Algoritma DFS untuk Penjadwalan Pembangunan Rumah Sederhana

Nisrina Yumna Khairunnisa (13517017)
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13517017@std.stei.itb.ac.id

***Abstract*—Penjadwalan dan rumah merupakan dua hal yang penting dalam hidup manusia. Penjadwalan membuat hidup manusia berjalan lebih efisien. Sementara rumah merupakan tempat yang semua manusia miliki, tidak harus fisik, namun secara bahasa rumah seharusnya berbentuk fisik. Dalam implementasi pembuatan rumah fisik, dibutuhkan suatu penjadwalan sehingga hasilnya sesuai. Pada kuliah IF2211 Strategi Algorima, salah satu strategi yang dipelajari adalah *Depth First Search.* Algoritma ini dapat melakukan penjadwalan sehingga algoritma sesuai untuk menyelesaikan masalah rumah dan penjadwalan yang telah disebutkan sebelumnya. Pada makalah ini, akan dijelaskan lebih lanjut bagaimana algoritma *Depth First Search* dapat menyelesaikan persoalan rumah dan penjadwalan**

Keywords—topological sort; Depth First Search; penjadwalan rumah;

I. PENDAHULUAN

Menurut KBBI, rumah merupakan bangunan untuk tempat tinggal. Namun sejatinya bagi beberapa orang rumah bukan hanya merupakan tempat tinggal, namun memiliki makna yang lebih dalam. Misalnya, bagi beberapa orang, rumah merupakan tempat berpulang, menciptakan ketenangan, menciptakan keamanan, ataupun menciptakan kebahagian dalam hati. Oleh karena itu, rumah seharusnya merupakan tempat yang paling nyaman dan aman bagi setiap orang. Nyaman dan aman bukan berarti rumah yang mewah, namun maksudnya memiliki arti yang besar bagi orang tersebut.

Untuk mendapatkan rumah yang sesuai dengan kriteria di atas, maka sepertinya akan lebih cocok jika kita membuat rumah sendiri. Maksud dari membuat rumah sendiri bukan berarti kita yang benar-benar membangun, tetapi maksudnya kita yang menentukan bentuk rumah kita seperti apa. Dalam proses pembuatan tersebut, akan dilewati fase pembangunan fisik rumah yang di dalamnya terdiri dari pembuatan dan penggabungan berbagai komponen rumah, seperti fondasi, tembok, jendela, pintu, dan masih banyak lagi.

Jika dilihat, banyak hal yang harus direncenakan untuk membangun sebuah rumah impian agar benar-benar seperti yang diinginkan. Hal itu menyebabkan proses pembangunan tersebut akan memakan waktu yang sangat banyak. Apalagi, ditambah jika pada pertengahan permbangunan ada hal yang ternyata tidak sesuai keinginan. Untuk mencegah hal-hal tersebut terjadi, maka diperlukan perencanaan pembangunan yang benar-benar matang. Perencanaan termasuk merencanakan bagian rumah apa yang akan dibangun terlebih dahulu agar waktu yang dibutuhkan untuk membangun rumah tersebut merupakan waktu yang paling efektif.

Pada penyusunan jadwal untuk pembangunan rumah, ada hal yang harus didahulukan untuk dibangun baru bisa melanjutkan membangun komponen rumah yang lainnya. Persolan ini dapat diselesaikan dengan pengurutan menggunakan *topological sorting.* Komponen rumah yang akan dibangun dapat dianggap sebagai sebuah simpul pada sebuah graf, sementara komponen yang harus dibangun sebelumnya (*prerequisite*) dapat digambarkan sebagai simpul sebelum komponen tersebut

Berdasarkan penjelasan-penjelasan di atas, maka untuk memperingan kerjaan dalam membangun rumah yang benar-benar sesuai kedinginan dapat dibantu dengan algoritma *Depth First Search*. Algoritma ini dapat digunakan karena algoritma ini dapat melakukan *topological sort* seperti yang dibuthkan untuk penjadwalan pembangunan. Dengan algoritma *Depth First Search,* maka akan diketahui urutan pembangunan yang efektif sehingga seharusnya pembangunnan akan diselesaikan dengan tepat waktu.

