Menghitung Banyak Cara Penyebaran Covid-19 pada Suatu Daerah Bermodel Struktur Data Pohon dengan Menggunakan *Dynamic Programming* dan *Combinatorics*

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

***Abstract*—Di masa pandemi seperti saat ini, penyebaran Covid-19 sangatlah meresahkan. Bisa saja risiko ancaman akan penyebaran Covid-19 pada daerah sekitar sudah sangat besar. Dengan perkiraan tersebut kita akan memodelkan lingkungan daerah tempat tinggal dengan menggunakan struktur data pohon. Dengan memodelkan daerah tempat tinggal dengan cara tersebut, banyak cara penyebaran dengan asumsi sumber hanya berasal dari satu titik dapat dihitung. Salah satu metode untuk menghitung banyak kemungkinan penyebaran ini adalah dengan menggunakan *dynamic programming* dan *combinatorics*. Dengan menggunakan teknik ini, perhitungan cara penyebaran dengan sumber yang tidak menentu dapat diselesaikan dengan kompleksitas waktu yang rendah.**

Keywords—Covid-19; pohon; dynamic programming; combinatorics

I. PENDAHULUAN

Penyebaran Covid-19 adalah salah satu masalah nyata yang sedang dihadapi olah umat manusia saat ini. Hal ini disebabkan antara lain karena mudahnya penyebaran virus ini dapat terjadi. Dengan hanya berdekatan, seseorang yang sehat dapat tertular virus corona dari orang lain yang sedang mengalami sakit corona. Dengan adanya fakta ini, permasalahan virus corona ini tentu saja menjadi masalah yang kompleks dan tidak dapat diselesaikan dengan instan oleh pemerintahan negara manapun.

Pada masa karantina, seseorang akan diperintahkan untuk tetap berdiam diri di dalam rumah. Dengan semua orang mematuhi perintah ini, kita dapat berasumsi bahwa seseorang akan jarang untuk berkegiatan di luar rumah kecuali ketika ada keperluan yang sangat penting. Dengan asumsi ini, akan dimodelkan sebuah daerah tempat tinggal dengan sebuah struktur data pohon. Selain itu, karena seluruh orang akan beraktifitas di dalam rumah pada sebagian besar waktunya, akan diasumsikan juga penyebaran tidak akan datang secara tiba-tiba yang artinya penyebaran hanya dapat di mulai dari satu titik sumber saja yang kemudian dapat menyebar ke titik-titik yang bertetanggaan.

Dari model tersebut, permasalahan penghitungan banyak cara penyebaran dapat didefinisikan sebagai masalah penomoran simpul-simpul sebuah pohon dengan aturan tertentu. Penomoran setiap simpul harus dimulai dari suatu simpul yang akan diberikan nomor 1 yang menyatakan sumber penyebaran dan kemudian penomoran akan dilanjutkan dengan men-*traverse* pohon dan memberikan nomor pada simpul-simpul yang dilewati dengan meng-*increment* nomor terakhir yang sudah digunakan.

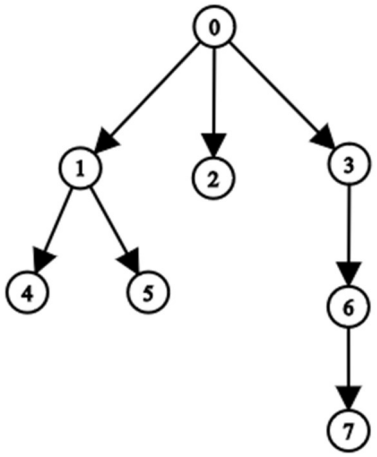
Setelah masalah didefinisikan sebagai permasalahan penomoran simpul-simpul pada pohon, banyak cara penomoran dapat dihitung secara naif yaitu dengan melakukan *brute-force/exhaustive search*. Akan tetapi, cara tersebut akan memakan waktu sangat lama yaitu dalam kompleksitas waktu *non-polynomial*.

