Wydział Elektryczny

POLITECHNIKA WARSZAWSKA

Języki i metody programowania 2

Dokumentacja projektowa: `labfinder`

Oliwia Pawelec, Jakub Żebrowski

prowadzący zajęcia:

Dr inż. Radosław Roszczyk

08.03.2024

Cel projektu

Celem projektu jest opracowanie programu o nazwie `labfinder`, znajdującego najkrótszą drogę od wybranego punktu wejściowego do wybranego punktu wyjściowego przez labirynt.

Program działa w trybie nieinteraktywnym (wsadowym). Labirynt, przez który będzie opracowana ścieżka, będzie znajdować się w pliku wyznaczonym przez użytkownika w argumentach wywołania programu. Wynik programu będzie umieszany na konsoli lub w pliku, który użytkownik wyznaczy w kolejnym argumencie wywołania.

Na projekt został nałożony limit pamięciowy 512 kB w czasie całego działania. Również projekt musi obsługiwać pliki binarne, których specyficzna struktura została opisana w założeniach projektowych napisanych przez prowadzących program zajęć. Jej opis jest dostępny w ostatniej sekcji tego dokumentu: *Dodatek: struktura specyficznego pliku binarnego.*

Opis funkcjonalności

- **Wczytanie pliku**: program wczytuje plik, którego nazwę podaje użytkownik. Program sprawdza również czy plik istnieje - jeśli nie, daje komunikat o tym.
- **Odkodowanie wczytanego labiryntu**: jeśli użytkownik podał plik z kodowaniem binarnym, program wyłuska z niego informacje na temat struktury labiryntu albo zwróci komunikat, że kodowanie jest nieznane/niepoprawne.
- **Walidacja struktury labiryntu**: program sprawdza, czy dany przez użytkownika labirynt jest poprawny: czy jest wejście/wyjście, równa liczba kolumn/wierszy etc.
- **Znalezienie ścieżki**: program znajduje najkrótszą ścieżkę przez labirynt od punktu startowego do punktu końcowego. Zapamiętuje przechodzoną ścieżkę.
- **Zapis ścieżki**: program zapisuje wybraną przez siebie ścieżkę (w formacie opisanym w rozdziale *Wykorzystywane pliki*) do wskazanego przez użytkownika pliku wyjściowego - jeśli plik o takiej nazwie nie istnieje to go tworzy - albo wypisuje wszystko na konsolę, jeśli użytkownik nie wybrał pliku docelowego.

Wykorzystywane pliki

Plik wejściowy: obsługiwane są dwa formaty plików: tekstowy i binarny

1. W formacie **tekstowym**, labirynt powinien występować w formacie, który zawiera definicje punktów reprezentujących odpowiednio:
   - P – punkt wejścia do labiryntu
   - K - punkt wyjścia z labiryntu,
   - X - ściana
   - spacja - miejsce, po którym można się poruszać

   Plik ten składać się powinien z pojedynczych znaków, których położenie x, y reprezentuje faktycznie położenie w labiryncie.

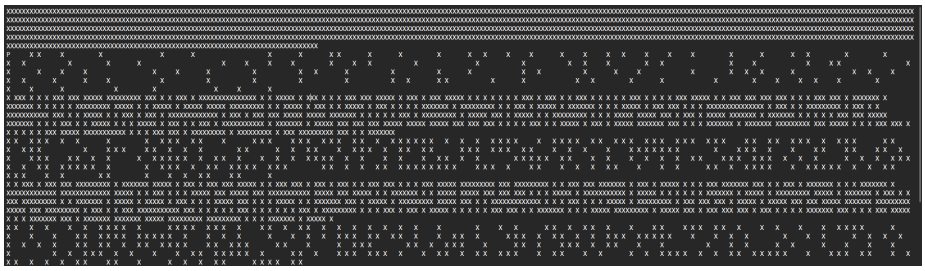

Figure 1: Przykładowy wygląd pliku wejściowego w formacie **tekstowym**

2. W formacie **binarnym**, labirynt powinien wpasowywać się strukturą w narzucony standard, opisany w ostatniej sekcji dokumentacji *Dodatek: struktura specyficznego pliku binarnego*

Plik wyjściowy: zawiera najkrótszą trasę przez wybrany labirynt, opisaną w formacie listy wykonanych kroków, na przykład:

```
START
FORWARD 1
TURNLEFT
FORWARD 4
TURNRIGHT
FORWARD 3
STOP
```

Tu również obsługiwane są oba (**tekstowe** i **binarne**) rodzaje plików. Kodowanie plików zachodzi w analogiczny sposób co do obsługi pliku wsadowego. Jeśli użytkownik zechce rozwiązanie w formie **tekstowej**, do pliku wyjściowego wypisana zostanie lista kroków. Jeśli jednak w formie **binarnej**, do pliku wyjściowego zostanie zapisany zakodowany w formie binarnej labirynt, oraz pod nim zakodowaną (również binarne) listę kroków do przejścia najkrótszej trasy.

