

Dokumentacja systemu do przetwarzania labiryntów

Arseni Kiyko

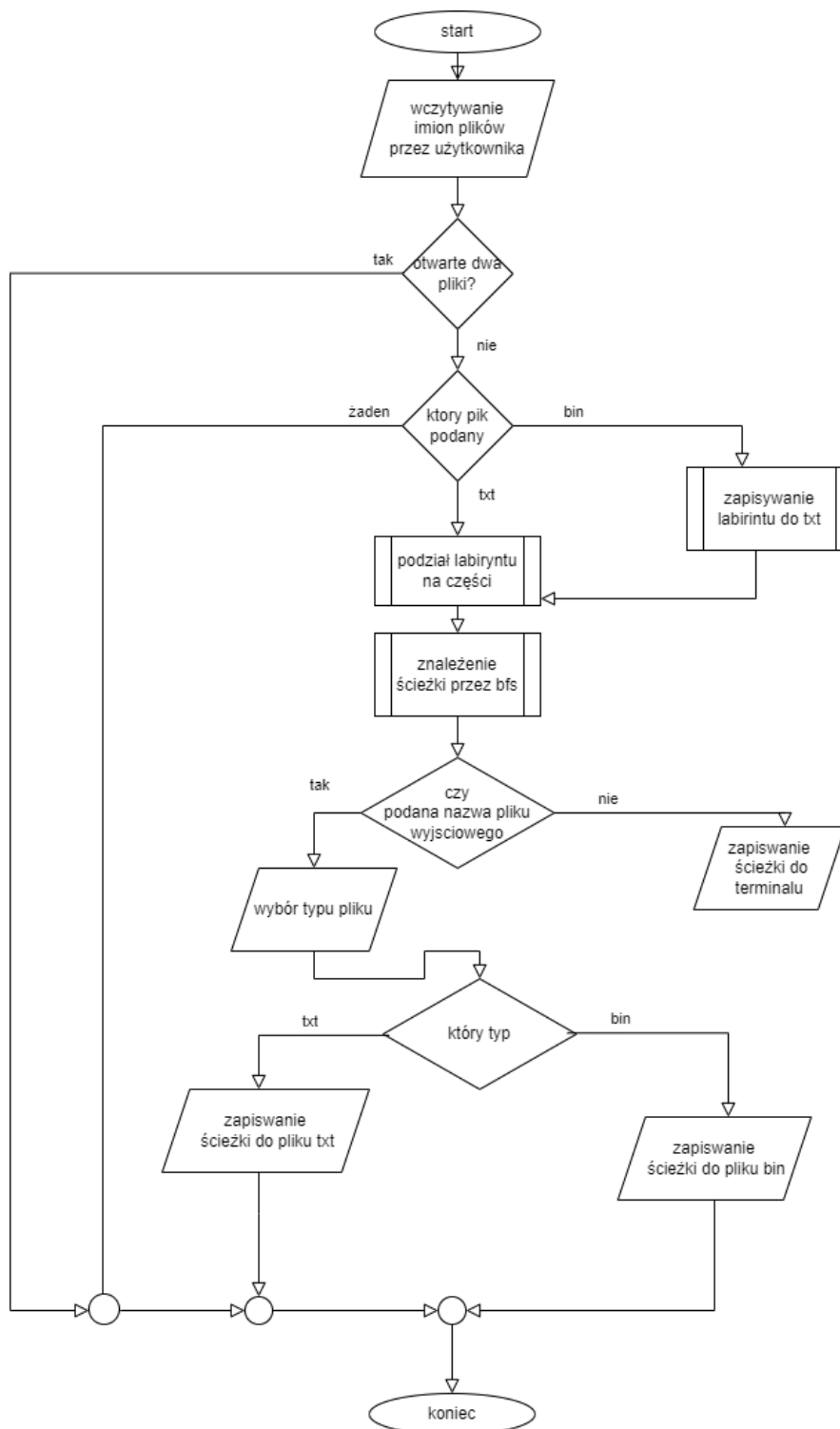
12 czerwca 2024

Streszczenie

Ten dokument zawiera kompleksowy opis systemu do przetwarzania labiryntów, włączając w to opis funkcjonalności, dane wejściowe i wyjściowe, ograniczenia systemu oraz szczegółowy opis poszczególnych modułów. Została również dodana sekcja dotycząca schematów algorytmów używanych w programie do analizy i rozwiązywania labiryntów.