Maka dari itu, perlulah penggunaan cara lain agar tidak terjadi perhitungan yang berulang. Salah satu cara untuk mencegah ini adalah dengan menggunakan *dynamic programming*. Dengan menggunakan paradigma program dinamis, permasalahan pada subpohon dapat diselesaikan dan kemudian hasil perhitungan ini dapat disimpan sehingga hal seperti perhitungan berulang tidak perlu dilakukan. Namun, dengan menggunakan *dynamic programming* muncul permasalahan ketika ingin menggabungkan sub-solusi. Penggabungan sub-solusi akan memerlukan *combinatorics* dalam menggabungkannya.

II. TEORI DASAR

A. Struktur Data Pohon

Pohon adalah sebuah struktur data non-linear. Sebuah pohon merepresentasikan struktur hierarkikal yang dimiliki oleh simpul-simpulnya yang salah satunya diantara mereka berperan sebagai akar.



Gambar 1 Contoh pohon

Beberapa istilah dalam struktur data pohon adalah sebagai berikut:

1. Simpul
   Simpul adalah sebuah elemen yang menyusun sebuah struktur data pohon yang didalamnya bisa menyimpan informasi dan biasanya diberikan identitas unik berupa indeks.
2. Akar
   Akar adalah sebuah simpul yang memiliki letak paling tinggi pada struktur data pohon.
3. Lintasan (*path*)
   Lintasan adalah simpul-simpul yang harus dilewati untuk berpindah dari suatu simpul *a* ke simpul *b.*
4. Tingkat (*level*)
   Tingkat sebuah simpul pada suatu pohon adalah besar jarak simpul tersebut terhadap akar.
5. Tinggi atau Kedalaman
   Tinggi atau kedalama dari sebuah pohon adalah tingkat maksimum dari seluruh simpul-simpul yang dimiliki oleh pohon.
6. *Predecessor*
   Predecessor adalah istilah yang menyatakan simpul yang berada satu level di atas suatu simpul yang sedang diacu.
7. *Successor*
   Successor adalah istilah yang menyatakan simpul yang berada satu level di bawah suatu simpul yang sedang diacu.
8. Orang tua (*parent*)
   Predecessor satu level di atas suatu simpul.
9. Anak (*child*)
   Successor satu level di bawah suatu simpul.
10. 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*.
11. Saudara kandung (*Sibling*)
    Simpul-simpul yang memiliki parent dan leluhur yang sama.
12. Upapohon/subpohon (*subtree*)
    Bagian dari pohon yang berupa suatu simpul beserta keturunannya dan memiliki karakteristik dari pohon itu sendiri.
13. Derajat
    Derajat sebuah simpul pada pohon menyatakan banyak anak yang dimiliki atau banyak simpul yang bertetanggaan.
14. Daun
    Daun adalah sebuah simpul pada pohon yang tidak memiliki anak.
15. Simpul Dalam
    Simpul dalam adalah simpul pada pohon yang memiliki anak dan juga memiliki orang tua.

B. Dynamic Programming

Dynamic programming atau program dinamis adalah teknik penyelesaian masalah dengan memecah atau mengurangi masalah kedalam sub-sub masalah yang lalu akan dicari solusi dari sub-sub masalah tersebut untuk kemudian dikombinasikan sehingga didapat solusi dari masalah. Meskipun hal ini terdengar mirip dengan metode *divide and conquer* dan *decrease and conquer*, terdapat perbedaan bahwa pada permasalah yang dipecahkan *dynamic programming* memiliki sub-sub masalah yang bisa saling beririsan.

Teknik *dynamic programming* biasanya digunakan untuk menyelesaikan masalah pencarian nilai minimum atau maksimum dari suatu masalah. Tidak hanya hal tadi, teknik ini juga dapat digunakan pada masalah optimisasi lainnya yang memiliki beberapa karakteristik yang mirip. Karakteristik masalah yang bisa ditemui pada masalah-masalah optimisasi adalah

1. Masalah membutuhkan informasi dari sub-sub masalah untuk menentukan solusi
2. Terdapat banyak kemungkinan yang menjadi kandidat dari solusi
3. Sub masalah bisa memiliki irisan antara satu dengan lainnya

Dari karakteristik masalah tadi, *Dynamic programming* memiliki keunggulan dalam memecahkannya. Keunggulan ini terletak pada kemampuan menyimpan solusi sub masalah sehingga penentuan solusi dapat dilakukan dengan memperhitungkan serangkaian keputusan yang telah diambil sebelumnya. Selain itu, kemampuan ini juga menyebabkan tidak diperlukan untuk menyelesaikan sub-sub masalah yang beririsan secara berulang-ulang.

