Analisis Penggunaan Algoritma Pencarian Melebar (BFS) dan Algoritma Pencarian Mendalam (DFS) dalam Teori Graf

Ella Madanella Dwi Mustika

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jalan Ganesha no. 10, Bandung
E-mail: if15022@students.if.itb.ac.id

ABSTRAK

Graf merupakan representasi yang umum digunakan dalam pencarian solusi suatu persoalan. Berbagai masalah aplikasi dunia nyata (*real-world problems*) dapat dikelompokkan menjadi suatu masalah klasik dalam teori graf, misalnya masalah utilitas dan masalah pengaturan topologi bintang pada jaringan komputer LAN sama-sama merupakan persoalan graf bipartit, sehingga pemecahan kedua persoalan tersebut dapat dilakukan dengan pendekatan yang sama.

Pencarian solusi persoalan dengan representasi graf dapat dilakukan dengan cara mengunjungi (*traverse*) simpul-simpul dalam graf secara sistematis. Algoritma pencarian melebar (breadth-first search / BFS) dan algoritma pencarian mendalam (depth-first search / DFS) merupakan algoritma traversal di dalam graf yang umum digunakan. Dalam teori graf, BFS dapat memecahkan masalah pengecekan graf bipartit, pencarian komponen terhubung dalam graf, pencarian jarak terpendek pada graf tak berbobot, serta pencarian diameter pohon. Sementara itu, DFS dapat memecahkan masalah pengurutan topologi dan pencarian komponen terhubung dan terhubung kuat dalam graf. Masalah-masalah teori graf yang diselesaikan dengan algoritma BFS dan DFS inilah yang akan dibahas dalam makalah ini. Dengan pemecahan masalah-masalah klasik teori graf tersebut, diharapkan teori graf dapat memecahkan persoalan dunia nyata dengan lebih mangkus.

Kata kunci: Breadth-First Search, Depth-First Search, Teori Graf

1. PENDAHULUAN

Pemecahan masalah dengan representasi graf dilakukan pertama-tama dengan membuat representasi objek masalah sebagai simpul dalam graf serta membuat representasi hubungan antar objek dengan garis yang menghubungkan simpul-simpul tersebut. Setelah itu, setiap simpul dalam graf dikunjungi secara sistematis (*traverse*).

Dalam graf terdapat dua macam algoritma traversal, yaitu:

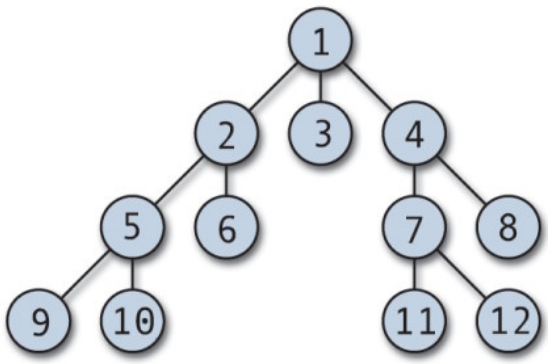
a. Algoritma pencarian melebar (breadth-first search / BFS)
b. Algoritma pencarian mendalam (depth-first search / DFS)

Meskipun kedua algoritma ini memiliki tujuan yang sama, yaitu mengunjungi setiap simpul dalam graf, pendekatan yang dilakukan oleh keduanya masing-masing berbeda, sehingga aplikasi penggunaan masing-masing algoritma ini juga berbeda sesuai kebutuhan masalah.

II. ALGORITMA PENCARIAN MELEBAR (BFS)

Algoritma pencarian melebar (BFS) mengunjungi setiap simpul dalam graf mulai dari simpul akar, lalu dilanjutkan dengan mengunjungi simpul-simpul yang bertetangga dengan simpul akar tersebut. Setelah itu, untuk setiap simpul yang sudah dikunjungi tersebut, dikunjungi simpul-simpul yang bertetangga dengannya dan belum dikunjungi, demikian seterusnya sampai seluruh simpul berhasil dikunjungi.

