DOKUMENTACJA PROJEKTOWA PROGRAMU ZNAJDUJĄCEGO NAJKRÓTSZĄ ŚCIEŻKĘ W LABIRYNCIE

Autorzy: Paweł Myszka, Tymon Piwowarski

7 kwietnia 2024

1 Funkcjonalność projektu

Głównym zadaniem programu jest odnalezienie najkrótszej trasy od wejścia do wyjścia z labiryntu i zapisanie jej w pliku wyjściowym w formie kolejnych kroków.
Program jako wejście musi przyjmować pliki tekstowe oraz binarne.
Wynikiem działania programu jest plik tekstowy bądź binarny (w przypadku binarnego plik zawiera również skompresowaną reprezentacje labiryntu). Format wszystkich plików jest ściśle określony w specyfikacji.

2 Pliki wykorzystywane przez program

2.1 pliki wejściowe:

- plik tekstowy - zawiera tekstową reprezentację labiryntu, w której 'X' oznacza ścianę, ' ' przejście, 'P' wejście, 'K' wyjście
- plik binarny - zawiera skompresowaną reprezentację labiryntu (kompresja dokonana przy użyciu RLE8)

2.2 pliki wyjściowe:

- plik tekstowy - zawiera kroki opisujące najkrótszą scieżkę w labiryncie oddzielone newlineami (postaci TURNRIGHT/TURNLEFT/FORWARD [LICZBA KROKÓW])
- plik binarny - zawiera skompresowaną reprezentację labiryntu oraz najkrótszą ścieżkę (postaci W/E/N/S [LICZBA KROKÓW])

2.3 pliki tymczasowe:

- graph.bin - plik binarny przechowujący labirynt w formie grafu (listy sąsiedztwa) gdzie kolejne pola numerowane są 0, 1, 2...n, każde pole ma 4 sąsiadów, a wartość -1 oznacza brak przejścia
- parent.bin - plik binarny przechowujący numer rodzica dla kolejnych pól, wypełniany jest podczas działania algorytmu bfs i wykorzystywanu przy rekonstrukcji scieżki
- path.bin - plik binarny, w którym przechowywana jest ścieżka w formie kolejnych indeksów pól, istotnym jest, że ścieżka przechowywana jest od końca
- queue.bin - plik binarny, który wykorzystywany jest do przechowywania cześci kolejki (używanej przy bfs) chwilowo nie mieszczącej się w pamięci ram
- lab.txt - plik tekstowy zawierający reprezentację labiryntu, tworzony w przypadku podania pliku wejściowego w formie binarnej w celu umożliwienia wykorzystania funckji działających na pliku tekstowym

3 Podział na moduły

bfs.c, file_io.c, file_vector.c, data.c, queue.c, main.c, metadata.h

3.1 bfs.c

Moduł zawierający funkcję traverse() będącą implementacją algorytmu bfs. Funkcje pomocnicze: reload_parent(), reload_graph() wykorzystywane są do wczytywania odpowiednich fragmentów labiryntu poczas eksploracji. Wynikiem działania modułu jest plik parent.bin.

3.2 file_io.c

Moduł obsługujący wejście i wyjście programu. Zawieraja funkcje path_to_binary(), path_to_txt() - wypisujące ścieżkę do plików odpowiednio binarnego i tekstowego, compress_lab_to_binary() - funkcja kompresująca reprezentację labiryntu i wypisująca ją do pliku binarnego, lab_info_txt(), lab_info_binary() - zbierające informacje z plików wejściowych, graph_to_bin_file() - znajdująca reprezentacje grafową labiryntu na podstawie pliku .txt

3.3 file_vector.c

Moduł zawierający zestaw funkcji wykorzystywanych do obsługi plików binarnych, w których przechowywane są wektory zmiennych typu int.

3.4 data.c

Moduł zawierający zbiór funkcji wykonujących operacje na danych, konwertujących ich na inny format. funkcja binary_to_txt() - konwertuje wejsciowy plik binarny na plik tekstowy, coords_to_node() - oblicza nr pola na podstawie ich koordynatow w pliku tekstowym.

3.5 queue.c

Moduł zawierający implementację kolejki, której część może być przechowywana w tymczasowym pliku binarnym.

3.6 main.c

Moduł główny wywołujący odpowiednie funkcje, zapewniający oczekiwane działanie programu.

3.7 metadata.h

Plik nagłówkowy, w którym przechowywane są stałe np. reprezentujące nazwy plików tymczasowych.

4 Wykorzystywane algorytmy

4.1 Algorytm bfs

Głównym algorytmem, który odpowiada za działanie programu jest bfs (breadth-first-search), czyli algorytmem przeszukiwania grafu "w szerz". Odwiedza on wierzchołki grafu w kolejności oddalenia od wierzchołka pierwszego.
Do tego wykorzystywana jest kolejka (w naszym przypadku cześciowo przechowywana w pliku tymczasowym), w której zapisywane są informacje o kolejnych wierchołkach do odwiedzenia.
Podczas działania algorytmu, przy dodawaniu wierzchołka do kolejki, w pliku parent.bin zapisywana jest informacja o poprzedniku (rodzicu) danego wierzchołka.
Program wykorzystuje funkcje reload_parent() oraz reload_graph() do wczytywania odpowiednich informacji o labiryncie z plików tymczasowych.

4.2 Algorytm zapisywania labiryntu w formie grafu

Algorytm ten pozwala na wykorzystanie przyjaźniejszej komputerowi reprezentacji labiryntu w formie grafu (listy sąsiedztwa).
Przechodzi on jednokrotnie po tekstowym pliku wejściowym i w przypadku napotkania znaku oznaczającego przejście między polami aktualizuje odpowiednią wartość w pliku binarnym.
Pole takie identyfikowane jest w sposób następujący: numerując od 0, jesli parzystość numeru linii jest inna niż parzystość numeru kolumny (w pliku .txt) to pole oznacza przejście.

4.3 Algorytm znajdowania informacji o labiryncie z pliku .txt

Algorytm ten przechodzi po pliku wejściowym dwukrotnie.
Za pierwszym razem znajduje on rozmiar pliku (liczbę wierszy i kolumn). Informacja takie konieczna jest do identyfikacji wejścia i wyjścia z pliku. Podczas tego przejścia weryfikowana jest również poprawność formatu pliku.
Za drugim razem algorytm identyfikuje współrzędne wejścia i wyjścia.

4.4 Algorytm wypisywania ścieżki

Na początku kolejno odwiedzane pola zapisywane są w pliku path.bin.
Zaczynając od wyjścia algorytm przechodzi po kolejnych wierzchołkach ustawiając **aktualny wierzchołek = rodzic aktualnego wierchołka**. Skutkuje to odwróconą kolejnościa ścieżki.
Następnym etapem działania algorytmu jest przekształcenie ciągu liczb reprezentujących kolejne pola na zrozumiałe dla człowieka kroki.
Przechodząc po kolejnych polach, algorytm oblicza wektor przesunięcia i jeżeli jest on różny od poprzedniego - nastąpiła zmiana kierunku. W takim wypadku wypisuje zapisaną aktualnie liczbę kroków i zeruje ją.