

Dokumentacja projektowa w C

Mateusz Czarnecki, Maciej Wieteska

9 maja 2024

1 GITHUB

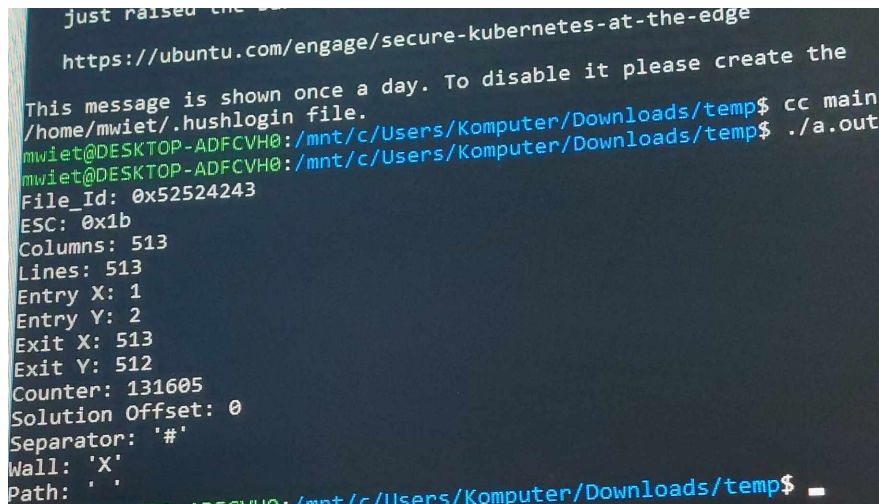
<https://github.com/MaciekWieteska/CrazyMaze.git>

2 Opis funkcjonalności

Podstawową funkcjonalnością projektu jest znajdowanie najkrótszej ścieżki labiryntu. Oprócz pliku wejściowego w formacie txt program obsługuje też labirynt zakodowany binarnie według standardu zbliżonego do RLE 8. Wynikiem działania programu jest plik tekstowy z listą wykonanych kroków. Program bazuje na czytaniu plików tekstowych i zapisywaniu przetworzonych informacji w tymczasowych plikach tekstowych aż do momentu uzyskania pliku końcowego wynik.txt. Generator labiryntu w formacie tekstowym znajduje się na stronie <http://tob.iem.pw.edu.pl/maze/>.

3 Opis plików początkowych, tymczasowych i końcowych

- maze.txt
Jest to plik, w którym znajduje się labirynt zakodowany tekstowo. Przykładowy plik znajduje się na rysunku 3.
- maze.bin
Plik, w którym znajduje się labirynt zakodowany binarnie zgodnie ze standardem zbliżonym do RLC 8.



Rysunek 1: Wygląd zdekodowanego headera w maze.bin

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 zawierające 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: Instrukcja pliku binarnego

- `maze_decoded.txt`
Jest to tymczasowy plik z labiryntem w formie tekstowej, który powstał w wyniku dekodowania pliku binarnego. Struktura jest identyczna jak pliku tekstowego na rysunku 3
- `graf.txt` Tymczasowy plik z labiryntem po przejściu algorytmu BFS zapisany w formie tekstowej. Jego przykładowa forma została przedstawiona na rysunku 4.
- `wynik_temp.txt`
Jest to plik w którym znajduje się ścieżka od końca do początku w labiryncie. Ma ona format: L7U2L2U2L2D2L8D2L8U2R2U2L, gdzie numer oznacza liczbę kroków w danym kierunku. Litery oznaczają D-dół, U-góra, L-lewo, R-prawo
- `wynik.txt`
W tym pliku znajduje się lista następujących po sobie kroków w labiryncie od początku do końca. Jego dokładny format znajduje się na rysunku 5.
- `Makefile`
Plik służący do automatycznej kompilacji wymaganych w programie plików. Jego wynikiem jest plik `a.out`.

```

a.out: main.o wymiary.o grafSolver.o graf2.o
    gcc main.o wymiary.o grafSolver.o graf2.o -o a.out

main.o: main.c funkcje.h
    gcc -c main.c

wymiary.o: wymiary.c funkcje.h
    gcc -c wymiary.c

grafSolver.o: grafSolver.c funkcje.h
    gcc -c grafSolver.c

graf2.o: graf2.c funkcje.h
    gcc -c graf2.c

