**Pengaplikasian Algoritma BFS dan DFS dalam Fitur *People You May Know* Jejaring Sosial Facebook**

**LAPORAN TUGAS BESAR**

Diajukan Untuk Memenuhi Tugas IF 2211 Strategi Algoritma

Semester II 2020/2021

A person standing in front of a crowd

Description automatically generated

Disusun oleh

**Reihan Andhika Putra (13519043)**

**Kadek Dwi Bagus Ananta Udayana (13519057)**

**RyoRichardo (13519193)**

**TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

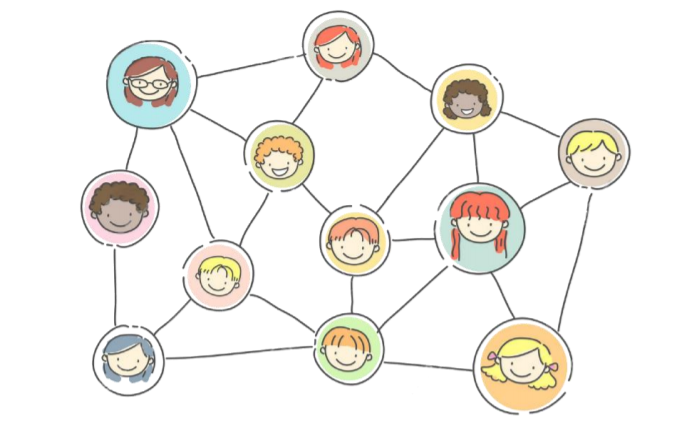
**2021**

**BAB I**

**DESKRIPSI TUGAS**

* 1. **Latar Belakang**

Bermain media sosial merupakan aktivitas yang sangat menyenangkan. Media sosial adalah sebuah media daring, dengan para penggunanya bisa dengan mudah berpartisipasi, berbagi, dan menciptakan isi blog, jejaring sosial, wiki, forum dan dunia virtual. Dari berbagai jenis media sosial, jejaring sosial merupakan jenis yang paling diminati oleh orang banyak. Popularitas jejaring sosial seperti facebook, twitter, Instagram, dan lainnya semakin meningkat dalam beberapa tahun terakhir. Pengguna media sosial pun berasal dari berbagai kalangan, mulai dari anak SD hingga orang tua berusia lanjut. Salah satu manfaat dari jejaring sosial adalah pengguna dapat berkomunikasi dengan orang-orang dari berbagai belahan dunia, baik orang yang sudah dikenal, maupun orang yang belum pernah dikenal sebelumnya.



Gambar 1. Ilustrasi graf pertemanan pada social network Facebook

(Sumber: <https://www.freecodecamp.org/news/deep-dive-into-graph-traversals-227a90c6a261/>)

Facebook sebagai salah satu pelopor jejaring sosial sejak tahun 2004 kini telah menembus angka 2,6 miliar pengguna. Fitur friend recommendation menjadi sangat penting karena banyaknya jumlah pengguna Facebook. Fitur bernama "People You May Know" ini dapat memberikan pengguna rekomendasi teman yang sebaiknya di-add, misalnya teman sekolah, teman kuliah, mantan pacar, atau orang yang kita kenal lewat suatu kegiatan tertentu.

Faktor utama saran pertemanan melalui fitur tersebut adalah berdasarkan mutual friend yang dimiliki oleh kedua akun pengguna. Misalnya pengguna A dan C belum berteman di facebook, tetapi keduanya berteman dengan pengguna B, berarti A dan C memiliki mutual friend yang sama, yaitu B. Semakin banyak mutual friend yang dimiliki antar kedua akun, maka semakin tinggi rekomendasi akun tersebut untuk di-add.

Selain itu, di setiap social media, termasuk Facebook, pengguna dapat mengeksplorasi akunakun pengguna lainnya yang tidak memiliki mutual friends sama sekali. Akan tetapi, dengan menelusuri graf pertemanan antar akun sehingga kita dapat mengetahui 'jarak' antar akun agar bisa saling terhubung dan berteman.

* 1. **Deskripsi tugas:**

Dalam tugas besar ini, Kamu akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan beberapa fitur dari People You May Know dalam jejaring sosial media (Social Network). Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Kamu dapat menelusuri social network pada akun facebook untuk mendapatkan rekomendasi teman seperti pada fitur People You May Know. Selain untuk mendapatkan rekomendasi teman, Kamu juga diminta untuk mengembangkan fitur lain agar dua akun yang belum berteman dan tidak memiliki mutual friends sama sekali bisa berkenalan melalui jalur tertentu.

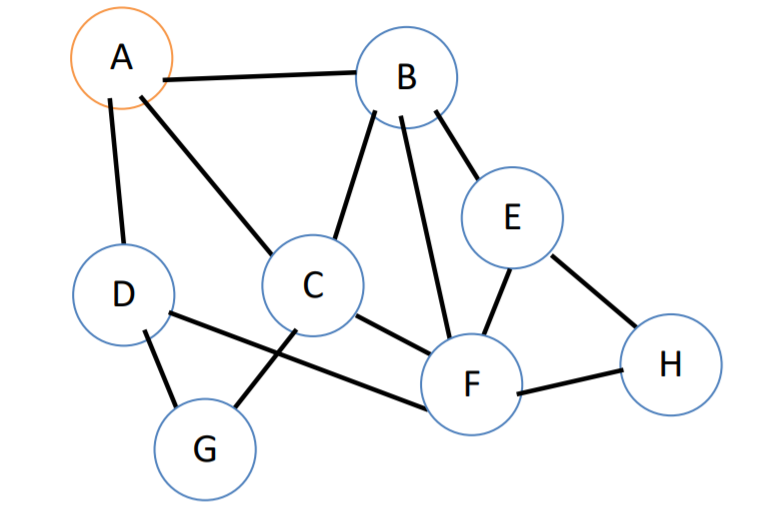
**Contoh Input dan Output Program**

Contoh berkas file eksternal:

|  |
| --- |
| 13  A B  A C  A D  B C  B E  B F  C F  C G  D G  D F  E H  E F  F H |

Gambar 2. Contoh input berkas file eksternal

Visualisasi graf pertemanan yang dihasilkan dari file eksternal:



Gambar 3. Contoh visualisasi graf pertemanan dari file eksternal

Untuk **fitur friend recommendation**, misalnya pengguna ingin mengetahui daftar rekomendasi teman untuk akun A. Maka output yang diharapkan sebagai berikut

|  |
| --- |
| Daftar rekomendasi teman untuk akun A:  Nama akun: F  3 mutual friends:  B  C  D  Nama akun: G  2 mutual friends:  C  D  Nama akun: E  1 mutual friend:  B |

Gambar 4. Hasil output yang diharapkan untuk rekomendasi akun A

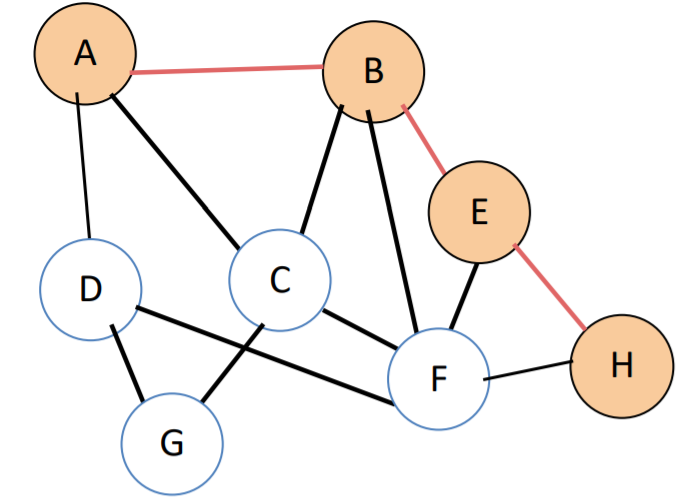
Untuk **fitur explore friends**, misalnya pengguna ingin mengetahui seberapa jauh jarak antara akun A dan H serta bagaimana jalur agar kedua akun bisa terhubung. Berikut output graf dengan penelusuran BFS yang dihasilkan.