Berikut gambar yang mengilustrasikan urutan simpul yang dikunjungi pada algoritma BFS:



Gambar 1. Ilustrasi urutan kunjungan simpul pada algoritma BFS

Dari gambar di atas, dapat dilihat bahwa dengan algoritma BFS, setiap simpul pada tingkat x dikunjungi lebih dahulu sebelum simpul pada tingkat di bawahnya (x+1).

2.1. Cara Kerja Algoritma BFS

Dalam algoritma BFS, simpul anak yang telah dikunjungi disimpan dalam suatu antrian. Antrian ini digunakan untuk mengacu simpul-simpul yang bertetangga dengannya yang akan dikunjungi kemudian sesuai urutan pengantrian.

Untuk memperjelas cara kerja algoritma BFS beserta antrian yang digunakannya, berikut langkah-langkah algoritma BFS:

1. Masukkan simpul ujung (akar) ke dalam antrian
2. Ambil simpul dari awal antrian, lalu cek apakah simpul merupakan solusi
3. Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.
4. Jika simpul bukan solusi, masukkan seluruh simpul yang bertetangga dengan simpul tersebut (simpul anak) ke dalam antrian
5. Jika antrian kosong dan setiap simpul sudah dicek, pencarian selesai dan mengembalikan hasil "solusi tidak ditemukan"
6. Ulangi pencarian dari langkah kedua

2.2. Properti Algoritma BFS

Karena seluruh simpul yang telah dikunjungi harus disimpan, maka kompleksitas ruang algoritma BFS adalah O(jumlah simpul+jumlah sisi). Hal ini menunjukkan bahwa algoritma BFS memerlukan ruang besar dan tidak cocok untuk pemecahan masalah besar.

Angka yang sama menunjukkan kompleksitas waktu terburuk, di mana setiap simpul harus diperiksa. Kompleksitas waktu terbaik, O(1), didapat jika solusi ditemukan pada pencarian pertama.

Walaupun memiliki kompleksitas ruang dan waktu yang kurang baik, algoritma BFS dapat mencari solusi secara tuntas, berarti algoritma ini akan menemukan solusi terbaik dan terdekat bagaimanapun bentuk graf tersebut.

2.3. Aplikasi Algoritma BFS dalam Teori Graf

Karena algoritma BFS menggunakan graf sebagai media representasi persoalan, tidak sulit untuk mengaplikasikan algoritma ini dalam persoalan-persoalan teori graf. Berikut contoh-contoh penggunaan algoritma BFS dalam persoalan teori graf:

2.3.1. Pengecekan Graf Bipartit

Graf bipartit adalah graf yang seluruh simpulnya dapat dipisah menjadi dua bagian, serta setiap sisi graf menghubungkan dua simpul yang masing-masing berada pada bagian yang berbeda, sehingga tidak ada simpul pada bagian yang sama yang bertetangga.

BFS digunakan untuk mengecek apakah suatu graf merupakan graf bipartit atau bukan dengan memulai pencarian pada simpul manapun, lalu memberi nilai 0 dan 1 pada setiap simpul yang dikunjungi. Angka ini menandakan bagian di mana simpul tersebut berada.

Contoh pemberian nilai pada graf: Jika simpul awal diberi nilai 0, maka seluruh simpul yang bertetangga dengannya diberi nilai 1, lalu seluruh simpul yang bertetangga dengan simpul bernilai 1 tersebut diberi nilai 0, dan seterusnya.

Dalam pemberian nilai ini, jika dicapai simpul yang telah dikunjungi sebelumnya, dicek apakah simpul tersebut berada pada tingkat yang sama dengan simpul ayahnya, dengan cara pengecekan nilai kedua simpul tersebut yang bertetangga tersebut.

Jika simpul bertetangga dengan simpul yang bernilai sama dengannya, maka dapat disimpulkan bahwa graf bukan merupakan graf bipartit. Jika sampai akhir pemberian nilai hal tersebut tidak terjadi, maka graf tersebut merupakan graf bipartit.

2.3.2. Pencarian Komponen Terhubung dalam Graf

Dua buah komponen dinyatakan terhubung dalam graf jika terdapat lintasan antara keduanya. Dalam algoritma BFS, seluruh simpul yang dikunjungi pasti merupakan simpul yang terhubung dengan simpul akar, karena simpul-simpul tersebut hanya dikunjungi jika bertetangga dengan simpul ayahnya. Seluruh simpul yang terhubung dengan simpul akar juga pasti dikunjungi.