```

4 Podział systemu na moduły

Moduły występujące w projekcie pt. CrazyMaze to:

- Moduł Main(main.c)
- Moduł odpowiedzialny za deszyfrowanie plików zakodowanych binarnie(w pliku main.c)
- Moduł odpowiedzialny za zliczanie ilości kolumn i wierszy w labiryncie(wymiary.c)
- Moduł odpowiedzialny za szukanie rozwiązania labiryntu i wypisywanie go(graf2.c)
- Moduł zawierający spis wszystkich funkcji wykorzystywanych w programie(funkcje.h)
- Moduł odpowiedzialny za zapisywanie wyniku w końcowym formacie (grafSolver.c)

5 Opis implementacji modułów

5.1 Moduł MAIN

Moduł Main poprzez main.c odpowiada za „zbiorcze“ zgromadzenie wszystkich funkcji programu w jednym miejscu oraz ich wywoływanie, gdy zajdzie taka potrzeba. Na samym początku tego modułu znajdują się załączone stosowne pliki nagłówkowe oraz główna funkcja main. Następnie program korzysta z funkcji getopt, która jest odpowiedzialna za uruchamianie programu ze stosownymi argumentami wejściowymi. Program wykorzystuje następujące flagi: -h: odpowiada za wyświetlenie instrukcji obsługi dla użytkownika i -l: wymaga podania nazwy pliku, którego zawartość ma być odczytana przez program. Dodatkowo, w module main.c, podczas odczytu pliku, program automatycznie sprawdza format wskazanego pliku w celu oceny, czy należy wykorzystać dekodery plików binarnych. W przypadku braku potrzeby użycia funkcji deszyfrującej, program natychmiastowo przechodzi do rozwiązywania labiryntu. Reszta tego modułu to odpowiednio wywołane funkcje służące do rozwiązywania labiryntu. Do modułu należy też plik Makefile.

5.2 Moduł odpowiedzialny za odszyfrowywanie blików binarnych na pliki w formacie txt

Funkcja w tym module najpierw wczytuje 40 bajtów z pliku wejściowego f do headera. Następnie alokuje pamięć na tablicę struktur slowo, gdzie każda struktura przechowuje trzy bajty. Liczba elementów tej tablicy jest równa wartości pola counter w headerze. Potem po kolei wczytuje dane z pliku do struktury slowo, po czym sprawdza współrzędne currX, currY, exitX, exitY, entryX i entryY. Jeżeli określone zmienne są w tym samym położeniu, to tam określamy początek 'P' oraz koniec 'K' (jest to ustalone w headerze). Program tak działa, aż do odczytania całego pliku, a następnie zwalnia pamięć i kończy swoje działanie.

5.3 Moduł odpowiedzialny za zliczanie ilości kolumn i wierszy w labiryncie

Moduł ten jest odpowiedzialny za zliczanie ilości kolumn i wierszy w pliku tekstowym. Na początku program przyjmuje jako argument plik, z którego będzie czytywał ilość wierszy i kolumn. Program zlicza wiersze używając funkcji fgetc(), która zwiększa zmienną z o 1 przy każdym napotkaniu znaku nowej linii \n. Proces ten trwa do momentu osiągnięcia końca pliku. Podobnie zlicza kolumny, również korzystając z funkcji fgetc(), ale tym razem na podstawie ilości wierszy. Jeśli liczba kolumn w bieżącym wierszu jest większa niż dotychczasowa wartość, program dokonuje aktualizacji zmiennej przechowującej ilość kolumn. Po zakończeniu iteracji po pliku, program kończy swoje działanie, przekazując uzyskane informacje (liczbę wierszy i kolumn) do innych modułów.