Berikut output yang diharapkan untuk penelusuran menggunakan BFS.

|  |
| --- |
| Nama akun: A dan H  2nd-degree connection  A → B → E → H |

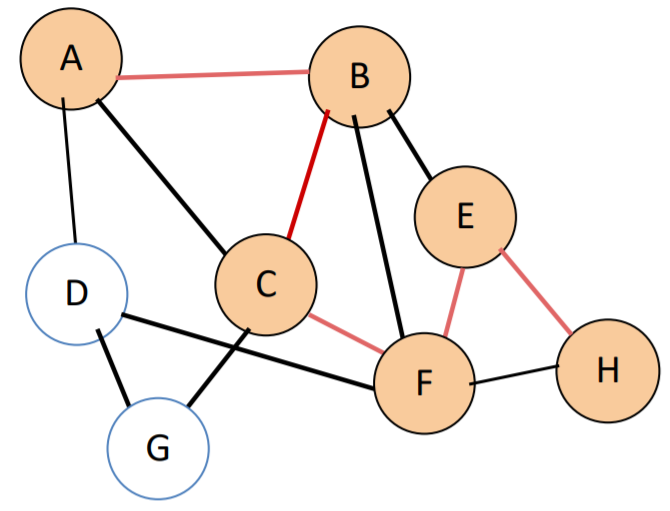
Gambar 5. Hasil output akun Nth degree connection akun A dan H menggunakan BFS

Perhatikan busur antara akun A dan H, terbentuk **salah satu jalur koneksi** sebagai berikut: A-BE-H (ada beberapa jalur lainnya, seperti A-D-F-H, dll, urutan simpul untuk ekspan **diprioritaskan berdasarkan abjad**). Akun A dan H tidak memiliki mutual friend, tetapi kedua akun merupakan 2nd-degree connection karena di antara A dan H ada akun B dan E yang saling berteman. Sehingga akun H dapat terhubung sebagai teman dengan jalur melalui akun B dan akun E. Jalur koneksi dari A ke H menggunakan BFS digambarkan dalam bentuk graf sebagai berikut.



Gambar 6. Hasil visualisasi jalur koneksi menggunakan BFS

Sedangkan untuk penggunaan algoritma DFS, diperoleh jalur lainnya, yaitu A-B-C-F-E-H yang digambarkan dalam bentuk graf sebagai berikut



Gambar 7. Hasil visualisasi jalur koneksi menggunakan DFS

Pada fitur explore friends, apabila terdapat dua buah akun yang tidak bisa saling terhubung (tidak ada jalur koneksi), maka akan ditampilkan bahwa akun tersebut tidak bisa terhubung melalui jalur koneksi yang sudah dimilikinya sekarang sehingga orang tersebut memang benar-benar harus memulai koneksi baru dengan orang tersebut.

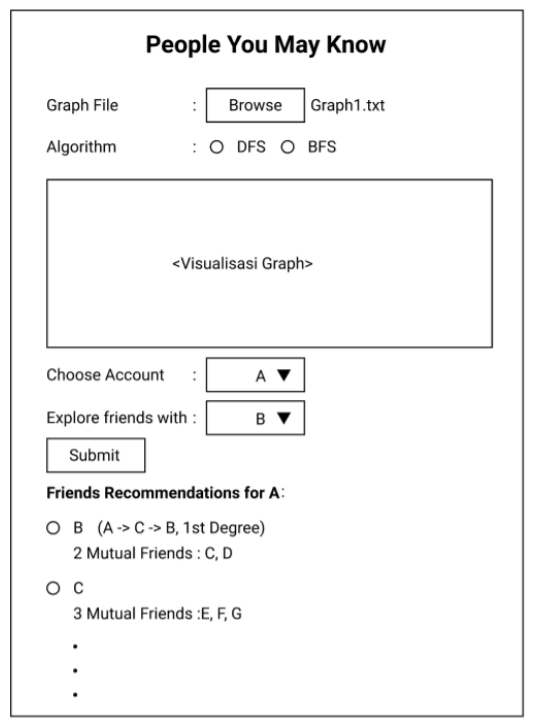
Misalnya terdapat dua orang baru, yaitu J dan I yang hanya terhubung antara J-I. Maka jalur koneksi yang dibentuk dari A ke J adalah.

|  |
| --- |
| Nama akun: A dan J  Tidak ada jalur koneksi yang tersedia  Kamu harus memulai koneksi baru itu sendiri. |

Gambar 8. Hasil output tidak ada jalur koneksi antara A dan J

* 1. **Spesifikasi Program**

Aplikasi yang akan dibangun dibuat berbasis GUI. Berikut ini adalah contoh tampilan dari aplikasi GUI yang akan dibangun.



Gambar 9. Tampilan layout dari aplikasi desktop yang dibangun

Spesifikasi GUI:

1. Program dapat menerima input berkas file eksternal dan menampilkan visualisasi graph.

2. Program dapat memilih algoritma yang digunakan.

3. Program dapat memilih akun pertama dan menampilkan friends recommendation untuk

akun tersebut.

4. Program dapat memilih akun kedua dan menampilkan jalur koneksi kedua akun dalam

bentuk visualisasi graf dan teks bertuliskan jalur koneksi kedua akun.

5. GUI dapat dibuat **sekreatif** mungkin asalkan memuat 4 spesifikasi di atas.

Program yang dibuat harus memenuhi **spesifikasi wajib** sebagai berikut:

1. Buatlah program dalam bahasa **C#** untuk melakukan penelusuran social network facebook sehingga diperoleh daftar rekomendasi teman yang sebaiknya di-add. Penelusuran harus memanfaatkan algoritma **BFS dan DFS**.
2. Awalnya program menerima sebuah berkas file eksternal yang berisi informasi pertemanan di facebook. Baris pertama merupakan sebuah integer N yang adalah banyaknya pertemanan antar akun di facebook. Sebanyak N baris berikutnya berisi dua buah string IF2211 Strategi Algoritma - Tugas Besar 2 7 (A, B) yang menunjukkan akun A dan B sudah berteman (lebih jelasnya akan diberikan contoh pada bagian 3).
3. Program kemudian dapat menampilkan **visualisasi graf pertemanan** berdasarkan informasi dari file eksternal tersebut. Graf pertemanan ini merupakan **graf tidak berarah** dan **tidak berbobot**. Setiap akun facebook direpresentasikan sebagai sebuah node atau simpul pada graf. Jika dua akun berteman, maka kedua simpul pada graf akan dihubungkan dengan sebuah busur.

Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk melakukan visualisasi adalah **MSAGL** (<https://github.com/microsoft/automatic-graph-layout>) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada: <https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1tPL30LA/edit?usp=sharing>

1. Terdapat dua fitur utama, yaitu:
2. Fitur friend recommendation
3. Program menerima sebuah pilihan akun dari user yang hendak dicari rekomendasi temannya. Pemilihan nama akun akan diterima melalui **GUI**. Cara pemilihan dibebaskan, bisa input dari keyboard atau meng-klik langsung sebuah node dari graf.
4. Program akan menampilkan daftar rekomendasi teman seperti pada fitur People You May Know facebook berupa nama akun tersebut secara **terurut** mulai dari **mutual friend terbanyak** antar kedua akun beserta daftar nama akun mutual friend.
5. Fitur explore friends
6. Dua akun yang tidak memiliki mutual friend, masih memiliki peluang untuk berteman jika kedua akun mempunyai **common Nth degree connection**, yaitu **jalur** yang menghubungkan kedua akun yang terpisah sejauh N akun (node pada graf).
7. Program menerima pilihan dua akun yang belum berteman.
8. Program akan menampilkan nilai N-th degree connection antar kedua akun dan memberikan jalur melalui akun mana saja sampai kedua aku bisa terhubung.
9. Dari graph yang sudah dibentuk, aplikasi harus dapat menyusun jalur koneksi hasil explore friends antara akun satu dengan akun yang ingin dituju Aplikasi juga harus dapat menunjukkan langkah-langkah pencarian, baik dengan algoritma BFS maupun DFS.
10. Jika tidak ditemukan jalur koneksi sama sekali antar kedua akun karena graf not fully connected, maka tampilkan informasi bahwa kedua akun tidak dapat terhubung.
11. Mahasiswa **tidak diperkenankan** untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS.

