Nama : Miftakhul Amal

NIM : 5311421009

**ANALISIS PRAKTEK MODUL 4**

1. Tentukan bagaimana algoritma BFS di atas dapat menentukan node ke 8, 6, dan 7  
    Algoritma BFS (Breadth-First Search) digunakan dalam kode di atas untuk menjelajahi graf berdasarkan tingkat jarak (level) dari node awal (n3) ke node-node lain dalam graf. Untuk menentukan jarak dari node n3 ke node 8, 6, dan 7, kita dapat melihat nilai atribut `distance` dari masing-masing node setelah menjalankan algoritma BFS.

Dalam hasil yang Anda berikan:

- Node 8 memiliki `distance` 3.

- Node 6 memiliki `distance` 2.

- Node 7 memiliki `distance` 3.

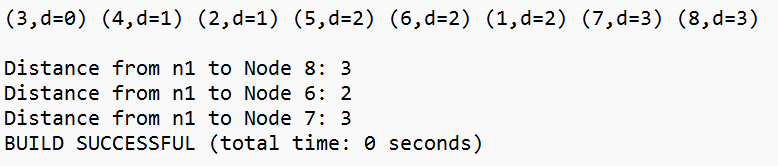
Ini berarti bahwa jarak dari node n3 ke node 8 adalah 3, jarak dari node n3 ke node 6 adalah 2, dan jarak dari node n3 ke node 7 adalah 3. Algoritma BFS secara efisien mencari jarak terpendek dalam graf dari node awal ke node-node lainnya.

Hasil ini didapatkan dengan mengikuti langkah-langkah berikut dalam algoritma BFS:

* Node n3 (awal) memiliki `distance` awal 0, dan kemudian kita menghitung jarak ke node tetangga dari n3 (yaitu n4, n2).
* Node n4 dan n2 memiliki `distance` awal yang ditetapkan sebagai `distance` n3 + 1, yaitu 1.
* Kemudian, kita menjelajahi lebih lanjut ke node-node tetangga dari n4 (yaitu n3, n5, n6) dan node tetangga dari n2 (yaitu n1, n3). `distance` dari node-node ini diperbarui sesuai dengan jarak dari node n3 (yang telah diubah) ditambah 1.
* Proses ini berlanjut hingga seluruh node dalam graf dijelajahi dan `distance` dari masing-masing node diperbarui sesuai dengan jarak terpendek dari node awal (n3).

Jadi, dengan menggunakan algoritma BFS, kita berhasil menentukan jarak dari node n3 ke node 8, 6, dan 7 sesuai dengan `distance` yang ditunjukkan dalam hasil yang Anda berikan.

**Hasil Praktek Simulasi :**



1. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.4 dapat dibentuk.

Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 5.

Dalam kode di atas, algoritma BFS digunakan untuk menjelajahi graf yang diwakili oleh adjacency list. Algoritma ini mencari node-node yang dapat dijangkau dari node awal (n1) dalam urutan yang mempertahankan tingkat jarak dari node awal. Untuk menentukan bagaimana algoritma BFS dapat menemukan node 5, kita dapat melihat langkah-langkah berikut:

