Dokumentacja Projektowa
Projektu Labirynt

Polina Nesterova
01192875@pw.edu.pl

Martyna Kochalska
01187553@pw.edu.pl

04 kwietnia 2024

1 Opis funcjonalności

Program umożliwia znalezienie najkrótszej ścieżki labiryncie przy ograniczeniu użycia do 512kB pamięci. Mapę labiryntu dostaje w 2 formach, tekstowej gdzie jeden znak jest ścianą albo przejściem, albo binarnej, która korzysta z opisanego w sekcji "Opis plików" sposobu zapisu. Dodatkowo program umożliwia wykonanie z dodatkowymi parametrami, takimi jak:

- `-f [input filename]` - parametr umożliwający podanie pliku wejściowego z labiryntem. W przypadku jego braku labirynt zostaje rozwiązany dla pliku przykładowego.
- `-o [output filename]` - dla plików txt: parametr przekierowywujący ścieżkę programu z konsoli do pliku, dla plików bin: parametr zmieniający miejsce zapisu najkrótszej ścieżki z pliku inputowanego na plik podany (program kopiuje wcześniejszą część pliku wejściowego i dodaje do niego wynik działania programu).
- `-h` - wyświetlenie pomocy

2 Opis plików wykorzystywanych przez program

- `Pliki Wejściowe` – program korzysta z plików zawierających informacje o labiryncie. Możliwe są dwa rodzaje plików wejściowych: tekstowy (z rozszerzeniem '.txt') oraz binarny (.bin).

  Plik tekstowy zapisany powinien być za pomocą znaków 'X', ' ', 'P' i 'K', gdzie znaki te oznaczają kolejno ścianę, przejście, początek i koniec labiryntu.

  Plik binarny składa się z 4 głównych sekcji:

  1. Nagłówka pliku

2. Sekcji kodująca zawierająca powtarzające się słowa kodowe
3. Nagłówka sekcji rozwiązania
4. Sekcji rozwiązania zawierające powtarzające się kroki które należy wykonać aby wyjść z labiryntu

Dokładny opis pliku binarnego jest możliwy do pobrania tutaj: https://isod.ee.pw.edu.pl/

- `Pliki Wyjściowe` – program generuje plik wyjściowy, w którym zapisywane są wyniki działania programu. Taki plik jest opcjonalny i jego nazwa może być podana przez użytkownika jako argument wywołania.
- `Pozostałe pliki` - Program tworzy pliki półtymczasowe przechowywujące podzielone fragmenty labiryntu w formie plików tekstowych. Pliki te przechowywane są w podfolderze projektu /chunks, który to jest czyszczony przy każdorazowym uruchomieniu programu. Dodatkowo w tym folderze tworzy pojedyczny plik path.txt, pełniący funkcję stosu dla ścieżek.

3 Podział na moduły

Program składa się kolejno z modułów:

1. `File loading` - wczytywanie plików do programu oraz sprawdzenie poprawności ich formatu.
2. `Txt chunks handling` - Podział pliku txt na fragmenty, wygenerowanie plików tekstowych.
3. `Bin chunks handling` - Zdekodowanie pliku binarnego, wygenerowanie plików kompatybilnych (tekstowych) z resztą programu.
4. `Pathfinding` - Znalezienie najkrótszej ścieżki w labiryncie.
5. `Saving and encoding` - Zapis ścieżki do pliku, przepisanie labiryntu na wymaganą formę dla pliku binarnego.

4 Opis działania modułów

- `Moduł 1 - File loading`

  Główne funkcjonalności:

  1. Wyświetlenie informacji o użyciu programu
  2. Wczytanie plików wejściowych oraz wyjściowych
  3. Sprawdzenie poprawności formatu pliku (zarówno tekstowego, jak i binarnego)

  Funkcje załączone w pliku nagłówkowym:

– void process_input(int argc, char *argv[], char **input_filename, char **output_filename) – przetwarza argumenty podane przy uruchamianiu programu, w tym nazwę pliku wejściowego i opcjonalnie pliku wyjściowego za pomocą getopta. Sprawdza poprawność argumentów i formatu plików.

  Obsługa błędów charakterystycznych dla funkcji:

  * Przy wykryciu niepoprawnych argumentów: wyświetla komunikat help oraz kończy program.

