SPRAWOZDANIE		Data wykonania: *06.01.2024*
Tytuł Mini-Projektu	**Wykonał:**	**Sprawdził:**
Mini Projekt 3	*Jakub Haraszkiewicz* *331696*	*dr inż. Konrad Markowski*

Spis treści

Cel projektu

Stworzenie programu generującego losowy labirynt i wyznaczającego ścieżkę od początku do końca.

Rozwiązanie problemu

Stworzono program w języku C drukujący losowy labirynt o losowych lub podanych wymiarach. Program używa jednego z trzech dostępnych algorytmów generujących i wyznacza ścieżkę między losowo wybranymi punktami na górze i na dole.
Zaimplementowane algorytmy:
Depth-first search (DFS) – algorytm chodzi po nieodwiedzonych komórkach i usuwa na swojej drodze ściany, kiedy komórka na której się znajduje nie ma nieodwiedzonych sąsiadów, algorytm cofa się aż dojdzie do takiej która ma. Ma tendencję do tworzenia długich, krętych i mało rozgałęzionych korytarzy.
Algorytm Prima – algorytm dodaje sąsiadów odwiedzonych komórek na listę i w losowej kolejności dodaje je do labiryntu usuwając ściany między nimi a odwiedzoną komórką. Ma tendencję do tworzenia korytarzy z wieloma krótkimi rozgałęzieniami.
Algorytm Wilsona – algorytm generuje losowe niezapętlające się ścieżki i dodaje je do labiryntu. Pierwsza biegnie między losowo wybranymi komórkami, następne biegną od losowej niedodanej komórki do napotkania komórki będącej już w labiryncie.

Szczegóły implementacyjne

Pliki

Kod jest rozdzielony między kilka plików o różnej funkcjonalności:

labirynt.c/.h – podstawowe struktury i funkcje używane w pozostałych plikach

DFS.c/.h – funkcja implementująca zrandomizowany algorytm depth-first search do generacji labiryntu

prim.c/.h - funkcja implementująca zrandomizowany algorytm Prima do generacji labiryntu oraz jedna funkcja skracająca funkcję główną.

wilson.c/.h - funkcja implementująca zrandomizowany algorytm Wilsona do generacji labiryntu oraz jedna funkcja skracająca funkcję główną.

solver.c/.h – dwie działające razem funkcje rozwiązujące labirynt przy użyciu algorytmu depth-first search, funkcja wypisująca diagram przejść rozwiązania.

druk.c/.h – funkcje drukujące labirynt

main.c – funkcja main sterująca całością programu, funkcja start interpretująca podane argumenty i zgłaszająca błędy oraz funkcja help wypisująca pomoc.

Struktury

Komórka

typedef struct komorka zawiera informacje o pojedynczej komórce.
Pola:
int status – w zależności od tej wartości program rozpoznaje komórkę jako część bariery wokół labiryntu, komórkę nieodwiedzoną (nie jest jeszcze częścią labiryntu), komórkę odwiedzoną przez algorytm generujący (część labiryntu), komórkę odwiedzoną przez algorytm rozwiązujący, komórkę należącą do ścieżki (używane przy algorytmie Wilsona i algorytmie rozwiązującym) lub komórkę na liście używanej przez algorytm Prima.
int rodzaj – służy do oznaczenia komórek startowej i końcowej rozwiązania.
int numer – numer komórki licząc od lewej do prawej i z góry na dół.
float gora, dol, prawo, lewo – wagi przejść między komórkami z zakresu 0 – 10. 0 oznacza brak przejścia, czyli ścianę (jest to domyślna wartość).
int x, y – współrzędne komórki w labiryncie
int numernaliscie – numer komórki na liście, wartość przydatna przy usuwaniu komórek ze środka listy

Lista

typedef struct lista zawiera dwa pola:
komorka** elementy – tablica wskaźników na komórkę
int rozmiar – liczba elementów zapisanych w tablicy

Labirynt

typedef struct labirynt zawiera informacje o całym labiryncie
Pola:
komorka** komórki – dwuwymiarowa tablica komórek tworząca labirynt
komorka* start, stop – wskaźniki na komórki startową i końcową rozwiązania
lista lista – lista używana przy algorytmach Prima i Wilsona
int x, y – wymiary labiryntu

Droga

typedef struct droga tworzy stos komórek należących do drogi (rozwiązania lub drogi w algorytmie Wilsona). Pola:
komorka* step – element stosu
struct droga* next – wskaźnik na następną część stosu
float waga – waga jednego przejścia wykorzystywana do sumowania wag rozwiązania

