Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Dokumentacja projektu zespołowego

Wizualizacja wyszukiwania drogi w labiryncie

Filip Stępień Nr indeksu: 094117
Rafał Grot Nr indeksu: 094046

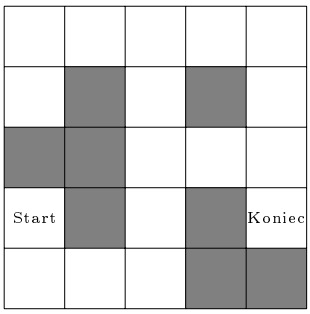
Informatyka, grupa 3ID11B

27 maja 2025

Spis treści

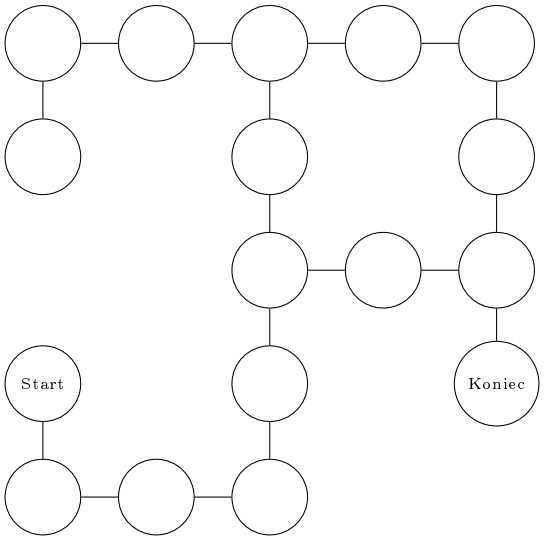
1 Wstęp

Celem projektu jest stworzenie aplikacji umożliwiającej generowanie dwuwymiarowego labiryntu oraz wizualizację procesu wyszukiwania ścieżki pomiędzy dwoma punktami. Labirynt w kontekście projektu to struktura siatki, gdzie każde pole może stanowić przejście lub ścianę.



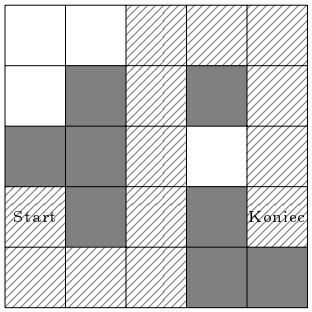
Rysunek 1: Przykładowy labirynt 5x5 z zaznaczonym startem i końcem, gdzie białe pole - przejście, czarne - ściana.

Można zauważyć, że taka struktura jest reprezentacją grafu, gdzie pola odpowiadają wierzchołkom, a krawędzie łączą sąsiadujące pola przejściowe.

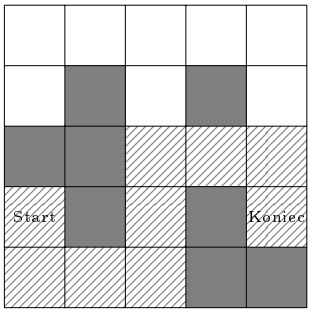


Rysunek 2: Graf reprezentujący labirynt z Rysunku 1.

W projekcie istotne jest porównanie różnych algorytmów wyszukiwania ścieżki, które pozwalają znaleźć trasę między dwoma punktami. W najlepszym przypadku celem jest znalezienie ścieżki *optymalnej*, czyli takiej, która minimalizuje liczbę kroków, co w kontekście grafu o jednakowych wagach krawędzi sprowadza się do znalezienia drogi o minimalnej długości.



(a) Ścieżka nieoptymalna



(b) Ścieżka optymalna

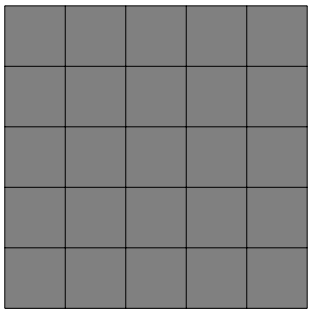
Rysunek 3: Porównanie ścieżek w labiryncie.

2 Generowanie labiryntu

2.1 Algorytm Prima

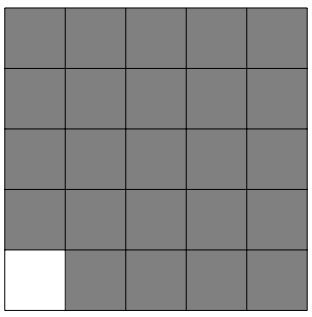
Algorytm Prima to metoda generowania labiryntów wykorzystująca technikę tworzenia minimalnego drzewa rozpinającego (MST) dla grafu reprezentującego planszę labiryntu. W kontekście generowania labiryntu działanie algorytmu przebiega następująco:

1. Na początku tworzona jest plansza, w której wszystkie pola są oznaczone jako ściany.



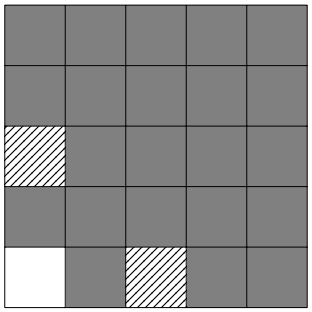
Rysunek 4: Cała plansza stanowi ściany.