Pada umumnya, terdapat dua cara untuk mengimplementasikan *dynamic programming* yaitu secara *top-down* dan secara *bottom-up*. *Top-down* adalah penyelesaian masalah dengan melihat masalah dimulai dari bagian yang terbesar yaitu masalah itu sendiri dan kemudian dilanjutkan dengan melihat sub masalah yang dimiliki. Sementara, pada *bottom-up* penyelesaian masalah dilakukan dimulai dari sub-masalah terkecil yang akan dimanfaatkan untuk menyelesaikan sub masalah yang lebih besar.

Mengimplementasikan *dynamic programming* dengan cara *top-down* dilakukan dengan *memoization*. *Memoization* pada dasarnya adalah menyimpan hasil perhitungan sebuah pemanggilan fungsi pada sebuah struktur data seperti array contohnya. Dengan menggunakan *memoization*, pada penyelesaian suatu fungsi yang perhitungannya membutuhkan relasi terhadap fungsi itu sendiri dapat dilakukan dengan cepat karena pemanggilan suatu fungsi maksimal hanya akan dilakukan sekali saja. Pada kasus *dynamic programming*, *memoization* dapat dilakukan dimulai dengan mendefinisikan solusi *permasalahan* dalam bentuk fungsi rekursif.

Sementara, untuk mengimplementasikan *dynamic programming* dengan cara *bottom-up* dapat dimulai dengan mendefinisikan solusi masalah dalam bentuk fungsi rekursif. Setelah itu, penyelesaian fungsi rekursif harus dilakukan secara

berurutan dari sub masalah yang terkecil yaitu basis ke sub masalah yang lebih besar.

Dari bahasan tersebut, didapat langkah-langkah pengembangan algoritma program dinamis adalah sebagai berikut

1. Karakteristikkan struktur solusi optimal
2. Definisikan secara rekursif nilai solusi optimal
3. Hitung nilai solusi optimal secara maju atau mundur
4. Konstruksi solusi optimal

C. *Combinatorics*

Combinatorics atau kombinatorial adalah cabang dari ilmu matematika yang utamanya mempelajari perhitungan yang berhubungan dengan suatu struktur, diantaranya adalah suatu pengurutan atau konfigurasi. Salah satu aplikasi dari kombinatorial adalah untuk menghitung banyaknya cara dalam mengatur objek-objek tertentu dengan aturan tertentu.

Di dalam menghitung banyaknya cara mengonfigurasi suatu hal, terdapat dua kaidah yang digunakan dalam *combinatorics*, yaitu

1. Kaidah perkalian (*rule of product*)

   Jika terdapat beberapa percobaan yang terpisah dan dari masing-masing percobaan terpisah tersebut memiliki banyak kemungkinan jawaban $p_1, p_2, \dots, p_n$, maka bila seluruh percobaan itu dilakukan, akan terdapat banyak kemungkinan jawaban sebesar $p_1 \times p_2 \times \dots \times p_n$.

2. Kaidah penjumlahan (*rule of sum*)

   Jika terdapat beberapa percobaan dan dari masing-masing percobaan tersebut memiliki banyak kemungkinan jawaban $p_1, p_2, \dots, p_n$, maka bila hanya satu percobaan yang dilakukan, akan terdapat banyak kemungkinan jawaban sebesar $p_1 + p_2 + \cdots + p_n$.

Menghitung banyak pengurutan yang dilakukan terhadap suatu himpunan barang unik yang *finite*, dapat dilakukan dengan menggunakan permutasi. Hal ini sesuai dengan kaidah perkalian, karena konfigurasi dilakukan dengan memilih barang satu per satu hingga seluruh barang tersusun. Hal ini menyebabkan kandidat barang yang akan dipilih terus berkurang pada setiap pemilihan. Misal terdapat *n* barang unik yang ingin disusun, maka pada pemilihan barang yang di posisi pertama ada n kemungkinan, kemudian untuk posisi kedua ada $n-1$ kemungkin, begitu seterusnya hingga hanya tersisa barang terakhir. Dengan mengikuti kaidah perkalian, didapat bahwa untuk menyusun r barang yang dipilih dari n barang unik, terdapat $n \times (n-1) \times \dots \times (n-r+1)$ kemungkinan yang bernilai sama dengan $P(n,r) = \frac{n!}{(n-r)!}$.

