Aplikasi Induksi Matematika Dalam Menentukan Kompleksitas Algoritma Disjoint Set Union (DSU)

Kinantan Arya Bagaspati / 13519044
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519044@std.stei.itb.ac.id

***Abstract*—Persoalan yang sering muncul dalam keseharian masyarakat pada umumnya bisa ditafsir dalam salah satu dari banyak model matematika, dan tentunya bisa ditemukan solusi yang efisien terhadap sebuah persoalan tersebut dengan susunan langkah-langkah logika yang terstruktur dan seringkali disebut dengan nama algoritma. Salah satu dari banyak model matematika yang dapat menyelesaikan banyak persoalan ialah graf. Diantara banyak permasalahannya, terdapat suatu struktur data yang mampu menyelesaikan dua masalah sekaligus secara efisien menggunakan algoritma dengan kompleksitas kecil yang terjamin oleh pembuktian menggunakan induksi, yakni struktur data Disjoint Set Union.**

***Keywords*—Algoritma, Struktur Data, Kompleksitas, Induksi, Graf, Pohon.**

I. PENDAHULUAN

Bidang ilmu informatika merupakan bidang ilmu yang tidak dapat terpisahkan dengan bidang ilmu matematika. Banyak sekali persoalan umum yang solusinya dapat dikatakan sebagai irisan dari kedua bidang, yakni matematika dan informatika ini. Meskipun dalam praktik menekuni masing-masing bidang, tentunya terdapat banyak perbedaan, misalnya pada prinsip bahwa matematika lebih menitikberatkan pada bukti yang kokoh dan rigor dalam setiap teorinya, sedangkan informatika lebih menitikberatkan pada aplikasi setiap algoritmanya. Namun keragaman dari kedua bidang ini dapat menimbulkan sudut pandang baru dalam menyelesaikan permasalahan yang ada. Dalam hal ini masalah tersebut ialah menyusun algoritma terhadap suatu model matematika berupa graf, yang terbukti efisien secara matematis.

Algoritma merupakan hal yang menjadi salah satu esensi dalam bidang ilmu informatika. Dewasa ini, orang-orang yang menekuni bidang ilmu ini berlomba-lomba membuat algoritma lebih efisien dari sebelumnya. Seberapa efisien sebuah algoritma secara kasarnya dapat dilihat dari besarnya kompleksitas algoritma tersebut. Namun menurunkan kompleksitas sebuah algoritma tidak semudah yang dibayangkan, karena menurunkan kompleksitas suatu hal seringkali diikuti mengorbankan efisiensi hal lain. Hal ini merupakan hal yang umum dalam dunia Informatika. *Disjoint Set Union* atau dapat disingkat DSU ialah salah satu wujud dari struktur data seimbang yang memiliki kompleksitas minim secara menyeluruh dari semua fiturnya.

Permaslahan umum yang ditangani oleh struktur data DSU ialah 2 submasalah berbeda sekaligus yang muncul dalam permodelan graf dalam matematika. Persoalan pertama ialah algoritma untuk menentukan apakah dua simpul terhubung. Persoalan kedua ialah algoritma untuk menambahkan sisi dalam sebuah graf. Kedua persoalan ini terlihat sederhana apabila dilihat sebagai kesatuan yang berbeda, namun apabila ditangani sekaligus, perlu struktur data yang tidak sesederhana struktur data primitif graf yang selama ini sudah umum digunakan.

Sebagai contoh, coba selesaikan masalah ini dengan struktur *adjacency list* yakni struktur yang menyimpan daftar simpul-simpul yang bertetangga dengan suatu simpul berindeks i untuk setiap i diantara 1 hingga n, dengan n ialah banyaknya simpul. Persoalan kedua jelas dapat dengan mudah ditemukan algoritma yang berjalan dalam kompleksitas waktu O(1), yakni algoritma penambahan elemen list. Namun algoritma yang dapat menyelesaikan persoalan pertama memiliki kompleksitas minimal O(n), misalnya *Breadth First Search* (BFS) dan *Depth First Search* (DFS).

Di sisi lain, apabila efisiensi algoritma difokuskan pada persoalan pertama, struktur yang paling cocok digunakan ialah struktur array yang menyimpan identitas komponen terhubung tempat suatu simpul berada untuk setiap simpul yang ada. Tentunya terdapat algoritma trivial yang dapat menyelesaikan persoalan pertama dengan kompleksitas waktu O(1), yakni algoritma perbandingan sederhana. Namun untuk menyelesaikan persoalan kedua, apabila 2 simpul yang dihubungkan berada pada komponen terhubung yang berbeda, algoritma yang ada harus dapat melakukan pengecekan terhadap identitas komponen terhubung setiap simpul, dan mengubahnya sesuai komponen mana yang akhirnya terhubung menjadi 1 komponen. Ini tentunya mempunyai kompleksitas O(n).

DSU merupakan struktur data yang dapat menyelesaikan kedua persoalan ini masing-masing dalam O(log(n)). Berikut akan digali lebih dalam mengenai detail struktur data dan algoritma yang digunakan dalam DSU, sekaligus beserta pembuktian mengenai besar kompleksitasnya.