1 Opis ogólny

Program rozpoczyna działanie od odczytu pliku, który zawiera dane labiryntu. Dane te mogą być reprezentowane w formacie tekstowym lub binarnym. Program analizuje plik, sprawdzając jego zgodność z określonymi wymaganiami formatowymi.



Rysunek 1: Schemat działania algorytmu

2 Główne etapy przetwarzania

Wczytywanie i walidacja pliku:

Użytkownik wskazuje ścieżkę do pliku z danymi labiryntu za pomocą argumentów linii poleceń. Program sprawdza obecność pliku i jego poprawność za pomocą funkcji z modułu `file_loading`.

Przetwarzanie danych labiryntu:

Jeśli plik labiryntu przekracza 50 linii, jest dzielony na części za pomocą modułu `msplit`. Dla plików binarnych używana jest specjalna obróbka przez moduł `msplitbin`, która obejmuje konwersję danych do formatu tekstowego.

Szukanie ścieżki:

Program wykorzystuje algorytm przeszukiwania wszerz (BFS), zaimplementowany w module `mbfs`, do znalezienia najkrótszej ścieżki od wejścia do wyjścia labiryntu. W trakcie przeszukiwania program uwzględnia strukturę labiryntu, omijając ściany i inne przeszkody.

Formatowanie i wyświetlanie wyników:

Wyniki poszukiwań ścieżki są formatowane i mogą być wyświetlane w pliku tekstowym lub binarnym, w zależności od określonych przez użytkownika parametrów. Ta funkcjonalność jest realizowana w module `mwyjscie`. Użytkownik może wybrać, gdzie i w jakim formacie zapisać wynik (na ekranie, w pliku tekstowym, w pliku binarnym).

Zarządzanie katalogami i plikami:

Program zarządza plikami tymczasowymi i katalogami za pomocą modułu `mdir`, który umożliwia tworzenie potrzebnych katalogów i usuwanie ich zawartości po zakończeniu działania programu.

3 Opis algorytmu

W tej części dokumentacji przedstawiono szczegółowy opis algorytmu używanego do przeszukiwania i rozwiązywania labiryntów. Program stosuje algorytm przeszukiwania wszerz (BFS - Breadth-First Search), który jest idealnie dopasowany do znajdowania najkrótszej ścieżki w strukturze takiej jak labirynt.

3.1 Przeszukiwanie wszerz (BFS)

Algorytm BFS rozpoczyna przeszukiwanie od zadanego punktu startowego (wejście labiryntu) i eksploruje wszystkie sąsiednie komórki na tym samym poziomie przed przejściem do komórek na następnym poziomie głębokości. To podejście gwarantuje, że jeśli istnieje rozwiązanie, BFS znajdzie najkrótszą możliwą ścieżkę do wyjścia labiryntu.

3.1.1 Proces algorytmu

Algorytm BFS wykorzystuje kolejkę do śledzenia wszystkich lokacji w labiryncie do przeszukania:

1. Zainicjuj kolejkę włożeniem punktu startowego labiryntu.
2. Dopóki kolejka nie jest pusta, wykonuj:
 - Usuń element z przodu kolejki.
 - Oznacz element jako odwiedzony.
 - Dla każdego nieodwiedzonego sąsiada odwiedzonego elementu:
 - Jeśli jest to wyjście, zakończ przeszukiwanie.
 - Inaczej, dodaj sąsiada do kolejki.

4 Używanie programu

Kompilacja programu

Aby skompilować program, użyj polecenia:

```
make
```

To polecenie spowoduje zbudowanie projektu zgodnie z instrukcjami określonymi w Makefile i stworzy plik wykonywalny `program`.

Uruchamianie programu

Program wykorzystuje argumenty linii poleceń do sterowania wejściem i wyjściem. Oto przykłady użycia:

Uruchomienie z wyświetleniem pomocy:

```
./maze_solver -h
```

Wyświetli informacje o tym, jak używać programu, w tym opis dostępnych argumentów.