**BAB II**

**LANDASAN TEORI**

1. **. Graf dan Graf Traversal**

Sebuah graf adalah pasangan himpunan V dan E, dimana V adalah himpunan *vertex* / simpul dari graf tersebut, dan E sebagai himpunan *edge* / sisi dari simpul pada V yang menyatakan keterhubungan dari dua simpul di V.

Berdasarkan himpunan sisinya, sebuah graf dapat dikategorikan menjadi dua tipe, yaitu:

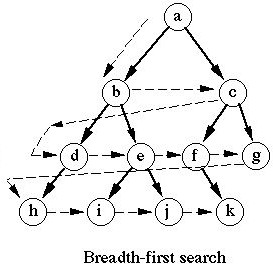
* 1. Graf berarah, yaitu sebuah graf dengan sisi yang memiliki arah. Artinya, jika sebuah sisi *e* menghubungkan simpul *a* dan *b*, maka dapat dikatakan ada hubungan antara *a* ke *b*, tapi belum tentu ada hubungan dari *b* ke *a*.
  2. Graf tak berarah, yaitu kebalikan dari graf berarah. Artinya, untuk setiap sisi pada himpunan E, jika ia menghubungkan simpul *a* dan *b*, maka diimplikasikan bahwa *b* ke *a* juga terhubung.

Pada graf tak berarah, jika *a* dan *b* terhubung, artinya mereka saling bertetangga.Traversal Graf adalah pengunjungan tiap vertex / node / simpul yang dapat dikunjungi dari graf G(V, E) dimana V adalah himpunan vertex dan E adalah himpunan edge dari graf G. Algoritma graf traversal digunakan untuk melakukan pencarian simpul yang tepat.

1. **. BFS (*Breadth First Search*)**

Breadth First Search adalah algoritma yang melakukan pengunjungan *node* pada suatu

graf G dengan graf Traversal secara “melebar”. Artinya, pencarian dilakukan dengan berdasarkan kedalaman pencarian *node*.



Gambar 10. Ilustrasi Breadth First Search pada sebuah graf.

Dengan Breadth First Search, pencarian dilakukan dengan menggunakan sebuah struktur

data *queue*, yaitu dengan teknik FIFO (First In First Out). Selain itu, penyimpanan dari tipa sisi yang bertetangga disimpan pada sebuah larik dua dimensi. Pada makalah ini, hanya akan dibahas graf yang memiliki sisi yang tidak berarah (*undirected edge*). Dengan data penyimpan simpul yang sedang dikunjungi berupa *queue*, misal *q*, langkah pencarian dilakukan dengan *step-step* berikut:

* 1. Kunjungi simpul pertama, misal *v*. Masukkan *v* ke antrian *q*.
  2. Pop antrian *q*. Misal hasil *pop* dari *q* adalah *v’*, maka kunjungi semua simpul yang *adjacent* / bertetangga dengan simpul *v’* dan belum pernah dikunjungi sebelumnya dengan memasukkan semuanya ke antrian *q*.
  3. Jika antrian *q* habis, artinya pencarian tidak ditemukan. Jika belum habis, lanjutkan ke langkah 4.
  4. Jika simpul ditemukan, hentikan pencarian. Jika belum ditemukan, kembali ke langkah 2.

Pada Gambar 1, jika kita mencari simpul *j*, maka dengan Breadth First Search, urutan

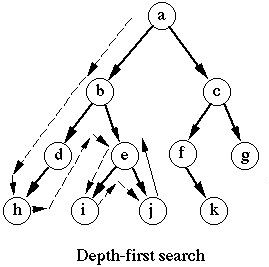
pencariannya adalah *a, b, c, d, e, f, g, h, i, j.* Misalnya b adalah banyak simpul dan d adalah kedalaman traversal, Breadth First Search memiliki beberapa properti, yaitu:

* Completeness? Ya (selama banyaknya simpul terbatas )
* Optimality? Ya, jika langkah proporsional dengan biaya. Dalam kasus pencarian *peer*, tidak.
* Kompleksitas waktu: O(bd)
* Kompleksitas ruang: O(bd)

1. **. DFS (Depth First Search)**

Depth First Search juga merupakan algoritma graf Traversal. Bedanya dengan Breadth

First Search terletak pada runtutan langkah yang diambil ketika mengambil keputusan untuk mengunjungi sebuah simpul. Seperti namanya, Depth First Search mencari simpul dengan cara mengunjungi simpul sedalam mungkin. Ketika sudah kehabisan arah, ia baru berbalik dan mengunjungi simpul sebelumnya, dan mencari simpul lain yang belum dikunjungi. Kemampuan ini disebut juga *backtracking*.



Gambar 11. Ilustrasi Depth First Search pada sebuah graf.

Sama seperti bagian Breadth First Search, algoritma Depth First Search pada makalah ini akan menyelesaikan pencarian pada graf tak berarah. Untuk Depth First Search, struktur data yang digunakan untuk menyimpan simpul adalah *stack*/ Selain itu, penyimpanan dari tipa sisi yang bertetangga disimpan pada sebuah larik dua dimensi. Langkah pencarian dilakukan dengan *step-step* berikut:

* 1. Kunjungi simpul pertama, misal *v*. Masukkan *v* ke tumpukan / *stack s*.
  2. Cek nilai *top* dari *s*. Misal *top* dari *s* adalah *v’*. Pilih satu simpul saja dari semua simpul yang bertetangga dengan *v’* dan belum dikunjungi. Pemilihan simpul tidak berdasarkan apapun, namun sebaiknya heuristik pemilihan simpul pada setiap langkah konsisten. Misal hasil pemilihan simpul yang bertetangga dengan *v’* adalah *p*, *push p* ke *s*.
  3. Jika *top* dari s, yaitu simpul *p*, adalah simpul yang dicari, maka hentikan pencarian. Jika tidak, lanjutkan ke langkah 4.
  4. Jika *top* dari *s*, yaitu simpul *p,* masih mempunyai tetangga, ulangi lagi langkah 2.
  5. Jika *top* dari *s*, yaitu simpul *p,* tidak mempunyai tetangga, maka *pop s* dan ulangi langkah 4. Langkah inilah yang disebut *backtracking*, yaitu kembali sebanyak 1 level secara kedalaman ke simpul sebelumnya.
  6. JIka *stack* sudah habis, maka simpul tidak ditemukan. Hentikan pencarian.

Misalnya b adalah banyak simpul dan m adalah kedalaman terjauh, maka Depth First

Search memiliki beberapa properti, yaitu:

* Completeness? Ya (selama banyaknya simpul terbatas )
* Optimality?Ya, jika langkah proporsional dengan biaya. Dalam kasus pencarian *peer*, tidak.
* Kompleksitas waktu: O(bm)
* Kompleksitas ruang: O(bm)

1. **. C# Pengembangan Aplikasi Berbasis Dekstop**

*Desktop application* atau aplikasi desktop adalah suatu aplikasi yang dapat berjalan

sendiri atau independen tanpa menggunakan *browser* atau koneksi internet disuatu komputer otonom. Aplikasi berbasis *desktop* merupakan aplikasi yang dijalankan pada masing-masing komputer atau klien. Aplikasi berbasis *desktop* harus diinstall terlebih dahulu ke dalam komputer agar dapat digunakan. Berdasarkan pengertian diatas penulis menyimpulkan bahwa aplikasi *desktop* adalah aplikasi yang berjalan pada komputer yang dapat digunakan secara langsung ketika kode program selesai dikompilasi. Bahasa c# menyediakan opsi untuk melakukan pengembangan aplikasi berbasis desktop. Dengan menggunakan visual studio sebagai IDE dan menggunakan .NET sebagai *framework*, kita dapat membuat aplikasi berbasis desktop dengan fitur fitur yang lengkap.