Gambar 1. Ilustrasi Langkah-langkah pembuatan rumah.
Sumber: http://everydaymom.info/steps-in-building-a-house/steps-in-building-a-house-steps-to-building-a-house-steps-for-building-a-house-contractor-steps-building-house-

steps-in-8-list-the-steps-involved-in-building-the-house-of-quality

II. Dasar Teori

A. Algoritma Penelusuran Graf

Graf merupakan struktur data yang linear, oleh karena itu dapat dilakukan penelusuran dengan relatif tidak sulit. Algoritma penelusuran graf merupakan algoritma yang mengunjungi semua simpul pada graf. Ada dua cara untuk melakukan penelusuran graf, yaitu:

- *Uninformed/Blind Search* (pencarian tanpa informasi)

  Algoritma yang termasuk kepada penelusuran jenis ini adalah *Depth First Seacrh, Breadth First Search, Depth Limited Search, Iterative Deepening Search,* dan *Uniform Cost Search.*

- *Informed Search* (pencarian dengan informasi)

  Algoritma yang termasuk kepada penelusuran jenis ini adalah *Best First Search* dan A*.

B. Algoritma Topological Sorting

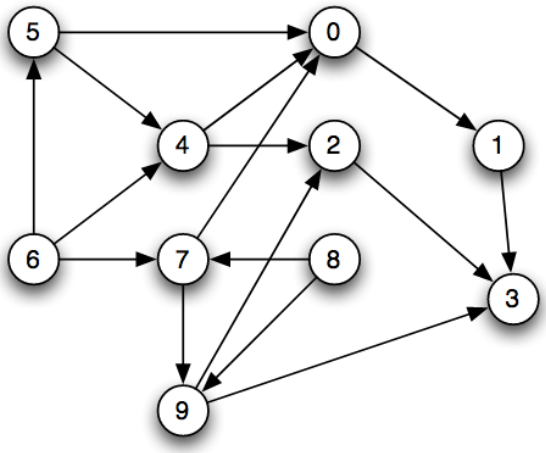
Topological sorting atau yang bisa juga disebut *topological ordering* merupakan algoritma yang mengurutkan suatu graf secara linear. Algorima ini melakukan penelusuran semua simpul yang pada ada graf sehingga terbentuk sebuah urutan yang aturannya ditentukan. Kasus algoritma ini digunakan yaitu ketika ada berbagai kejadian dan ada berbagai *constriaint* (*predence constraints*) sehingga "kejadian A harus diselesaikan sebelum kejadian B mulai diselesaikan". Tujuan dari *topological sorting* yaitu menemukan urutan kejadian-kejadian yang ada berdasarkan *dependency* yang ada atau menentukan apakah ada urutan kejadian yang mungkin untuk menyelesaikan masalah pengurutan yang ada.

Dalam bahasa formal, *topological sort* dapat didefiniskan sebagai berikut:

- Misal ada graf berarah $G(V,E)$ dengan simpul V merepresentasikan sebuah kejadian dan setiap sisi $(u,v) \in E$ yang artinya kejadian u harus diselesaikan sebelum v.
- Dicari urutan simpul 1, ..., $|V|$ sehingga untuk semua sisi (u,v), u ada sebelum v pada hasil pengurutan
- Hasil pengurutan tersebut disebut *topological sort*
- *Topological sort* pasti dapat menyelesaikan semua kejadian, jika grafnya merupakan DAG (*directed acyclic graph*)

Sebagai contoh ilustrasi, perhatikan Gambar 2 dan deskripsi berikut:

- Kejadian 0 harus selesai sebelum kejadian 1 is dimulai.
- Kejadian 1 dan kejadian 2 harus selesai sebelum kejadian 3 dimulai.
- Kejadian 4 harus selesai sebelum kejadian 0 atau kejadian 2 dimulai.
- Kejadian 5 harus selesai sebelum kejadian 0 atau kejadian 4 dimulai.
- Kejadian 6 harus selesai sebelum kejadian 4, kejadian 5 atau kejadian 7 dimulai.
- Kejadian 7 harus selesai sebelum kejadian 0 atau kejadian 9 dimulai.
- Kejadian 8 harus selesai sebelum kejadian 7 atau kejadian 9 dimulai.
- Kejadian 9 harus selesai sebelum kejadian 2 atau kejadian 3 dimulai.