Podział na moduły

Program został podzielony na moduły:

- obsługa wejścia/wyjścia: obsługa plików wejściowych/wyjściowych, sprawdzanie błędów
- operacje na bitach: ułatwienie operacji na bitach (dostanie się do konkretnego bita, ustawienie, sprawdzenie jego wartości)
- odkodowywanie plików: odczyt struktury labiryntu z pliku, zapis tej struktury w programie
- najkrótsza ścieżka: znajdowanie najkrótszej ścieżki przez podany labirynt
- zakodowywanie plików: zapis struktury labiryntu do pliku wraz z wybraną trasą przez ten labirynt

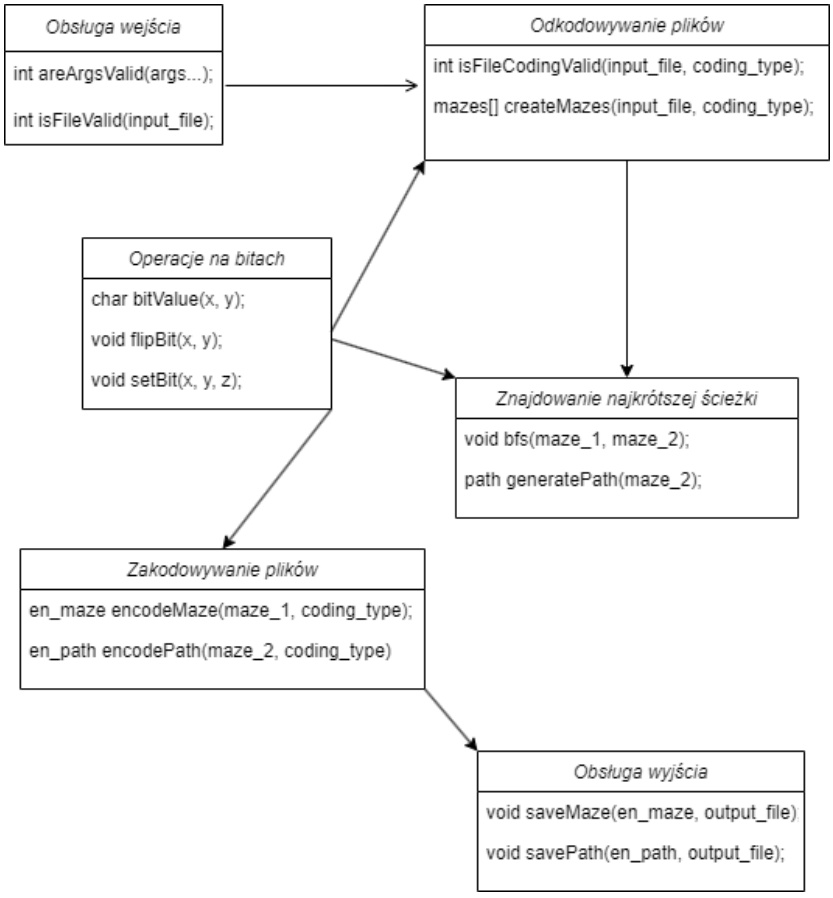

Figure 2: Uproszczony podział na moduły

Obsługa wejścia/wyjścia

W tym module znajduje się funkcja sprawdzająca dodatkowe argumenty - wyrzuca komunikat o błędzie kiedy np. jakichś argumentów będzie brakować.

```
while ((c = getopt(argc, argv, "in:inc:out:outc")) != -1) {
    switch (c) {
        ...
    }
}
```

Obsługa we/wy zajmuje się również wyświetlaniem komunikatu pomocy, kiedy podane argumenty będą niepoprawne.

```
// wyświetla jak użyć programu (argumenty etc)
void help(char* input_filename) {
    ...
}
```

Operacje na bitach

Moduł będzie składał się z makr zajmujących się operacjami na bitach: sprawdzeniem wartości danego bita na x,y pozycji w macierzy, ustawieniem jego wartości na konkretną, zmienieniem jej na przeciwną.