– bool is_valid_maze_format(const char *filename) - Funkcja dla sprawdzenia poprawności labiryntu zawartego w pliku tekstowym. Jest odpowiedzialna za: sprawdzenie czy wszystkie linie mają tą samą długość, Czy granice labiryntu składają się z 'X'Liczenie liczby wejść (P' i wyjść 'K' w labiryncie), czy jest dokładnie jedno wejście i wyjście w labiryncie.

  Obsługa błędów charakterystycznych dla funkcji:

  * Przy wykryciu niepoprawności labiryntu: wyświetla komunikat mówiący o błędzie oraz kończy program.
  * Nieprawidłowy format labiryntu: wyświetla komunikat mówiący o błędzie oraz kończy program.
  * Nieprawidłowy format pliku wejściowego (inny niż .txt/.bin): wyświetla komunikat mówiący o błędzie oraz kończy program.
  * Nieprawidłowa długość kolumn / wierszy: wyświetla komunikat mówiący o błędzie oraz kończy program.
  * Niekonsystentna długość linii w labiryncie: wyświetla komunikat mówiący o błędzie oraz kończy program.
  * Plik nie składa się wyłącznie ze spacji, Xsów, oraz jednego "P" i "K": wyświetla komunikat mówiący o błędzie oraz kończy program.

– bool is_valid_binary_maze_format(const char *filename) Ta funkcja jest analogiczna do 'is_valid_maze_format', ale działa dla plików binarnych. Sprawdza, czy plik jest plikiem binarnym. Dokonuje to poprzez odczytanie każdego bajtu z pliku i sprawdzenie, czy znajduje się on w zakresie od 0x07 do 0x7E. Jeśli jakikolwiek znak w pliku nie mieści się w tym zakresie, plik jest traktowany jako binarny. Zwraca true jeśli warunki są spełnione, w przeciwnym razie false.

  Obsługa błędów charakterystycznych dla funkcji:

  * Przy wykryciu niepoprawności labiryntu: wyświetla komunikat mówiący o błędzie oraz kończy program.
  * Nieprawidłowy format labiryntu: wyświetla komunikat mówiący o błędzie oraz kończy program.
  * Nieprawidłowy format pliku wejściowego (inny niż .txt/.bin): wyświetla komunikat mówiący o błędzie oraz kończy program.

    - Nieprawidłowa długość kolumn / wierszy: wyświetla komunikat mówiący o błędzie oraz kończy program.
    - Plik nie składa się z założonych sekcji: : wyświetla komunikat mówiący o błędzie oraz kończy program.

- `Moduł 2 - Txt chunks handling`

  Główne funkcjonalności:

  1. Wczytanie pliku wejściowego po części do pamięci
  2. Podział pliku na segmenty i ich zapis do folderu /chunks
  3. Wczytywanie chunku do pamięci

  Funkcje załączone w pliku nagłówkowym:

  – int txt_file_to_txt_chunks(char *filepath, int16_t col, int16_t row, int16_t chunk_rows_counter) - funkcja otrzymująca ścieżkę do pliku wejściowego, dzieląca plik wejściowy na segmenty, zapisująca je w formacie [numer].txt, zwracająca ilość wygenerowanych chunków w folderze chunk.

    Obsługa błędów charakterystycznych dla funkcji:

    - Przy wykryciu błędu (standardowe błędy opisane niżej) zwraca -1

  – char **read_txt_chunk(char *filepath, int16_t col, int16_t row, int16_t chunk_row_size) - funkcja otrzymująca ścieżkę do chunku razem z jego paramterami, zwracająca go w formie tablicy dwuwymiarowej.

- `Moduł 3 - Bin chunks handling`

  Główne funkcjonalności:

  1. Odczyt danych zapisanych w pliku wejściowym binarnym
  2. Konwersja formy pliku binarnego na mapę tekstową złożoną ze spacji, X'ów, "K" i "P".
  3. Podział pliku na segmenty i ich zapis w folderze /chunks

  Funkcje załączone w pliku nagłówkowym:

  – int bin_file_to_txt_chunks(char *filepath, int16_t col, int16_t row, int16_t chunk_rows_counter) funkcja otrzymująca ścieżkę do pliku wejściowego, zmienijąca formę pliku na tą z pliku tekstowego, zwracająca ilość wygenerowanych chunków w folderze /chunks

    Obsługa błędów charakterystycznych dla funkcji:

    - Przy wykryciu błędu (standardowe błędy opisane niżej) zwraca -1

- void delete_files_in_directory(const char *directory_path); funkcja optrzymująca ścieżkę do folderu który ma wyczyścić, sprawdzająca jego istnienie.