II. LANDASAN TEORI

A. Induksi Matematika

Induksi matematika ialah salah satu metode pembuktian suatu proposisi atau klaim yang diharapkan berlaku untuk setiap suatu kondisi terpenuhi, dengan memanfaatkan properti

keterurutan dari kondisi tersebut. Metode induksi ini terkesan dapat dikatakan sebagai pembuktian secara rekursif karena seperti memiliki efek domino dalam pelaksanaannya. Efek domino yang dimaksud ialah efek yang menyebabkan pembuktian satu kasus proposisi menyebabkan secara bertahap setiap kasus proposisi terbukti satu per satu.

Prinsip pembuktian dengan induksi matematika secara garis besar dapat dilihat dalam dua bagian saja. Bagian pertama ialah pembuktian basis proposisi, yakni satu atau lebih kasus yang perlu dibuktikan terlebih dahulu sebagai modal dalam pembuktian induksi. Bagian kedua ialah penyusunan aturan-aturan induksi yang benar dan lengkap, yakni setiap aturan tersusun atas logika yang runtut dan benar, serta apabila aturan ini digunakan dengan benar, maka dari sejumlah basis proposisi yang sudah ada, dapat disimpulkan bahwa proposisi benar untuk setiap kasus.

Berikut tiga klasifikasi induksi yang sering dijumpai dalam pembuktian matematika. Klasifikasi dilakukan berdasarkan perbedaan basis dan aturan yang digunakan, dengan memisalkan proposisi yang akan dibuktikan terhadap bilangan n ialah p(n).

1. Induksi Sederhana

   Induksi jenis ini sesuai namanya memiliki prinsip yang paling mendasar. Basis proposisi yang digunakan hanya satu kasus saja, yakni saat n=0, yakni p(0). Aturan induksi yang dibuktikan ialah aturan paling sederhana, yakni apabila p(k) benar, p(k+1) benar. Contoh aplikasi induksi jenis ini ialah pembuktian jumlah jabat tangan dari n orang ialah n(n-1)/2, serta $1^2+2^2+\ldots+n^2 = n(n+1)(2n+1)/6$.
2. Induksi yang Dirampatkan

   Induksi ini jelas lebih kompleks daripada induksi sederhana. Basis proposisi yang digunakan bisa lebih dari satu kasus dan tidak harus kasus trivial seperti n=0. Contohnya ialah membuktikan bahwa semua bilangan yang lebih dari sama dengan 8 dapat dinyatakan dalam penjumlahan sejumlah 3 dan sejumlah 5, dengan menggunakan basis p(8), p(9), dan p(10), serta aturan apabila p(k) benar, p(k+3) juga benar. Aturan induksi yang digunakan juga dapat lebih dari satu. Contohnya ialah membuktikan ketaksamaan Arithmetic Mean tidak lebih kecil dari Geometric mean dari n bilangan, dengan basis p(2), dan aturan apabila p(k) benar, maka p(2k) dan p(k-1) benar.
3. Induksi Kuat

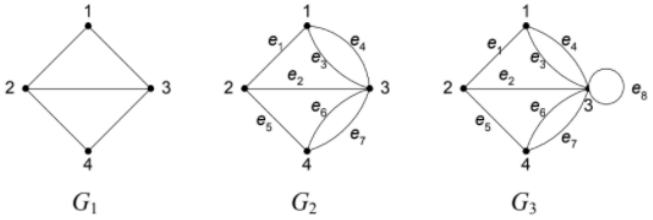
   Hal yang membedakan induksi jenis ini dengan induksi lainnya ialah aturan yang digunakan cukup spesial. Aturan yang digunakan ialah apabila p(1) hingga p(k) benar, maka p(k+1) benar. Ini membuat opsi pembuktian p(k+1) menjadi lebih banyak dibandingkan induksi sederhana. Namun basis yang harus dibuktikan juga harus diperhatikan cukup apabila diterapkan opsi-opsi dalam pembuktian setiap p(n) dalam induksi, dan hal ini merupakan kesalahan yang sering terjadi atau dilupakan.

B. Pohon dalam Graf

Graf adalah suatu struktur matematika yang berupa kumpulan titik-titik (biasa disebut simpul/*vertex*) yang tiap pasangan titiknya dapat dihubungkan dengan satu atau lebih garis (biasa disebut sisi/*edge*) atau tidak terhubung sama sekali. Formalnya, suatu graf G terdiri dari 2 himpunan (V, E). $V = \{v_1,v_2,\ldots,v_n\}$ ialah himpunan berisi n simpul pada graf, sedangkan $E = \{e_1,e_2,\ldots,e_m\}$ ialah himpunan sisi yang masing masing menghubungkan 2 buah simpul yang tidak harus berbeda. Untuk memudahkan dan menghilangkan keambiguan, semua graf dan sisi yang dibahas pada makalah ini ialah tidak berarah (*undirected*).

Sebuah sisi e = (a,b) dikatakan e bersisian dengan simpul a dan b. Dua simpul a dan b dikatakan bertetangga apabila terdapat satu sisi e yang bersisian dengan simpul a dan simpul b. Sepasang sisi e_1 dan e_2 dinamakan sisi ganda apabila kedua sisi tersebut menghubungkan sepasang simpul yang sama, formalnya $e_1 = (a,b)$ dan $e_2 = (a,b)$. Sebuah sisi e dinamakan sisi gelang apabila sisi tersebut menghubungkan 2 simpul yang sama.

