Maze Master

Dokumentacja implementacyjna

https://github.com/JGolaszewski/MazeMaster

Jędrzej Gołaszewski i Szymon Stasiak

April 11, 2024

Funkcjonalność programu

Celem tego programu jest umożliwienie użytkownikowi znalezienia najkrótszej ścieżki przez labirynt, który został stworzony na podstawie pliku wygenerowanego na stronie http://tob.iem.pw.edu.pl/maze/. Program wygeneruje precyzyjny ciąg instrukcji, który prowadzi od punktu początkowego do punktu wyjściowego. Dzięki niemu użytkownik może sprawnie i efektywnie odnaleźć właściwą drogę. Jednocześnie, stanowi on fundament dla następnego programu który będzie wizualizował labirynt i zaznaczał na nim ścieżkę.

Użytkowanie

Ten program służy do rozwiązywania labiryntów zapisanych w plikach tekstowych. Rozwiązanie jest zapisywane do innego pliku tekstowego lub wyświetlane w konsoli, jeśli nie zostanie podany plik wyjściowy.

./bin -i <plik_wejściowy> [-o <plik_wyjściowy>] [opcje]

Argumenty programu

-i <plik_wejściowy> lub -in <plik_wejściowy>: Określa nazwę pliku tekstowego zawierającego labirynt do rozwiązania.

-o <plik_wyjściowy> lub -out <plik_wyjściowy>: Określa nazwę pliku tekstowego, do którego program zapisze rozwiązanie labiryntu. Jeśli ta opcja nie zostanie podana, rozwiązanie zostanie wyświetlone w konsoli.

-v lub -verbose: Włącza wyświetlanie dodatkowych informacji podczas działania programu.

-d lub -debug: Włącza tryb debugowania, umożliwiając analizę szczegółowych informacji na temat działania programu.

-h lub -help: Wyświetla pomoc dotyczącą sposobu użycia programu.

-q lub -quit: Natychmiastowe wyjście z programu.

Architektura Systemu

System na jakim działa, systemowe funkcje jakies ewentualnie podzial na pliki programu, jezyk w jakim jest napisany wersja jezyka (c2x), foldery wejście wyjście bin itp

Środowisko uruchomieniowe

Program jest przystosowany do działania w środowisku Unixowym

Testy programu były prowadzone w:

Subsystem Kali Linux for Windows 11

Linux Ubuntu

Wersja C

Program jest napisany w języku programowania C w standardzie C23 (C2X);

Makefile

Ten Makefile jest narzędziem do automatyzacji procesu kompilacji programu napisanego w języku C. Składa się z trzech głównych celów: "build", "clean" i "test". Cel "build" kompiluje pliki źródłowe do pliku wykonywalnego, przy użyciu zdefiniowanych flag kompilacji. "Clean" usuwa plik wykonywalny, a "test" uruchamia program z określonymi argumentami w celu przetestowania go. Zmienne definiują nazwę pliku wykonywalnego, katalog wyjściowy i parametry kompilacji. Makefile ten zapewnia prosty i elastyczny sposób zarządzania procesem kompilacji i testowania programu w języku C.

Opis plików

Pliki Wejściowe:

- Plik binarny

Plik binarny zawiera nagłówek pliku, sekcje kodującą, nagłówek sekcji kodowania oraz sekcja wyniku.

Nagłówek pliku składa się z :

- 32 bitów id pliku(0x52524243)
- 8 bitów znak ESC
- Liczba kolumn 16 bitów
- Liczba linii 16 bitów
- Pozycja x wejścia 16 bitów
- Pozycja y wejścia 16 bitów
- Pozycja x wyjścia 16 bitów
- Pozycja y wyjścia 16 bitów
- Pamięć zarezerwowaną 96 bitów
- Licznik słów kodowych 32 bitów
- Wskaźnik na rozwiązanie 32 bitów
- Początek słowa kodowego (Separator) 8 bitów
- Definicja ściany labiryntu 8 bitów

- Definicja ścieżki labiryntu 8bitów

  W sumie: 420 bitów

  Slowa kodowe:

- Separator 8 bitów
- Wartość 8 bitów (Ściana / Ścieżka)
- Liczba wystąpień 8 bitó (0 – to jedno wystąpienie)

Nagłówek rozwiązania:

- Id sekcji rozwiązania 32 bitów (0x52524243)
- Liczba kroków do przejścia 8 bitów (0 – to jeden krok)

W sumie 40 bitów

Krok rozwiązania:

- Kierunek 8 bitów (N, E, S, W)
- Licznik pól do przejścia (0 – to jedne pole)
- Plik tekstowy

Plik tekstowy zawiera labirynt zapisany tekstowo gdzie jeden znak odpowiada jednemu polu na siatce labiryntu. Znak '#' - odpowiada ścianie labiryntu a ' ' - odpowiada ścieżce labiryntu, 'K' - odpowiada końcu a 'P' - odpowiada początkowi.