  Obsługa błędów charakterystycznych dla funkcji:

  * Błąd z otworzeniem/znalezniem folderu: wyświetlenie komunikatu o błędzie i zakończenie programu.
  * Błąd z usunięciem plików: wyświetlenie komunikatu o błędzie i zakończenie programu.

- `Moduł 4 - Pathfinding`

  Główne funkcjonalności:

  1. Przeskanowanie labiryntu w poszukiwaniu ścieżek
  2. Znalezienie najkrótszej ścieżki i wypisanie jej.

  Funkcje załączone w pliku nagłówkowym:

  - void solve_maze(int chunk_counter, int16_t col, int16_t row, int16_t chunk_rows_counter) - funkcja sterująca całym dalszym rozwiązaniem labiryntu. W argumencie dostaje liczbę wygenerowanych uprzednio chunków oraz informacje na temat chunkow.

- `Moduł 5 - Saving and encoding`

  Główne funkcjonalności:

  1. Przekierowanie rozwiązania do pliku tekstowego w przypadku odpowiedniego argumentu wejściowego
  2. Zakodowanie rozwiązanie w prawidłowym formacie dla pliku binarego oraz zapis.

  Funkcje załączone w pliku nagłówkowym:

  - void save_shortest_path(bool is_o, char **output_filename) - funkcja obsługująca wypisanie/zapisanie najkrótszej ścieżki. W zależności od otrzymanych argumentów zapisuje ścieżkę w pliku tekstowym albo wypisuje ją na konsolę.
  - void bin_file_saving(char **input_filename, char **output_filename, bool is_o,) - funckja dopisująca najkrótszą ścieżkę do pliku binarnego w odpowiednim formacie, w zależności od otrzymanego argumentu: dopisuje na końcu pliku wejściowego, albo kopiuje plik wejściowy do podanego pliku wyjściowego i do niego załącza rozwiązanie.

    Obsługa błędów charakterystycznych dla funkcji:

    * Błąd podczas kopiowania pliku: wyświetlenie komunikatu o błędzie i zakończenie programu.

Dodatkowo program powinien obsługiwać błędy dla ogólnych przypadków, takich jak:

1. Brak możliwości otworzenia pliku: wyświetlenie komunikatu mówiącego o błędzie oraz zakończenie programu.
2. Problem z zaalokowaniem pamięci: wyświetlenie komunikatu mówiącego o błędzie oraz zakończenie programu.

5 Opis wykorzystywanych algorytmów

Do znalezienia najkrótszej ścieżki wykorzystywany jest w programie algorytm Djikstry. Algorytm ten znajduje najkrótsze ścieżki w grafie o nieujemnych wagach krawędzi. Jest to algorytm zachłanny, który wybiera najbliższy wierzchołek w każdej iteracji, co prowadzi do znalezienia najkrótszej ścieżki do wszystkich pozostałych wierzchołków w grafie.

Algorytm ten można opisać za pomocą listy kroków:

1. S $\leftarrow$ Ø (zbiór S ustawiamy jako pusty)
2. Q $\leftarrow$ wszystkie wierzchołki grafu
3. Utwórz n elementową tablicę d (tablica na koszty dojścia)
4. Utwórz n elementową tablicę p (tablica poprzedników na ścieżkach)
5. Tablicę d wypełnij największą wartością dodatnią
6. d [v] $\leftarrow$ 0 (koszt dojścia do samego siebie jest zawsze zerowy)
7. Tablicę p wypełnij wartościami -1 (-1 oznacza brak poprzednika)
8. Dopóki Q zawiera wierzchołki, wykonuj kroki K09. . . K12
9. Z Q do S przenieś wierzchołek u o najmniejszym d [u]
10. Dla każdego sąsiada w wierzchołka u :wykonuj kroki K11. . . K12 (przeglądamy sąsiadów przeniesionego wierzchołka)
11. Jeśli w nie jest w Q, to następny obieg pętli (szukamy sąsiadów obecnych w Q)
12. Jeśli d [w] $>$ d [u] + waga krawędzi u – w, to:

    d [w] $\leftarrow$ d [u] + waga krawędzi u – w

    p [w] $\leftarrow$ u

    (sprawdzamy koszt dojścia. Jeśli mamy niższy, to modyfikujemy koszt i zmieniamy poprzednika w na u)
13. Zakończ