2. Następnie wybierane jest losowe pole i oznaczane jako przejście.



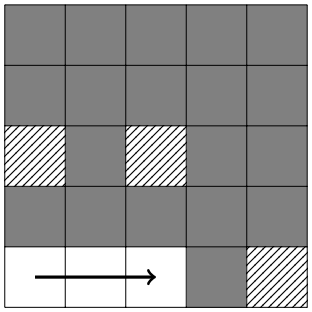
Rysunek 5: Losowe pole startowe.

3. Do zbioru krawędzi dodawane są sąsiednie komórki, do których można przejść bezpośrednio z miejsca startowego. Za sąsiednie uznaje się komórki oddalone o jedno pole w pionie lub poziomie. W ten sposób korytarze zawsze będą miały szerokość jednego pola.



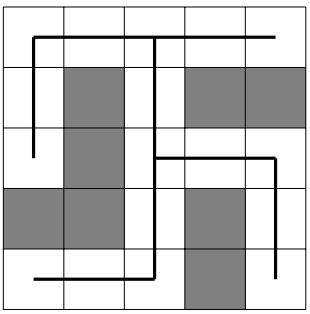
Rysunek 6: Wybór sąsiednich pól.

4. Losowana jest jedna krawędź ze zbioru. Jeśli prowadzi ona do nieodwiedzonego pola, to tworzy się przejście pomiędzy bieżącym polem a nowym (usuwana zostaje ściana między nimi), a nowe pole zostaje oznaczone jako przejście. Do zbioru krawędzi dodawani są sąsiedzi nowego pola.

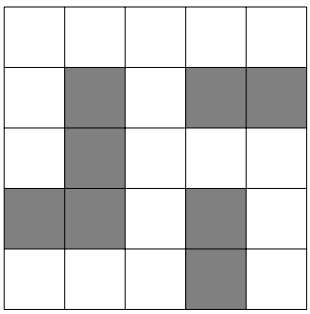


Rysunek 7: Utworzenie krawędzi do sąsiedniego pola.

5. Proces powtarza się, dopóki zbiór krawędzi nie będzie pusty.



(a) Wyznaczanie kolejnych krawędzi labiryntu.



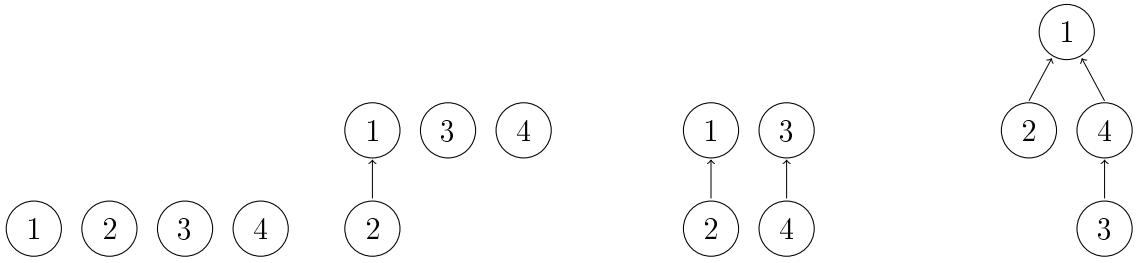
(b) Labirynt powstały na podstawie wyznaczonych krawędzi.

Rysunek 8: Kolejne kroki działania algorytmu.

2.2 Algorytm Kruskala

Algorytm Kruskala, podobnie jak algorytm Prima, opiera się na tworzeniu minimalnego drzewa rozpinającego. Kluczowym elementem jego działania jest wykorzystanie struktury danych *Disjoint Set* (zbiorów rozłącznych).

Struktura *Disjoint Set* reprezentuje rozłączne zbiory elementów za pomocą drzew. Na początku każdy element tworzy pojedynczy, jednoelementowy zbiór, którego reprezentantem jest korzeń drzewa. Główna operacja na tej strukturze, *union*, łączy dwa zbiory, tworząc jedno drzewo, w którym korzeń jednego zbioru staje się potomkiem korzenia drugiego. W ten sposób zbiory są scalane. Schematyczne przedstawienie scalania zbiorów znajduje się na Rysunku 9.

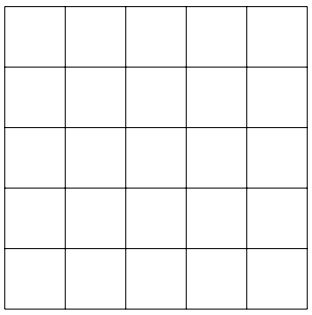


(a) Krok 1 (b) Krok 2 (c) Krok 3 (d) Krok 4

Rysunek 9: Kolejne kroki scalania zbioru

Generowanie labiryntu przebiega następująco:

1. Tworzona jest siatka, w której wszystkie pola są oznaczone jako przejścia.



Rysunek 10: Cała plansza stanowi przejścia.

