Spis treści

[Cel projektu 2](#_Toc88968998)

[Rozwiązanie problemu 2](#_Toc88968999)

[Szczegóły implementacyjne 2](#_Toc88969000)

[Sposób wywołania programu 2](#_Toc88969001)

[Wnioski i spostrzeżenia 2](#_Toc88969002)

# Cel projektu

Projekt można podzielić na 2 główne etapy:

1. Wygenerowanie labiryntu i przepisanie go na graf, z czego

* labirynt powinien być wielkości od 2x2 do 10x10, użytkownik powinien móc sam zdecydować o oczekiwanej wielkości
* Generacja labiryntu jest całkowicie losowa (pseudolosowa)
* Wyświetlone powinny być wszystkie połączenia grafu.

1. Przypisanie wag do każdego wierzchołka grafu i znalezienie ścieżki o najmniejszej wadze.
   * Wagi powinny być o wartości zmiennoprzecinkowej z zakresu 0-10

# Rozwiązanie problemu

Na początku poszukałam algorytmów umożliwiających generację labiryntu. Zdecydowałam się na algorytm Depth First Search, polecany jako jeden z prostszych. W zrozumieniu kroków bardzo pomógł film obrazujący algorytm:  
<https://www.youtube.com/watch?v=YBI2SDnleEw&t=3s>

1. Najpierw algorytm ustawia wszystkie wierzchołki na nieodwiedzone oraz przejścia na zabudowane.
2. Następnie zaczynając od startu, jeśli ma wokół siebie nieodwiedzone wierzchołki, losuje wśród nich kolejny na który przechodzi, niszcząc równocześnie ścianę i oznaczając stary jako odwiedzony. Krok ten zapisuje na stosie.
3. Jeśli algorytm natrafi na sytuację, w której wszystkie wierzchołki są odwiedzone, cofa się usuwając ostatnie kroki ze stosu aż do momentu gdy już będzie mógł wykonać krok 2.
4. Jeśli przy okazji kroku 3. algorytm wróci do startu, generacja została zakończona.

Minusem tego algorytmu jest to, że tworzy on wyłącznie jedną ścieżkę od wejścia do wyjścia. Aby urozmaicić mój labirynt, pod koniec generacji losowo niszczę pozostałe ściany. Dzięki temu ilość ścieżek w labiryncie stała się różnorodna

Wygenerowany labirynt wyświetliłam.

Następnie zamieniłam mój labirynt w postaci macierzy na postać listy sąsiedztwa i to wyświetliłam. Przy okazji tego kroku nadałam każdemu wierzchołkowi wagę od 0.00 – 10.00

Na koniec pozostało znaleźć ścieżkę o najmniejszej wadze. Wykorzystałam do tego algorytm Dijkstra, jako opiera się w sumie na wagach.

1. Najpierw algorytm ustawia wszystkim wierzchołkom koszt dojścia na Null oraz ustawia je na nieodwiedzone
2. Następnie dla początkowego wierzchołka ustawia koszt dojścia na jego wagę i oznacza go jako odwiedzonego
3. Następnie dla wszystkich wierzchołków połączonych z nim oblicza i przypisuje wagę dojścia, czyli koszt dojścia wcześniejszego wierzchołka plus waga obliczanego.
4. Wśród wierzchołków nieodwiedzonych z przypisanym kosztem dojścia wybrać ten o najniższym koszcie i oznaczyć go jako odwiedzonego. Wykonać dla niego krok 3.

Gdy krok 4 odnajdzie i oznaczy jako odwiedzony wierzchołek końcowy, można przejść do wypisywania ścieżki. Aby to zrobić :

1. Zaczynając od wierzchołka końcowego poszukujemy wierzchołków sąsiadujących najniższym przypisanym koszcie dojścia. Przejście na nie zapisujemy. Powtarzamy kroki aż algorytm dojdzie do wierzchołka początkowego.

Tak otrzymaną ścieżkę wyświetliłam w kolejności odwrotnej, aby uzyskać drogę od startu do końca.