5.4 Moduł rozwiązań

Moduł rozwiązań związany jest ze znalezieniem i wypisaniem rozwiązania końcowego. Odpowiadają za to pliki graf.c Działanie algorytmu opisano dokładniej w następnym rozdziale. W skrócie, badane jest otoczenie każdego odwiedzonego pola, jeżeli w pole graniczące z badanym można pójść następuje dodanie go do kolejki. Po odnalezieniu K w otoczeniu dodawany jest ostatni krok i algorytm przestaje działać. Dodatkowo na bieżąco wpisywany jest odwrócony kierunek chodzenia po labiryncie tak, by po odnalezieniu końca móc wrócić na start "po śladach" i wpisać przebieg trasy do pliku. Gdy to nastąpi usuwane są pliki tymczasowe a program się kończy.

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
P   X   X                               X                               X   X
X XXX X X X XXXXXXXXXXX XXXX X XXX XXXXXXXXXXX XXXX XXXX XXXX XXXX X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X X XXXXXXXXXXX X XXX XXX XXXXXXX XXX X X X XXXXXXXXXXX XXXX X X XXXX
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X XXXXXXX XXX X X XXX X X XXX X X XXX XXX X X XXX XXXXXXX X XXXXX X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
XXXXXXXX XXX XXX X X XXXXX X XXXXXXX XXX XXX XXXXXXX XXX XXXXXXX X X X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X X X XXXXX X XXXXX X XXXXXXX XXX XXX X XXXXX XXXXX X XXXXXXX X XXX X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X X XXXXXXX X XXXXXXX X X X X XXX XXXXXXX X X X X XXX XXXXX X X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X XXX XXX XXXXX XXX X XXXXXXX X XXXXX X X XXXXXXXXXXX XXXX X X XXXXX
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X XXXXX XXX XXXXX X X X XXX XXX X X XXX X XXX XXX XXX XXX XXX XXX
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X X XXXXXXX X XXX XXXXX X XXX X XXXXXXX XXX XXXXXXX XXX XXXXXXX X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X XXX X XXX XXX X X X XXXXX XXXXX XXX XXXXXXXXXXX XXX XXXXXXX X X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X XXX X XXX XXXXXXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX X
X XXX X X XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X XXX XXX XXXXXXX XXX XXXXX X XXX X X XXX X X XXXXXXX X XXXXXXX X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX X
X X X X X X X X X X X X X X X X X X X X X X X X X X X X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