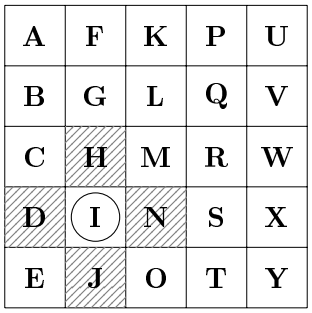
2. Każde pole planszy jest osobnym zbiorem w strukturze *Disjoint Set*. Na tym etapie żadna komórka nie jest jeszcze połączona z inną.

A	**F**	**K**	**P**	**U**
B	**G**	**L**	**Q**	**V**
C	**H**	**M**	**R**	**W**
D	**I**	**N**	**S**	**X**
E	**J**	**O**	**T**	**Y**

Rysunek 11: Każdy zbiór jest oznaczony unikalną literą.

3. Komórki planszy są rozpatrywane w losowej kolejności:

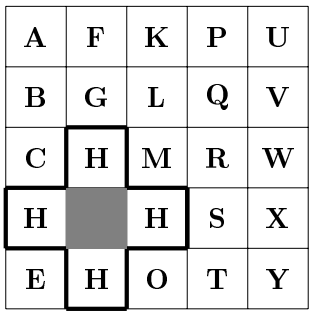
   (a) Dla wybranej komórki określa się bezpośrednich sąsiadów. W tym przypadku są to pola bezpośrednio przyległe w górę, dół, lewo lub prawo - inaczej niż przyjęto w algorytmie Prima.



Rysunek 12: Losowa komórka (oznaczona kółkiem) i jej sąsiedzi (zakreskowani).

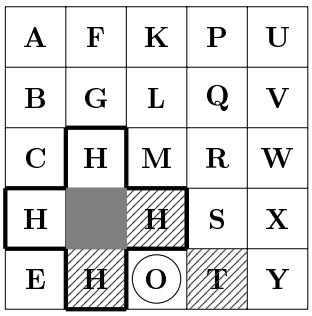
   (b) Jeśli wszyscy sąsiedzi należą do różnych zbiorów:

      i. Wylosowana komórka staje się ścianą.
      ii. Sąsiedzi zostają połączeni w jeden zbiór.

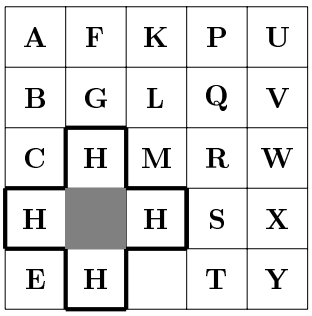


Rysunek 13: Łączenie komórek w zbiory. Reprezentantem zbioru jest pierwszy dodany element (tutaj komórka H), choć może to być dowolna komórka z grupy.

   (c) Jeśli chociaż dwaj sąsiedzi należą do tego samego zbioru, komórka nie zostaje oznaczona jako ściana – pozostaje przejściem.



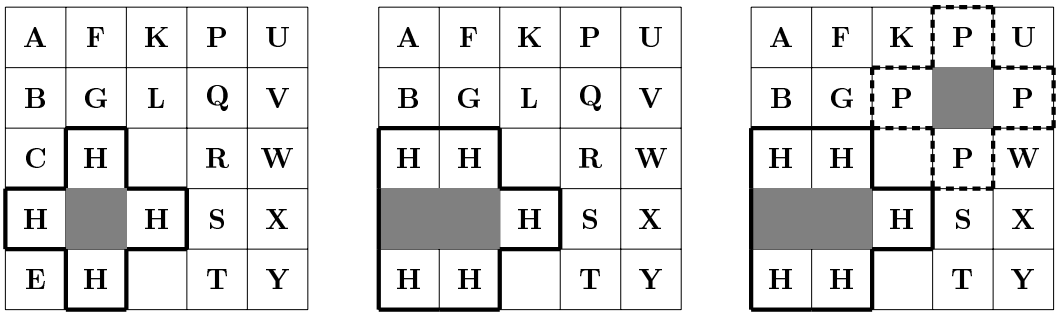
(a) Wylosowanie kolejnej komórki.



(b) Pozostawienie przejścia.

Rysunek 14: Wylosowana komórka ma już dwóch takich samych sąsiadów - scalanie nie następuje.

4. Proces powtarza się, aż każda komórka zostanie przetworzona.

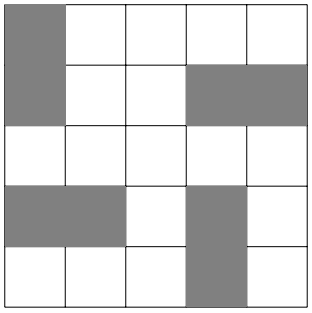


(a) Kolejna komórka bez unikalnych sąsiadów.

(b) Rozszerzenie połączonego zbioru.

(c) Scalenie kolejnego zbioru.

Rysunek 15: Przykładowe kolejne iteracje algorytmu.



Rysunek 16: Przykładowy wygląd końcowego, wygenerowanego labiryntu.