W tym momencie wszystkie założenia projektu zostały zrealizowane.

# Szczegóły implementacyjne

Najpierw pobieram od użytkownika wielkość labiryntu i sprawdzam czy liczba jest w zakresie 2-19. (zawiera to w sobie zakres 2-10)

Obraz zawierający zrzut ekranu, tekst, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Następnie pobieram czy użytkownik chce wyświetlić labirynt wygenerowany w wersji z nazwą numeryczną wierzchołków czy bez.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Po pobraniu inputu zaczynam inicjalizację labiryntu w formie macierzy rozmiar x rozmiar mającą w środku structa przechowywującego informacje jak na zdjęciu poniżej

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

Następnie inicjuję stos tworząc go w bardzo prymitywny sposób (tworząc tablice o rozmiarze rozmiar2 x 2 i wypełniając go wartością -1. (Później nauczyłam się robić stos za pomocą linked listy, jednak tutaj zachowałam już i ten prymitywny sposób).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

Same funkcje do obsługi stacku zostały opisane w ten sposób, później będą one wykorzystywane:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Następnie wylosowałam początek i koniec labiryntu i zaczęłam jego generację

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Generacja labiryntu jest implementacją algorytmu Depth First Search.

Pierwszy while mówi, aby wykonywać jego zawartość aż stos nie będzie pusty (Krok 4 mojego opisu DPS – wykonywanie póki algorytm nie wróci do startu)

W warunku drugiego while’a zawiera się funkcja lab\_przebijanie\_scian (odpowiadająca krokowi 2 mojego opisu DPS) a w samym warunku wykonuje się logika odpowiadająca krokowi 3 mojego opisu DPS, która wykonuje się w momencie gdy lab\_przebijanie\_ścian zwraca -1 czyli nie ma możliwych ruchów do wykonania. Funkcja ta wygląda tak:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

Najpierw bada ona dla każdego wierzchołka ile on ma sąsiadów, którzy nie zostali do tej pory odwiedzeni lub nie są ścianą.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

Następnie spośród dostępnych kierunków losuje jeden i burzy ścianę pomiędzy wierzchołkami, przechodząc na kolejny.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Na koniec zwraca -1 lub 1, w zależności czy są dostępne kierunki do wylosowania.

Po wygenerowaniu labiryntu trzeba było go wyświetlić w postaci macierzy (2n + 1) x (2n + 1) (aby móc ładnie wyświetlić ściany w konsoli.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

W samej funkcji zaczęłam od wypełniania macierzy

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Korzystając z danych z labiryntu dalej ustawiałam kratki w macierzy. Potrzebne były w sumie tylko informacje o stanie ścian na górze i po prawej od wierzchołka, pozostałe były redundantne.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Tutaj nastąpiła poprawa algorytmu Depth First Search poprzez dodanie możliwości tworzenia pozostałych dróg. Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Następnie wyświetliłam całość labiryntu (na zdjęciu poniżej sposób wyświetlania labiryntu z ponumerowanymi wierzchołkami.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Następnie zaczęłam przepisywać labirynt na formę listy sąsiedztwa

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Zaczęłam od inicjacji. W tym kroku również wylosowałam wagę dla każdego wierzchołka.

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, zrzut ekranu

Opis wygenerowany automatycznie

Sprawdziłam liczbę połączeń dla każdeg o wierzołka i przypisałam ją do .ilepołączeń

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Zainicjowałam tablice .sąsiedzi (przechowywującą id wierzchołków sąsiadów) oraz tablice .sąsiedzi\_adresy będącą pointerem na sąsiedne wierzchołki.

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

Po inicjacji wypełniłam te tablice.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Następnie wyświetliłam gotową listę sąsiedztwa.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Tą funkcję zakończyłam o przypisanie wartości do wierzchołków o ich koordynatach i domyślnemu ustawieniu ich na nieodwiedzone.

Następnie wzięłam się za znajdywanie najkrótszej drogi.