**BAB III**

**ANALISIS PEMECAHAN MASALAH**

**3.1. Langkah-Langkah Pemecahan Masalah**

Langkah-langkah untuk mencari solusi permasalahan ini adalah sebagai berikut:

1. Memahami permasalahan yang diberikan, yaitu realisasi fitur Explore Friend dan Get Friend Recommendation dengan algoritma BFS dan DFS.
2. Memahami konsep Graf, BFS, DFS, dan implementasinya dalam bahasa C#.
3. Melakukan mapping permasalahan menjadi elemen-elemen dalam algoritma BFS dan DFS.
4. Merancang dan mengimplementasi struktur data yang mendukung.
5. Merancang dan mengimplementasi algoritma BFS dan DFS dalam realisasi fitur Explore Friend dan Get Friend Recommendtaion.
6. Merancang dan mengimplementasi GUI yang dapat menerima input dan memberikan output kepada user.
7. Menguji dengan beberapa kemungkinan kasus yang terjadi ketika menggunakan fitur Explore Friend dan Get Friend Recommendation

**3.2. Mapping Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS**

1. Persoalan Explore Friend
2. Problem state: Mencari orang yang ingin di-explore
3. Initial state: Simpul orang yang melakukan Explore Friend
4. Solution state: Simpul orang yang ingin di-explore
5. State space: Seluruh simpul yang terbentuk saat pencarian dengan BFS/DFS.
6. State space tree: Pohon yang tersusun dari simpul-simpul pada State space
7. Solution space: Himpunan seluruh simpul pada Solution state
8. Solution: Path yang terbentuk dari Initial state ke Solution state
9. Persoalan Get Friend Recommendation
10. Problem state: Mencari simpul orang yang memiliki mutual dengan simpul awal
11. Initial state: Simpul orang yang melakukan Get Friend Recommendation
12. Solution state: Simpul orang yang memiliki mutual dengan Initial state
13. State space: Seluruh simpul yang terbentuk saat pencarian dengan BFS/DFS.
14. State space tree: Pohon yang tersusun dari simpul-simpul pada State space
15. Solution space: Himpunan seluruh simpul pada Solution state
16. Solution: Path yang terbentuk dari Initial state ke Solution state

**3.3. Ilustrasi Kasus**

Dalam tugas besar ini, Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Kami dapat menelusuri social network untuk mendapatkan rekomendasi teman. Selain untuk mendapatkan rekomendasi teman, kami juga diminta untuk mengembangkan fitur lain agar dua akun yang belum berteman dan tidak memiliki mutual friends sama sekali bisa berkenalan melalui jalur tertentu.

Contoh Inputan

13

A B B E D F BF

A C C F E H

A D C G E F

B C D G F H

Untuk fitur friend recommendation, misalnya pengguna ingin mengetahui daftar rekomendasi teman untuk akun B. Maka output yang diharapkan sebagai berikut

Daftar rekomendasi teman untuk akun A:

Nama akun: H

2mutual friends: EF

Nama akun: D

2 mutual friends: A F

Nama akun: G

1 mutual friend: C

Untuk fitur explore friends, misalnya pengguna ingin mengetahui seberapa jauh jarak antara akun B dan D serta bagaimana jalur agar kedua akun bisa terhubung. Berikut output dengan penelusuran BFS yang dihasilkan.

Nama akun: B dan D

2nd-degree connection B → A → D

Sedangkan untuk penggunaan algoritma DFS, diperoleh juga jalur yang sama yaitu B → A → D

Dalam kedua algoritma ini, urutan abjad akan diprioritaskan.

**BAB IV**

**IMPLEMENTASI DAN PENGUJIAN**

1. **Implementasi Program**

Notasi Algoritmik BFS Explore Friend

|  |
| --- |
| **function** *exploreFriend*(G: **Graph**, person1, person2: **Node)** → **List<string>, bool**  **{I.S.:** G, person1, person2 terdefinisi  **F.S.:** memberikan output berupa List nama Node-Node jalur koneksi person1  ke person2 dan Bool true jika ada koneksi, false jika tidak.**}**  **Kamus:**  current: **Element**  next: **Element**  visited: **List<string>**  queue: **Queue<Element>**  friend: **string**  fnode: **Node**  **Algoritma:**  current ← person1  Enqueue current ke dalam queue  **while** (nama current ≠ nama person2 **and** jumlah element queue > 0) **do**  tambahkan nama current ke visited  current ← Dequeue element teratas queue  **for** (friend as semua friends dari current)  fnode ← Node pada G dengan nama friend  next ← fnode  hapus semua friends dari next yang berada pada visited  **for** (friend as semua connection dari current)  tambahkan friend ke connection dari next  tambahkan nama current ke connection dari next  Enqueue next ke dalam queue  **if** (jumlah elemen queue > 0) **then**  current ← Peek element teratas queue  tambahkan nama current ke connection dari current  **if** (nama current = nama person2) **then**  → connection, true  **else**  → connection, false |

Notasi Algoritmik DFS Explore Friend

|  |
| --- |
| **function** *exploreFriend*(G: **Graph**, person1, person2: **Node,** found: **Bool)** → **List<string>,** found **Bool**  **{I.S.:** G, person1, person2 terdefinisi  **F.S.:** memberikan output berupa List nama Node-Node jalur koneksi person1  ke person2 dan found = true jika ada koneksi, false jika tidak.**}**  **Kamus:**  current, second\_el: **Element**  visited, connection: **List<string>**  Stack Stack**<Element>**  friend: **string**  fnode: **Node**  **Algoritma:**  current ← person1  found ← false  push current ke dalam stack  **while** (nama current ≠ nama person2 **and** jumlah elemen stack > 0) **do**  tambah nama current ke visited  current ← pop elemen teratas  **for** (friend as semua friends dari current)  **if** (friend belum di visit) **then**  fnode ← node(friend)  second\_el ← Element(fnode)  **for** (second\_friend as connection(current))  connection(Second\_el) ← onnection(Second\_el)+ Element(fnode)  connection(Second\_el) ← connection(Second\_el)+ name(current)  push second\_el ke stack  **while** (jumlah stack > 0 dan belum di visit) **do**  current ← pop elemen teratas  **if** (jumlah stack > 0) **then**  current ← peek elemen teratas  connection(current) ← connection(current) + name(current)  **if** (current = person2) **then**  found ← true  → connection(current), found |

Notasi Algoritmik BFS Get Friend Recommendation

|  |
| --- |
| **function** *friendRecommendation*(G: **Graph**, person: **Node)** → **string**  **{I.S.:** G, person terdefinisi  **F.S.:** memberikan output berupa pesan rekomendasi teman dari Node input  sesuai format yang diminta.**}**  **Kamus:**  queue: **Queue<Node>**  friend: **string**  friend2: **string**  fnode: **Node**  fnode2: **Node**  count: **integer**  recommend: **Dictionary<string, int>**  container: **string**  **Algoritma:**  **for** (friend as semua friends dari person)  fnode ← Node pada G dengan nama friend  Enqueue fnode ke dalam queue  count ← jumlah element queue  **while** (count > 0) **do**  fnode ← Dequeue element teratas queue  **for** (friend2 as semua friends dari fnode)  fnode2 ← Node pada G dengan nama friend2  Enqueue fnode2 ke dalam queue  count ← count – 1  **while** (jumlah element queue > 0) **do**  fnode ← Dequeue element teratas queue  **if** (nama fnode ≠ nama person **and** fnode bukan friends dari person) **then**  **if** (fnode belum ada pada recommend) **then**  tambahkan <fnode, 1> pada recommend  **else**  recommend[fnode] ← recommend[fnode] + 1  urutkan recommend berdasarkan value terurut menurun  **for** (friend as Key pada recommend)  container ← container + friend + “\n” + recommend[friend] + “ mutual”  **if** (recommend[friend] = 1) **then**  container ← container + “friend: “  **else**  container ← container + “friends: “  fnode ← Node pada G dengan nama friend  hapus friends pada fnode jika tidak ada di friends pada person  **for** (friend2 as semua friends dari fnode)  container ← container + friend2 + “ “  container ← container + “\n”  → container |