Gambar 2. Graf Berarah Asiklik.
Sumber: https://www.inf.ed.ac.uk/teaching/courses/inf2b/algnotes/note10.pdf

Untuk graf tersebut jika dilakukan *topological sorting* maka hasilnya adalah 8, 6, 7, 9, 5, 4, 2, 0, 1, 3. Prioritas untuk memroses sebuah simpul terlebih dahulu yaitu yang memiliki sisi masuk paling sedikit. Jika suatu simpul sudah dirposes, maka simpul dihapus dari graf asal. Setelah penghapusan baru dilanjutkan *topological sorting* dengan graf yang baru hingga semua simpul telah diproses (graf kosong).

Berikut ini adalah *pseudocode* untuk melakukan *topological sorting* pada graf berarah yang asiklik:

```
function Topsort(G)
   T ← empty list
   Z ← empty queue
   in ← dictionary mapping all vertices to 0
   for each v ∈V do
      for each u adjacent to v do
         increment in[v]
   for each v ∈V do
      if in[v] = 0 then
         add v to Z
   while S is not empty do
      v ← Z.remove
      append v to T
      for each u adjacent to v do
         decrement in[u]
         if in[u] = 0 then
            add u to Z
   return T
```

Kompleksitas waktu dari algoritma ini secara keseluruhan yaitu $\Theta(1) + \Theta(V+E) + \Theta(V) + \Theta(V+E) = \Theta(V+E)$

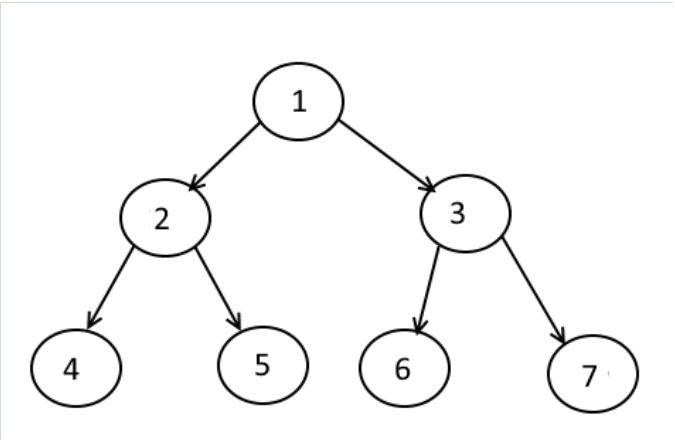
C. Algoritma *Depth First Search*

Pada abad ke-19, matematikawan dari Perancis yang bernama Charlex Pierre Tremaux menemukan sebuah algoritma penelusuran graf berarah yang bernama *Tremau Algorithm.* Kemudian algoritma tersebut merupakan algoritma yang efisien untuk mencari jalan keluar sebuah labirin dan pasti berhasil untuk semua labirin yang memiliki penyelesaian jalan keluar. Kemudian, setelah berratus-ratus tahun akhirnya algoritma tersebut dikenal sebagai *Depth First Search.*

Algoritma DFS, yang merupakan singkatan dari *Depth First Search,* merupakan salah satu algortima yang digunakan untuk melakukan traversal di dalam graf dan *topological sorting*. Ciri dari pencarian solusi dengan DFS yaitu melakukan ekspansi simpul-simpil terdalam lebih dahulu. Pertama-tama simpul akar dibangkitkan pertama kali, kemudian simpul pada level kedua, lalu simpul pada level ketiga, dan seterusnya. Sehingga, pencarian akan mengikuti sebuah lintasan tunggal yang semakin dalam dari akar. Ketika pencarian sampai pada sebuah simpul yang tidak memiliki tetangga, maka pencarian dilanjutkan ke lintasan lainnya.

