POLITECHNIKA ŚLĄSKA

WYDZIAŁ INŻYNIERII BIOMEDYCZNEJ

PROJEKT BIOCYBERNETYKA

Generowanie labiryntu za pomocą uczenia maszynowego

Autorzy:
Emilia Wojciechowska
Michał Drozd

Opiekun:
dr inż. Zuzanna Miodonska

Zabrze D.M.Y

1 Cel pracy

Celem naszego projektu jest stworzenie skryptu generującego labirynt za pomocą różnych algorytmów. Tworzenie labiryntu jest wizualizowane. Zaimepletowany oraz zwizualizowany zostanie także skrypt przechodzący stworzony labirynt. Wyniki generacji labiryntu zostaną opisane oraz omówione.

2 Użyte algorytmy

2.1 Depth-first search

Przeszukiwanie w głąb, w skrócie DFS jest to algorytm przeszukiwania grafu. Jest on najlepszym, najwygodniejszym algorymtem zarówno do tworzenia jak i rozwiązywania labiryntów. Polega na analizie wszystkich krawędzi wychodzących z danego wierzchołka. Po zbadaniu wszystkich krawędzi wychodzących z danego wierzchołka algorytm powraca do wierzchołka, z którego dany wierzchołek został odwiedzony. Algorytm ten ma dobrą złożoność, nie zabiera za wiele pamięci. Tworzenie labiryntu polega na tak długim wybieraniu poszczególnych wierzchołków i tworzenia labiryntu, aż natrafi się na jakąś przeszkodę. Wtedy algorytm wraca do miejsca, z którego będzie mógł kontynuować tworzenie w inną stronę, i tak aż do zapełnienia całej powierzchni.

2.2 Binary Tree

Jest to jeden z najprostrzych algorytmów. Tworzenie opiera się na jedej komórce, nie na całym labiryncie. W komórce losowo wybierany jest kierunek: północ lub zachód. Można tworzyć nieskończenie duże labirynty przy bardzo małej ilości pamięci. Problemem tego algorytmu jest to, że ma on silne przekątne. Ponadto dwa z czterech stron labiryntu będą połączone jednym korytarzem. Jednak można to zaakceptować w naszej sytuacji.

2.3 Side Winer

1.Przejdź przez siatkę w wierszu, zaczynając od komórki od 0,0. 2.Dodaj bieżącą komórkę do zestawu „uruchom”. 3.Dla bieżącej komórki losowo zdecyduj, czy wyrzeźbić wschód, czy nie. 4.Jeśli fragment został wyrzeźbiony, ustaw nową komórkę jako komórkę bieżącą i powtórz kroki 2-4. Jeśli przejście nie zostało wyrzeźbione, wybierz dowolną komórkę z zestawu przebiegów i wytnij przejście na północ. Następnie opróżnij zestaw uruchomieniowy, ustaw następną komórkę w wierszu jako bieżącą komórkę i powtórz kroki 2-5. Kontynuuj, aż wszystkie rzędy zostaną przetworzone.

2.4 Wilson

1.Wybierz dowolny wierzchołek losowo i dodaj go do UST. 2.Wybierz dowolny wierzchołek, który nie znajduje się jeszcze w UST i wykonuj błądzenie loso-

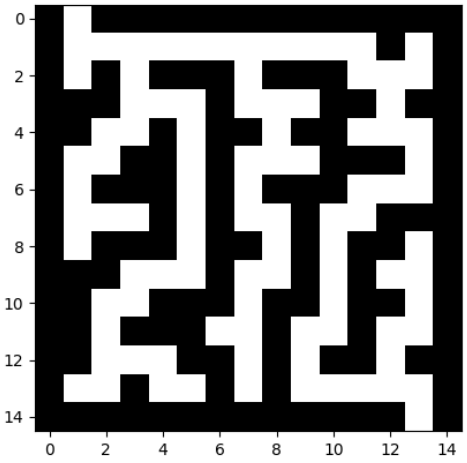
we, aż napotkasz wierzchołek, który znajduje się w UST. 3.Dodaj wierzchołki i krawędzie dotknięte podczas błądzenia losowego do UST. Powtarzaj 2 i 3, aż wszystkie wierzchołki zostaną dodane do UST.