Jika, perhitungan kemungkinan dilakukan pada pengambilan barang tetapi tanpa memperhitungkan urutan pengambilan, dapat digunakan formula kombinasi.

$$C(n,r) = \frac{n!}{(n-r)!\,r!}$$

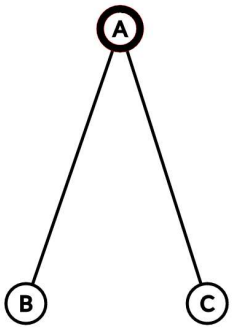
Ide formula kombinasi dapat diturunkan dengan menggunakan permutasi. Misal terdapat n buah barang unik, dan ingin dihitung berapa banyak kemungkinan untuk mengambil r buah barang dari himpunan tersebut. Maka, hal ini dapat dihitung dengan menggunakan permutasi $P(n,r)$ yaitu yang berarti mengurutkan r buah barang dari himpunan kemudian dengan membagi hasil tersebut dengan banyak kemungkinan untuk mengurutkan r. Sehingga akan didapat persamaan $C(n,r) = \frac{P(n,r)}{r!}$.

III. Pemodelan dan Pendefinisian Masalah

A. *Pemodelan Masalah*

Tempat tinggal pada permasalahan penyebaran Covid-19 dimodelkan dengan struktur data pohon. Masing-masing simpul pada struktur data pohon ini akan merepresentasikan satu buah rumah, sementara masing-masing sisi pada pohon akan merepresentasikan jalan yang menghubungkan tepat dua buah rumah. Selain itu, sisi pada pohon tidak berarah, hal ini menyesuaikan pada fakta bahwa sebuah jalan dapat ditempuh dalam dua arah.

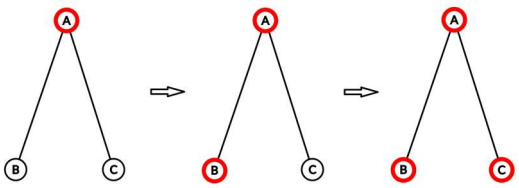
Sementara, penyebaran Covid-19 pada daerah tempat tinggal akan dimulai dari satu buah simpul yang kemudian akan menyebar ke sekitar. Misal terdapat daerah tempat tinggal dengan 3 rumah seperti berikut



Gambar 2 Model tempat tinggal 1

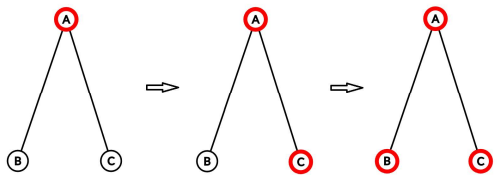
Titik awal penyebaran dapat dimulai dari simpul manapun. Sebagai contoh, misalnya titik awal penyebaran adalah simpul A, Maka akan ada dua kemungkinan penyebaran

1. $\{A\} \rightarrow \{A,B\} \rightarrow \{A,B,C\}$



Gambar 3 Model tempat tinggal skenario 1

2. $\{A\} \rightarrow \{A,C\} \rightarrow \{A,B,C\}$



Gambar 4 Model tempat tinggal skenario 2

B. Pendefinisian Masalah

pada suatu daerah tempat tinggal yang di dalamnya terdapat n buah rumah bernomor 1 sampai n, maka permasalahan dapat didefinisikan sebagai sebuah pohon dengan n buah simpul dan dihubungkan dengan sisi-sisi yang tidak berarah yang kemudian akan dicari banyak cara penomoran pada pohon dimulai dari sebuah simpul sembarang dengan aturan penomoran dimulai dari angka 1 dan dilanjutkan hingga angka ke-n dengan penomoran pada simpul diberikan pada simpul yang bertetanggaan terhadap salah satu simpul yang sudah diberi nomor. Misal kemungkinan penomoran pohon dimulai dari simpul i dinotasikan dengan $count(i)$, maka perhitungan yang akan dicari adalah

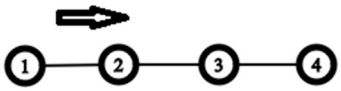
$$result = \sum_{i=1}^{n} count(i)$$

IV. Analisis dan Perancangan Algoritma

A. Solusi Dengan Satu kemungkinan Sumber

Untuk menyelesaikan permasalahan akan dianalisis bagaimana cara menghitung kemungkinan jika hanya ada satu pilihan sumber yaitu simpul i.

