Specyfikacja Projektu Labirynt

Marcel Opałko i Jakub Hamerlik

01.04.2024

Spis Treści

1 Specyfikacja programu

- System operacyjny: **linux / unix**
- Użyty jezyk programowania: **C**
- Implementacja i standard jezyka: **C99 ISO/ANSI**
- Zgodne kompilatory: **GCC, clang**
- Maksymalna pamieć operacyjna: **512 kB**

2 Sposób uruchomienia programu

- Uruchamiamy plik makefile, który kompiluje wszystkie moduły. Nastepnie wpisujemy:
  ./a.out [nazwa_pliku_wejsciowego] [nazwa_pliku_wyjsciowego] [wersja_zapisu_pliku]

  wersja_zapisu_pliku = 1 to wyjście w formie mapy binarnej
  wersja_zapisu_pliku = 0 to wyjście w formie tekstowej

3 Opis funkcjonalności, które powinny być zrealizowane

3.1 Wczytywanie Labiryntu

Program powinien umożliwiać użytkownikowi wczytanie labiryntu z pliku tekstowego lub binarnego.

- Obsługa błedów podczas wczytywania danych, takich jak nieprawidłowy format pliku lub brak pliku.

3.2 Analiza Labiryntu

Po wczytaniu labiryntu, program powinien przeprowadzić analize jego struktury, identyfikujac punkt wejścia, punkt wyjścia, ściany i dostepne przestrzenie.

3.3 Znajdowanie Drogi przez Labirynt

Implementacja algorytmu przeszukiwania grafu lub innego odpowiedniego algorytmu, który znajdzie najkrótsza ścieżke od punktu wejścia do punktu wyjścia.

- Zapewnienie optymalnego działania algorytmu zgodnie z ograniczeniami dotyczacymi zużycia pamieci.

3.4 Generowanie Listy Kroków

Po znalezieniu drogi, program powinien generować liste kroków, które prowadza od punktu wejścia do punktu wyjścia.

- Każdy krok powinien być opisany w sposób czytelny dla użytkownika, jako "FORWARD", "TURNLEFT", "TURNRIGHT" razem z liczba kroków.

3.5 Zapisywanie Listy Kroków

Wygenerowana lista kroków powinna być zapisywana do pliku tekstowego w czytelnej formie.

- Obsługa błedów zwiazanych z zapisem danych do pliku.

3.6 Obsługa Plików Binarnych

Program powinien mieć możliwość wczytania i zapisania labiryntu w postaci pliku binarnego zgodnie z wymaganiami specyfikacji.

- Odpowiednie przetwarzanie danych binarnych z uwzglednieniem podanej struktury pliku binarnego.

3.7 Zarzadzanie Pamiecia

Zapewnienie efektywnego zarzadzania pamiecia, unikajac wycieków pamieci i nadmiernego zużycia zasobów.

- Regularne zwalnianie zaalokowanej pamieci po jej użyciu.
- Program powinien działać zgodnie z ograniczeniami dotyczacymi zużycia pamieci, nie przekraczajac 512 kB w czasie całego działania.

3.8 Kompatybilność z Systemem Linux

Program musi być kompatybilny z systemem Linux, aby działać poprawnie w tym środowisku.

- Testowanie i rozwiazywanie ewentualnych problemów zwiazanych z kompatybilnościa systemu.

3.9 Makefile

Dołaczony plik Makefile powinien umożliwiać łatwa kompilacje programu oraz automatyzować proces budowania projektu.

3.10 Elastyczność Architektury

Program powinien być zaprojektowany w sposób umożliwiajacy elastyczna modyfikacje i dostosowanie do ewentualnych zmian wymagań projektowych w trakcie semestru.

- Odpowiednie zastosowanie struktur danych i modułowego projektowania.

4 Szczegółowe opisy wszystkich plików

4.1 Plik Wejściowy (Labirynt)

Program bedzie wczytywał labirynt z pliku tekstowego lub binarnego zawierajacego definicje punktów labiryntu. Maksymalny rozmiar labiryntu to 2049x2049 znaków.

- Obsługa Pliku Tekstowego - Program musi obsługiwać wczytywanie labiryntu z pliku tekstowego.
- Obsługa Pliku Binarnego - Program musi obsługiwać wczytywanie labiryntu z pliku binarnego.

4.2 Plik Wyjściowy (Lista Kroków)

Po znalezieniu drogi przez labirynt, program bedzie generował liste wykonanych kroków i zapisywał je do pliku tekstowego. Każdy krok jest opisany jako "FORWARD", "TURNLEFT", "TURNRIGHT" razem z ewentualna liczba kroków przy "FORWARD".