Jenis sisi yang diperbolehkan ada dalam sebuah graf menentukan jenis graf tersebut, diantaranya terdapat 2 jenis, yakni graf sederhana (*simple graph*), dan graf tak sederhana (*unsimple graph*). Graf sederhana ialah graf yang tidak memperbolehkan sisi ganda atau sisi gelang. Sementara graf tidak sederhana dibagi lagi menjadi dua yakni graf ganda (*multi graph*) dan graf semu (*pseudo graph*). Graf ganda ialah graf yang memperbolehkan sisi ganda namun tidak sisi gelang. Sementara graf semu memperbolehkan sisi ganda dan sisi gelang.

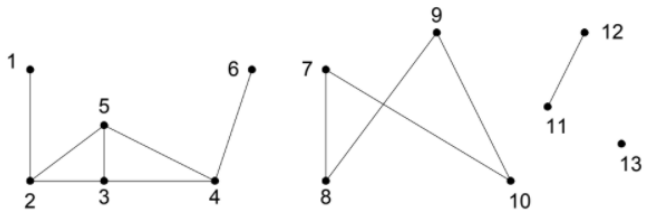


Gambar 2.1 Contoh graf dari masing-masing jenis
Sumber: Slide materi kuliah

Pada contoh di atas, G1 adalah graf dengan V = {1,2,3,4} dan E = {(1,2), (1,3), (2,3), (2,4), (3,4)} merupakan graf sederhana karena tidak memiliki sisi ganda maupun sisi gelang. G2 adalah graf dengan V = {1,2,3,4} dan E = {(1,2), (2,3), (1,3), (1,3), (2,4), (3,4), (3,4)} merupakan graf ganda karena memiliki sisi ganda yakni 2 buah (1,3) dan 2 buah (3,4). G3 adalah graf dengan V = {1,2,3,4} dan E = {(1,2), (2,3), (1,3), (1,3), (2,4), (3,4), (3,4), (3,3)} merupakan graf semu karena mempunyai sisi gelang yakni (3,3).

Lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan v_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0,e_1,v_1,e_2,v_2,\ldots,v_{n-1},e_n,v_n$ sedemikian sehingga $e_1 =(v_0, v_1)$, $e_2 =(v_1, v_2)$, ... , $e_n =(v_{n-1}, v_n)$ adalah sisi-sisi dari graf G. Sementara sirkuit ialah lintasan yang memiliki simpul awal dan simpul akhir yang sama. Panjang suatu sirkuit sama dengan panjang lintasan yang membentuknya.

Dua buah simpul a dan b yang tidak harus berbeda dalam sebuah graf dikatakan terhubung apabila terdapat lintasan dengan simpul awal a dan simpul akhir b. Komponen terhubung dalam suatu graf ialah bagian graf (*subgraph*) yag berisi simpul-simpul yang saling terhubung. Formalnya, 2 simpul berada dalam komponen terhubung yang sama jika dan hanya jika kedua simpul tersebut terhubung. Sebagai contoh, graf di bawah ini mempunyai 4 buah komponen terhubung, yakni {1,2,3,4,5,6}, {7,8,9,10}, {11,12}, dan {13}.



Gambar 2.2 Contoh graf dengan 4 komponen terhubung
Sumber: Slide materi kuliah

Misalkan G = (V, E) adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki m = n – 1 buah sisi.
4. G tidak mengandung sirkuit dan memiliki m = n – 1 buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan (jembatan adalah sisi yang bila dihapus menyebabkan graf terpecah menjadi dua komponen).

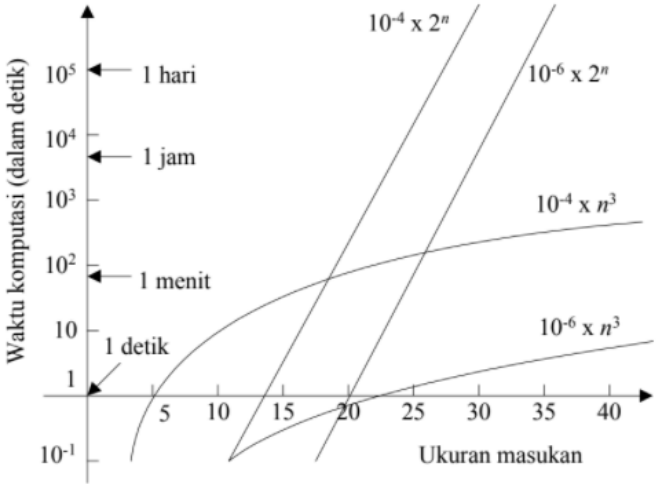
Dalam sebuah graf yang berupa pohon, terdapat satu buah simpul yang dikatakan *root* yang dapat diibaratkan sebagai akar dari pohon tersebut. Setiap simpul x yang bukan merupakan *root* memiliki *parent* simpul y yakni simpul yang bertetangga dengan x, serta y termasuk dalam lintasan yang menghubungkan x dengan *root*. *Parent* dari *root* ialah simpul itu sendiri. *Depth* atau kedalaman dari suatu pohon ialah jarak dari lintasan antara *root* dan simpul terjauh yang ada dalam pohon tersebut.