Rysunek 3: Labirynt przed użyciem BFS’a

```

graf.txt — Notatnik
Plik Edycja Format Widok Pomoc
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
PLLLLLXDLXDLLLLLLLLLLLLLLLLLLX X X X
XUXOXUXDXUXDXXXXXXXXXXXUXOXOX XXX XXXXXXXXXXX XXXX XXXX XXXX X
XUXDXULLXULLDXRRRRDXUXRRRRUX X X X X X X X X X X X X X X X
XUXDXXXXXXXXXDXUXDXDXXXXXXXXX XXX X X X XXXXXXXXXXX XXXX X X XXXXX
XUXRRRRRRRRRRDXUXDLXRRUX X X X X X X X X X X X X X
XUXXXXXXXXXXXDXUXDXOXOXUX XXX X X XXX XXX X X XXX XXXXXXX X XXXX X
XULLLLLLXUXDLXUXDXRRUXUX X X X X X X X X X X X X X X X
XXXXXXXXXXDXUXDXUXOXOX X XXXXXXX XXX XXX XXXXXXX XXX XXXXXXX X X X
DXRRDXULLLXRRUXRRUX X X X X X X X X X X X X X X X X
DXUXDXOXOXUXUXOXOXOX XXXXXXX XXX XXX X XXXXX XXXXXXX XXXX X XXX X
DXUXRRRRRRRUXDXRRUX X X X X X X X X X X X X X X X X
DXUXXXXXXXXXXXDXUXDXOXOXOX X X X XXX XXXXXXXXXXXDXDX X XXX XXXX X
XRRUXDLXRRRRUXRRUX X X X X XDLLLLLLLXULLX X X X X X
XUXOXDXOXUXOXOXOXDX XXXXXXX X XXXXX X XXXXXXXXXXXXXOXOX X X XXXX X
XUXRRRRDXULLXRRULLDX X X X X X XDXDLLLLLL XUL X X X X
XUXOXOXDXOXUXOXOXDX X XXX XXX X XXX XDXXXXXXXXX XXXXXXXX X X XXX
XUXDLLLLLLXULLXDXULLX X X X X XDXRRDLLLX X X X X
XUXDXXXXXXXXXXXDXUXDXOXOX X XXX X XXXXXXXXXXOXOXOXOXOXOX X XXXXXXXX X
XUXRRDXRRRRRRRUXRRRULLLLX X X X XDLLLLLLXDLX X X X
XUXOXDXUXOXOXOXOXOXOXOX XXX XXX XDXOXOXDXOX XXX XXXXXXXX X XXX X
XULLLXUXRRUXDLXDLLLXUX X X XRRRRDXURDX X X X X X X X
XXXXXXXXXXDXUXDXOXOXOXOX X X XXXXXDXOXOX XXXX XXX XXXX XXX X
XDLXUXUXDXDLXURDXUXDLX X XRRUXDLXULLDX X X X X X X X
DXOXUXOXOXOXOXOXOXDXUXDXOXOX XXXOXDXOXOXDXOX X XXX XXXXXXXXXX X
DXRRUXULLLXDXULLDXUXDXULLLLX XRRDXRRURDX XDX X X X
DXUXOXOXOXOXDXOXOXDXUXDXOXOXOX XXXOXOXOXOXOXOXOXOXOXOX XXX X X
DXULLLXRRRRDXULLLXUXRRDXULLLLX XDLXDLLLXDXRRRRRRRRRRDLX X X
DXOXOXOXOXOXOXOXOXOXOXOXOXOX XXXDXUXOXOXOXOXOXOXOXOXOXOXOXOXOX
XRRDLXULLLXRRRRDXRRULLXRRDXRRUX XDXUXRRUXDXUXDLLLLLLXULLX X
XUXDXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOXOX
XUXRRRRRRRRULLLLXULLLLLLLLXULLLLLLLLXULLLLLLLLX RRRRRULLLLLLL
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

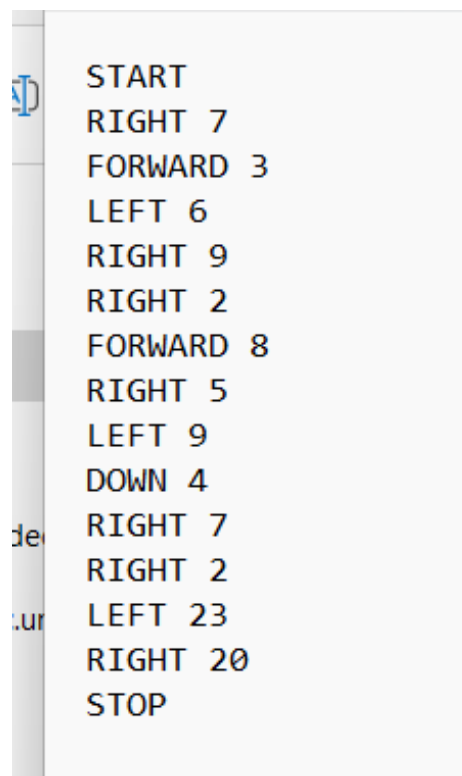
Rysunek 4: Labirynt po użyciu BFS’a

5.5 Moduł skorowidza

Moduł skorowidza zawiera w sobie definicje wszystkich wykorzystywanych w programie funkcji.

5.6 Moduł odpowiedzialny za zapisywanie wyniku w końcowym formacie

Jest to moduł przetwarzający plik z wypisanymi krokami po labiryncie na opis zgodny z wymaganiami projektu. Tworzy on z opisu tablicę po której iteruje od końca i zgodnie z algorytmem wypisuje liczbę oraz słowo(LEFT, RIGHT itp.) zależnie od litery oraz liczby obok. W ten sposób powstaje plik z opisem tj. FORWARD4, LEFT5.

A screenshot of a text editor window. The editor has a light gray background and a vertical scrollbar on the left. The text is black and consists of a sequence of navigation commands for a maze. The commands are: START, RIGHT 7, FORWARD 3, LEFT 6, RIGHT 9, RIGHT 2, FORWARD 8, RIGHT 5, LEFT 9, DOWN 4, RIGHT 7, RIGHT 2, LEFT 23, RIGHT 20, and STOP. The text is left-aligned and spans approximately 15 lines.

