut:

$$adjV = \{1, 3, 2, 4\}$$

Anggota himpunan *adjV* tersebut tidak memuat *vertex* 5 dan 6 sehingga jumlah anggota himpunan *adjV* adalah empat sedangkan jumlah *vertex* adalah enam. Dengan kata lain *vertex* 6

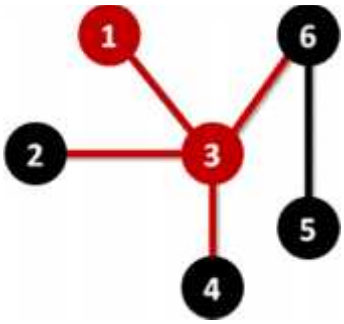
dan *vertex* 5 berada dalam *component* yang berbeda dan graf ini memiliki lebih dari satu *component* seperti diperlihatkan pada Gambar 4.

Jika graf pada Gambar 1 ditambahkan sebuah *edge* yang menghubungkan *vertex* 3 dan *vertex* 6 seperti diperlihatkan pada Gambar 5, maka diperoleh *simple graph* dengan enam *vertex*, yaitu:

$$V = \{1,2,3,4,5,6\}$$

dan lima *edge*, yaitu:

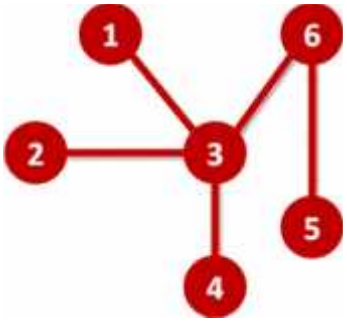
$$E = \{13, 23, 34, 36, 56\}$$



Gambar 5. *Vertex* 3 dan *vertex* 6 terhubung

Pada himpunan *E* diperoleh *vertex-vertex* yang berpasangan dengan *vertex* 3 yaitu *vertex* 1, 2, 4 dan 6, maka *vertex* 2, 4 dan 6 ditambahkan ke himpunan *adjV*.

$$adjV = \{1,3,2,4,6\}$$



Gambar 6. Graf hanya memiliki satu *component*

Karena pada himpunan *E* tidak ada lagi *vertex* yang berpasangan dengan *vertex* 2 dan *vertex* 4 kecuali *vertex* 3 maka selanjutnya akan dicari *vertex* yang berpasangan dengan *vertex* 6. Dari himpunan *E* diperoleh *vertex-vertex* yang berpasangan dengan *vertex* 6 yaitu *vertex* 3 dan *vertex* 5, maka himpunan *adjV* ditambahkan dengan *vertex* 5. Selanjutnya algoritma akan berhenti karena jumlah anggota himpunan *adjV* sama dengan jumlah anggota himpunan *V* dan diperoleh himpunan *adjV* sebagai berikut:

$$adjV = \{1,3,2,4,6,5\}$$

Dari himpunan *adjV* dapat dilihat bahwa anggota himpunan *adjV* tersebut memuat semua *vertex* dan jumlah anggota himpunan *adjV* sama dengan jumlah *vertex* atau dengan kata lain semua *vertex* tersebut berada dalam satu *component* yang sama, seperti diperlihatkan pada Gambar 6.

Kesimpulan

Algoritma konektivitas graf dapat menentukan keterhubungan suatu graf pada *simple graph* dengan ruang lingkup himpunan. Suatu *simple graph* dinyatakan terhubung (*connected*) jika graf tersebut hanya memiliki satu *component*, sebaliknya jika suatu *simple graph* memiliki lebih dari satu *component*, maka algoritma akan menyimpulkan bahwa graf tersebut tidak terhubung (*disconnected*).

Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada pihak-pihak yang telah memberikan ide, saran dan masukan-masukan lainnya, diantaranya:

1. Al Aiyub, atas penelitian sebelumnya yang telah banyak memberikan inspirasi untuk kelancaran penelitian ini.
2. Kepada anggota Grup Riset Multimedia yang tidak dapat disebutkan nama satu per satu.

Referensi

1. Goodaire, Edgar G. and Parmenter, Michael M., 2002, *Descrete mathematics with graph theory*, Prentice-Hall, USA.
2. Wallis, W.D., 2007, *A Beginner's Guide to Graph Theory*, Second Edition, Birkhauser, Boston.
3. Wilson, Robin J., 1996, *Introduction to Graph Theory*, Fourth edition, Longman Group Ltd, London.
4. Bondy, J. A. and Murty, U. S. R., 2008, *Graph Theory*, Springer, Berlin.