C. Kompleksitas Algoritma

Kompleksitas algoritma ialah suatu hal yang menentukan seberapa efektifnya suatu algoritma dalam menyelesaikan suatu masalah berdasarkan seberapa besarnya sumber daya yang diperlukan. Sumber daya yang dimaksud ialah waktu dan memori yang dibutuhkan oleh algoritma tersebut. Sesuai sumber dayanya, Kompleksitas algoritma dibagi menjadi dua, yakni kompleksitas waktu dan kompleksitas memori.

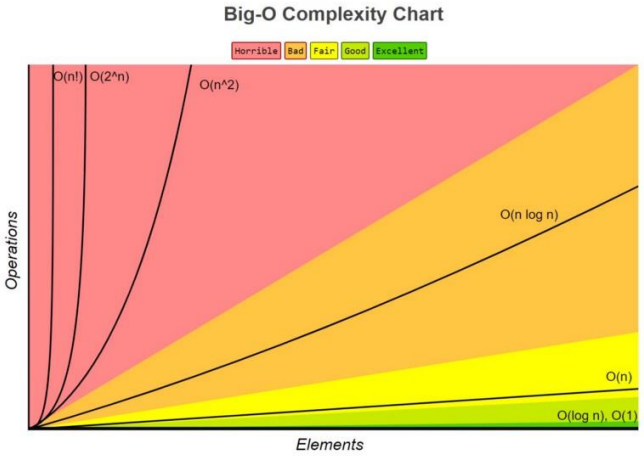
Kompleksitas waktu suatu algoritma dihitung berdasarkan satuan terkecilnya yang berupa banyaknya operasi primitif yang dilakukan algoritma tersebut. Operasi primitif diantaranya memberi nilai suatu variabel, operasi matematika (+, -, *, %, /), perbandingan, akses dan alokasi memori, dan sejenisnya. Saat ini, operasi primitif dieksekusi komputer dengan waktu rata-rata setara dengan 10^{-9} detik. Sementara

Kompleksitas memori suatu algoritma ialah banyaknya memori yang digunakan selama algoritma berlangsung. Besarnya memori yang digunakan tiap tipe data tentunya berbeda-beda mengingat kembali tipe data primitif, misalnya char (1 byte), short (2 byte), int dan float (4 byte), serta double dan long long (8 byte).



Gambar 2.3 Grafik ukuran masukan terhadap waktu komputasi secara logaritmik, untuk setiap kompleksitas waktu program
Sumber: Slide materi kuliah

Kompleksitas suatu algoritma tentunya sebanding pada kompleksitas masalah yang diselesaikan (dinyatakan sebagai n). Inilah mengapa kompleksitas algoritma jarang dihitung secara spesifik beserta konstanta-konstanta dan koefisien-koefisiennya. Karena semakin besar n, semakin tidak signifikan pengaruh konstanta dan koefisien tersebut, namun semakin terlihat bahwa pertumbuhan kompleksitas yang ada bergantung pada n. Permasalahan sekarang jadi berubah menjadi formula apa yang digunakan untuk mengaitkan kompleksitas masalah dengan kompleksitas algoritma. Oleh karena itu, digunakan notasi Big-O, Big-Ω, dan Big-Θ.



Gambar 2.4 Grafik efisiensi tiap kompleksitas
Sumber: freecodecamp.org

Notasi Big-O, Big-Ω, dan Big-Θ ini umumnya memiliki nilai diantara fungsi umum berikut ini, terurut dari rendah ke besar: 1, log(n), sqrt(n), n, n log(n), n sqrt(n), n^2, …, 2^n, dan n!. T(n) = O(f(n)) (dibaca "T(n) adalah O(f(n)), yang artinya T(n) berorde paling besar f(n)) bila terdapat konstanta C dan n_0 sedemikian sehingga $T(n) \leq C\ f(n)$ untuk $n \geq n_0$. T(n) = Ω(g(n)) (dibaca "T(n) adalah Omega (g(n))" yang artinya T(n) berorde paling kecil g(n)) bila terdapat konstanta C dan n_0 sedemikian sehingga $T(n) \geq C(g\ (n))$ untuk $n \geq n_0$. T(n) = Θ(h(n)) (dibaca "T(n) adalah tetha h(n)") yang artinya T(n) berorde sama dengan h(n) jika T(n) = O(h(n)) dan T(n) = Ω(h(n)).

Tabel 2.1 Algoritma umum setiap kompleksitas

Kompleksitas	Algoritma Umum
O(1)	Operasi primitif
O(log(n))	Binary search, FPB dengan metode gauss
O(sqrt(n))	Cek prima
O(n)	Traversal sederhana
O(n log(n))	Merge sort, Sieve of eratosthenes
$O(n^2)$	Bubble, Insertion, Selection sort
$O(n^3)$	Perkalian matriks, Eliminasi Gauss Jordan
O(n!)	Determinan dengan rekursi

D. Disjoint Set Union

Seperti yang telah dikemukakan sebelumnya, DSU ialah salah satu wujud dari struktur data seimbang yang memiliki kompleksitas O(log(n)) dalam menyelesaikan persoalan penambahan sebuah sisi dalam graf, serta pemeriksaan apakah dua simpul terhubung atau tidak. Inti dari DSU yang sebenarnya ialah menyimpan susunan setiap komponen terhubung dalam bentuk pohon dengan meminimalkan *depth*nya.