* Awalnya, semua node diatur sebagai berwarna putih (WHITE), yang berarti mereka belum dijelajahi, jaraknya tak terbatas (Integer.MAX\_VALUE), dan belum memiliki predecesor.
* Algoritma dimulai dari node awal, yaitu n1. Node n1 diubah menjadi warna abu-abu (GRAY) yang menunjukkan bahwa sedang dalam proses penjelajahan, jaraknya diatur menjadi 0, dan predecesor diatur menjadi null.
* Node n1 kemudian dimasukkan ke dalam antrian (queue) `q` untuk diteruskan.
* Algoritma melanjutkan dengan mengeluarkan node n1 dari antrian. Kemudian, itu memeriksa tetangga-tetangga dari n1 yang belum dieksplorasi. Dalam kasus ini, n1 memiliki dua tetangga yang belum dijelajahi, yaitu n2 dan n3.
* Node n2 dan n3 dimasukkan ke dalam antrian dan diubah menjadi warna abu-abu. Jaraknya diperbarui sesuai dengan jarak dari n1 ditambah 1, sehingga menjadi 1. Predecesor mereka diatur sebagai n1.
* Algoritma kemudian melanjutkan dengan mengeluarkan n2 dari antrian dan memeriksa tetangga-tetangga n2 yang belum dijelajahi, yaitu n1, n4, dan n5.
* Node n4 dan n5 dimasukkan ke dalam antrian dan diubah menjadi warna abu-abu dengan jarak masing-masing adalah 2 dan predecesor adalah n2.
* Algoritma terus berlanjut dengan mengunjungi tetangga-tetangga n3 dan seterusnya hingga semua node yang dapat dijangkau telah dijelajahi.

Jadi, algoritma BFS menemukan node 5 (dan node-node lainnya) dengan mengikuti langkah-langkah penjelajahan dari node awal (n1) sesuai dengan tingkat jarak. Node 5 ditemukan setelah melewati node 1, node 2, dan node 4 dalam urutan yang mempertahankan jarak minimum. Node 5 ditemukan dengan jarak 2 dari node awal.

**Hasil Praktek Simulasi :**  


1. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.5 dapat dibentuk.

Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 9.

Dalam kode di atas, algoritma BFS digunakan untuk menjelajahi graf yang diwakili oleh adjacency list. Algoritma ini mencari node-node yang dapat dijangkau dari node awal (n1) dalam urutan yang mempertahankan tingkat jarak dari node awal. Untuk menentukan bagaimana algoritma BFS dapat menemukan node 9, kita dapat melihat langkah-langkah berikut:

* Awalnya, semua node diatur sebagai berwarna putih (WHITE), yang berarti mereka belum dijelajahi, jaraknya tak terbatas (Integer.MAX\_VALUE), dan belum memiliki predecesor.
* Algoritma dimulai dari node awal, yaitu n1. Node n1 diubah menjadi warna abu-abu (GRAY) yang menunjukkan bahwa sedang dalam proses penjelajahan, jaraknya diatur menjadi 0, dan predecesor diatur menjadi null.
* Node n1 kemudian dimasukkan ke dalam antrian (queue) `q` untuk diteruskan.
* Algoritma melanjutkan dengan mengeluarkan node n1 dari antrian. Kemudian, itu memeriksa tetangga-tetangga dari n1 yang belum dieksplorasi. Dalam kasus ini, n1 memiliki tiga tetangga yang belum dijelajahi, yaitu n2, n3, dan n4.
* Node n2, n3, dan n4 dimasukkan ke dalam antrian dan diubah menjadi warna abu-abu. Jarak mereka diperbarui sesuai dengan jarak dari n1 ditambah 1, sehingga menjadi 1. Predecesor mereka diatur sebagai n1.
* Algoritma kemudian melanjutkan dengan mengunjungi tetangga-tetangga n2, n3, dan n4 dan seterusnya hingga semua node yang dapat dijangkau telah dijelajahi.
* Node 9 ditemukan setelah melewati node 5 dalam urutan yang mempertahankan jarak minimum. Node 9 ditemukan dengan jarak 3 dari node awal (n1).

Jadi, algoritma BFS menemukan node 9 dengan mengikuti langkah-langkah penjelajahan dari node awal (n1) sesuai dengan tingkat jarak.

**Hasil Praktek Simulasi :**



1. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.5 dapat dibentuk.

Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 9.