Notasi Algoritmik DFS Get Friend Recommendation

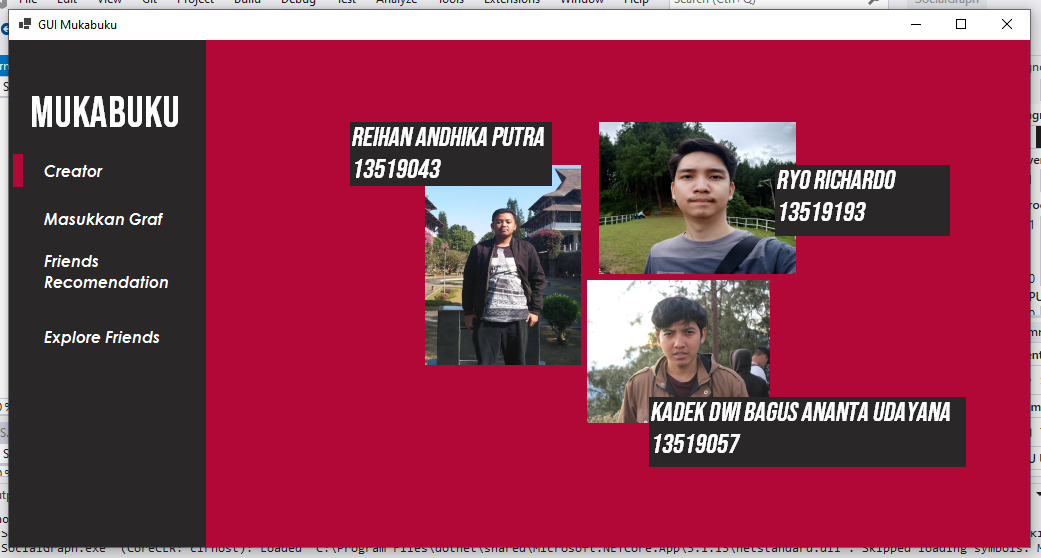
|  |
| --- |
| **function** *friendRecommendation*(G: **Graph**, person: **Node)** → **string**  **{I.S.:** G, person terdefinisi  **F.S.:** memberikan output berupa pesan rekomendasi teman dari Node input  sesuai format yang diminta.**}**  **Kamus:**  friend: **string**  friend2: **string**  fnode: **Node**  recommend: **Dictionary<string, int>**  container: **string**  **Algoritma:**  **for** (friend as semua friends dari person)  fnode ← Node pada G dengan nama friend  **for** (friend2 as semua friends dari fnode)  **if** (nama fnode ≠ nama person **and** fnode bukan friends person) **then**  **if** (fnode belum ada pada recommend) **then**  tambahkan <fnode, 1> pada recommend  **else**  recommend[fnode] ← recommend[fnode] + 1  urutkan recommend berdasarkan value terurut menurun  **for** (friend as Key pada recommend)  container ← container + friend + “\n” + recommend[friend] + “ mutual”  **if** (recommend[friend] = 1) **then**  container ← container + “friend: “  **else**  container ← container + “friends: “  fnode ← Node pada G dengan nama friend  hapus friends pada fnode jika tidak ada di friends pada person  **for** (friend2 as semua friends dari fnode)  container ← container + friend2 + “ “  container ← container + “\n”  → container |

1. **Penjelasan Struktur Data**

|  |  |  |
| --- | --- | --- |
| **Class Node** | **Atribut:**   1. string name 2. List<string> friends | **Method:**   1. addFriends 2. countFriends 3. ctor, getter, setter |
| **Penjelasan Singkat:**  Node adalah kelas yang merepresentasikan vertex pada graf pertemanan yang akan dibuat. Node juga merepresentasikan orang dalam graf pertemanan. Suatu node memiliki nama dan list yang berisikan kumpulan nama temannya. List kumpulan nama teman merepresentasikan *edge.* | | |
| **Class Graph** | **Atribut:**   1. List<node> vertex | **Method:**   1. addEdge 2. addNode 3. ctor, getter, setter |
| **Penjelasan Singkat:**  Graf adalah kelas yang merepresentasikan graf pada graf pertemanan yang akan dibuat. Graf mempunyai lis yang berisikan kumpulan node(orang). | | |
| **Class Parser** | **Atribut:**   1. Graph result 2. Int numOfEdge 3. List<string> uniqueName 4. String[]files | **Method:**   1. ctor |
| **Penjelasan Singkat:**  Parser adalah kelas yang merepresentasikan mesing untuk membaca input dari file.txt dan diubah ke dalam representasi graf. Parser mempunyai graf, banyaknya *edge*, list yang berisikan nama orang yang unik di dalam graf dan list yang berisikan input dari file.txt. Nantinya pemanggilan ctor dari parser akan disertai *path* dari file yang akan dibuka. Semua atribut di parser bersifat statis sehingga bisa dipanggil tanpa instansiasi di program utama/GUI. | | |
| **Class Visualizer** | **Atribut:**   1. Viewer MSAGL | **Method:**   1. Visualize |
| **Penjelasan Singkat:**  Visualizer adalah kelas yang merepresentasikan visualizer dari graf yang akan dibentuk. Kelas ini bertujuan untuk menggambar graf sesuai spesifikasi dan memberikan atribut warna sesuai kebutuhan. | | |
| **Class Element** | **Atribut:**   1. Node person 2. List<string> connection | **Method:**   1. addConnection 2. getName 3. ctor |
| **Penjelasan Singkat:**  ElQueue adalah kelas yang menjadi elemen queue pada BFS dan stack pada DFS. Kelas ini bertujuan untuk melengkapi kelas Node dengan menambahkan atribut connection yang berupa daftar nama Node yang pernah dikunjungi ketika proses BFS/DFS berlangsung. | | |
| **Class Queue <T>** | **Atribut:**   1. List<T> collection | **Method:**   1. Enqueue 2. Dequeue 3. Peek |
| **Penjelasan Singkat:**  Queue adalah kelas generik yang sudah diimplemetasi dalam modul System.Collection.Generic yang tersedia dalam bahasa C#. Untuk menggunakannya, kita hanya perlu memanggil namespace System.Collection.Generic dalam program kita. Queue memiliki method Enqueue (menambah element ke dalam Queue), Dequeue (mengeluarkan element teratas Queue), dan Peek (mengintip element teratas Queue). Kelas Queue dimanfaatkan dalam kelas BFS. | | |
| **Class Stack <T>** | **Atribut:**   1. List<T> collection | **Method:**   1. Push 2. Pop 3. Peek |
| **Penjelasan Singkat:**  Stack adalah kelas generik yang sudah diimplemetasi dalam modul System.Collection.Generic yang tersedia dalam bahasa C#. Untuk menggunakannya, kita hanya perlu memanggil namespace System.Collection.Generic dalam program kita. Stack memiliki method Push (menambah element ke dalam Stack), Pop (mengeluarkan element teratas Stack), dan Peek (mengintip element teratas Stack). Kelas Stack dimanfaatkan dalam kelas DFS. | | |
| **Class Dictionary <TKey, TValue>** | **Atribut:**   1. KeyValuePair<TKey, TValue> current | **Method:**   1. ContainsKey 2. Add |
| **Penjelasan Singkat:**  Dictionary adalah kelas generik yang sudah diimplemetasi dalam modul System.Collection.Generic yang tersedia dalam bahasa C#. Untuk menggunakannya, kita hanya perlu memanggil namespace System.Collection.Generic dalam program kita. Dictionary memiliki method ContainsKey (memeriksa apakah Key input sudah ada dalam Dictionary) dan Add (menambah pasangan Key-Value ke dalam Dictionary). Kelas Dictionary dimanfaatkan dalam kelas BFS dan DFS. | | |
| **Class BFS** | **Atribut: -** | **Method**   1. friendRecommendation 2. exploreFriend |
| **Penjelasan Singkat:**  BFS adalah kelas yang beranggotakan method untuk menjalankan fungsi **friend recommendation** dan **explore friend** secara BFS. Method friendRecommendation menerima input Graph dan sebuah Node, serta memberi output string daftar friend recommendation Node tersebut. Method exploreFriend menerima input Graph, 2 buah Node, dan boolean, serta memberi output List<string> nama Node yang dikunjungi selama penelusuran dari Node input pertama ke Node input kedua secara BFS. | | |
| **Class DFS** | **Atribut: -** | **Method**   1. friendRecommendation 2. exploreFriend |
| **Penjelasan Singkat:**  DFS adalah kelas yang beranggotakan method untuk menjalankan fungsi **friend recommendation** dan **explore friend** secara DFS. Method friendRecommendation menerima input Graph dan sebuah Node, serta memberi output string daftar friend recommendation Node tersebut. Method exploreFriend menerima input Graph, 2 buah Node, dan boolean, serta memberi output List<string> nama Node yang dikunjungi selama penelusuran dari Node input pertama ke Node input kedua secara DFS. | | |

