

DOKUMENTACJA PROJEKTOWA PROGRAMU ZNAJDUJĄCEGO NAJKRÓTSZĄ ŚCIEŻKĘ W LABIRYNCIE

Autorzy: Paweł Myszka, Tymon Piwowarski

8 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 reprezentację 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)

```
XXXXXXXXXXXXXXXXXXXXX
P                      X
XXX X  XXX  XXXXX X
X  X      X      X
X XXX X  XXX  XXX X
X X      X      X X
X XXX X X  XXX  XXX
X  X      X      X
XXXXX  XXXXX X X X
X                      X X
X XXX  XXX X  XXX X
X                      X
X X  XXXXX  XXX X X
X X      X      X X X
X X  XXX  XXX X X X
X                      X K
XXXXXXXXXXXXXXXXXXXXX
```

Rysunek 1: przykład pliku wejściowego .txt

Dane wejściowe w formacie binarnym podzielone są na 4 główne sekcje:

1. Nagłówek pliku
2. Sekcja kodująca zawierająca powtarzające się słowa kodowe
3. Nagłówek sekcji rozwiązania
4. Sekcja rozwiązania zawierająca powtarzające się kroki które należy wykonać aby wyjść z labiryntu

Sekcja 1 i 2 są obowiązkowe i zawsze występują, sekcja 3 oraz 4 są opcjonalne. Występują jeśli wartość pola *Solution Offset* z nagłówka pliku jest różna od 0.

Nagłówek pliku:

Nazwa pola	Wielość w bitach	Opis
File Id	32	Identyfikator pliku: 0x52524243
Escape	8	Znak ESC: 0x1B
Columns	16	Liczba kolumn labiryntu (numerowane od 1)
Lines	16	Liczba wierszy labiryntu (numerowane od 1)
Entry X	16	Współrzędne X wejścia do labiryntu (numerowane od 1)
Entry Y	16	Współrzędne Y wejścia do labiryntu (numerowane od 1)
Exit X	16	Współrzędne X wyjścia z labiryntu (numerowane od 1)
Exit Y	16	Współrzędne Y wyjścia z labiryntu (numerowane od 1)
Reserved	96	Zarezerwowane do przyszłego wykorzystania
Counter	32	Liczba słów kodowych
Solution Offset	32	Offset w pliku do sekcji (3) zawierającej rozwiązanie
Separator	8	słowo definiujące początek słowa kodowego – mniejsze od 0xF0
Wall	8	słowo definiujące ścianę labiryntu
Path	8	słowo definiujące pole po którym można się poruszać
Podsumowanie	420	Sumarycznie nagłówek ma rozmiar 40 bajtów

Słowa kodowe:

Nazwa pola	Wielość w bitach	Opis
Separator	8	Znacznik początku słowa kodowego
Value	8	Wartość słowa kodowego (Wall / Path)
Count	8	Liczba wystąpień (0 – oznacza jedno wystąpienie)

Sekcja nagłówkowa rozwiązania

Nazwa pola	Wielość w bitach	Opis
Direction	32	Identyfikator sekcji rozwiązania: 0x52524243
Steps	8	Liczba kroków do przejścia (0 – oznacza jeden krok)

Krok rozwiązania:

Nazwa pola	Wielość w bitach	Opis
Direction	8	Kierunek w którym należy się poruszać (N, E, S, W)
Counter	8	Liczba pól do przejścia (0 – oznacza jedno pole)

Pola liczone są bez uwzględnienia pola startowego.

Rysunek 2: format pliku wejściowego .bin

2.2 pliki wyjściowe:

- plik tekstowy - zawiera kroki opisujące najkrótszą ścieżkę 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])

```
1 FORWARD 3
2 TURNRIGHT
3 FORWARD 1
4 TURNLEFT
5 FORWARD 4
6 TURNRIGHT
7 FORWARD 1
8 TURNLEFT
9 FORWARD 1
10 TURNRIGHT
11 FORWARD 1
12 TURNLEFT
13 FORWARD 2
14 TURNLEFT
15 FORWARD 1
16 TURNLEFT
17 FORWARD 1
18 TURNRIGHT
19 FORWARD 1
20 TURNRIGHT
```

Rysunek 3: format pliku wejściowego .bin

Uwaga: format pliku wyjściowego binarnego zawarty w powyżej w opisie wejściowego pliku .bin

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, dane przechowywane są w pliku jako ciąg int-ów
- **parent.bin** - plik binarny przechowujący numer rodzica dla kolejnych pól, wypełniany jest podczas działania algorytmu bfs i wykorzystywany przy rekonstrukcji ścieżki, dane przechowywane są w pliku jako ciąg int-ów
- **path.bin** - plik binarny, w którym przechowywana jest ścieżka w formie kolejnych indeksów pól, istotnym jest, że ścieżka przechowywana jest od końca, dane przechowywane są w pliku jako ciąg int-ów
- **queue.bin** - plik binarny, który wykorzystywany jest do przechowywania części kolejki (używanej przy bfs) chwilowo nie mieszczącej się w pamięci ram, dane przechowywane są w pliku jako ciąg int-ów
- **lab.txt** - plik tekstowy zawierający reprezentację labiryntu, tworzony w przypadku podania pliku wejściowego w formie binarnej w celu umożliwienia wykorzystania funkcji działających na pliku tekstowym, format pliku adekwatny do formatu pliku wejściowego .txt

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.