3 Specyfikacja wewnętrzna

Python 3.10.0 Biblioteki:Matplotlib,Tkinter,Numpy,Time

4 Kod

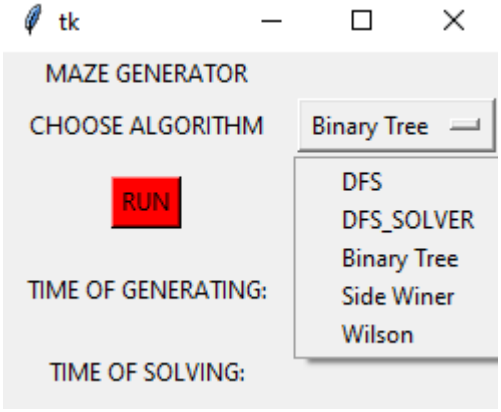
Dla każdego algorytmu stworzyliśmy klasę, która generuje labirynt. W każdym przypadku możliwe jest ustalenie wielkości labiryntu, oraz zobaczenie wizualizacji. Algorytmy zostały napisane w języku Python, wizualizaja powstała za pomocą biblioteki Matplotlib. Przykładowy labirynt o wielkości 15x15 za pomocą algorytmu DFS wygląda następująco:



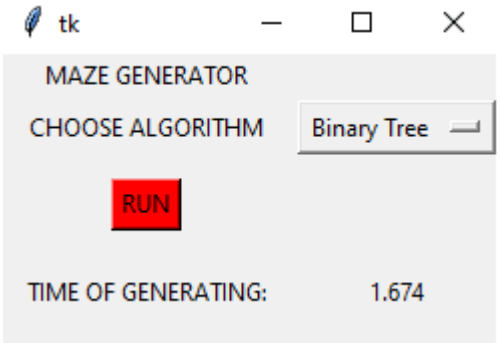
Rysunek 1: Przykładowy labirynt DFS

Aby ułatwić generowanie labiryntów stworzyliśmy GUI za pomocą Tkinter, czyli graficzny interfejs umożliwiający wybór danego algorytmu, oraz po generacji wyświetlający czas z dokładnością do części tysięcznych pokazujących ile trwało zrobienie pełnego labiryntu.

Jedną opcją do wyboru jest przejście wygenerowanego labiryntu DFS. Pokazywany jest czas tworzenia oraz rozwiązywania. Algorytm rozwiązywania działa analogicznie do algorytmu tworzenia, metoda wizualizacji została ulepszona o kolory, które ułatwiają śledzenie ścieżki.



Rysunek 2: GUI



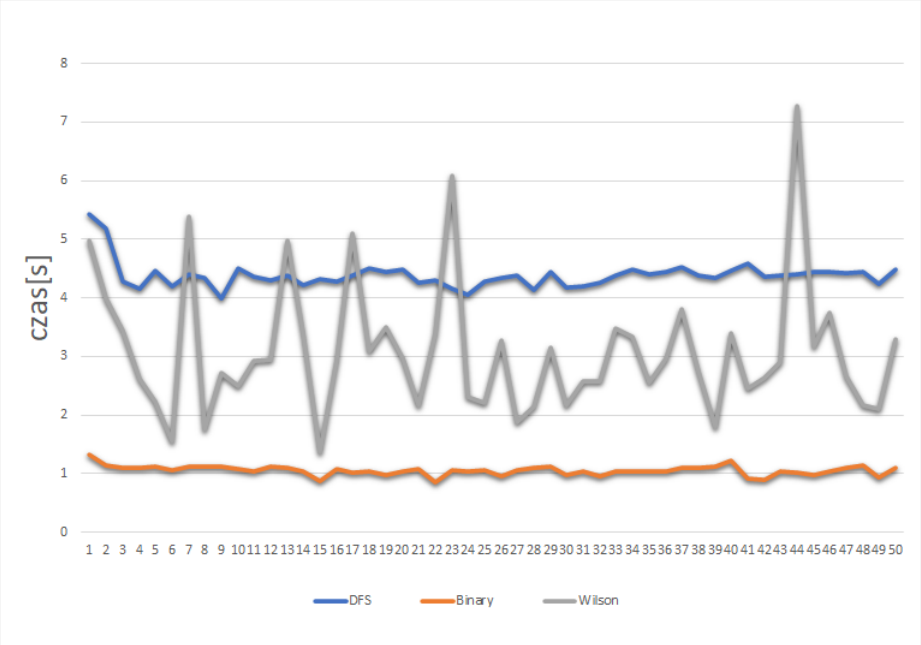
Rysunek 3: Pokazany czas generacji labiryntu za pomocą algorytmu Binary Tree