Pada kasus terburuk, untuk menemukan sebuah solusi, DFS harus membangkitkan semua simpul pada pohon ruang status untuk menemukan solusi. Jika setiap simpul membangkitkan *b* buah simpul baru, dan batas maksimum kedalaman pohon ruang status adalah *m,* maka kompleksitas waktu algoritma DFS pada kasus terburuk adalah $O(b^m)$. Kompleksitas ruang algoritma DFS adalah O(*bm),* karena hanya perlu disimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangan.

Sebagai contoh implementasi dari algoritma DFS, perhatikan Gambar 3.



Gambar 3. Graf Berarah Asiklik.
Sumber: https://algorithms.tutorialhorizon.com/depth-first-searchtraversal-in-binary-tree/

Untuk gambar tersebut, maka hasil penelusuran akhirnya adalah 1, 2, 4, 5, 3, 6, 7. Untuk mendapatkan hasil tersebut, dilakukan langkah-langkah berikut:

1. Kunjungi simpul 1
2. 2 merupakan tetangga 1 dengan indeks terkecil, maka kunjungi 2
3. 4 merupakan tetangga 2 dengan indeks terkecil, maka kunjungi 4
4. Sudah *dead end* di 4 karena tidak memiliki sisi keluar sehingga dilakukan *backtracking* ke 2 kembali
5. 5 merupakan tetangga 2 yang tersisa dari yang belum dikunjungi, maka kunjungi 5
6. Sudah *dead end* di 5 karena tidak memiliki sisi keluar sehingga dilakukan *backtracking* ke 2 kembali
7. Semua tetangga 2 sudah dilewati sehingga *backtracking* ke 1
8. 3 merupakan tetangga 1 yang tersisa dari yang belum dikunjungi, maka kunjungi 3
9. 6merupakan tetangga 3 dengan indeks terkecil, maka kunjungi 6
10. Sudah *dead end* di 6 karena tidak memiliki sisi keluar sehingga dilakukan *backtracking* ke 3 kembali
11. 7 merupakan tetangga 3 yang tersisa dari yang belum dikunjungi, maka kunjungi 7
12. Sudah *dead end* di 7 karena tidak memiliki sisi keluar sehingga dilakukan *backtracking* ke 3 kembali
13. Semua tetangga 3 sudah dilewati sehingga *backtracking* ke 1
14. Semua tetangga 1 sudah dilewati dan tidak dapat dilakukan *backtracking*

Naturalnya algoritma DFS merupakan algoritma rekursif yang memanfaatkan algoritma *backtracking*. Algoritma *backtracking* dilakukan ketika sudah tidak mungkin untuk meng-ekspan simpun lebih mendalam. *Backtracking* dalam arti merupakan pencarian jalan hingga *dead end*. Namun walaupun DFS naturalnya bersifat rekursif, DFS juga dapat diimplementasikan dalam bentuk iteratif. Untuk DFS yang iteratif diimplementasikan menggunakan struktur data *stack* untuk menyimpan simpul. Berikut *pseudocode* DFS untuk baik yang iteratif maupun rekursif:

```
procedure DFS-iterative (G,s)

   let S be stack
   S.push( s      mark s as visited.
   while ( S is not empty)
      v  =  S.top( )
      S.pop( )
      for all neighbours w of v in Graph G
         if w is not visited
            S.push( w )
            mark w as visited
procedure DFS-recursive(G, s)
   mark s as visited
   for all neighbours w of s in Graph G
      if w is not visited
         DFS-recursive(G, w)
```

Jika diimplementasikan menggunakan adjacenty list, maka kompleksitas keseluruhan dari algoritma DFS ini adalah *O(V+E)*.

III. Penyelesaian Penjadwalan Pembangunan Rumah dengan Algoritma DFS

A. Langkah Penyelesaian

Pencarian solusi dilakukan dengan cara menerima input pengguna. Input pengguna akan berupa jumlah komponen yang dibutuhkan untuk membangun rumah yang diinginkan dan komponen-komponen rumah yang akan dibangun. Pengguna hanya dapat memasukkan komponen yang ada dalam pilihan, yaitu fondasi, tembok, atap, jendela, pintu, dekorasi, dan saluran air. Program akan mengetahui *prerequisite* untuk setiap komponen sehingga kemudian akan dibentuk graf berarah. Fondasi dapat dibangun tanpa sebelumnya membangun apapun, tembok dibangun hanya jika fondasi sudah dibangun, atap dibangun hanya jika tembok sudah dibangun, jendela dibangun hanya jika tembok sudah dibangun, pintu dibangun hanya jika tembok sudah dibangun, dekorasi dibangun hanya jika atap, jendela, atau pintu sudah dibangun, dan saluran air dibangun hanya jika tembok sudah dibangun. Bagian yang menunjuk merupakan *prerequisite* dari komponen yang ditunjuk.