Struktur yang digunakan ialah pohon karena struktur tersebut merupakan struktur dengan sisi minimal yang telah merepresentasikan keterhubungan setiap pasangan simpul. Apabila penambahan sisi dilakukan pada 2 sisi yang sudah terhubung, maka tidak akan terjadi apa-apa karena penambahan sisi baru pada sebuah pohon akan menimbulkan adanya sirkuit sehingga bukan lagi berupa pohon. Selanjutnya, cara meminimalkan *depth* dari setiap pohon ialah dengan algoritma penambahan sisi yang selalu menjamin bahwa kedalaman tiap pohon dengan n buah simpul ialah log(n), yang akan dibuktikan dengan induksi pada bab selanjutnya.

Terdapat 3 prosedur utama dalam program DSU yang tidak ada dalam prosedur yang digunakan pada program pohon pada umumnya. Prosedur pertama ialah inisialisasi, yakni pada awal tidak ada sisi sama sekali, setiap simpul memiliki *parent* dirinya sendiri, dan bisa dianggap sebagai rootnya masing-masing. Prosedur kedua ialah memeriksa apakah 2 buah simpul terhubung, menggunakan algoritma yang sederhana yakni mencari apakah *root* dari simpul yang pertama sama dengan *root* dari simpul kedua sehingga membuktikan kedua simpul berada dalam satu pohon atau tidak. Prosedur ketigalah ialah prosedur menghubungkan kedua simpul. Prosedur inilah yang akan menjadi inti pembahasan mengapa apabila algoritma ini diterapkan, tiap pohon yang memiliki n buah simpul aka memiliki *depth* maksimal log(n).

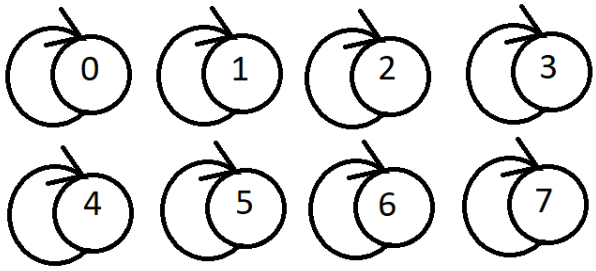
.

III. Aplikasi Induksi Matematika Dalam Menentukan Kompleksitas Algoritma Disjoint Set Union (DSU)

A. Struktur Data DSU dan Algoritma yang Digunakan

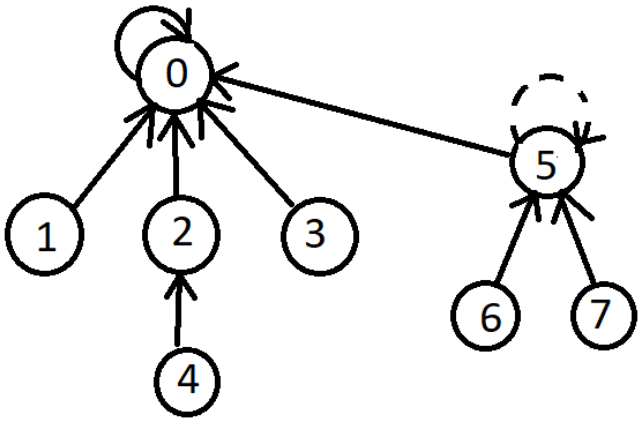
DSU merupakan struktur data yang memiliki properti menyimpan informasi apabila 2 simpul sembarang terhubung dalam graf atau tidak, serta mengupdate informasi tersebut setiap ada sisi baru ditambahkan dalam graf dengan keduanya memiliki kompleksitas maksimal sebanyak O(log n), dengan n menyatakan banyaknya simpul dalam graf. Oleh karena itu dapat ditebak dari namanya, esensi dari DSU ialah proses menggabungkan 2 komponen terhubug berbeda sambil tetap mempertahankan struktur data yang ada (Uniting 2 Disjoint Sets).

Pertama akan dibahas struktur data dari DSU secara detail. Misalkan graf yang tersedia memiliki sebanyak n simpul, yang diberi indeks 0 sampai dengan n-1. Struktur data ini menyimpan indeks *parent* dari masing-masing simpul dalam sebuah array par[n]. Dalam setiap saat selama program berjalan, suatu simpul yang *parent*nya ialah dirinya sendiri disebut *root*, serta dapat dijamin bahwa setiap komponen terhubung hanya akan memiliki 1 *root* saja.



Gambar 3.1 Kondisi awal DSU untuk graf bersimpul 8
Sumber: Dokumen pribadi

Pada awalnya *parent* dari masing-masing simpul ialah simpul itu sendiri (par[i]=i) seperti pada gambar di atas, sehingga terdapat n komponen terhubung berbeda dan tiap simpul adalah *root* dari komponen terhubungnya masing-masing. Depth dari sebuah komponen terhubung adalah jarak maksimal suatu simpul dalam komponen terhubung dengan *root* dari komponen terhubung tersebut. Suatu simpul a yang bukan *root* dikatakan berjarak b dengan *root*nya apabila par[par[…par[a]…]] = *root* (par muncul sebanyak b kali). Struktur data ini juga menyimpan banyak simpul yang terdapat dalam satu komponen terhubung dari setiap simpul, ke dalam sebuah array set[n].