5 Wyniki i testowanie

Testowanie algorytmów polega na porównywaniu czasów tworzenia pomiędzy labiryntami.

Pomiar	DFS	Binary	Wilson
1	5,437	1,321	4,983
2	5,173	1,14	3,983
3	4,28	1,093	3,437
4	4,158	1,109	2,609
5	4,468	1,125	2,234
6	4,204	1,062	1,546
7	4,405	1,125	5,392
8	4,343	1,118	1,749
9	3,983	1,125	2,718
10	4,499	1,078	2,499
11	4,359	1,046	2,923
12	4,297	1,125	2,952
13	4,391	1,094	4,967
14	4,218	1,047	3,39
15	4,327	0,875	1,371
16	4,281	1,078	2,937
17	4,374	1,015	5,092
18	4,499	1,047	3,093
19	4,452	0,969	3,5
20	4,484	1,047	2,968
21	4,265	1,078	2,171
22	4,296	0,844	3,39
23	4,155	1,065	6,092
24	4,046	1,046	2,312
25	4,271	1,062	2,203
26	4,343	0,953	3,28
27	4,374	1,062	1,875
28	4,14	1,109	2,14
29	4,436	1,125	3,155
30	4,186	0,984	2,157
31	4,202	1,031	2,578
32	4,249	0,953	2,577
33	4,389	1,047	3,484
34	4,483	1,046	3,327
35	4,405	1,046	2,562
36	4,452	1,047	2,968
37	4,53	1,094	3,812
38	4,39	1,109	2,749
39	4,343	1,122	1,796
40	4,454	1,218	3,39
41	4,577	0,906	2,453
42	4,359	0,89	2,64
43	4,374	1,047	2,906
44	4,405	1,015	7,264
45	4,452	0,968	3,171
46	4,439	1,047	3,737
47	4,421	1,109	2,655
48	4,436	1,14	2,171
49	4,233	0,937	2,109
50	4,483	1,093	3,297
Średnia	4,3844	1,05664	3,09528
Odchylenie	0,2289247	0,0844743	1,1504359

Rysunek 4: Porównanie czasu generacji labiryntów za pomocą różnych algorytmów



Rysunek 5: Wykres przedstawiający zależności czasów dla różnych algorytmów

6 Podsumowanie

Wszystkie pomiary zostały przeprowadzone dla rozmiaru labiryntu 15 na 15. Zostało wykonane po 50 prób pomiarów. Najdłużej tworzył się labirynt za pomocą Wilsona. Jest to spowodowane dość ciekawym rozwiązaniem algorytmu, który losuje poszczególne punkty oraz drogę do nich. Labirynt tworzony jest skomplikowanie, nie zawiera pomyłek, jednak czas jest znacznie najdłuższy. Im więcej labiryntu jest stworzone, tym czas i szukanie dróg stają się coraz dłuższe. Metoda Drzewa Binarnego spowodowała najkrótszy czas, jednak labirynt jest bardzo prosty, mało rozbudowany. Natomiast DFS pozwala nam renderować skomplikowane przejścia w labiryncie, a dodatkowo czas działania jest przystępny, pulsuje się w okolicach 4,3s.

7 Źródła

- https://python.plainenglish.io/maze-generation-algorithms-with-matrices-in-python-i-33bc69aacbc4
- https://weblog.jamisbuck.org/under-the-hood/