Oleh karena itu, seluruh simpul yang dikunjungi dalam algoritma BFS dan masuk ke dalam pembentukan pohon BFS adalah komponen terhubung terbesar pada graf yang mencakup simpul awal (akar).

2.3.3. Pencarian Jarak Terpendek pada Graf Tak Berbobot

Pada graf tak berbobot, jarak terpendek antara dua simpul hanya ditentukan oleh jumlah simpul atau jumlah sisi yang berada di lintasan antara dua simpul tersebut. Untuk mendapatkan jarak terpendek seperti ini, algoritma BFS dapat digunakan untuk menyelesaikan persoalan.

Pohon yang dibentuk dalam algoritma BFS memiliki sisi yang menghubungkan simpul-simpul pada setiap tingkat. Tidak mungkin ada sisi dalam graf yang terhubung dan berada di luar tingkat-tingkat pada pohon tersebut. Hal ini menandakan bahwa pohon algoritma BFS menampilkan jarak terpendek antara simpul akar dan

simpul-simpul lainnya dalam graf. Setiap simpul terhubung dengan akar, sehingga jarak terpendek dari simpul ke akar sama dengan tingkat simpul tersebut.

2.3.4. Pencarian Diameter Pohon

Diameter dari sebuah pohon adalah lintasan terpanjang dari seluruh jarak terpendek antara dua simpul pada suatu pohon. Dalam pencarian diameter pohon, digunakan algoritma BFS seperti yang telah diterangkan di atas untuk mendapatkan jarak terpendek dari setiap simpul. Hasil dari penghitungan jarak terpendek masing-masing tersebut digunakan bersama dengan satu variabel global yang menyatakan panjang maksimum untuk mendapatkan diameter pohon.

Berikut algoritma untuk mencari diameter pohon:

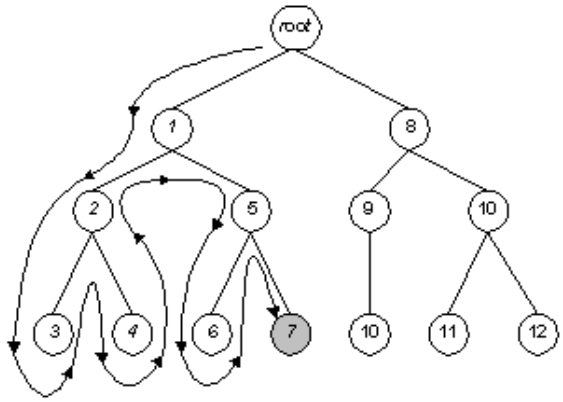
1. Inisialisasi panjang maksimum = 0
2. Lakukan algoritma BFS pada pohon
3. Masukkan hasil jarak terpendek pada suatu simpul ke variabel temp
4. Jika panjang maksimum < temp, maka panjang maksimum = temp
5. Ulangi dari langkah 2 sampai jarak terpendek seluruh simpul telah dicari
6. Kembalikan panjang maksimum

Panjang maksimum yang dikembalikan merupakan diameter pohon yang dicari.

III. ALGORITMA PENCARIAN MENDALAM (DFS)

Algoritma pencarian mendalam (DFS) mencari solusi dengan mengunjungi simpul akar, lalu simpul-simpul yang bertetangga dengan simpul akar (setingkat di bawahnya), terus sampai simpul paling dalam pada bagian tersebut. Setelah itu, dicari simpul yang telah dikunjungi pada tingkat terdekat dan terdalam, lalu simpul yang bertetangga dengan simpul ini dikunjungi, demikian seterusnya sampai seluruh simpul telah dikunjungi.

Berikut gambar yang mengiilustrasikan urutan simpul yang dikunjungi pada algoritma BFS:



Gambar 2. Ilustrasi urutan kunjungan simpul pada algoritma DFS

Dari gambar di atas, dapat dilihat bahwa dengan algoritma DFS, setiap anak simpul pertama yang bertetangga dengan simpul akar dikunjungi sampai tingkat terdalamnya lebih dahulu, lalu seluruh simpul pada subpohon tersebut, sebelum simpul lain yang juga bertetangga dengan simpul akar.