Gambar 3.2 Prosedur penambahan edge yang menghubungkan vertex 4 dengan 6, sehingga parent dari vertex 5 berubah
Sumber: Dokumen pribadi

Terdapat pula prosedur penambahan sisi yang harus diikuti agar struktur DSU tetap terjaga propertinya. Misalkan sisi baru yang ditambah ini menghubungkan simpul a dengan b. Apabila simpul a dan b sudah berada dalam komponen terhubung yang sama (dapat dicek dengan menghitung terus menerus parent a

hingga mencapai *root*nya, kemudian cek apakah sama dengan *root* dari b), maka abaikan sisi ini (karena DSU hanya memperdulikan informasi 2 simpul sembarang terhubung atau tidak). Sedangkan apabila simpul a dan b sudah berada dalam komponen terhubung yang berbeda, misalkan berturut-turut set X dan set Y. Misalkan komponen terhubung X memiliki x buah simpul dan komponen terhubung Y memiliki y buah simpul (set[root[X]] = x dan set[root[X]] = y). Apabila x>y, maka ubah nilai parent dari *root* Y menjadi *root* X (par[root[Y]] = root[X]), sehingga *root* dari komponen terhubung gabungan ialah *root* awal dari X. Jika tidak, lakukan sebaliknya (par[root[X]] = root[Y]), sehingga *root* dari komponen terhubung gabungan ialah *root* awal dari Y.

B. Kompleksitas Waktu DSU

Dapat diperhatikan bahwa proses penambahan sisi dan proses pengecekan 2 simpul a dan b terhubung atau tidak berbanding lurus dengan depth maksimal dari komponen terhubung yang mengandung a dan komponen terhubung yang mengandung b. Algoritma dari struktur DSU ini mengatakan bahwa kompleksitas dari program ialah sebesar O(log n) dengan n ialah banyaknya simpul dalam graf. Selanjutnya akan dibuktikan mengapa terjadi hasil perhitungan demikian.

Sekarang cukup dibuktikan mengapa dengan menjaga struktur ini, suatu komponen terhubung yang terdiri dari sebanyak a simpul akan memiliki depth sebesar maksimal log(a). Metode pembuktian yang akan digunakan ialah metode induksi kuat. Pertama ditinjaulah kasus basis, yakni apabila a=1, jelas depth dari komponen terhubung berisi 1 titik pasti bernilai 0, sehingga sesuai dengan hal yang ingin dibuktikan. Kemudian asumsikan bahwa pernyataan/proposisi benar untuk a=1,2,3,…,b untuk suatu bilangan asli b, lalu akan dibuktikan pernyataan tersebut juga benar untuk a=b+1.

Sekarang tinjau sebuah komponen terhubung dengan b+1 simpul. Sebelumnya, pasti komponen terhubung tersebut dibentuk dari 2 komponen terhubung X dan Y yang keduanya memiliki jumlah simpul tidak lebih besar dari b+1, dan dapat dimisalkan 2 komponen terhubung tersebut berturut-turut memiliki x simpul dan y simpul, dengan x tidak lebih kecil dari y. Berdasarkan asumsi induksi, karena x dan y tidak lebih besar dari b, maka komponen terhubung X dan Y berturut-turut memiliki depth log(x) dan log(y). Karena x tidak lebih kecil dari y, maka menurut algoritma DSU, *root* dari Y lah yang dihubungkan ke *root* dari X, sehingga komponen terhubung gabungan akan memiliki depth sebesar min(log(x), log(y)+1). Namun jelas log(x) < log(x+y), serta log(y)+1 = log(2y) ≤ log(x+y) = log(b+1), sehingga terbukti bahwa pernyataan juga benar untuk a=b+1. Maka dengan metode induksi, telah terbukti bahwa komponen terhubung yang terdiri dari sebanyak a simpul akan memiliki depth sebesar maksimal log(a), berlaku untuk setiap a bilangan asli.

C. Kompleksitas Memori DSU

Terdapat 2 array yang digunakan dalam algoritma DSU ini. Array pertama ialah par[n] yang menyimpan *parent* dari masing-masing simpul. Array kedua ialah set[n] dengan set[i] menyimpan banyaknya simpul pada komponen terhubung dengan *root* simpul i. Sehingga mudah dipastikan bahwa bayak memori yang digunakan setara dengan 2n sehingga memiliki kompleksitas O(n).

IV. STUDI KASUS

Suatu contoh soal klasik yang jelas dapat diselesaikan menggunakan DSU ialah sebagai berikut. Dipunyai graf dengan n simpul berindekskan 1 sampai n. Terdapat pula sejumlah q *query* yang masing masing dapat menyambungkan 2 simpul a dan b dengan satu sisi, atau menanyakan apakah 2 simpul a dan b terhubung atau tidak. Input akan berisi q+1 baris dengan baris pertama memberikan nilai n dan q dengan $1 \leq n$, $q \leq 3.10^5$, dan q baris selanjutnya berisi nilai c, a, dan b. Apabila c bernilai 0 maka program menjawab apakah 2 simpul a dan b terhubung atau tidak dengan mengoutput nilai 1 apabila terhubung, 0 jika tidak. Sementara apabila c bernilai 1 maka program menyambungkan 2 simpul a dan b dengan satu sisi.