Dalam kode di atas, algoritma BFS digunakan untuk menjelajahi graf yang diwakili oleh adjacency list. Algoritma ini mencari node-node yang dapat dijangkau dari node awal (n6) dalam urutan yang mempertahankan tingkat jarak dari node awal. Untuk menentukan bagaimana algoritma BFS dapat menemukan node 3, kita dapat melihat langkah-langkah berikut:

* Awalnya, semua node diatur sebagai berwarna putih (WHITE), yang berarti mereka belum dijelajahi, jaraknya tak terbatas (Integer.MAX\_VALUE), dan belum memiliki predecesor.
* Algoritma dimulai dari node awal, yaitu n6. Node n6 diubah menjadi warna abu-abu (GRAY) yang menunjukkan bahwa sedang dalam proses penjelajahan, jaraknya diatur menjadi 0, dan predecesor diatur menjadi null.
* Node n6 kemudian dimasukkan ke dalam antrian (queue) `q` untuk diteruskan.
* Algoritma melanjutkan dengan mengeluarkan node n6 dari antrian. Kemudian, itu memeriksa tetangga-tetangga dari n6 yang belum dieksplorasi. Dalam kasus ini, n6 memiliki dua tetangga yang belum dijelajahi, yaitu n2 dan n7.
* Node n2 dan n7 dimasukkan ke dalam antrian dan diubah menjadi warna abu-abu. Jarak mereka diperbarui sesuai dengan jarak dari n6 ditambah 1, sehingga menjadi 1. Predecesor mereka diatur sebagai n6.
* Algoritma kemudian melanjutkan dengan mengunjungi tetangga-tetangga n2 dan n7 dan seterusnya hingga semua node yang dapat dijangkau telah dijelajahi.

Node 3 ditemukan setelah melewati node 4 dalam urutan yang mempertahankan jarak minimum. Node 3 ditemukan dengan jarak 3 dari node awal (n6). Jadi, algoritma BFS menemukan node 3 dengan mengikuti langkah-langkah penjelajahan dari node awal (n6) sesuai dengan tingkat jarak.

**Hasil Praktek Simulasi :**



**ANALISIS PRAKTEK MODUL 5**

1. Pelajari class EightPuzzleSearch, EightPuzzleSpace, dan Node.  
    Class EightPuzzleSearch merupakan implementasi algoritma pencarian pada masalah 8-Puzzle. Masalah 8-Puzzle adalah masalah permainan papan di mana kita harus menggerakkan ubin-ubin yang teracak pada sebuah papan 3x3 sehingga membentuk pola yang diinginkan. Dalam kasus ini, "0" mewakili ubin kosong yang dapat digerakkan.
   1. Class Node:

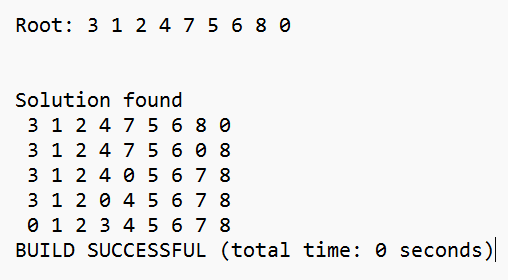
Class ini digunakan untuk merepresentasikan setiap simpul dalam pohon pencarian. Setiap simpul memiliki atribut sebagai berikut:

* `state`: Menyimpan konfigurasi papan permainan (dalam bentuk array 1 dimensi).
* `cost`: Menyimpan biaya atau nilai g(n) dari simpul ini.
* `parent`: Menyimpan referensi ke simpul induk dalam pohon pencarian.
* `successors`: Menyimpan daftar dari simpul-simpul anak.
* Terdapat metode `equals` untuk membandingkan apakah dua simpul memiliki konfigurasi yang sama.
* Metode `getPath` digunakan untuk mendapatkan jalur dari simpul awal hingga simpul ini.
  1. Class EightPuzzleSearch:
* Class ini berisi logika utama dari algoritma pencarian untuk menyelesaikan masalah 8-Puzzle.
* Terdapat dua metode heuristik yang dapat digunakan: `h1Cost` (menghitung jumlah ubin yang terbalik) dan `h2Cost` (menghitung jarak Manhattan antara ubin-ubin).
* Terdapat metode `hCost` yang dapat digunakan untuk memilih heuristik yang akan digunakan.
* Terdapat metode `getBestNode` untuk mendapatkan simpul dengan biaya terendah dari daftar simpul.
* Terdapat metode `getPreviousCost` untuk mendapatkan biaya dari simpul yang sama pada daftar terbuka atau tertutup.
* Metode `run` adalah metode utama yang menjalankan algoritma pencarian A\* untuk menemukan solusi.
* Terdapat metode `printPath` yang digunakan untuk mencetak jalur solusi ke layar.
* Metode `main` digunakan untuk memulai proses pencarian.
  1. Class EightPuzzleSpace:

Class ini mewakili ruang pencarian 8-Puzzle. Ini berisi aturan dan operasi yang berkaitan dengan masalah 8-Puzzle, seperti mendapatkan simpul awal dan simpul tujuan, serta menghasilkan daftar simpul suksesor.

Algoritma A\* digunakan dalam class EightPuzzleSearch untuk mencari solusi dengan menghitung nilai biaya f(n) dari setiap simpul, yang merupakan penjumlahan biaya sejauh ini (g(n)) dan estimasi biaya ke tujuan (h(n)) berdasarkan heuristik yang digunakan.

**Hasil Simulasi Praktek :**



1. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya

Gambar 8. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal

state. Analisa dan bedakan dengan solusi pada point 1.

Untuk menentukan langkah-langkah yang diperlukan agar puzzle mencapai goal state, dapat melakukan pencarian solusi dengan menggunakan algoritma pencarian seperti A\* (A star) yang telah diimplementasikan dalam kode program EightPuzzleSearch di atas. Algoritma A\* adalah salah satu algoritma pencarian yang digunakan untuk menemukan solusi terpendek dari sebuah masalah.

Dalam kasus Eight Puzzle, dapat memulai dari initial state (getRoot()) dan mencari solusi dengan langkah-langkah berikut:

* Inisialisasi open list dengan node initial (root) dan closed list kosong.
* Ambil node terbaik dari open list berdasarkan nilai fungsi heuristik (hCost) yang lebih rendah.
* Periksa apakah node tersebut merupakan goal state. Jika iya, maka telah menemukan solusi dan dapat menghentikan pencarian.
* Jika node bukan goal state, ekspansi node tersebut untuk mendapatkan semua node potensi yang dapat dicapai dengan satu langkah.
* Periksa setiap node potensi yang dihasilkan dari ekspansi. Jika node tersebut belum ada dalam open list atau memiliki biaya yang lebih rendah dari node yang sama dalam open list, tambahkan node tersebut ke open list.
* Tandai node yang telah diekspansi sebagai sudah ditutup (closed) dan hapus dari open list.
* Ulangi langkah 2-6 hingga menemukan goal state atau tidak ada node lagi dalam open list.  
  **Hasil Praktek Simulasi :**

A screenshot of a computer game

Description automatically generated

1. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya

Gambar 5.9. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai

goal state. Analisa dan bedakan dengan solusi pada point 1 dan 2.

Algoritma ini mencari solusi dengan mengeksplorasi berbagai keadaan ubin (node) dan menggunakan heuristik (h1 atau h2) untuk menentukan prioritas penelusuran. Heuristik ini digunakan untuk memprediksi biaya yang diperlukan untuk mencapai goal state.