Jika pohon berbentuk linear, Ketika simpul yang menjadi sumber adalah yang berderajat 1, maka hanya akan ada 1 kemungkinan penyebaran.



Gambar 5 Penomoran linear

Sementara, jika simpul yang dipilih menjadi sumber adalah yang berderajat 2, akan diperlukan kombinatorik untuk menghitung kemungkinan penomoran selanjutnya. Banyak kemungkinan yang ada, akan ditentukan dengan banyak simpul yang ada pada sebelah kiri dari simpul yang telah diambil dan banyak simpul yang ada pada sebelah kanan dari simpul yang telah diambil. Misal, banyak simpul yang berada di kiri adalah n dan yang berada di sebelah kanan adalah m, maka untuk memberi penomoran selanjutnya bisa dilakukan pergerakan ke kiri sebanyak n dan pergerakan ke kanan sebanyak m. Hal ini akan menghasilkan sebuah formula banyak kemungkinan.

$$count(i) = \frac{(n+m)!}{n!\,m!}$$

Hal ini mirip dengan formula dari kombinasi, karena memang pada dasarnya tahap ini adalah melakukan pemilihan n buah atau m buah benda dari $(n+m)$ pilihan benda. Perhatikan bahwa formula ini berlaku juga untuk pemilihan sumber simpul berderajat 1.

Jadi, dari pembahasan tersebut didapat bahwa jika pohon berbentuk linear dan memiliki n buah simpul, maka

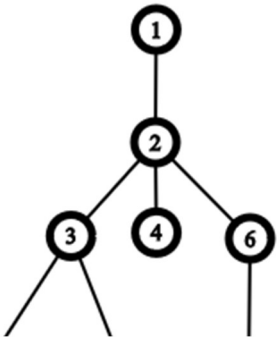
$$count(i) = C(n-1, k),$$

dengan nilai k adalah banyak simpul di sebelah kanan sumber atau simpul di sebelah kiri sumber.

Setelah menyelesaikan subset dari masalah, terlihat bahwa untuk menggabungkan solusi sub masalah(pada kasus ini adalah sub masalah sebelah kiri dan kanan) akan dibutuhkan kombinatorik.

Selanjutnya, yang akan dicari adalah bagaimana cara untuk mendefinisikan permasalahan ini ke dalam bentuk rekursif. Permasalahan ini berada pada suatu struktur data pohon. Maka, sebelum memulai penyelesaian, akan ditentukan terlebih dahulu sebuah simpul yang akan dijadikan sebagai akar (*root*) dan juga notasi untuk melambangkan solusi dari sub masalah. Hal ini dilakukan agar masalah dapat dianalisis menjadi sub masalah yang berada dalam bentuk subpohon. Maka dari itu, akan didefinisikan sebuah notasi untuk melambangkan solusi dari sebuah sub masalah dengan menggunakan $dp(i)$ untuk subtree berakar di simpul i.

Pada pembahasan selanjutnya, simpul yang akan dijadikan sebagai akar adalah simpul bernomor 1.

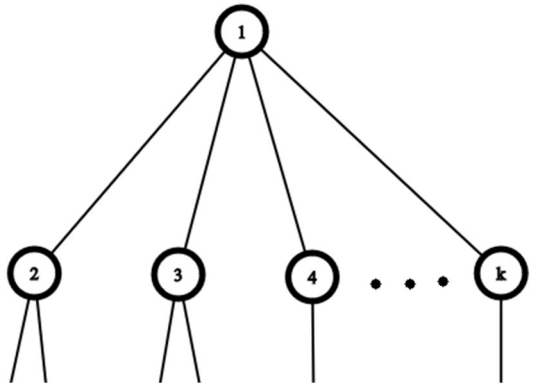


Gambar 6 Analisis rekursif kasus 1

Jika akar pohon memiliki derajat 1, maka dapat ditentukan dengan mudah hubungan rekursifnya terhadap anaknya. Pada kasus gambar 6, hubungan akar dengan simpul lain yang merupakan anaknya adalah

$$dp(1) = dp(2)$$

Tetapi hubungan rekursif akan memiliki kasus berbeda jika derajat akar bukanlah 1, sehingga hubungan ini tidak dapat digunakan secara general.