```
START  
RIGHT 7  
FORWARD 3  
LEFT 6  
RIGHT 9  
RIGHT 2  
FORWARD 8  
RIGHT 5  
LEFT 9  
DOWN 4  
RIGHT 7  
RIGHT 2  
LEFT 23  
RIGHT 20  
STOP
```

Rysunek 5: Wypisanie kroków najkrótszej ścieżki w labiryncie

6 Opis wykorzystywanych algorytmów

- Algorytm BFS:
Przykładowy algorytm w pseudokodzie:

```
int graf2(FILE* in, FILE* out, int wiersze, int kolumny, FILE*
    wynik){
    zaalokuj pamiec na kolejke(400 000 bitow)
    przepisz labirynt do pliku out
    cofnij wskaznik na poczatek
    przesun wskaznik do punktu P w labiryncie
    zbadaj otoczenie i okresl kierunek poczatkowy
    while(1){
        zbadaj otoczenie(gora, dol, prawo lewo )
        jezeli pole obok to 'K' petla konczy sie
        jezeli nie to postaw na polu przeciwny kierunek do badanego//np
            jezeli badana jest gora to wpisany zostanie D(dol)
        dodanie wszystkich pol ' ' w otoczeniu do kolejki
    }
    zresetowanie wskaznika w pliku out
    Przesun wskaznik do punktu K w labiryncie
    Zbadaj otoczenie K(w poszukiwaniu U/D/L/R)
    ustaw kierunek poruszania sie na ten zgodny z otrzymanym
        wynikiem
    while(1){
        jezeli obecny kierunek = kierunek na polu to kroki++
        jezeli obecny kierunek != kierunek na polu to wypisuje kroki i
            kierunek do pliku. kroki = 0, kierunek to litera odczytana
            z pola
        jezeli obecne pole to 'P' nastepuje wypisanie krokow i kierunku
            do pliku. Petla konczy sie
    }
    zwolnienie pamieci;
    return 0;
}
```

- Algorytm dekodujący pliki zakodowane binarnie:
Przykładowy algorytm w pseudokodzie:

```
int odczyt(wskaznik do czytanego pliku *f, wskaznik to
    tworzonego pliku *out) {
    okreslenie zmiennych currX i currY
    odczytanie 40 bajtow z pliku f i zapisanie pod wskaznikiem
        header
    return -1
    zaalokowanie pamieci na strukture slowa(licznik w headerze *3)
    return -1
    jezeli(wczytane slowa nie rownaja sie obecnemu licznikowi slow)
        zwolnij slowa
    kiedy(liczba slow w headerze wieksza od obecnego i){
    kiedy(ilosc slow wieksza od y){
    jezeli(currX i entryX header sa rowne i currY i entryY headera
        sa rowne)
        wypisz w tym miejscu 'P'
    jezeli(currX i exitX headera sa rowne i currY i exitY headera
        sa rowne)
        wypisz w tym miejscu 'K'
    }
    currX++
    }
    jezeli(currX == liczba kolumn w headerze + 1)
    currX = 1
    currY = 0
}
zwalniamy pamiec
return 0
```

7 Opis obsługi błędów w programie

7.1 Potencjalne błędy, które mogą wystąpić:

- Błąd: Nie można otworzyć pliku. W tej sytuacji program informuje użytkownika o braku możliwości otwarcia pliku, po czym kończy działanie programu.
- Błąd: Nie znaleziono pliku wejściowego. Program wypisuje informację o problemie, po czym kończy natychmiastowo swoje działanie.
- Błąd: Nie podano pliku początkowego. Program wypisuje informację o braku pliku początkowego, po czym kończy swoje działanie.