3.1. Cara Kerja Algoritma DFS

Dalam algoritma DFS, simpul yang telah dikunjungi disimpan dalam suatu tumpukan (*stack*). Antrian ini digunakan untuk mengacu simpul-simpul yang akan dikunjungi sesuai urutan tumpukan (masuk terakhir, keluar pertama) dan mempermudah proses runut-balik jika simpul sudah tidak mempunyai anak (simpul pada kedalaman maksimal).

Untuk memperjelas cara kerja algoritma DFS beserta tumpukan yang digunakannya, berikut langkah-langkah algoritma DFS:

1. Masukkan simpul ujung (akar) ke dalam tumpukan
2. Ambil simpul dari tumpukan teratas, lalu cek apakah simpul merupakan solusi
3. Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.
4. Jika simpul bukan solusi, masukkan seluruh simpul yang bertetangga dengan simpul tersebut (simpul anak) ke dalam tumpukan
5. Jika tumpukan kosong dan setiap simpul sudah dicek, pencarian selesai dan mengembalikan hasil “solusi tidak ditemukan”
6. Ulangi pencarian dari langkah kedua

3.2. Properti Algoritma DFS

Berbeda dengan algoritma BFS yang hanya melakukan pencarian solusi pada graf sampai menemukannya, sehingga membutuhkan waktu yang lama, algoritma DFS menggunakan metode heuristik, sehingga solusi dapat ditemukan lebih cepat.

Kompleksitas ruang algoritma DFS juga jauh lebih rendah dari algoritma BFS, karena hanya menyimpan simpul-simpul pada suatu subpohon. Akan tetapi, kompleksitas waktu algoritma DFS sama dengan kompleksitas waktu algoritma BFS. Algoritma DFS juga dapat mencari solusi secara tuntas seperti algoritma BFS, berarti algoritma ini akan menemukan solusi terbaik dan terdekat bagaimanapun bentuk graf tersebut. Walaupun demikian, masih terdapat kelemahan algoritma DFS, yaitu jika pada graf yang sangat besar kedalaman graf terus bertambah dan tidak diketahui akhirnya, sehingga memori tidak cukup. Hal ini dapat diatasi dengan pengembangan algoritma DFS yaitu pencarian mendalam berulang (*Iterative Deepening Search - IDS*).

3.3. Aplikasi Algoritma DFS dalam Teori Graf

Seperti halnya algoritma BFS, algoritma DFS juga dapat digunakan dengan mudah pada teori graf. Bahkan banyak persoalan yang tidak dapat diselesaikan algoritma BFS yang dapat diselesaikan oleh algoritma DFS. Berikut contoh-contoh penggunaan algoritma BFS dalam persoalan teori graf:

3.3.1. Pencarian Komponen Terhubung dalam Graf

Sama seperti pada algoritma BFS, dalam algoritma DFS, seluruh simpul yang dikunjungi juga pasti merupakan simpul yang terhubung dengan simpul akar, karena simpul-simpul tersebut hanya dikunjungi jika bertetangga dengan simpul ayahnya, sehingga seluruh simpul yang terhubung dengan simpul akar juga pasti dikunjungi.

Dalam hal pencarian komponen terhubung dalam graf, algoritma BFS dan DFS menghasilkan pencarian yang memiliki kecepatan dan kebutuhan memori sama, karena setiap simpul harus dikunjungi. Hasil pencarian juga sama, walaupun proses yang dilakukan berbeda. Kedua algoritma dapat menghasilkan komponen terhubung terbesar pada graf yang mencakup simpul awal (akar).

Oleh karena itu, kedua algoritma dapat digunakan untuk menyelesaikan persoalan ini dengan mangkus.

3.3.2. Pengurutan Topologi

Dalam teori graf, pengurutan topologi dari graf DAG (graf berarah tanpa siklus) adalah pengurutan linear simpul-simpul di dalam graf yang bersesuaian dengan urutan di mana x dikunjungi sebelum y jika terdapat lintasan dari x ke y. Pengurutan topologi dapat diaplikasikan pada penjadwalan sekuensial suatu pekerjaan.