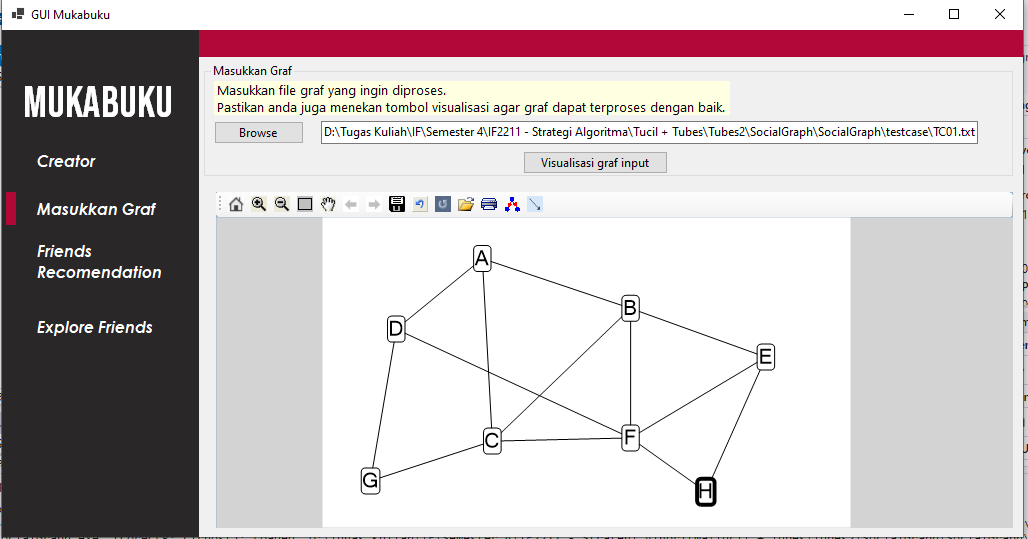
1. **Tata Cara Penggunaan Program**

Untuk dapat menjalankan program, kamu dapat mengklik file executable (.exe) yang ada pada folder *binary* yang telah disediakan atau kamu dapat membuka project lewat visual studio dengan meng-open file .sln. Cara lengkap dapat disimak di README.MD yang telah disediakan



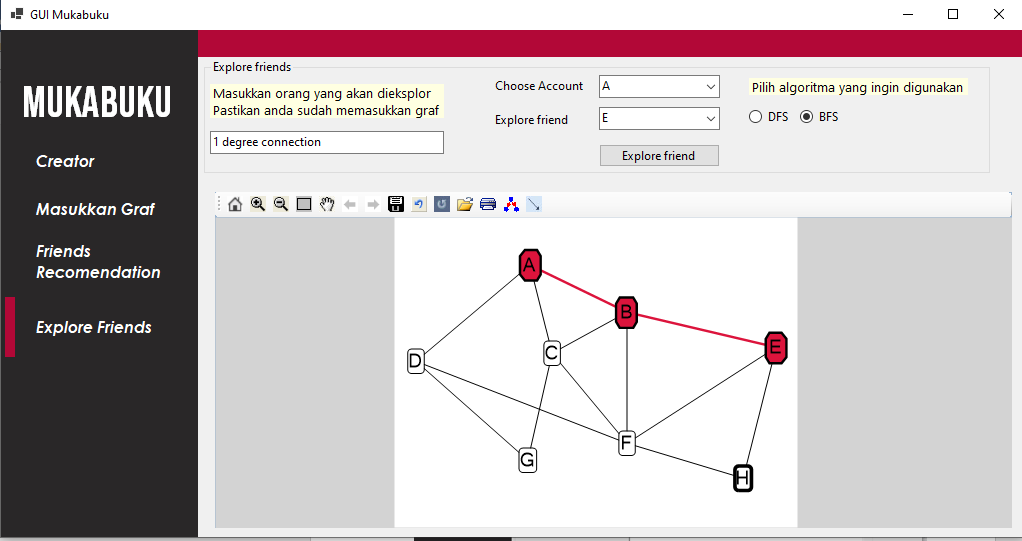
*Gambar 10. Halaman utama program*

Saat membuka aplikasi, halaman pertama yang akan muncul adalah halaman kreator. Halaman ini berisi data diri dari pembuat aplikasi. Di sisi kiri terdapat sidebar untuk melakukan navigasi antar halaman. Tekan tombol Masukkan Graf untuk memasukkan graf yang ingin dibaca dari file eksternal.



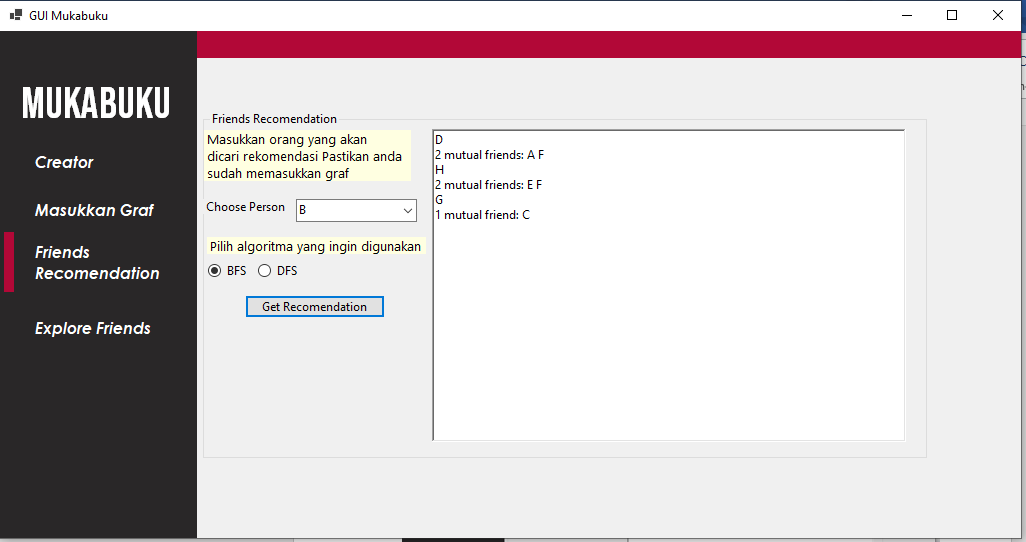
*Gambar 11. Halaman masukkan graf*

Setelah menekan tombol masukkan graf maka kamu akan berpindah ke halaman untuk memasukkan graf dari file eksternal. Pada kondisi awal akan terdapat kotak biru tempat graf akan digambar. Untuk mengambil file eksternal, tekanlah tombol browse dan popup pencarian file akan keluar. Carilah file graf yang ingin kamu load dan pastikan format filenya sudah benar. Jika sudah maka klik open file tersebut dan pathnya kana terisi di *textbox.* **Tekan tombol Visualisasi graf input untuk memasukkan data graf ke dalam aplikasi**. Apabila tombol tidak ditekan maka data dari graf tidak masuk ke aplikasi. Setelah tombol ditekan maka gambar graf akan muncul di layar. Setelah memvisualisasikan graf, kamu bisa menekan menu Explore Friends untuk melakukan eksplorasi pertemanan.



*Gambar 11. Halaman explore friends*

Setelah menekan tombol explore friend maka kamu akan masuk ke halaman untuk mencari derajat pertemanan. **Pastikan kamu sudah meng-*input* dan memvisualisasikan graf di halaman masukkan graf** **atau kamu tidak akan bisa memilih orang untuk di eksplorasi**. Pada halaman ini kamu dapat memilih dua orang di graf untuk dicek derajat pertemanannya. Kamu juga dapat memilih algoritma pengecekannya yaitu BFS atau DFS. Setelah kamu memasukkan pilihanmu, tekanlah tonbol *Explore Friend*. Visualisasi graf dan derajat pertemanan akan ditampilkan di layar. Apabila kamu memilih orang yang sama atau tidak memilih algortima maka akan ditampilkan pesan kesalahan. Setelah memvisualisasikan graf, kamu bisa menekan menu *Friends Recomendation* untuk mencari rekomendasi pertemanan.