Odkodowywanie plików

Moduł najpierw zajmuje się sprawdzaniem czy kodowanie jest poprawne - poniżej przykładowy fragment kodu, który sprawdza, czy liczba wierszy jest wszędzie taka sama.

```
char is_file_coding_valid(FILE* input_file, char coding_type) {
    ...
    if (rows_count != rows[1])
        return 1; // invalid
    ...
    return 0; // valid
}
```

Moduł musi zawierać funkcję transformującą zakodowany labirynt za labirynt w formie macierzy bitowej `maze[256][256]` i `maze[128][128]`.

```
void decode_file(FILE* input_file, char coding_type) {
    ...
}
```

Najkrótsza ścieżka

W module pojawia się funkcja z implementacją BFS.

```
void bfs(char maze_struct[256][256], char maze_flags[128][128]) {
    // forma kolejki, do której będzie się dodawać kolejne sąsiednie komórki
    while (queue.size != 0) {
        ...
    }
    ...
}
```

Na podstawie labiryntu z flagami, zostanie opracowana ścieżka:

```
void create_path(path* some_path, char maze_2[128][128]) {
    ...
}
```

Zakodowywanie plików

W tym module labirynt z macierzy bitowej zamieniany jest na format zapisywalny do pliku.

```
void encode_maze(FILE* output_file, char coding_type) {
    ...
}
```

Dopisywana jest również trasa najkrótszej ścieżki do pliku wyjściowego.

```
void encode_path(FILE* output_file, char coding_type) {
    ...
}
```

Wykorzystane algorytmy

Przechowywanie labiryntu

Labirynt jest przechowywany w dwóch macierzach bitowych. Są to dwie 2-wymiarowe tablice o rozmiarach 256x256 i 128x128 bajtów. W pierwszej przechowywane są informacje na temat wyglądu labiryntu (rozmieszczenie ścianek, komórek), a w drugiej flagi (odwiedzona/nieodwiedzona) dla każdej komórki.

Algorytm przechodzenia

Do znalezienia najkrótszej ścieżki został użyty algorytm BFS (ang. *Breadth First Search*). Operuje on na drugiej macierzy 128x128 i kolejce obok, do której kolejno są dodawani sąsiedzi. Zmienia on flagi komórek odwiedzonych na '1'. Po znalezieniu punktu wyjściowego, ustawia resztę flag (nie uwzględnionych w trasie) na '0'. Na podstawie tego ustalana jest ścieżka - zapisywana do pliku wyjściowego.

Argumenty wywołania programu

Program może być wywoływany w trzy sposoby:

labfinder przedstawia informacje o poprawnym wywołaniu programu oraz o jego działaniu.

labfinder -in <wejscie> -inc <typ kodowania {t|b}> wczytuje podany plik wejściowy w kodowaniu określonym jako t (**tekstowe**) albo b (**binarne**), a następnie wypisuje listę kroków na `stdout`.

labfinder -in <wejscie> -inc <{t|b}> -out <wyjscie> -outc <{t|b}> wczytuje plik wejściowy o nazwie `<wejscie>` o określonym kodowaniu, a następnie tworzy plik `<wyjscie>` i wypisuje do niego listę kroków w określonym kodowaniu

Program nie wymaga dodania rozszerzenia do nazw plików wejściowych i wyjściowych, wymaga jedynie podania rodzaju kodowania pliku wsadowego.

Komunikaty błędów

Program będzie komunikował błędy, gdy:

- będzie brakowało argumentów: np. z nazwą pliku wsadowego, z informacją o kodowaniu pliku wsadowego - informacja o pliku wyjściowym i o jego kodowaniu może zostać pominięta - wówczas na konsolę zostanie wypisana lista kroków przejścia w formacie tekstowym
- argumenty wywołania nie będą poprawne (np. wskazane przez użytkownika pliki nie będą istniały)
- kodowanie pliku z labiryntem nie będzie w poprawnym formacie
- labirynt będzie uszkodzony: będzie brakowało punktu start/koniec lub będzie ich więcej niż po jednym, nie będzie istniała ścieżka prowadząca z punktu początkowego do końcowego, liczba wierszy/kolumn będzie zmienna

Testowanie

Program powinien być w stanie przejść testy wykonane przy użyciu programu *valgrind*, również przy użyciu komendy

```
/usr/bin/time -v
```

Dodatek: struktura specyficznego pliku binarnego

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.