Dengan menggunakan algoritma DFS, pengurutan topologi dilakukan dengan algoritma berikut:

1. Lakukan algoritma DFS pada graf G
2. Inisialisasi *list* kosong
3. Jika suatu simpul telah selesai dipanggil seluruh anaknya, masukkan simpul ke dalam l*i*st
4. Kembalikan list yang telah berisi semua simpul.

Urutan setiap simpul didapat dari posisinya dalam *list*, mulai dari awal *list*.

3.3.3. Pencarian Komponen Terhubung Kuat

Jika algoritma BFS dapat mencari komponen terhubung pada graf tak berarah, algoritma DFS dapat melakukan sesuatu yang lebih rumit, yaitu mencari komponen terhubung kuat pada graf berarah.

Graf berarah disebut terhubung kuat jika untuk setiap pasangan simpul A dan B terdapat lintasan dari A ke B serta dari B ke A. Komponen terhubung kuat pada graf berarah merupakan subgraf terhubung kuat maksimal pada suatu graf. Komponen ini membentuk partisi dari graf.

Dalam pencarian komponen terhubung kuat, digunakan dua kali algoritma DFS, yaitu pada graf serta graf transpose dari graf tersebut. Berikut algoritma DFS yang digunakan dalam pencarian komponen terhubung kuat:

1. Lakukan algoritma DFS pada graf G
2. Cari graf transpose dari G: G^T
3. Lakukan algoritma DFS pada graf G^T secara terbalik
4. Kembalikan output hutan DFS yang terbentuk pada langkah 3 sebagai suatu komponen terhubung kuat

IV. KESIMPULAN

Algoritma BFS dan DFS dapat memecahkan masalah-masalah klasik dalam teori graf dengan mangkus. Pemilihan algoritma yang digunakan untuk memecahkan masalah disesusaikan dengan kebutuhan karena algoritma BFS dan DFS memiliki kelebihan dan kekurangan masing-masing.

Sebagian masalah, seperti pengecekan graf bipartit lebih tepat diselesaikan dengan menggunakan algoritma BFS, sedangkan masalah lainnya seperti pencarian komponen terhubung kuat lebih tepat diselesaikan dengan menggunakan algoritma DFS. Ada pula masalah yang dapat diselesaikan sama baiknya oleh algoritma BFS dan DFS seperti pencarian komponen terhubung dalam graf.

Dengan terpecahkannya masalah-masalah klasik teori graf tersebut, terbukti bahwa pemecahan masalah-masalah aplikasi dunia nyata dengan cara pembuatan representasi masalah dalam graf dan penggunaan algoritma traversal dalam graf merupakan cara yang mangkus.

REFERENSI

[1] *Basic Graph Algorithms*. cs.anu.edu.au/student/comp3600/apac/basic_graph_algorithms.pdf. Tanggal akses: 22 Mei 2007, pukul 19.20.
[2] Eppstein, David. *BFS and DFS*. http://www.ics.uci.edu/~eppstein/161/960215.html. Tanggal akses: 22 Mei 2007, pukul 19.15.
[3] Muhammad, Rashid Bin. Breadth First Search. http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm. Tanggal akses: 22 Mei 2007, pukul 19.05.
[4] Munir, Rinaldi. (2007). *Diktat Kuliah IF2251 Strategi Algoritmik.* Program Studi Teknik Informatika, Institut Teknologi Bandung. 2007.
[5] Munir, Rinaldi. (2006). *Diktat Kuliah IF2153 Matematika Diskrit.* Program Studi Teknik Informatika, Institut Teknologi Bandung. 2006.
[6] Wikipedia, *Breadth-First Search*, http://en.wikipedia.org/wiki/Breadth_first_search. Tanggal akses: 22 Mei 2007, pukul 19.00.
[7] Wikipedia, *Depth-First Search*, http://en.wikipedia.org/wiki/Depth_first_search. Tanggal akses: 22 Mei 2007, pukul 19.10.