Nagłówki funkcji:

```
int traverse(int, int, point_t);
```

wynikiem jej działania jest 0, gdy ścieżka została znaleziona, 1 gdy ścieżki brak.

3.2 file_io.c

Moduł obsługujący wejście i wyjście programu. Zawierają 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()` - znajdującą reprezentację grafową labiryntu na podstawie pliku `.txt`

Nagłówki funkcji:

```
int lab_info_txt(char*, point_t*, point_t*, point_t*, int*);
```

```
int lab_info_binary(char*, point_t*, point_t*, point_t*);
```

```
int path_to_txt(char*, int, int, point_t, int);
```

```
void path_to_binary(char*, int, int, point_t);
```

```
int compress_lab_to_binary(char*, char*, point_t, point_t, point_t);
```

lab_info_txt() i **lab_info_binary** zwracają 0 gdy wszystko jest ok, 1 gdy nie można czytać pliku wejściowego, 2 - gdy format pliku wejściowego jest niepoprawny
path_to_txt() i **compress_lab_to_binary()** zwracają 1 gdy nie można czytać pliku wyjściowego, 0 gdy wszystko ok.

3.3 file_vector.c

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

Nagłówki funkcji:

```
void init_file_vector(char*, int, int);
```

```
int* read_file_vector(char*, int, int);
```

```
void update_file_vector(char*, int, int);
```

```
int read_file_position(char*, int);
```

```
void delete_temp_files(char**, int);
```

`read_file_vector()` zwraca wskaźnik do wektora wczytanych intów

`read_file_position()` zwraca wartość wczytanego inta

3.4 data.c

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

Nagłówki funkcji:

```
int coords_to_node(point_t, point_t);
void binary_to_txt(char*, char*, point_t);
void graph_to_bin_file(char*, point_t);
```

`coords_to_node()` zwraca inta będącego nr wierzchołka

3.5 queue.c

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

Nagłówki funkcji:

```
Queue_t* init_queue(int, int);
void push(Queue_t*, int);
int pop(Queue_t*);
void display(Queue_t*);
void destroy_queue(Queue_t* queue);
```

```
/* struktura zawierajaca informacje
   o elemencie kolejki */
typedef struct node{
    int value;
    struct node* next;
} node_t;

typedef struct{
    int internal_size; /* rozmiar kolejki w pamieci RAM */
    int external_size; /* rozmiar kolejki w pamieci ROM */
    int internal_capacity; /* pojemnosc kolejki w pamieci RAM */
    int external_capacity; /* pojemnosc kolejki w pamieci ROM */
    int external_offset; /* indeks pierwszego elementu w pamieci ROM */
    node_t* top;
} Queue_t;
```

Rysunek 4: struktury zaimplementowane w module queue

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 częściowo przechowywana w pliku tymczasowym), w której zapisywane są informacje o kolejnych wierzchołkach 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, jeśli 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

Plik tekstowy:

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.

Plik binarny:

Dane wczytywane są według specyfikacji pliku wejściowego załączonego powyżej.

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 wierzchołka**. Skutkuje to odwróconą kolejnością 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.

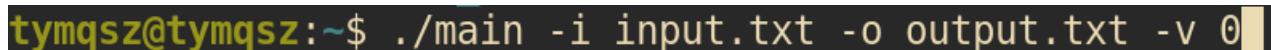
Do pliku binarnego dane wypisywane są według formatu opisanego powyżej (przy pliku wejściowym).

5 Sposób wywołania

Do kompilacji programy wykorzystywany jest plik makefile.

Aby uruchomić program należy wywołać plik main podając argumenty:

```
-i [plik wejścia]  
-o [plik wyjścia]  
-v [0/1] (verbose)
```



```
tymqsz@tymqsz:~$ ./main -i input.txt -o output.txt -v 0
```

Rysunek 5: przykład uruchomienia programu

6 Obsługa sytuacji wyjątkowych

- folder "input" nie istnieje - kod błędu 13
- brak podanej nazwy pliku wejściowego jako argumentu - kod błędu 17
- brak możliwości utworzenia pliku wyjściowego - kod błędu 69
- brak możliwości otwarcia pliku wejściowego - kod błędu 2137
- niepoprawny format pliku wejściowego - kod błędu 19
- brak istnienia ścieżki w labiryncie - kod błędu 11