Przetwarzanie pliku tekstowego labiryntu:

```
./program -f labyrinth.txt -o solution
```

Tutaj `labyrinth.txt` to plik labiryntu do przetworzenia, a `solution.txt` to plik do zapisania wyniku.

Przetwarzanie binarnego pliku labiryntu:

```
./program -b labyrinth.bin -o solution
```

5 Dane wejściowe

Program akceptuje na wejściu:

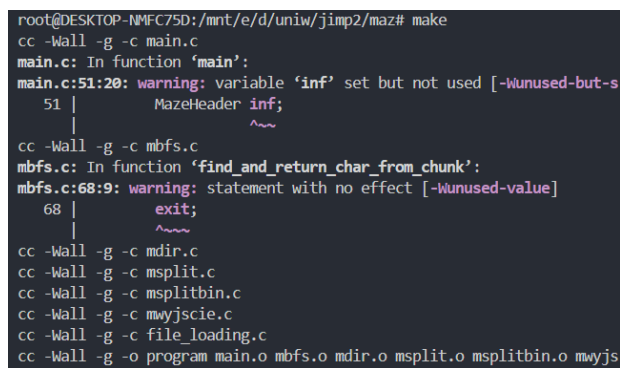
- Nazwę pliku z labiryntem w formacie tekstowym (*.txt) lub binarnym (*.bin).
- Opcjonalne parametry do kontrolowania wyjścia wyników lub do wyświetlania pomocy (-h).

6 Dane wyjściowe

Na wyjściu program może generować:

- Plik tekstowy z rozwiązaniem labiryntu, pokazujący sekwencję kroków do osiągnięcia wyjścia.
- Plik binarny z rozwiązaniem.
- Wyjście na konsolę do bezpośredniego przeglądania wyników.

przykłady



```
root@DESKTOP-NMFC75D:/mnt/e/d/uniw/jimp2/maz# make
cc -Wall -g -c main.c
main.c: In function 'main':
main.c:51:20: warning: variable 'inf' set but not used [-Wunused-but-s
51 |         MazeHeader inf;
    |
cc -Wall -g -c mbfs.c
mbfs.c: In function 'find_and_return_char_from_chunk':
mbfs.c:68:9: warning: statement with no effect [-Wunused-value]
68 |         exit;
    |
cc -Wall -g -c mdir.c
cc -Wall -g -c msplit.c
cc -Wall -g -c msplitbin.c
cc -Wall -g -c mwyjscie.c
cc -Wall -g -c file_loading.c
cc -Wall -g -o program main.o mbfs.o mdir.o msplit.o msplitbin.o mwyjs
```

Rysunek 2: kompilacja programu

```

root@DESKTOP-NMFC75D:/mnt/e/d/uniw/jimp2/maze# ./program -f maze1.txt
Directory content deleted successfully
Directory content deleted successfully
Directory content deleted successfully
Całkowita liczba fragmentów: 1
Całkowita liczba linii: 21
Długość linii: 21
Współrzędne 'P': (1, 0)
Współrzędne 'K': (19, 20)
Loaded chunk 1/1
(19, 20)
START
FOWARD 3
TURNRIGHT
FOWARD 2
TURNLEFT
FOWARD 4
TURNLEFT
FOWARD 2
TURNRIGHT
FOWARD 2
TURNRIGHT
FOWARD 2
TURNLEFT

```

Rysunek 3: wyjście do terminalu

7 Ograniczenia

Labirynt musi być przedstawiony w akceptowalnym formacie, z wyraźnie określonymi symbolami dla ścian (X), przejść (), wejścia (P) i wyjścia (K). Wymiary labiryntu nie mogą przekraczać 1024*1024 komórek z powodu ograniczeń pamięci i wydajności.

8 Moduły programu

Program składa się z kilku modułów:

1. file_loading

Opis: Moduł file_loading odpowiada za wczytywanie i wstępną walidację plików labiryntu. Zapewnia funkcje do czytania plików, sprawdzania ich obecności oraz weryfikacji zgodności z oczekiwanym formatem.