Algorytmy generacji

DFS

Algorytm dfs jest zaimplementowany w funkcji rekurencyjnej.
Typ funkcji: integer
Argumenty: dwuwymiarowa tablica komórek (labirynt), współrzędne aktualnej komórki, losowa liczba służąca randomizacji algorytmu
Na samym początku funkcja zwraca 0, jeśli aktualna komórka była już odwiedzona.
Głównym elementem funkcji jest pętla while trwająca tak długo jak aktualna komórka posiada nieodwiedzonych sąsiadów. Program rozpoczyna następne iteracje funkcji w losowych sąsiadujących komórkach i sczytuje zwracane wartości. Jeśli zwrócono 1 – tworzy się przejście między komórkami, jeśli 0 – komórka była już odwiedzona więc przejście nie powstaje. Trwa to aż do spełnienia warunku końca pętli.
Na końcu funkcja zwraca wartość 1.

Prim

Typ funkcji: void
Argumenty: struktura labirynt, współrzędne początkowej komórki, losowa liczba służąca randomizacji algorytmu
Na początku pierwsza komórka jest oznaczana jako odwiedzona (należąca do labiryntu).
Następnie tworzona jest lista, w której przechowywane będą komórki sąsiadujące z komórkami odwiedzonymi i dodawani są do niej sąsiedzi komórki początkowej.
Rozpoczyna się pętla (1) while trwająca tak długo jak w liście są elementy. Program wybiera losową komórkę z listy i zaczyna następną pętlę (2) w której losuje komórkę obok. Jeżeli jest to komórka należąca do labiryntu to tworzy w jej kierunku przejście i kończy pętlę (2), jeżeli nie, powtarza pętlę (2) do skutku. Następnie komórka zostaje usunięta z listy i oznaczona jako odwiedzona, a jej sąsiedzi są do listy dodani. Proces jest powtarzany do spełnienia warunku końca pętli (1).

Wilson

Algorytm Wilsona jest zaimplementowany za pomocą dwóch funkcji: generacja_wilson() (funkcja główna) i randomwalk() (tworzy losową ścieżkę od wybranej niedodanej komórki do komórki już dodanej). Obie te funkcje są typu void i jako argumenty przyjmują strukturę labirynt i losową liczbę do randomizacji, randomwalk() przyjmuje też współrzędne losowej niedodanej komórki.

generacja_wilson():

Tworzona jest lista ze wszystkimi komórkami. Program usuwa losową komórkę z listy, dodaje ją do labiryntu i rozpoczyna pętlę while trwającą do usunięcia z listy wszystkich komórek. W pętli losowo wybrana komórki jest usuwana z listy, oznaczana jako część ścieżki i w jej współrzędnych wywoływana jest funkcja randomwalk(). Następnie wybrana komórka jest oznaczana jako część labiryntu i pętla powtarza się do spełnienia warunku kończącego ją.

randomwalk():

Program tworzy strukturę droga oraz wskaźniki na aktualną komórkę (k) i komórkę (n), która będzie następnym krokiem w ścieżce. Wskaźnik na k jest przekazywany do drogi. Rozpoczyna się pętla while trwająca do dojścia algorytmu do komórki należącej do labiryntu. W pętli:

1. losowa komórka obok k zostaje zapisana jako n
2. jeśli n jest poza granicami labiryntu pętla zaczyna się od nowa
3. jeśli n należy już do drogi program cofa się na drodze aż dojdzie do poprzedniego wystąpienia n
4. jeśli poprzednie warunki nie zostały spełnione n zostaje zapisana jako k i dodana do drogi

Następnie rozpoczyna się kolejna pętla, która tworzy przejścia między komórkami na drodze, oznacza je jako odwiedzone i usuwa je z listy nieodwiedzonych.

Algorytm rozwiązujący DFS

Algorytm rozwiązujący jest podzielony na dwie funkcje typu float: funkcję główną solver() i funkcję pomocniczą f(). Obie przyjmują jako argumenty dwuwymiarową tablicę komórek (labirynt), wskaźnik na wskaźnik na strukturę droga, współrzędne aktualnej komórki i wskaźnik na float przechowujący sumę wag rozwiązania. Funkcja f() przyjmuje dodatkowo float przechowujący wagę pojedynczego przejścia z aktualnej komórki. Funkcje solver() i f() są w ciągu rekurencyjnym, wywołują się wzajemnie.

solver():

Głównym zadaniem funkcji solver() jest wywoływanie funkcji f() w dostępnych komórkach sąsiadujących z aktualną. Następnie zwraca wartość zwracaną przez f() czyli aktualną sumę wag w przypadku znalezienia rozwiązania lub 0 w przypadku nieznalezienia rozwiązania.

f():

Funkcja f() posiada wyrażenie if które wykonuje się w dwóch przypadkach

1. aktualna komórka jest komórką końcową
2. funkcja solver() wywołana w tej komórce zwróciła coś innego niż 0 co może nastąpić tylko jeżeli została już znaleziona komórka końcowa.