4.3 Plik Konfiguracyjny (Makefile)

Aby umożliwić kompilacje programu, bedzie wymagany plik Makefile, który zawiera instrukcje kompilacji i łaczenia plików źródłowych.

- Zarzadzanie Modułami. W Makefile należy uwzgledнić moduły programu i odpowiednie zależności.

.

4.4 Pliki Binarny (Labirynt w Postaci Pliku Binarnego)

Program bedzie musiał mieć możliwość wczytania i zapisania labiryntu w postaci pliku binarnego.
Struktura pliku binarnego:

- Nagłówek pliku

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

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 rozwiazania

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 rozwiazania - Pola liczone sa bez uwzglednienia pola startowego

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)

Plik binarny bedzie zawierał odpowiednie struktury danych do reprezentacji labiryntu.

4.5 Pliki Tymczasowe

Program tworzy pliki tymczasowe podczas działania zapisujac w nich:

- temp - tablica plików tymczasowych zawierajacych możliwości przejść pomiedzy komórkami. Każdy plik (oprócz ostatniego) składa sie z 10 wierszy z 1024 liczbami.
- temp1 - tablica plików tymczasowych zawierajacych wagi komórek. Składa sie z takich samych elementów jak tablica temp.

5 Wykorzystane funkcje:

5.1 Funkcja wczyt

Funkcja wczyt bedzie przyjmować plik wejściowy i przetwarzać ten plik. Tworzymy tablice dwuwymiarowa intów 1024x10. Funkcja wczytuje 3 wiersze wejścia i sprawdza dla każdej komórki, w która strone można przejść. Wartość komórki sprawdzanej zależy od możliwości przejścia do innych, każde przejście oznacza dodanie odpowiedniej liczby: N-2, E-4, S-8, W-16. Wyniki zapisujemy w tablicy i przepisujemy trzeci wiersz do pierwszego oraz wczytujemy nastepne dwa. Powtarzamy ta czynność (10 razy) aż zapełnimy tablice. Pełna tablice zapisujemy jako plik tymczasowy. Usuwamy tablice i tworzymy nowa. Powtarzamy taki proces aż przejdziemy po całym pliku wejściowym.

5.2 Funkcja podmien

Funkcja podmien - bedzie zmieniać tablice używane podczas pracy działania funkcji DFS oraz wypis. Funkcja ta bedzie podmieniała wartości w tablicy roboczej 1024x30 z plików tymczasowych tak, żeby program zawsze miał (o ile to możliwe nie ostatni i pierwszy blok) jeden blok nad i pod blokiem środkowym.

5.3 Funkcja DFS

Funkcja DFS bedzie używać zmodyfikowanego algorytmu DFS aby przejść sie po labiryncie i znaleźć w nim najkrótsza droge. DFS działa poprzez rekurencyjne przeszukiwanie grafu, eksplorujac jedna gałaź w głab przed powrotem. Funkcja bedzie działała jak zwykły DFS z jedna zmiana, bedziemy tworzyć tablice wag. Zaczynajac od wejścia do labiryntu każda komórka bedzie dostawała wage, dziecko dostaje wage o jeden wieksza od rodzica. Zmieni sie również kryterium wejścia do dziecka, zamiast patrzeć czy zostało ono odwiedzone, bedziemy sprawdzać czy możemy zmniejszyć wage dziecka.

5.4 Funkcja wypis

Funkcja wypis bedzie intepretować wyniki funkcji DFS za pomoca utworzonej wcześniej tabeli wag. Zaczynajac od wyjścia labiryntu, zawsze bedziemy wchodzić do komórki o jeden mniejszej od wagi aktualnej komórki. W ten sposób przejdziemy sie po najkrótszej drodze z wejścia do wyjścia.

6 Wykorzystane algorytmy:

6.1 DFS

DFS - deep-first search to algorytm przeszukujacy graf. Algorytm przechodzi sie po grafie wchodzac w głab jednej gałezi, aż nie dojdzie do ślepego zaułku. Wtedy wraca sie do punktu, w którym może wejść w głab innego odgałezienia. Algorytm przechodzac sie zaznacza że odwiedził dana komórke aby uniknać zapetleń działania programu.
Nasz program wykorzystuje zmodyfikowana wersje tego programu. Zamiast zaznaczać, że odwiedziliśmy dana komórke, zaznaczamy jej odległość od wejścia. Robimy to poprzez działanie: odległość ojca + 1. Algorytm może wchodzić do odwiedzonych już komórek tylko wtedy, kiedy wchodzac, bedzie zmniejszał odległość dziecka. Wynikiem przejścia naszego algorytmu jest powstała tablica zawierajaca odległości wszystkich komórek od wejścia. Aby uzyskać wynik, nasz program zaczynajac od wyjścia, cofa sie po komórkach z kolejnymi, malejacymi o jeden, wartościami.