Przykładowy plik labiryntu:

Pliki programu:

- main.c

Main łączy moduły programu w spójną całość.

- graph.c oraz graph.h

Zawiera implementację struktury wierzchołków oraz funkcje służące do odczytu ich z tymczasowego node_temp //Odnośnik do plikow tymczasowych//

- BFS.c, BFS.h, readPath.c oraz readPath.h

Zawiera implementację modułu BFS //Odnośnik do modułu// wraz z odczytywaniem rozwiązania.

- interface.c, interface.h oraz reports.h

Zawiera implementację modułu komunikacji z użytkownikiem //Odnośnik do modułu// oraz makra do zgłaszania błędów.

- parser.c oraz parser.h

Zawiera implementację modułu //Odnośnik do modułu// parsera który zamienia plik labiryntu na grafowy jego odpowiednik.

- Macros.h

Zawiera makra stałych w programie i funkcje pomocnicze. //Odnośnik do akapitu o stałych i funkcjach pom//

Struktury programu

W programie wykorzystywane są dwie struktury własne

- Node (node_t)

Struktura node opisuje wierzchołek grafu (kafelek ścieżki w labiryncie).

W jej skład wchodzą zmienne: x (położenie x), y (położenie y), adj(zmienna trzymająca połączenia z kolejnymi wierzchołkami w sposób bitowy tzn.: od lewej 1 bit przejście w lewo, 2 bit przejście w prawo, 3 bit przejście w górę, 4 bit przejście w dół).

Implementacja struktury wygląda następująco:

//kod

- Queue (queue_t)

Struktura queue jest wrapperem na wskaźnik na plik(jest to plik tymczasowy w którym są przetrzymywane dane kolejki) poszerzona o 1 bit utrzymujący informację o tym czy kolejka jest aktualnie pusta.

Implementacja struktury wygląda następująco:

//kod

Założenia i rozwiązania systemu

Przygotowanie danych do algorytmu

Dane z pliku txt //Odnośnik do pliku tekstowego wejscia// i binarnego //Odnosnik do pliku wejsciowego bin// zawierające opis labiryntu są przetwarzane na formę grafu. Każde pole jest przetwarzane na osobny wierzchołek i przechowywane na siatce o rozmiarach x*y gdzie x to ilość kolumn labiryntu a y ilość wierszy. Każdy element nawet pole ze ściana ma swój wierzchołek jednak pola ściany są pojedynczymi wierzchołkami bez żadnych krawędzi.
Pola puste labiryntu te po których możemy chodzić posiadają opis w formie bitowej opisujących ich sąsiadów to znaczy możemy się z tego dowiedzieć w którym kierunku możemy pójść dalej do góry do dołu w prawo czy na lewo. Opis ten jest tworzony podczas parsowania //Odnośnik do parsowania//.

Użyty algorytm

BFS (Breadth-First Search) w rozwiązywaniu labiryntów

Algorytm rozwiązujący labirynt opiera się na BFS (Breadth-First Search), ponieważ jest to efektywna metoda znajdowania najkrótszej ścieżki w grafie nieskierowanym lub skierowanym, co jest istotne w przypadku poszukiwania wyjścia z labiryntu.

Kroki algorytmu:

Inicjalizacja: Rozpoczynamy od wierzchołka, który jest znanym wyjściem z labiryntu.

Przeszukiwanie: Wykonujemy algorytm BFS, sprawdzając kolejne wierzchołki labiryntu.

Oznaczanie kierunku: Podczas sprawdzania kolejnych wierzchołków grafu, oznaczamy kierunek, z którego przyszliśmy do danego wierzchołka. Dzięki temu będziemy mieli informację o ścieżce prowadzącej do danego wierzchołka.

Znalezienie wejścia: Kontynuujemy algorytm BFS aż do momentu znalezienia wejścia do labiryntu. Gdy to nastąpi, mamy pełną informację o ścieżce prowadzącej od wyjścia do wejścia.

Generowanie instrukcji: Znając ścieżkę od wyjścia do wejścia, możemy generować instrukcje krok po kroku, które należy wykonać, aby przejść przez labirynt.

Metoda parsowania

Metoda parsowania opiera się na zapisaniu wczytanych danych w sposób binarny jako poszczególne kolejne trzy wiersze z pliku następnie zidentyfikowanie połączeń pomiędzy pustymi polami i zapisanie ich zgodnie z formatem pliku temp_nodes //Odnośnik do plikow tymczasowych//.