Gambar 7 Analisis rekursif kasus 2

Untuk menentukan hubungan rekursif pada kasus gambar 7, diperlukan informasi lain selain hasil dari solusi sub-sub pohon ($dp(i)$ dengan $(2 \leq i \leq k)$). Informasi lain yang dibutuhkan ini adalah banyaknya simpul yang ada pada sub pohon. Dengan adanya informasi ini, solusi dari masalah dapat dirancang.

Untuk mempermudah, akan didefinisikan notasi $size(i)$ untuk menyatakan ukuran(banyak simpul) dari subpohon yang berakar pada simpul i, notasi $child(i)$ untuk melambangkan himpunan anak dari simpul i, dan notasi $parent(i)$ untuk menyatakan orang tua dari simpul i.

Solusi untuk $dp(i)$ adalah merupakan hasil kali dari $dp(j)$ $(j \in child(i))$ dikali dengan banyak kemungkinan konfigurasi dari elemen-elemen yang ada pada setiap subpohon.

Untuk menghitung konfigurasi dari elemen-elemen subpohon, pertama akan dihitung semua kemungkinan pengurutannya yaitu dengan formula $(\sum_{j \in chil\ (i)} size(j))!$. Perhatikan bahwa untuk pohon dengan akar di i, berlaku persamaan $\sum_{j \in child(i)} size(j) = size(i) - 1$. Sehingga semua kemungkinan pengurutannya adalah $(size(i) - 1)!$. Setelah melakukan tahap tadi, perhatikan bahwa untuk simpul pada masing-masing subpohon pengurutannya seharusnya tidak perlu dihitung kembali karena banyak pengurutannya sudah dihitung pada perhitungan $dp(j); (j \in child(i))$. Sehingga kemungkinan tadi masih harus dibagi dengan $\prod_{j \in child(i)} size(j)!$. Dari situ didapat bahwa banyaknya konfigurasi dari elemen-elemen subpohon adalah seperti berikut

$$konfigurasi\ anak\ simpul\ i = \frac{(size(i) - 1)!}{\prod_{j \in child(i)} size(j)!}$$

Dari bahasan sebelumnya didapatkan hubungan rekursif berikut

$$dp(i) = \left(\prod_{j \in chil\ (i)} dp(j) \right) \times \left(\frac{(size(i) - 1)!}{\prod_{j \in child(i)} size(j)!} \right),$$

sehingga,

$$dp(i) = (size(i) - 1)! \prod_{j \in child(i)} \frac{dp(j)}{size(j)!}$$

dan untuk $akar = i$ akan berlaku

$$count(i) = dp(i).$$

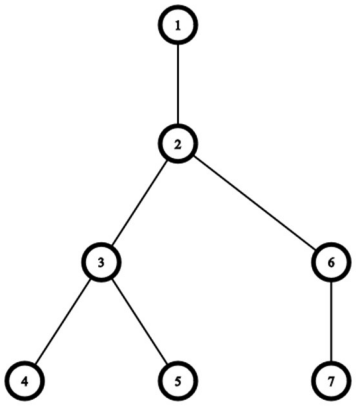
Jika $size(i) = 1$, maka berlaku

$$dp(i) = 1,$$

yang merupakan basis dari rekursi.

B. Solusi Permasalahan

Setelah pembahasan tadi, telah ditemukan cara untuk menghitung kemungkinan jika hanya terdapat satu kemungkinan sumber. Hal itu otomatis membuat kita mengetahui salah satu cara untuk menghitung kemungkinan keseluruhan yaitu dengan menjumlahkan semua kemungkinan pada sumber-sumber berbeda. Namun, jika hal itu dilakukan, akan banyak langkah berulang yang dilakukan.