Funkcje:

- `process_input(int argc, char *argv[], char **input_filename_bin, char **input_filename, char **output_filename)`: Przetwarza argumenty linii poleceń, identyfikując źródła danych wejściowych i ścieżkę dla wyników wyjściowych.
- `file_exists(const char *filename)`: Sprawdza, czy istnieje plik tekstowy.
- `file_exists_bin(const char *filename)`: Sprawdza, czy istnieje plik binarny.
- `is_valid_maze(const char *filename)`: Waliduje format pliku tekstowego labiryntu.
- `is_valid_binary_maze(const char *filename)`: Waliduje format pliku binarnego labiryntu.

Obsługa błędów:

- Brak pliku.

- Nieudane otwarcie pliku.
- Nieprawidłowy format pliku.

2. mdir

Opis: Moduł `mdir` zarządza katalogami używanymi przez program do przechowywania tymczasowych i wynikowych plików.

Funkcje:

- `create_directory(const char *path)`: Tworzy nowy katalog.
- `delete_directory_content(const char *path)`: Usuwa zawartość katalogu.
- `if_dir(const char *dir_name)`: Sprawdza istnienie katalogu i czyści go, jeśli już istnieje.

Obsługa błędów:

- Błędy dostępu do systemu plików.
- Błędy przy tworzeniu lub usuwaniu katalogów.

3. msplit

Opis: Moduł `msplit` odpowiada za podział pliku labiryntu na mniejsze części, co jest niezbędne przy dużym rozmiarze labiryntu.

Funkcje:

- `parse_maze(const char *file_path, Position *posP, Position *posK, int *rows, int *cols, int *num_chunks)`: Dzieli plik labiryntu na części, zapisując każdą część do osobnego pliku.

Obsługa błędów:

- Błędy odczytu pliku.
- Niezgodność rozmiarów linii w labiryncie.

4. mbfs

Opis: Moduł `mbfs` implementuje algorytm przeszukiwania wszerek (Breadth-First Search, BFS) do określenia najkrótszej ścieżki przez labirynt.

Funkcje:

- `bfs(int rows, int cols, int start_row, int start_col, int16_t chunk_row_counter, int how_many_chunks)`: Realizuje algorytm BFS w celu znalezienia ścieżki.
- `create_queue(int capacity), enqueue(Queue *queue, int row, int col), dequeue(Queue *queue), is_queue_empty(Queue *queue)`: Funkcje zarządzania kolejką dla BFS.

Obsługa błędów:

- Przekroczenie granic tablicy.
- Nieoczekiwane symbole w labiryncie oznaczające ściany lub zajęte trasy.

5. mwyjście

Opis: Moduł `mwyjście` formatuje i prezentuje wyniki pracy algorytmu w pliku lub na ekranie.

Funkcje:

- `txt_compress(const char *input_file, int num_letters, FILE *final_fp)`: Formatuje wyjście ścieżki w labiryncie, kompresując sekwencje kroków.

Obsługa błędów:

- Błędy otwarcia plików.
- Błędy zapisu do pliku.

6. msplitbin

Opis: Moduł `msplitbin` zapewnia obróbkę binarnych plików labiryntu, konwertując je na format tekstowy do dalszego przetwarzania.

Funkcje:

- `read_bin_file(char *filepath)`: Czyta binarny plik i konwertuje go na format tekstowy.
- `write_code_world_to_file(...)`: Zapisuje dane do pliku zgodnie z określonym formatem.

Obsługa błędów:

- Błędy odczytu binarnego pliku.
- Nieprawidłowy format danych w binarnym pliku.

9 Zakończenie

Dokumentacja ta dostarcza kompleksowego przeglądu funkcjonalności i struktury programu do przetwarzania labiryntów. Przez odpowiednią modularność, program jest elastyczny i efektywny w przetwarzaniu różnych formatów danych oraz w obsłudze różnorodnych scenariuszy użytkowania. Użytkownicy mogą dostosować działanie programu do swoich potrzeb, korzystając z różnorodnych opcji wejścia i wyjścia, które program oferuje.