Pilihan yang diberikan pengguna kemudian disimpan pada program daram bentuk sudah menjadi sebuah graf. Struktur data graf yang digunakan pada implementasi yaitu adjacency list karena berdasarkan dasar teori, kompleksitas algoritma DFS dengan ajdacency list merupakan implementasi yang paling efisien dibandingkan dengan struktur data yang lainnya.

Kemudian setelah tersimpan grafnya, maka dilakukan *topological sorting* dengan DFS. *Topological sorting* dengan DFS yang dilakukan sama seperti yang sudah dibahas pada dasar teori. Berikut potongan kodenya jika diimplementasikan dengan Java:

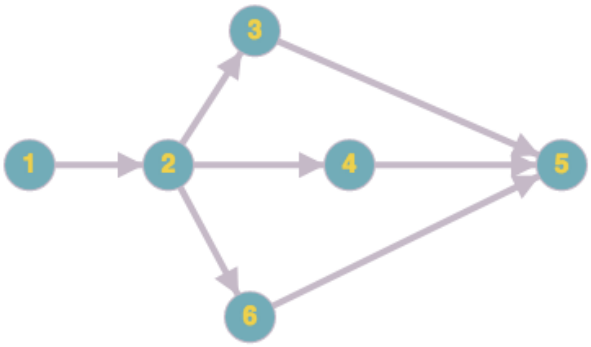
```
public void topSort(int s, int x){
   for (int i=0; i<s; i++){
      visited[i] = false;
   }
   if (!visited[x]){
      stack.add(x+1);
      dfs(x);
   }
}

public void dfs(int x){
   visited[x] = true;
   for (int i : G[x]){
      if (!visited[i]){
         stack.add(i+1);
         dfs(i);
      }
   }
}
```

Selain itu pada implementasi, struktur data yang digunakan yaitu:

```
private LinkedList<Integer>[] G;
private boolean[] visited;
private LinkedList<Integer> stack;
```

Agar implementasi program yang dibuat lebih jelas, maka dapat diilustrasikan sebagai berikut:

1. Pertama-tama pengguna memilih komponen apa yang ingin ada di rumahnya. Pada contoh di Gambar 4, dicontohkan pengguna memilih fondasi, tembok, atap, jendela, dekorasi, dan saluran air. Setelah itu untuk mempermudah pengolaan graf maka bukan nama dari komponen tersebut yang diproses namun komponen tersebut diberi nomor terurut. Misalnya untuk contoh komponen-komponen yang sudah disebutkan sebelemnya di paragraf ini fondasi dilambangkan 1, tembok dilambangkan 2, dan seterusnya hingga saluran air dilambangkan dengan angka 6 maka akan terbentuk sebuah graf seperti Gambar 4.



Gambar 4. Graf Berarah.
Sumber: penulis

Graf tersebut terbentuk berdasarkan program yang dibuat. Program otomatis mengetahui *prerequisite* komponen 1 (fondasi) atau komponen yang lainnya berdasarkan database yang dimiliki karena pemilihan komponen yang dapat digunakan juga terbatas. Sehingga akhirnya komponen itu ditentukan untuk ditempatkan pada posisi tersebut. Untuk komponen selanjutnya juga tidak beda jauh. Program otomatis mengenali bahwa jika ingin membuat atap, jendela,

atau saluran air harus dibuat tembok terlebih dahulu sehingga penempatanya seperti pada di Gambar 4.