Jeśli któryś z warunków został spełniony, aktualna komórka zostaje dodana na stos droga i zwrócona zostaje aktualna suma wag. Jeśli warunki nie zostaną spełnione funkcja zwraca 0.

Druk

druk():
Typ: void
Argumenty: struktura labirynt, wymiary labiryntu, int tryb
Funkcja druk() odpowiada za wywoływanie pozostałych funkcji drukujących labirynt. Najpierw wywoływana jest funkcja drukbariera() drukująca górną granicę. Następnie w pętli for wywoływanej tyle razy, ile rzędów komórek ma labirynt, wywoływane są na zmianę funkcje odpowiadające za drukowanie pionowych i poziomych ścian labiryntu. W zależności od tego jaki został podany int tryb wywoływane są funkcje drukujące labirynt pusty, z rozwiązaniem lub z numerami komórek. Na koniec znowu wywołana jest funkcja drukbariera().
Wszystkie pozostałe funkcje działają w podobny do siebie sposób. Jako argumenty przyjmują dwuwymiarową tablicę komórek (labirynt), liczbę kolumn w labiryncie i numer aktualnego rzędu. Używają pętli for do przechodzenia po kolumnach labiryntu i drukują spacje wewnątrz komórek i w miejscach przejść oraz znak Z w miejscach ścian. Znak Z jest zdefiniowany na początku pliku druk.c jako "\033[101m\033[47m \033[40m" (\033 to znak Escape). W ten sposób są ustawiane kolory ścian, tła i ścieżki/numerów, uniemożliwia to jednak drukowanie labiryntu do pliku, ponieważ kolory wyświetlają się poprawnie tylko w terminalu.

Inne funkcje

Losowanie

Każdy z algorytmów generujących używa funkcji losuj() do wybierania losowej sąsiadującej komórki. Funkcja losuj() zwraca dwu elementową tablicę liczb całkowitych; indeks 0 odpowiada kierunkowi pionowemu, a 1 poziomemu. Elementy mogą przybierać wartości -1, 0 lub 1.
-1 oznacza góra lub lewo, 1 oznacza dół lub prawo. Funkcja losuje dwie wartości: indeks 0 lub 1 i wartość -1 lub 1. Uzyskany w ten sposób kierunek funkcje generujące dodają do współrzędnych komórki i otrzymują współrzędne losowej komórki sąsiadującej.

Dodawanie i usuwanie z listy

Funkcja dodajdolisty() zmienia status komórki na 4 (znajduje się na liście), aktualizuje rozmiar listy, dodaje komórkę i nadaje komórce numer na liście równy aktualnemu rozmiarowi.
Funkcja usunzlisty() przesuwa wszystkie elementy na liście o numerze wyższym niż usuwana komórka o jedno miejsce w dół i aktualizuje ich numery na liście.
Funkcja dodajsasiadow() z prim.c uruchamia funkcję dodajdolisty() dla wszystkich sąsiadów podanej komórki o ile są oni nieodwiedzeni (status = 0).

Sposób wywołania programu

Program wywołuje się komendą ./Labirynt. Jako argumenty można podać wymiary oraz wybrane flagi:
-h – wyświetla pomoc z opisem programu i wszystkich flag (bez labiryntu)
-wilson – zostaje użyty algorytm Wilsona
-prim – zostaje użyty algorytm Prima
-dfs – zostaje użyty algorytm depth-first search
-sciezka – rysuje rozwiązanie na labiryncie
-numery - numeruje komórki i wypisuje graf przejść z uwzględnieniem wag
-pusty – rysuje labirynt bez rozwiązania i numerów

Należy podać tylko jedną flagę dotyczącą algorytmu generującego.

Domyślnie użyty jest algorytm Wilsona i drukowany jest labirynt ze ścieżką

Przykładowe wywołania:
./Labirynt
./Labirynt 20 20 -dfs -numery
./Labirynt -prim
./Labirynt -pusty -wilson

Wnioski i spostrzeżenia

Zadanie było przyjemne i dobrze mi się nad nim pracowało, było też jednak bardzo czasochłonne. Robiąc je zaimplementowałem 4 różne algorytmy generacji labiryntu (jednego nie zamieściłem w finalnej wersji) i dużo się przy tym nauczyłem. Dopiero algorytm Wilsona który zaimplementowałem jako ostatni dawał satysfakcjonujące rezultaty, jednak na poprzednie algorytmy poświęciłem już trochę czasu więc nie chciałem ich usuwać z projektu (algorytm Aldousa-Brodera zawieszał się przy labiryntach większych niż 8 na 8 więc go akurat usunąłem).