Algoritma Greedy adalah pendekatan dalam pemrograman di mana kita membuat keputusan yang terbaik (optimal) pada setiap langkah dengan harapan bahwa keputusan ini akan membawa kita pada solusi yang optimal secara keseluruhan. Dalam konteks labirin, ini berarti kita akan selalu mencoba bergerak ke arah yang terlihat paling menjanjikan pada setiap langkah.

**Penjelasan Kode Secara Umum**

1. **Membaca Labirin dari File:** Kode pertama-tama membaca labirin dari sebuah file teks dan menyimpan labirin tersebut dalam bentuk array dua dimensi. Labirin terdiri dari titik awal (**S**), titik akhir (**E**), dinding (**#**), dan jalur yang dapat dilalui (**.**).
2. **Mencari Jalur dengan Algoritma Greedy:** Algoritma Greedy di sini bekerja dengan mencoba semua kemungkinan jalur dari titik awal ke titik akhir. Pada setiap langkah, algoritma akan mencoba untuk bergerak ke salah satu dari empat arah (kanan, bawah, kiri, atas) yang dianggap paling menjanjikan.
3. **Menandai Jalur yang Sudah Dikunjungi:** Agar tidak kembali ke tempat yang sama dan menciptakan lingkaran tak berujung, algoritma menandai sel-sel yang sudah dikunjungi.
4. **Menemukan Semua Jalur:** Dengan cara ini, algoritma akan mencari semua jalur yang mungkin dari titik awal ke titik akhir dan menyimpan jalur-jalur tersebut.
5. **Mencari Jalur Terpendek dan Terpanjang:** Setelah menemukan semua jalur, algoritma kemudian mencari jalur yang terpendek dan terpanjang di antara semua jalur yang ditemukan.

Keterangan Tambahan:  
Algoritma Greedy mencoba semua jalur yang mungkin dengan bergerak ke arah yang terlihat paling menjanjikan pada setiap langkah. Meskipun mencoba semua jalur mungkin tampak tidak seperti pendekatan Greedy murni (karena Greedy biasanya tidak mencoba semua kemungkinan), dalam konteks ini, kita menggunakan Greedy dalam setiap langkah pencarian jalur. Algoritma memprioritaskan jalur yang belum dikunjungi dan menghindari jalur yang sudah dikunjungi untuk menemukan solusi yang valid.

**Hubungan Kode dan Greedy Algorithm**

// Inisialisasi dan Pembacaan Labirin:

void readMaze**(**char **\***filename**)** **{**

// Membaca file labirin dan menyimpan di array dua dimensi

**}**

//Validasi Gerakan:

//Fungsi isValidMove memastikan bahwa langkah berikutnya sah (tidak melewati dinding atau keluar dari batas labirin).

int isValidMove**(**int x**,** int y**,** int visited**[**MAX**][**MAX**])** **{**

// Mengecek apakah posisi (x, y) sah untuk dilalui

**}**

//Mencari Jalur (Fungsi Greedy):

//Fungsi findPaths menggunakan Depth-First Search (DFS) yang merupakan pendekatan Greedy untuk mencari jalur.

void findPaths**(**int x**,** int y**,** int visited**[**MAX**][**MAX**],** Path currentPath**)** **{**

**if** **(**x **==** end**.**x **&&** y **==** end**.**y**)** **{**

// Jika sudah mencapai titik akhir, simpan jalur ini

**}**

// Mencoba bergerak ke empat arah (kanan, bawah, kiri, atas)

**for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**

int newX **=** x **+** directions**[**i**][**0**];**

int newY **=** y **+** directions**[**i**][**1**];**

**if** **(**isValidMove**(**newX**,** newY**,** visited**))** **{**

// Tandai posisi ini sebagai dikunjungi

visited**[**newX**][**newY**]** **=** 1**;**

// Tambahkan posisi ini ke jalur saat ini

currentPath**.**points**[**currentPath**.**length**++]** **=** **(**Point**){**newX**,** newY**};**

// Rekursif mencari jalur dari posisi baru ini

findPaths**(**newX**,** newY**,** visited**,** currentPath**);**

// Kembali dan tandai posisi ini sebagai tidak dikunjungi

visited**[**newX**][**newY**]** **=** 0**;**

currentPath**.**length**--;**

**}**

**}**

**}**

//Memulai Pencarian:

//Fungsi solveMaze memulai pencarian dari titik awal dan memanggil fungsi findPaths.

void solveMaze**()** **{**

int visited**[**MAX**][**MAX**]** **=** **{**0**};**

Path initialPath **=** **{{**start**},** 1**};**

visited**[**start**.**x**][**start**.**y**]** **=** 1**;**

findPaths**(**start**.**x**,** start**.**y**,** visited**,** initialPath**);**

**}**