2. Setelah ditentukan bentuk grafnya, maka graf akan disimpan ke program dalam bentuk adjacency list karena struktur data tersebut merupakan struktur data yang membuat algoritmanya memiliki kompleksitas waktu yang tidak begitu tinggi.
3. Selanjutnya jika graf sudah disimpan di dalam program maka dapat dilakukan penelusuran DFS sehingga terbentuk sebuah urutan *topological sort* yang merupakan urutan bagaimana urutan pembangunan yang direkomendasikan.

B. Penghitungan Kompleksitas

Kompleksitas waktu pemrosesan graf hingga program dapat mengeluarkan jawaban dapat dilihat dari algoritma yang dipakai. Pada masalah ini digunakan algoritma *topological* sorting dengan DFS tanpa modifikasi sehingga kompleksitas waktu yang dimiliki program ini sama seperti kompleksitas waktu untuk algoritma *topological* sorting dengan DFS. Kompleksitasnya yaitu $O(V+E)$ jika ada sejumlah V simpul dan sejumlah E sisi pada graf yang akan diproses.

C. Contoh Analisis

Ada beberapa kemungkinan kasus yang terjadi saat penjadwalan pembangunan rumah, 2 diantaranya yaitu kasus berhasil membuat jadwal dan tidak berhasil membuat jadwal. Kedua kasus tersebut akan diuji pada bagian ini, sebagai berikut:

- Kasus Uji 1

Kasus uji ini akan menguji untuk kasus yang berhasil melakukan penjadwalan pembangunan rumah.

Untuk kasus masukan dengan pilihan komponen yang diinginkan seperti pada contoh di bagian B yaitu fondasi, tembok, atap, jendela, dekorasi, dan saluran air dengan graf seperti Gambar 4 maka hasil eksekusinya akan berbentuk:

```
macs-MacBook-Air-8:stima mac$ java bangunRumah
Mau dari berapa barang?
6
Pilihannya: fondasi, tembok, atap, jendela, pintu, dekorasi, saluran air
fondasi
tembok
atap
jendela
dekorasi
saluran air
Urutan membangun yang baik:
fondasi
tembok
atap
dekorasi
jendela
saluran air
```

Gambar 5. Hasil Eksekusi Kasus Uji 1.

Sumber: penulis

Ketika membangun rumah, fondasi harus selalu didahulukan daripada komponen lain. Setelah fondasi dilanjutkan tembok. Ini juga sesuai yang diinginkan karena membangun tembok baru bisa dilakukan setelah membangun fondasi. Selanjutnya yaitu atap. Atap juga sesuai karena atap baru dapat dibangun ketika sudah ada tembok, tidak mungkin atap dibangun langsung setelah fondasi tanpa ada tembok. Lalu dilanjutkan dekorasi. Jika rumah sudah ada tembok dan atap otomatis rumah tersebut sudah layak untuk diisi dan didekorasi sehingga langkah ini juga tidak menyalahi aturan. Setelah itu dibangun jendel. Hal ini sesuai karena pembangun jendela hanya dapat dilakukan jika tembok sudah dibangun. Lalu yang terakhir yaitu saluran air. Langkah pembangunan saluran air sesuai karena rumahnya sudah "berbentuk".

Sesuai dengan algoritma *Depth First Search*, algoritma pada program yang dibuat menelusuri graf ke dalam terlebih dahulu. Ketika suatu graf memiliki sisi lebih dari satu maka akan dikunjungi dari yang memiliki indeks terkecil terlebih dahulu. Pada kasus ini, atap diproses lebih dahulu daripada jendela dan saluran air karena atap memiliki indeks terkecil yaitu 3. Selanjutnya setelah 3 dirposes maka melanjutkan ke kedalaman yang lebih dalam. Yaitu dekorasi. Dekorasi merupakan sisi yang bersebelahan dengan 3 dan belum dikunjungi maka dia benar semakin dalam. Selanjutnya diproses jendela karena dekorasi merupakan *dead end* yang mengakibatkan adanya *backtracking*. Di jendela juga menemukan *dead end* sehingga dilanjutkan ke saluran air.

- Kasus Uji 2

Kasus uji ini akan menguji untuk kasus yang tidak berhasil melakukan penjadwalan pembangunan rumah karena tidak ditemukan urutan yang tepat sehingga tidak semua komponen terbangun pada akhirnya.