Zaczęłam od przypisania adresu pierwszego i ostatniego wierzchołka, oraz utworzenia tablicy droga, w której zostanie zwrócona pod koniec działania funkcji znajdywanie\_najkrotszej\_drogi

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Zaczęłam od inicjalizacji elementów, kosztu dojścia dla wierzchołka początkowego, listę potencjalnych sąsiadów Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Następnie nastąpiło wejście w pętlę implementacji algorytmu Djikstry odwzorującą kroki 3 i 4 mojego opisu. Wykonuje się ona do momentu, w którym zostanie odwiedzony wierzchołek końcowy.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Po tym momencie pojawił się problem, gdy musiałam wykonać „backtracking”, czyli krok 5 algorytmu. Coś w mojej implementacji sprawiało, że czasami program zwracał mi seg fault. Jako, że w debuggerze nie mogłam wywołać przez długi czas problemu, musiałam szukać błędu ręcznie. Dużo czasu pochłonął mi ten błąd. Ostatecznie jednak go znalazłam. Problem znajdował się tutaj:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Gdzie przez przypadek od razu „przeskakiwałam” przy pierwszym porównaniu sąsiednich wierzchołków na sąsiada, nie sprawdzając czy pozostałe mają mniejszy wagowy koszt dojścia. To sprawiło że w późniejszej części programu pojawiał się właśnie seg fault albo program się zapętlał w momencie gdy natrafił na ślepą uliczkę. Poprawiona pętla backtrackingu, czyli kroku 5 jest widoczna poniżej:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Na koniec zajęłam się zapisywaniem najkrótszej drogi do tablicy i wyświetlaniem jej w formie tekstowej.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Zostało mi tylko wyświetlenie najkrótszej drogi w formie graficznej.



Funkcja do tego była bardzo prosta, wykorzystywała już stare kawałki kodu, jedyne co to wymagała uzupełnienia starej tablicy o drogę.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Na koniec pozostało tylko zwalnianie pamięci na koniec programu

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

# Sposób wywołania programu

Program wystarczy włączyć za pomocą wpisania ścieżki do MazeOrginal.exe w cmd / powershellu i kliknięciu enter.

Następnie należy podać oczekiwany rozmiar labiryntu, wybrać czy chce się dostać labirynt z wyświetloną numeracją wierzchołków, po tym kroku reszta programu samodzielnie się wykona.

# Wnioski i spostrzeżenia

Duże trudności miałam z implementacją algorytmu Djikstry. Powodem nie był logiczny brak zrozumienia, a sama implementacja, w której w pewnym momencie wkradł się segmentation fault. Błąd ten był o tyle nieprzyjemny, że pojawiał się nieregularnie, raz na kilka wykonań, z jakiegoś powodu też nie chciał się pojawiać w debuggerze. Po kilku dniach udało mi się odnaleźć problem, wspomniałam o tym pobieżnie w sekcji implementacji.

Bardziej ogólne wnioski:

Projekt dla początkującej osoby zarówno w c jak i w samym programowaniu był wymagający i zdecydowanie trudny. Jednak nie była to poprzeczka za wysoko postawiona, możliwym było dowiezienie projektu na czas, szczególnie dzięki dobremu rozkładaniu terminów (dawanie zarówno wcześniejszego terminu oddania w ramach motywacji jak i bezkarnego dodatkowego terminu po świętach dla osób napotykających na problemy czy to z czasem własnym czy to z programem). Nauczyłam się podczas tego projektu dużo: roli i sposobu korzystania ze wskaźników, tablic dynamicznych, działania algorytmu First Depth Search oraz Djikstra, rozdzielania swojego kodu na kilka plików, działania i tworzenia struktur danych, rozbijania większych zadań na mniejsze, szukaniu pomocy w programowaniu. Były to rzeczy dla mnie wcześniej niejasne lub obce. Uważam, że jest kilka rzeczy które mogłabym zrobić lepiej, jednak na poziom umiejętności z którym zaczynałam oraz czas jaki mogłam poświęcić w ramach tego przedmiotu, projekt uważam za udany.