Langkah-langkah yang diambil oleh kode tersebut adalah:

* Inisialisasi initial state (getRoot) dan goal state (getGoal) dari teka-teki delapan ubin.
* Mencari jalur pencarian dari initial state ke goal state dengan menggunakan algoritma A\*.
* Menghitung biaya heuristik (hCost) untuk setiap node yang dieksplorasi. Heuristik ini dapat menggunakan metode h1 atau h2, di mana h1 menghitung berapa banyak ubin yang berbeda dengan goal state, sedangkan h2 mengukur jarak manhattan antara ubin-ubin dalam initial state dengan goal state.
* Menentukan node terbaik (getBestNode) untuk dieksplorasi selanjutnya berdasarkan biaya heuristik terendah.
* Mengeksplorasi node-node yang mungkin (getSuccessors) dari node saat ini dengan memindahkan ubin kosong ke atas, bawah, kanan, atau kiri jika memungkinkan.
* Memeriksa apakah solusi sudah ditemukan. Jika ya, maka pencarian berhenti.
* Jika solusi belum ditemukan, maka algoritma A\* akan terus mencari hingga menemukan solusi.

Untuk menganalisis langkah-langkah yang diperlukan untuk mencapai goal state, dapat menjalankan kode ini dengan initial state yang berbeda dan mengamati outputnya. Algoritma A\* akan mencari solusi dengan mengeksplorasi berbagai kombinasi langkah hingga mencapai goal state. Langkah-langkah yang diambil akan tergantung pada keadaan awal (initial state) dari teka-teki delapan ubin tersebut.

Perlu diingat bahwa algoritma A\* digunakan untuk menemukan solusi optimal dengan mempertimbangkan biaya heuristik. Langkah-langkah yang diambil dapat berbeda tergantung pada initial state dan goal state yang digunakan dalam teka-teki delapan ubin tersebut.

**Hasil Praktek Simulasi :**

A screenshot of a computer

Description automatically generated

1. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya

Gambar 5.10. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai

goal state. Analisa dan bedakan dengan solusi pada point 1, 2, dan 3.

Untuk menentukan langkah-langkah menuju goal state, perlu dilakukan analisis pada kode pertama, terutama pada metode `run()` yang merupakan inti dari algoritma pencarian A\*. Di dalam metode ini, program mencoba untuk menemukan solusi dengan melakukan ekspansi dari setiap node pada setiap iterasi.

Untuk setiap node yang diekspansi, langkah-langkah yang diambil untuk mencapai goal state adalah sebagai berikut:

* Program memilih node dengan biaya terkecil (cost) dari kumpulan node yang belum dieksplorasi.
* Jika node yang diekspansi adalah goal state, maka pencarian dihentikan dan solusi ditemukan.
* Jika node yang diekspansi bukan goal state, program akan menghasilkan node-node penerus dari node tersebut.
* Setiap node penerus akan dievaluasi dengan fungsi heuristik (dalam hal ini menggunakan h2Cost) dan biaya path dari root.
* Jika node penerus tidak ada di dalam kumpulan node yang belum dieksplorasi atau di dalam kumpulan node yang telah dieksplorasi, atau memiliki biaya path yang lebih rendah dari node yang ada di kumpulan node yang telah dieksplorasi, maka node penerus akan dimasukkan ke dalam kumpulan node yang belum dieksplorasi.
* Proses berlanjut hingga goal state ditemukan atau tidak ada node lagi yang bisa diekspansi.

Sementara itu, pada kode kedua (`EightPuzzleSpace`), terdapat definisi initial state dan goal state. Initial state adalah tata letak awal dari puzzle, sementara goal state adalah tata letak yang harus dicapai. Method `getSuccessors(Node parent)` digunakan untuk menghasilkan node-node penerus dari suatu node tertentu, yaitu node yang mungkin dicapai dari node tersebut dengan satu langkah.  
**Hasil Praktek Simulasi :**

A screenshot of a computer

Description automatically generated

1. Ubahlah initial dan goal state dari program dan class-class di atas sehingga bentuk initial dan

goal statenya Gambar 5.11. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state.  
**Hasil Praktek Simulasi :**Dalam Simulasi yang dilakukan berkaitan dengan mengubah initial dan goal state dari program dan class-class di atas sehingga bentuk initial dan goal statenya Gambar 5.11. tidak dapat diselesaikan karena proses running yang sangat lama.