*Gambar 12. Halaman friends recomendation*

Setelah menekan tombol *Friends Recommendation* maka kamu akan masuk ke halaman untuk mencari rekomendasi teman. **Pastikan kamu sudah meng-*input* dan memvisualisasikan graf di halaman masukkan graf** **atau kamu tidak akan bisa memilih orang untuk di cari rekomendasi teman**. Pada halaman ini kamu dapat memilih seseorang untuk dicari orang lain yang mungkin menjadi temannya. Setelah kamu memasukkan pilihanmu, tekanlah tonbol *Get Recomendation*. List rekomendadi teman yang terurut berdasarkan jumlah akan ditampilkan ke layar.

1. **Hasil Pengujian**
2. **Visualisasi Graf**

*Note: Di file aslinya, input adalah tiap pasangan satu baris*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **TC01.txt** | **Input:**  13  C F  C G  D G | | A B  F H  A D  D F  E H | | E F  B C  B E  B F  A C | | **Analisis:**   1. Program dapat menerima inputan dari file eksternal (.txt) dan mengubahnya ke bentuk graf. 2. Program dapat memvisualisasikan graf default yang ada di dokumen tubes. 3. Urutan penulisan pada inputan file eksternal tidak berpengaruh pada visualisasi graf dan pembentukan. |
| **Foto** | | | | | | | |
| **TC02.txt** | **Input:**  19  C D  D E  D T  D I | P D  P I  G P  G I  E Z | | E L  X Y  Y I  Y S  S G | | W B  W L  W X  Z A  Z B | **Analisis:**   1. Program dapat memvisualisasikan graf yang lebih kompleks |
| **Foto** | | | | | | | |
| **TC03.txt** | **Input:**  18  A B  B C  B F | B J  I J  K Z  K X  Z X | | W X  H K  Z H  C G  F J | | D E  F G  D F  W H  A C | **Analisis:**   1. Program bisa memvisualisasikan graf yang terputus |
| **Foto** | | | | | | | |

1. **Explore Friends ~ DFS**

*Note: Di file aslinya, input adalah tiap pasangan satu baris*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TC01.txt** | | **Input:**  13  C F  C G  D G | | | A B  F H  A D  D F  E H | | | E F  B C  B E  B F  A C | | | **Analisis:**   1. Program dapat memodelkan dan memvisualisasikan graf pada kasus yang disediakan asisten 2. Program dapat mensimulasikan pencarian jalur dengan algoritma DFS pada kasus yang disediakan asisten |
| **Foto DFS ke 1** | | | | | | | | | | | **Person 1:** A  **Person 2:** E  **Path yang dibentuk:** A-B-C-F-E  **Analisis DFS:**   1. DFS dapat mekukan backtrack yang seharusnya dari node F ke node D, tetapi karena tidak ada jalan sehingga backtrack ke arah E |
| **Foto DFS ke 2** | | | | | | | | | | | **Person 1:** E  **Person 2:** G  **Path yang dibentuk:** E-B-A-C-F-D-G  **Analisis DFS:**   1. DFS dapat melakukan track jalur ke arah yang benar |
| **TC02.txt** | **Input:**  19  C D  D E  D T | | | D I  P D  P I  G P  G I | | | E Z  E L  X Y  Y I  Y S | | | S G  W B  W L  W X  Z A  Z B | **Analisis:**  1) Program dapat memodelkan dan memvisualisasikan graf pada kasus dengan jumlah simpul yang banyak  2) Program dapat mensimulasikan pencarian jalur dengan algoritma DFS untuk semua kemungkinan masukkan |
| **Foto DFS ke 1** | | | | | | | | | | | **Person 1:** X  **Person 2:** D  **Path yang dibentuk:** X-W-B-Z-E-D  **Analisis DFS:**  1) DFS dapat melakukan track jalur ke arah yang benar |
| **Foto DFS ke 2** | | | | | | | | | | | **Person 1:** G  **Person 2:** A  **Path yang dibentuk:** G-I-D-E-L-W-B-Z-A  **Analisis DFS:**  1) DFS dapat mekukan backtrack yang seharusnya dari node D ke node C, tetapi karena tidak ada jalan sehingga backtrack ke arah E |
| **TC03.txt** | **Input:**  18  A B  B C  B F | | B J  I J  K Z  K X  Z X | | | W X  H K  Z H  C G  F J | | | D E  F G  D F  W H  A C | | **Analisis:**   1. Program dapat memodelkan dan memvisualisasikan graf pada kasus dengan graf terputus 2. Program dapat mensimulasikan pencarian jalur dengan algoritma DFS untuk semua kemungkinan masukkan |
| **Foto DFS ke 1** | | | | | | | | | | | **Person 1:** C  **Person 2:** J  **Path yang dibentuk:** C-A-B-F-J  **Analisis DFS:**  1) DFS dapat mekukan backtrack yang seharusnya dari node F ke node D dan node G, tetapi karena tidak ada jalan sehingga backtrack ke arah J |
| **Foto DFS ke 2** | | | | | | | | | | | **Person 1:** F  **Person 2:** K  **Path yang dibentuk: -**  **Analisis DFS:**   1. DFS tidak dapat menghasilkan jalur karena node F dan node K tidak berada pada graph yang sama |

1. **Explore Friends ~ BFS**

*Note: Di file aslinya, input adalah tiap pasangan satu baris*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TC01.txt** | | **Input:**  13  C F  C G  D G | | | A B  F H  A D  D F  E H | | | E F  B C  B E  B F  A C | | | **Analisis:**   1. Program dapat memodelkan dan memvisualisasikan graf pada kasus yang disediakan asisten 2. Program dapat mensimulasikan pencarian jalur dengan algoritma BFS pada kasus yang disediakan asisten |
| **Foto BFS ke 1** | | | | | | | | | | | **Person 1:** A  **Person 2:** H  **Path yang dibentuk:** A-B-E-H  **Analisis BFS:**   1. BFS berhasil membentuk path dengan benar 2. BFS berhasil membentuk path dengan jarak terpendek |
| **Foto BFS ke 2** | | | | | | | | | | | **Person 1:** E  **Person 2:** G  **Path yang dibentuk:** E-B-C-G  **Analisis BFS:**   1. BFS berhasil membentuk path dengan benar 2. BFS berhasil membentuk path dengan jarak terpendek |
| **TC02.txt** | **Input:**  19  C D  D E  D T | | | D I  P D  P I  G P  G I | | | E Z  E L  X Y  Y I  Y S | | | S G  W B  W L  W X  Z A  Z B | **Analisis:**   1. Program dapat memodelkan dan memvisualisasikan graf pada kasus dengan jumlah simpul yang banyak 2. Program dapat mensimulasikan pencarian jalur dengan algoritma BFS untuk semua kemungkinan masukkan |
| **Foto BFS ke 1** | | | | | | | | | | | **Person 1:** Y  **Person 2:** C  **Path yang dibentuk:** Y-I-D-C  **Analisis BFS:**   1. BFS berhasil membentuk path dengan benar 2. BFS berhasil membentuk path dengan jarak terpendek |
| **Foto BFS ke 2** | | | | | | | | | | | **Person 1:** S  **Person 2:** A  **Path yang dibentuk:** S-G-I-D-E-Z-A  **Analisis BFS:**   1. BFS berhasil membentuk path dengan benar 2. BFS berhasil membentuk path dengan jarak terpendek |
| **TC03.txt** | **Input:**  18  A B  B C  B F | | B J  I J  K Z  K X  Z X | | | W X  H K  Z H  C G  F J | | | D E  F G  D F  W H  A C | | **Analisis:**   1. Program dapat memodelkan dan memvisualisasikan graf pada kasus dengan graf terputus 2. Program dapat mensimulasikan pencarian jalur dengan algoritma BFS untuk semua kemungkinan masukkan |
| **Foto BFS ke 1** | | | | | | | | | | | **Person 1:** E  **Person 2:** A  **Path yang dibentuk:** E-D-F-B-A  **Analisis BFS:**   1. BFS berhasil membentuk path dengan benar 2. BFS berhasil membentuk path dengan jarak terpendek |
| **Foto BFS ke 2** | | | | | | | | | | | **Person 1:** A  **Person 2:** K  **Path yang dibentuk:** Tidak ada  **Analisis BFS:**   1. BFS tidak menemukan path karena Person 1 terpisah graf dengan Person 2 |