Untuk mendemonstrasikan hasil yang telah diperoleh pada bab 3 makalah ini, dapat dibuat program DSU dalam bahasa C++ untuk menyelesaikan persoalan klasik yang telah dikemukakan. Terdapat tambahan variabel global berupa operations yang nilai awalnya 0 dan setiap operasi primitif yang dilakukan program, nilainya bertambah. Variabel inilah yang menjadi pertimbangan kompleksitas waktu dari DSU sehingga dapat dicocokkan dengan hasil pada bab 3.

```
4   int parent[100000], st[100000], operations;
5
6   void init_dsu(int n){
7       for(int i=1; i<=n; i++){
8           parent[i] = i;
9       }
10      for(int i=1; i<=n; i++){
11          st[i] = 1;
12      }
13      operations = 0;
14  }
```

Gambar 4.1 Prosedur inisialisasi DSU
Sumber: Dokumen pribadi

Hanya hal-hal trivial saja yang tertulis dalam prosedur inisialisasi ini. Pertama ialah variabel global array parent, array set, dan operations. Array parent diinisialisasi sedemikian sehingga setiap simpul memiliki dirinya sendiri sebagai parent, serta banyaknya simpul dalam komponen terhubung semua bernilai 1, sebagaimana diinisialisasi dalam array set. Variabel operations juga diinisialisasi dengan nilai 0.

```
16  int root(int a){
17      int u, v;
18      u = a;
19      while(parent[u]!=u){
20          u = parent[u];
21          operations++;
22      }
23      return u;
24  }
25
26  void check_connect(int a, int b){
27      if(root(a) == root(b)){
28          cout << "connected" << endl;
29      }else{
30          cout << "not connected" << endl;
31      }
32  }
33
34  void join(int a, int b){
35      int u = root(a);
36      int v = root(b);
37      if(u != v){
38          if(st[u]<st[v]){
39              parent[u] = v;
40              st[v] += st[u];
41          }else{
42              parent[v] = u;
43              st[u] += st[v];
44          }
45          operations+=2;
46      }
47  }
```

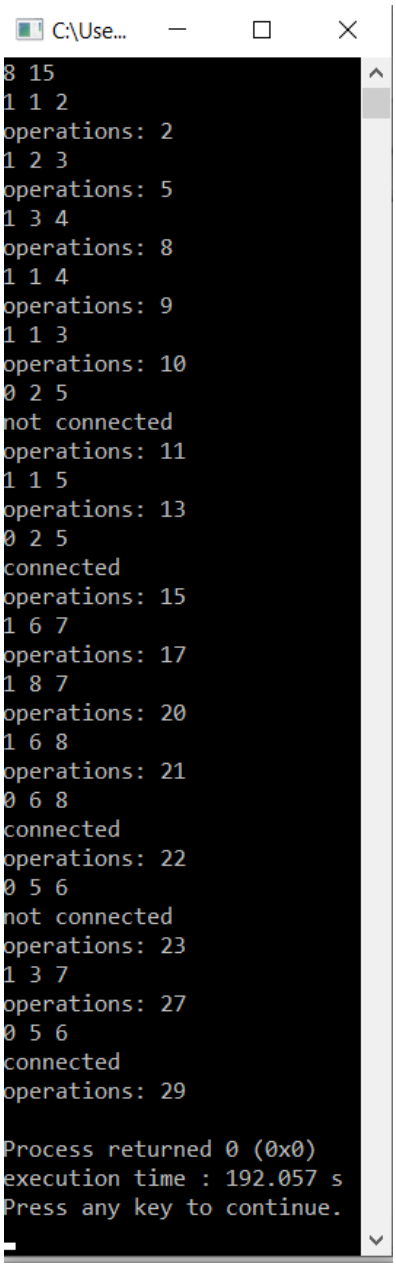
Gambar 4.2 Prosedur cek terhubung dan penambahan sisi
Sumber: Dokumen pribadi

Pada bagian kode prosedur check_connect dan join, dibentuk sebuah fungsi pembantu yakni root(a) untuk menelusuri simpul apa yang menjadi root dalam komponen terhubung yang mempunyai simpul a sebagai anggotanya. Fungsi root ini digunakan untuk memeriksa apakah a dan b berada pada komponen terhubung yang sama dalam prosedur check_connect. Selanjutnya fungsi root digunakan untuk melihat informasi banyaknya simpul pada kedua komponen terhubung a dan b dalam prosedur join. Tiap prosedur juga disertai penambahan nilai variabel global operations yang sesuai

```
49  int main(){
50      ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0)
51      int n, q, a, b, c;
52      cin >> n >> q;
53      init_dsu(n);
54      while(q>0){
55          cin >> c >> a >> b;
56          if(c==0){
57              check_connect(a, b);
58          }else{
59              join(a, b);
60          }
61          cout << "operations: " << operations << endl;
62          q--;
63      }
64  }
```

Gambar 4.3 Kode main
Sumber: Dokumen pribadi

Bagian yang tertulis dalam kode main tak lain ialah realisasi dari persoalan yang telah dijelaskan diawal bab ini. Hal yang tidak ada dalam program DSU pada umumnya ialah pemanggilan nilai operations setiap sebuah prosedur dijalankan. Berikut ialah contoh kasus yang digunakan. Mudah diperiksa secara manual bahwa program ini benar untuk setiap *query* yang ada.