Gambar 8 Analisis rekursif kasus 3

Jika perhitungan dilakukan dengan menghitung banyak kemungkinan pada sumber yang berbeda-beda, maka akan banyak sub masalah yang dihitung berulang kali. Pada gambar 8, jika sumber berada pada simpul 2, maka sub masalah $dp(3)$ akan dihitung yang padahal sub masalah $dp(3)$ juga akan dihitung jika sumber berada pada simpul 1. Hal inilah yang disebut dengan *overlap* dan membuat perhitungan dinilai sebagai hal yang boros.

Untuk mengatasi permasalahan ini, terdapat langkah-langkah yang harus dilakukan.

Pertama adalah hitung kemungkinan jika hanya terdapat satu kemungkinan sumber dan simpanlah solusi sub masalah/subpohon.

Kedua, kemungkinan dengan sumber yang berbeda dari akar dapat dicari dengan melakukan dfs pada pohon. Selagi melakukan dfs, solusi permasalah dapat dihitung dengan menerapkan formula berikut

$$count(i) = (n-1)! \cdot \left(\prod_{j \in chil\ (i)} \frac{dp(j)}{size(j)!} \right) \frac{count(parent(i)) \cdot size(i)! \cdot (n - size(i) - 1)!}{(n-1)! \cdot dp(i) \cdot (n - size(i))!}; (i \neq akar),$$

sehingga didapat formula,

$$count(i) = \left(\prod_{j \in child(i)} \frac{dp(j)}{size(j)!} \right) \frac{count(parent(i)) \cdot size(i)!}{dp(i) \cdot (n - size(i))}; (i \neq akar)$$

formula tersebut pada dasarnya adalah melakukan *reroot* atau mengubah akar untuk melakukan perhitungan sesuai dengan formula yang telah didapatkan pada pembahasan dengan satu sumber saja sebelumnya. Dengan menggunakan formula tersebut, solusi dari permasahan dapat dihitung dalam kompleksitas waktu linear $O(n)$.

C. Algoritma

Setelah mendapatkan hubungan rekursif dan formula penyelesaian, akan diimplementasikan algoritma tersebut pada bahasa pemrograman *C++*. Teknik untuk mengimplementasikan *dynamic programming* yang digunakan adalah *bottom-up*.

```
void solve_dp(int i, int parent){
    if (size[i] == 1){ // Basis
        dp[i] = 1;
        return;
    }
    dp[i] = factorial[size[i] - 1];
    // iterasi tetangga simpul i
    for (auto j : adjList[i]){
        if (j == parent) continue;
        solve_dp(j, i);
        dp[i] = (dp[i] * dp[j]) /
factorial[size[j]];
    }
}
```

Setelah itu itu, untuk mendapatkan kemungkinan total, harus dilakukan *reroot* terhadap pohon dengan melakukan dfs.

```
long long solve_count(int i, int parent){
    if (parent == -1) count[i] = dp[i]; // Akar
    else{
        _count[i] = (_count[parent] *
factorial[size[i]]) / ((n - size[i]) * dp[i]);
        // iterasi tetangga simpul i
        for (auto j : adjList[i]){
            if (j != parent) continue;
            _count[i] = (_count[i] * dp[j]) /
factorial[size[j]];
        }
    }
    long long result = _count[i];
    for (auto j : adjList[i]){
        if (j != parent) result += solve_count(j, i);
    }
    return result;
}
```

Pada implementasi untuk mengubah akar tree tersebut, penjumlahan hasil dilakukan secara simultan. Sehingga, pemanggilan fungsi $solve\_count(akar, -1)$ akan menghasilkan solusi dari perhitungan banyak kemungkinan penyebaran.

Dengan menggunakan implementasi tersebut, kompleksitas perhitungan banyak kemungkinan penyebaran dapat dihitung dalam kompleksitas waktu $O(n)$ dan dengan kompleksitas ruang $O(n)$.

Source Code at Github

https://github.com/mastree/CP_Journey/blob/master/STIMA

Video Link at Youtube

https://youtu.be/YgSaZVqXoKY

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] Cormen, T. H., Leiserson, C.E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. London: MIT Press.

[2] Munir, Rinaldi. 2009. Diktat Kuliah Strategi Algoritma. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung.

[3] Munir, Rinaldi. 2009. Matematika Diskrit. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung.

[4] Skiena S. S. (2012). The Algorithm Design Manual (Second ed.). London: Springer.

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, 2 Mei 2020

Muhammad Kamal Shafi - 13518113