Zamiana binarna odbywa się poprzez zamianę poszczególnych znaków w pliku wejściowym // Odnośnik pliki wejściowe// na ich bitowe reprezentacje (0 – wolna przestrzeń, 1 - ściana). Po wczytaniu 3 pierwszych wierszy program przystępuje do zamiany ich w bardziej czytelną formę krawędzi grafu. Jeżeli znak jest 0 sprawdza on połączenia we wszystkie 4 kierunki następnie zapisuje je do pliku tymczasowego temp_nodes //Odnośnik do plików tymczasowych// wraz z przestrzenia na zapis flagi odwiedzenia oraz kierunku rodzica. Następnie program zwalnia pamięć pierwszego wiersza i wczytuje kolejny powtarzając proces aż do wyczerpania wierszy w pliku.

Pliki tymczasowe

Program korzysta z 2 różnych plików tymczasowych aby ograniczyć zużycie pamięci procesora do maksymalnie 512 kB.

- Plik “temp_nodes”

- Zastosowanie:

  Przetrzymuje dane wierzchołka grafu (adj – 4 bity na możliwe kierunki przejścia, flag – 1 bit na flagę odwiedzenia, parent – 2 bity na kierunek w którym jest rodzic wierzchołka).

- Struktura:

  W pliku dane zapisywane są w sposób binarny po kolei od lewej do prawej, z góry na dół. Każda ściana z pliku jest odzwierciedlona jako pusta przestrzeń o rozmiarze wierzchołka a każda ścieżka jest przedstawiona jako odpowiednie dane wierzchołka //patrz zastosowanie//. Takie rozwiązanie pozwala na natychmiastowe odczytanie danych wierzchołka o pozycji x y bez przeszukiwania pliku. Zarówno puste jaki pełne ściany są odzwierciedlone w pliku jako N bitów (gdzie N jest ilością bitów które zajmuje struktura wierzchołka//Odnośnik do struktury node//)

- Plik "temp_queue"

- Zastosowanie:

  Przetrzymuje danych kolejki w pamięci pliku.

- Struktura:

  W pliku dane zapisywane są w sposób binarny, Każde kolejne N bitów (gdzie N jest ilością bitów które zajmuje struktura wierzchołka//Odnośnik do struktury node//) przetrzymuje dane kolejnych wierzchołków w kolejce. Dzięki temu że plik jest otwierany w trybie append+ możliwa jest optymalizacja pamięciowa nie zapisywania początku i końca danych kolejki, gdyż append automatycznie przechodzi na koniec kolejki podczas zapisywania do niej danych a kursor wskazuje na jej początek. Aby oszczędzić złożoność czasową stare elementy nie są usuwane w trakcie działania kolejki (plik jest usuwany po wywołaniu funkcji zamknięcia kolejki), a kursor wskazuje na realny pierwszy element.

Podział na moduły

-Parser

Moduł odpowiedzialny za parser składa się z 3 głównych funkcji:

- readLineBit

Jest to funkcja odpowiedzialna za przekształcenie wiersza z pliku na jej odpowiednik bitowy (Ściana -> 1, Ścieżka -> 0). Odbywa się to poprzez iterację przez poszczególne znaki i zapis ich w tablicy bajtów jako poszczególne bity.

- toGraph

Funkcja toGraph otrzymuje 3 kolejne linie zapisane binarnie analizując środkową z nich i zapisując ją do pliku tymczasowego temp_nodes //odnośnik// w co wchodzą połączenia pomiędzy pustymi polami oraz puste flagi dla algorytmu oraz puste przestrzenie pamięci o rozmiarze takim samym jak dane wierzchołka dla każdej ściany.

- parseFile

Jest to funkcja która łączy dwie poprzednie. Jako argument dostaje nazwę pliku wejściowego następnie wczytując trójkami wiersze z niego w sposób binarny, za pomocą funkcji readLineBit i wywołując funkcję toGraph co każdą wczytaną linijkę.

-BFS

Moduł odpowiedzialny za znalezienie najkrótszej ścieżki przejścia labiryntu składa się z funkcji bfs odpowiedzialnej za przeszukanie grafu i stworzenie relacji w kolejnych wierzchołkach. W trakcie wykonywania bfs zapisujemy z jakiego wierzchołka tu przyszliśmy .
Propozycja implementacji:

Drugą funkcją tego modułu jest funkcja readPath. Funkcja ta odpowiada za wypisanie kolejnych kroków, którymi należy się poruszać by odnaleźć najkrótszą ścieżkę. Funkcja ta iteruje przez kolejne wierzchołki rozpoczynając od wierzchołka w którym znajduje się wyjście z labiryntu. Iteruje korzystając z relacji rodzica którą stworzył bfs . Sprawdza z której strony został odwiedzony wierzchołek podczas wykonywania bfs a następnie przechodzi w tą stronę . W ten sposób poznajemy kolejne kroki i wykonujemy to do momentu dojścia do wejścia labiryntu.