Gambar 4.4 Hasil test case
Sumber: Dokumen pribadi

Setelah mempelajari berbagai kasus, dapat diperkirakan banyaknya operasi secara kasar dari DSU dibandingkan denngan dua struktur data lain yakni *Adjacency Matriks* dan *Connected Set* yang sempat disinggung dalam makalah ini. Berikut ialah tabel dari beberapa kasus beserta perbandingan banyaknya operasi dari ketiga struktur data yang disajikan dalam bentuk tabel. Mudah dilihat bahwa Kompleksitas DSU lebih kecil dalam kasus-kasus dengan jumlah query seimbang.

Tabel 4.1 Perbandingan Kompleksitas

Banyak query		Adjacency Matrix	DSU	Connected Set
Cek koneksi	Join			
0	1000	1000	2500	500000
200	800	10800	2000	360200
400	600	80600	1750	180400
600	400	180400	1750	80600
800	200	360200	2000	10800
1000	0	500000	2500	1000

V. KESIMPULAN

DSU merupakan salah satu struktur data seimbang yang memiliki kompleksitas minim secara menyeluruh dari dua masalah yakni soal menentukan apakah dua simpul terhubung atau tidak, serta soal menambahkan sisi pada graf. Kedua masalah telah terbukti dapat diselesaikan dengan algoritma yang keduanya memiliki kompleksitas O(log(n)). Dalam kasus yang memiliki jumlah *query* persoalan pertama seimbang dengan jumlah *query* persoalan kedua, tentunya performa struktur DSU akan jauh lebih efisien dibandingkan struktur *Adjacency List* yang memiliki kompleksitas O(1) pada persoalan kedua namun O(n) pada persoalan pertama, serta struktur yang menyimpan komponen terhubung setiap simpul yang memiliki kompleksitas O(1) pada persoalan pertama, namun O(n) pada persoalan kedua. Namun tetap harus diperhatikan bahwa DSU ini tidak selalu lebih baik daripada kedua struktur lainnya, yakni saat banyak *query* sangat timpang.

Keberadaan suatu struktur yang menyeimbangkan kompleksitas dari dua masalah tidak jarang dijumpai dalam dunia informatika. Sebagai contoh permasalahan yang menyangkut suatu array bilangan dengan persoalan pertama berupa mengubah satu elemen array menjadi bilangan yang lain, serta persoalan kedua berupa menanyakan jumlah semua elemen array yang indeksnya ada dalam suatu range (l, r). Dengan struktur array biasa, tentunya persoalan pertama dapat dicapai dalam O(1), namun persoalan kedua dicapai dalam O(n). Dengan struktur *prefix sum* (array yang menyimpan jumlah elemen pertama hingga elemen ke-i), tentunya persoalan kedua dapat dicapai dalam O(1), namun persoalan pertama dicapai dalam O(n). Solusi yang tepat ialah menggunakan struktur *fenwig tree* yang, sama seperti DSU, dapat menyelesaikan kedua persoalan tersebut dalam O(log(n)).

Fakta bahwa keberadaan struktur DSU yang menyeimbangkan kompleksitas kedua masalah merupakan hal yang sering muncul dalam dunia informatika, tentunya menambah nilai makalah ini, yang diharapkan dapat digunakan dalam beberapa kasus lain yang serupa. Struktur data DSU dan algoritma yang digunakan dalam makalah ini tentunya jauh dari kata paling efisien, sebagai contoh saja terdapat algoritma modifikasi DSU yang diklaim dapat menyelesaikan kedua persoalan dalam kompleksitas O(1) dalam sebagian besar kasus. Modifikasi yang digunakan ialah mengupdate *parent* suatu simpul setiap simpul tersebut diakses sehingga lama kelamaan setiap komponen terhubung akan memiliki *depth* 1. Sehingga tidak bisa dipungkiri bahwa DSU beserta pembuktian induksi matematika masih dapat dikembangkan seiring berjalannya waktu dan akan selalu berperan sebagai batu pijakan menuju algoritma di masa yang akan dating yang lebih efisien.

VI. UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan terimakasih kepada puji syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmatnya, penulis bisa menyelesaikan tugas makalah ini. Penulis juga menguncapkan terimakasih kepada Ibu Ulfa selaku dosen mata kuliah Matematika Diskrit, yang selama ini membimbing pembelajaran Matematika Diskrit yang sangat membantu pengerjaan makalah ini dan Bapak Rinaldi Munir, yang selama satu semester ini menyediakan website yang dapat dengan mudah diakses berisi materi-materi kuliah, latihan-latihan soal untuk kuis dan ujian, dan semua dokumen pembelajaran, soal dan lainnya yang tentunya berguna dalam proses pembelajaran Matematika Diskrit.

REFERENCES

[1] http://cep.lppm.itb.ac.id/assets/upload/file/pemrograman-kompetitif-dasar.pdf. Diakses pada 9 Desember 2020.
[2] https://www.freecodecamp.org/news/all-you-need-to-know-about-big-o-notation-to-crack-your-next-coding-interview-9d575e7eec4/. Diakses pada 9 Desember 2020.
[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Induksi-matematik-bagian2-2020.pdf. Diakses pada 9 Desember 2020.
[4] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf. Diakses pada 9 Desember 2020.
[5] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf. Diakses pada 9 Desember 2020.
[6] https://cp-algorithms.com/data_structures/disjoint_set_union.html. Diakses pada 9 Desember 2020.
[7] https://www.geeksforgeeks.org/disjoint-set-data-structures/. Diakses pada 9 Desember 2020.

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, 9 Desember 2020

Kinantan Arya Bagaspati