Untuk kasus uji 2 yang kedua dipilih komponen yang tidak berbungan sehingga tidak dapat dilakukan penjadwalan, yaitu fondasi dan pintu.

Gambar 6. Graf Kasus Uji 2.

Sumber: penulis

```
macs-MacBook-Air-8:stima mac$ java bangunRumah
Mau dari berapa barang?
2
Pilihannya: fondasi, tembok, atap, jendela, pintu, dekorasi, saluran air
fondasi
pintu
Urutan membangun yang baik:
Tidak ditemukan penjadwalan yang cocok
```

Gambar 7. Hasil Eksekusi Kasus Uji 2.

Sumber: penulis

Ketika membangun rumah, fondasi memang dibangun duluan tetapi pintu tidak dapat dibangun langsung setelah fondasi. Pintu membutuhkan tembok agar dapat dibangun pada rumah.

Sesuai dengan algoritma *Depth First Search*, algoritma pada program yang dibuat menelusuri graf ke dalam terlebih dahulu. Namun karena kedua simpul tidak berhubungan maka yang didantangi hanya simpul fondasi atau 1. Setelah itu program akan berhenti. Lalu program dapat mengeluarkan pesan kesalahakan karena tidak semua simpul didatangi sementara untuk penjadwalan pembangunan rumah ini seharusnya semua simpul didatangi.

IV. Kesimpulan

Dapat disimpulkan bahwa da berbagai macam algoritma yang dapat digunakan untuk menyelesaikan suatu masalah. Namun, untuk masalah penjadawalan pembangunan rumah ini algoritma terbaik yang digunakan yaitu algoritma *Depth First Search*. Pertama ditentukan syarat-syarat komponen rumah dibangun. Kemudian dari situ dapat dibuat graf berarahnya untuk menentukan apa saja yang harus dibangung beserta *prerequisite*-nya. Hasil graf tersebut kemudia ditelusuri dengan *Depth First Search* sehingga terbentuk suatu urutan yang disebut *topological sort*. Hasil urutan tersebut merupakan urutan yang sebaiknya dilakukan untuk pembangunan rumah sehingga lebih tepat sesuai dengan keinginan. Hal tersebut telah berhasil diimplementasikan dalam bentuk sebuah program kecil sangat sederhana yang dijelaskan pada bagian-bagian sebeleum ini. Selain itu, dapat disimpulkan juga bahwa algoritma *Depth First Search* ternyata dekat dengan kehidupan sehari-hari.

V. Ucapan Terima Kasih

Pertama-tama penulis bersyukur atas nikmat Allah SWT. karena dengan nikmat dan kehendak-Nya penulis dimudahkan dalam menyelesaikan makalah ini dengan baik dan tepat waktu. Penulis juga mengucapkan terima kasih kepada orang tua, sahabat, dan teman-teman penulis yang senantiasa memberi dukungan, motivasi, dan kasih sayang selama pengerjaan makalah ini. Penulis turut mengucapkan terima kasih kepada dosen Strategi Algoritma K2, Ibu Masayu Leylia Khodra, yang telah memberikan pelajaran tentang *Depth First Search* di mata kuliah Strategi Algoritma pada semester 4.

References

[1] A. Puntambekar, Analysis and Design of Algorithms. Mumbai: Technical Publications Pune : 2008.

[2] A. Puntambekar, Data Structure and Files. Mumbai: Technical Publications Pune : 2008.

[3] https://indonesia.hackerearth.com/konsep-depth-first-search-dfs/

[4] http://www.cs.utoronto.ca/~tabrown/csc263/2014W/week9.pdf

[5] https://courses.cs.washington.edu/courses/cse326/03wi/lectures/RaoLect20.pdf

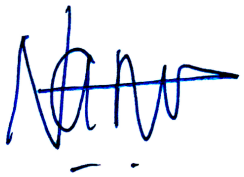
[6] https://www.inf.ed.ac.uk/teaching/courses/inf2b/algnotes/note10.pdf

[7]

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, 25 April 2019



Nisrina Yumna Khairunnisa 13517017