-Implementacja kolejki

Moduł odpowiedzialny za implementacje kolejki składa się z funkcji w pełni operujących na pliku kolejki wykonujące podstawową jej funkcjonalność (taką jak pop, push, create i destroy). Każda funkcjonalność jest rozdzielona na osobną funkcje:

- create_q

  Funkcja której zadaniem jest otworzenie pliku tymczasowego kolejki. Na początku otwiera go w trywie 'write' aby usunąć zawartość potencjalnie istniejącego pliku o tej samej nazwie, następnie otwiera go w trybie append+ aby można było swobodnie dodawać nowe elementy na końcu pliku a wskaźnik pliku żeby wskazywał na realny pierwszy element kolejki.

- qelete_q

  Funkcja która ma na celu zamknięcie i usunięcie tymczasowego pliku kolejki.

- push_q

Funckcja zapisuje położenie początku po czym dopisuje nowy element na koniec pliku i ustawia wskaźnik pliku z powrotem na realny element początkowy kolejki. Zmienia ona także zmienną kolejki isEmpty na fałsz.

- pop_q

Funkcja która wczytuje pierwszy realny element kolejki i zmienia początek na kolejny. Jej zadaniem jest również ustalenie czy kolejka jest pusta. Jeżeli nie uda pobrać się elementu ustawia ona zmienną isEmpty na prawdę.

-Moduł funkcjonalny

W skład tego modułu wchodzi funkcja parse_args jest odpowiedzialna ze pobranie i przeanalizowanie argumentów wywołania programu.

Funkcja print_help w przypadku podania przez użytkownika argumentu -h lub –help wyświetla pomoc

// Zdjecie pomocy wyświetlonej

Trzecią funkcja występująca w tym module jest funkcja openFile która odpowiada za sprawdzenie czy podany przez użytkownika plik istnieje i jest poprawny. Jeżeli powyższe warunki są spełnione funkcja zwraca otwarty plik

Ponadto w skład tego modułu wchodzi 5 makr do obsługi powiadomień w programie:

R_INFO – odpowiedzialne za wyświetlanie informacji
R_WARNING – odpowiedzialne za wyświetlanie ostrzeżeń
R_DEBUG – odpowiedzialne za wyświetlanie dodatkowych informacji w trybie debug
R_VERBOSE – odpowiedzialne za wyświetlanie dodatkowych informacji w trybie verbose
R_ERROR - odpowiedzialne za wyświetlanie informacji błędu i zakończenie programu

Stałe programu i funkcje pomocnicze

MAX_LINE_WIDTH 2049: Stała określająca maksymalną szerokość wiersza.

WALL_CHAR 'X': Stała reprezentująca znak ściany.

UINT unsigned int: Typ danych UINT jako unsigned int.

USHORT unsigned short: Typ danych USHORT jako unsigned short.

UCHAR unsigned char: Typ danych UCHAR jako unsigned char.

TEMP_NODE_FILENAME "./temp/tempNodes.txt": Nazwa pliku dla tymczasowych węzłów.

QUEUE_FILENAME "./temp/queue.txt": Nazwa pliku dla kolejki.

GET_BIT(byteArr, n): Funkcja pobierająca bit o indeksie n z tablicy bajtów byteArr.

BYTE_TO_BINARY_PATTERN "%c%c%c%c%c%c%c%c": Wzorzec formatowania dla konwersji bajtów na ciągi binarne.

BYTE_TO_BINARY(byte): Funkcja konwertująca bajt byte na ciąg binarny.

PRINT_BINARY(byte): Funkcja drukująca bajt byte jako ciąg binarny.

Możliwe błędy

Lista możliwych błędów które zakończą się niepowodzeniem programu:
- Brak dostępu do pliku wejścia

- Brak podania pliku wejścia

- Brak wystarczająco miejsca na dysku

- Błędny format labiryntu

Lista możliwych błędów które zakończą się ostrzeżeniem programu: - Brak dostępu do podanego pliku wyjścia (program stworzy plik o innej nazwie w katalogu out)

- Brak wyjścia z labiryntu (program poinformuje nas o braku wyjścia z labiryntu)