1. **Friends Recomendation**

*Note: Di file aslinya, input adalah tiap pasangan satu baris*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TC01.txt** | **Input:**  13  C F  C G  D G | | A B  F H  A D  D F  E H | | E F  B C  B E  B F  A C | **Analisis:**   1. Program dapat mencari mutual friend antara dua orang dengan algortima BFS dan DFS untuk graf dari asisten 2. Program dapat menghitung jumlah mutual Friend antara dua orang dan mensortir dari yang paling banyak. | | | |
| **Gambar Graf (Dari gambat ujicoba masukkan graf)** | | | | | | | | | |
| **BFS**  **Input Orang: C**  **Rekomendasi pertemanan:**  D  3 mutual friends: A F G  E  2 mutual friends: B F  H  1 mutual friend: F | | | | | | | **DFS**  **Input Orang: C**  **Rekomendasi pertemanan:**  D  3 mutual friends: A F G  E  2 mutual friends: B F  H  1 mutual friend: F | | |
| **BFS**  **Input Orang: F**  **Rekomendasi pertemanan:**  A  3 mutual friends: B C D  G  2 mutual friends: C D | | | | | | | **DFS**  **Input Orang: F**  **Rekomendasi pertemanan:**  A  3 mutual friends: B C D  G  2 mutual friends: C D | | |
| **TC02.txt** | **Input:**  19  C D  D E  D T  D I | P D  P I  G P  G I  E Z | | E L  X Y  Y I  Y S  S G | W B  W L  W X  Z A  Z B | **Analisis:**   1. Program dapat mencari mutual friend antara dua orang dengan algortima BFS dan DFS untuk graf yang lebih kompleks dari graf yang diberikan asisten | | | |
| **Gambar Graf (Dari gambat ujicoba masukkan graf)** | | | | | | | | | |
| **BFS**  **Input Orang: G**  **Rekomendasi pertemanan:**  G  2 mutual friends: I P  L  1 mutual friend: E  Z  1 mutual friend: E  Y  1 mutual friend: I | | | | | | | | **DFS**  **Input Orang: G**  **Rekomendasi pertemanan:**  G  2 mutual friends: I P  L  1 mutual friend: E  Z  1 mutual friend: E  Y  1 mutual friend: I | |
| **BFS**  **Input Orang: Z**  **Rekomendasi pertemanan:**  W  1 mutual friend: B  D  1 mutual friend: E  L  1 mutual friend: E | | | | | | | | **DFS**  **Input Orang: Z**  **Rekomendasi pertemanan:**  W  1 mutual friend: B  D  1 mutual friend: E  L  1 mutual friend: E | |
| **TC03.txt** | **Input:**  18  A B  B C  B F | B J  I J  K Z  K X  Z X | | W X  H K  Z H  C G  F J | D E  F G  D F  W H  A C | **Analisis:**   1. Program dapat mencari mutual friend antara dua orang dengan algortima BFS dan DFS untuk graf yang terpisah. | | | |
| **Gambar Graf (Dari gambat ujicoba masukkan graf)** | | | | | | | | | |
| **BFS**  **Input Orang: F**  **Rekomendasi pertemanan:**  C  2 mutual friends: B G  A  1 mutual friend: B  E  1 mutual friend: D  I  1 mutual friend: J | | | | | | | | | **DFS**  **Input Orang: F**  **Rekomendasi pertemanan:**  C  2 mutual friends: B G  A  1 mutual friend: B  E  1 mutual friend: D  I  1 mutual friend: J |
| **BFS**  **Input Orang: W**  **Rekomendasi pertemanan:**  K  2 mutual friends: H X  Z  2 mutual friends: H X | | | | | | | | | **DFS**  **Input Orang: W**  **Rekomendasi pertemanan:**  K  2 mutual friends: H X  Z  2 mutual friends: H X |

1. **Analisis Strategi BFS dan DFS**

Pada algoritma DFS, ide utama dalam algoritma ini adalah pembuatan stack. dengan metode LIFO atau Last In First Out. Sedangkan pada BFS, ide utama algoritma ini adalah menggunakan queue dengan metode FIFO atau First In First Out. Dalam penentuan algoritma mana yang lebih mahal dan algoritma mana yang lebih baik, kedua algoritma tersebut baik untuk masing-masing kasus. Algoritma DFS lebih baik jika mencari daun node paling jauh dalam sebuah Graph, tetapi algoritma BFS tidak akan efektif dalam hal ini. Sedangkan untuk mencari daun node yang tidak jauh dari node awal, maka algoritma BFS kemungkinan bisa lebih baik dari algoritma DFS, karena algoritma BFS memetakan semua kemungkinan yang terjadi. Contohnya pada kasus foto DFS ke 1 dalam Test Case 2 algoritma DFS, lebih baik menggunakan DFS daripada BFS, sedangkan pada kasus foto BFS ke 1 dalam Test Case 1 algoritma BFS, lebih baik menggunakan BFS daripada DFS.

**BAB V**

**PENUTUP**

* 1. **Kesimpulan**

Dari tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021 berjudul “Pengaplikasian Algoritma BFS dan DFS dalam Fitur *People You May Know* Jejaring Sosial Facebook”, kami berhasil membuat aplikasi dengan GUI untuk menampilkan derajat pertemanan dari beberapa orang yang direpresentasikan dengan graf dengan memanfaatkan algoritma BFS dan DFS. Program dibuat dengan kakas .NET dan menggunakan bahasa pemrograman C#.

* 1. **Saran**

Saran-saran yang dapat kami berikan untuk tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021 adalah:

* 1. Algoritma yang *dig*unakan pada Tugas Besar ini masih memiliki banyak kekurangan sehingga sangat memungkinkan untuk dilakukan efisiensi, misalnya dengan tidak menggunakan fungsi yang sama berulang-ulang. Oleh karena itu, dalam pengembangan program ini, masih bisa dilakukan efisiensi kinerja.
  2. Memperjelas spesifikasi dan batasan-batasan setiap program pada *file* tugas besar untuk mencegah adanya multitafsir dan kesalahpahaman pada proses pembuatan program.
  3. Penulisan *pseudocode* tampak kurang perlu dikarenakan program yang lumayan panjang dan membaca program lebih mudah daripada membaca *pseudocode* dengan asumsi program sudah *well commented.*
  4. **Refleksi**

Setelah menyelesaikan tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021, kami dapat merefleksikan beberapa hal, yaitu:

1. Komunikasi antar anggota kelompok berjalan dengan baik, sehingga tidak terjadi miskomunikasi atau kesalahpahaman selama pengerjaan.
2. Selama pengerjaan Tugas Besar, telah dibuat beberapa milestone untuk setiap orangnya dengan deadline yang bervariasi dan semua terselesaikan tepat waktu.
3. Selama proses pembuatan program, ketika ditemukan ketidaksesuaian saat proses eksperimen, temuan langsung dikomunikasikan kepada anggota kelompok lainnya dan bersama-sama mencari solusi.
4. Perlunya untuk mempelajari desktop application development agar kedepannya dapat membuat sebuah aplikasi dengan GUI yang lebih fungsional dan lebih user-friendly dari segi UI/UX.
5. Lebih merapikan source code program karena ada beberapa fungsi yang didefinisikan secara tidak modular.
6. Mengerjakan tugas besar dengan perasaan gembira, karena it’s not worth it if you’re not have fun.
   1. **Komentar**

Setelah menyelesaikan tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021, kami mempunyai beberapa komentar, yaitu:

1. Kak, kalau bisa tucil sama tubes selanjutnya tolong dipermudah ya soalnya april keos :D

**DAFTAR PUSTAKA**

Adegeo. (2020, October 26). Windows forms for .NET 5 Documentation. Diakses pada Maret

16, 2021, diambil dari <https://docs.microsoft.com/id-id/dotnet/desktop/winforms/>

Microsoft. (n.d.). Microsoft/automatic-graph-layout. Diakses pada Maret 17, 2021, diambil

dari <https://github.com/microsoft/automatic-graph-layout>

Munir, R. (n.d.). Materi Strategi Algoritma ITB 2021. Diakses pada Maret 15, 2021, diambil

dari <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/stima20-21.htm>

Fabian, Z. (n.d). Penerapan Algoritma BFS, DFS, dan A\* untuk P2P File Sharing dengan

Torrenting. Diakses pada Maret 17, 2021, diambil dari

<http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/stima19-20.htm#Makalah>