Dokumentacja Techniczna Programu "Labirynt"

Jakub Hamerlik i Marcel Opałko

2 Marzec 2024

1 Idea działania programu

Program pozwala na znalezienie drogi przez wczytany uprzednio labirynt, który jest plikiem tekstowym zawierającym definicje odpowiednich punktów

- P - punkt wejścia do labiryntu
- K - punkt wyjścia z labiryntu,
- X - ściana labiryntu
- spacja - ścieżka, po której można się poruszać

W naszym programie będziemy korzystać z algorytmu DFS (Deep First Search) do znajdywania drogi przez labirynt.

Maksymalny rozmiar labiryntu to 1024 x 1024 liczony po ścieżkach, a więc 2049 x 2049 jeśli chodzi o znaki. (włączając znak nowej linii)

Jeśli chodzi o architekturę programu - musi być ona w maksymalnym stopniu elastyczna, ponieważ w trakcie semestru może nastąpić zmiana wymagań dot. projektu.

Program w C nie powinien zużywać więcej pamięci niż 512kB w czasie całego swojego działania. Aby spełnić ten warunek, należy dobrze zarządzać pamięcią w programie, więc podejmiemy kroki:

- Będziemy wczytywać labirynt kawałek po kawałku.
- Ograniczymy złożoność obliczeniową poprzez np. usunięcie zbędnych pętli.
- Brak wielokrotnego zwalniania pamięci
- Zapobieganie wyciekom pamięci

Wyjściem z programu będzie lista wykonywanych kroków, np w postaci:

```
START
FORWARD X (X = liczba kroków w przód)
TURNLEFT
...
STOP
```

2 Wywołanie programu

1. Skompiluj program za pomocą pliku makefile (w katalogu projektu), wpisując komendę "make"
2. Uruchom program, podając odpowiednie parametry wejściowe.

   ```
   ./program nazwa_pliku_wejściowego nazwa_pliku_wyjściowego
   ```

   - *nazwa_pliku_wejciowego* - nazwa pliku z którego wczytywany jest labirynt
   - *nazwa_pliku_wyjciowego* - nazwa pliku do którego ma być zapisana ścieżka (lista) (brak podania tego argumentu skutkuje wypisaniem śćieżki na stdout)

3 Podział programu na moduły

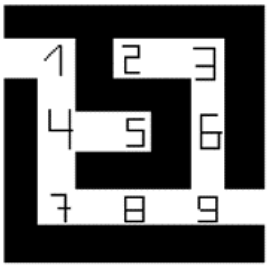
Program przechodzenia labiryntu będzie dzielił się na kilka kluczowych modułów:

- Funkcja wczytująca dane z pliku wejściowego i zmieniająca te dane na format wygodny do użycia w reszcie programu.
- Funkcja DFS przechodząca się po labiryncie reprezentowanego za pomocą grafu, szukająca przejścia od wejścia do wyjścia.
- Funkcja intepretująca ścieżkę DFS na format wyjściowy, czyli listę kolejnych kroków przejścia po labiryncie.
- Funkcja dzieląca labirynt na mniejsze części i zapisująca je w plikach chwilowych. Pozwoli to na zaoszczędzenie pamięci używanej podczas programu.

4 Opis podstawowych funkcji i struktur

Program zawierać będzie kilka struktur:

- Tablicę dwuwymiarową int'ów (T), która będzie zawierać dla każdego pola labiryntu listę pól, do których można z niego przejść. Np. dla labiryntu:



  Tablica będzie wyglądać:
  $1\{4\}, 2\{3\}, 3\{2, 6\}, 4\{1, 5, 7\}, 5\{4\}, 6\{3, 9\}, 7\{4, 8\}, 8\{7, 9\}, 9\{6, 8\}$

- Tablicę jednowymiarową bool'i (odw), która będzie zawierać informację w których komórkach nasz algorytm DFS już był. 0 oznacza komórkę nieodwiedzoną, 1 odwiedzoną.

Oprócz struktur program zawierać będzie również funkcje:

- Funkcja podziel – będzie używana w każdej innej funkcji tego programu. Będzie ona dzielić labirynt na pomniejsze kwadratowe segmenty (rozmiar każdego z nich jest jeszcze do ustalenia, przykładowo może mieć rozmiar 64x64) i zapisywać je w plikach chwilowych.

- Funkcja wczyt – będzie przyjmować plik wejściowy i przetwarzać ten plik na format opisany w strukturze powyżej (T).

- Funkcja DFS – będzie używać algorytmu DFS aby przejść się po labiryncie i znaleźć w nim przejście. Funkcja będzie zapisywać ścieżkę przejścia, aby ułatwić działanie funkcji intepretującej wyniki.

- Funkcja wypis – będzie intepretować wyniki funkcji DFS na format wyjściowy opisany w wymogach zadania, np.:
  START
  FORWARD 1
  TURNLEFT
  FORWARD 4
  TURNRIGHT
  FORWARD 3
  STOP