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

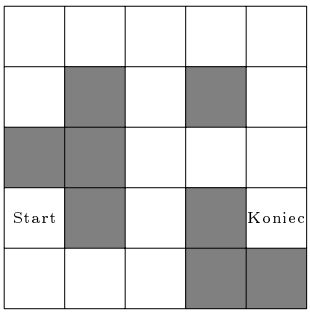
Informatyka, grupa 3ID11B

28 maja 2025

Spis treści

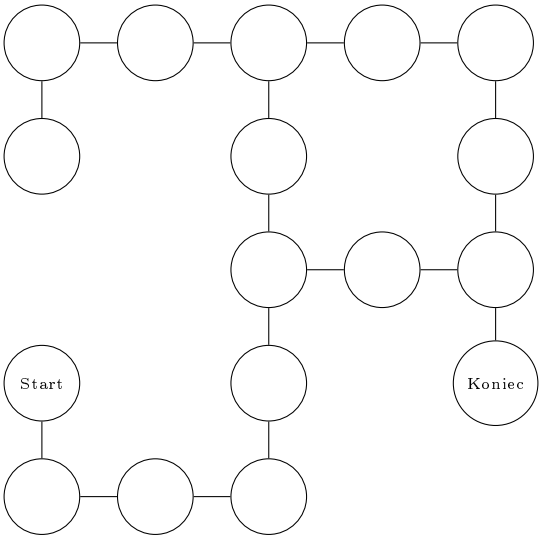
1 Wstęp

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. Przykładowy labirynt pokazano na Rysunku 1.



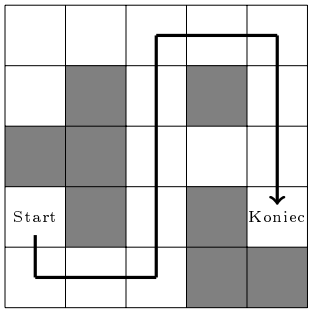
Rysunek 1: Przykładowy labirynt 5x5 z zaznaczonym startem i końcem, gdzie białe pole - przejście, czarne - ściana.

Można zauważyć, że taka struktura jest reprezentacją grafu, gdzie pola odpowiadają wierzchołkom, a krawędzie łączą sąsiadujące pola przejściowe. Reprezentacja labiryntu w formie grafu została przedstawiona na Rysunku 2.

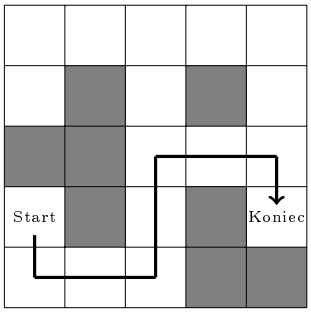


Rysunek 2: Graf reprezentujący labirynt z Rysunku 1.

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. Ścieżkę *optymalną* oraz *nieoptymalną* ukazuje Rysunek 3.



(a) Ścieżka nieoptymalna



(b) Ścieżka optymalna

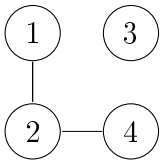
Rysunek 3: Porównanie ścieżek w labiryncie.

2 Generowanie labiryntu

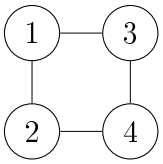
Do generowania labiryntów kluczowe jest zrozumienie idei drzew rozpinających.

Drzewo rozpinające [2] to taki podgraf oryginalnego grafu, który zawiera wszystkie jego wierzchołki, a jednocześnie jest spójny i nie zawiera cykli.

Spójność grafu [1] oznacza, że między każdą parą wierzchołków istnieje co najmniej jedna ścieżka, czyli można przejść z dowolnego wierzchołka do dowolnego innego, poruszając się po krawędziach grafu. Różnicę między grafem spójnym oraz niespójnym przedstawia Rysunek 4.



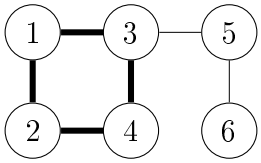
(a) Graf niespójny.



(b) Graf spójny.

Rysunek 4: Przykłady grafów spójnych i niespójnych.

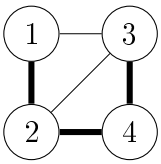
Cyklem [1] nazywamy ścieżkę zaczynającą się i kończącą w tym samym wierzchołku, w której żadna krawędź ani wierzchołek (poza początkiem i końcem) się nie powtarza. Obecność cykli oznacza, że można okrążyć pewien obszar grafu i wrócić do punktu startu inną drogą, co w kontekście labiryntu oznacza istnienie pętli. W drzewie rozpinającym takich pętli nie ma, co gwarantuje unikalną ścieżkę między dowolnymi dwoma wierzchołkami. Przykładowy graf zawierający cykl został pokazany na Rysunku 5.



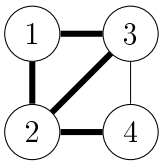
Rysunek 5: Graf zawierający cykl (pogrubiona linia).

Minimalne drzewo rozpinające [2] (ang. *MST, minimum spanning tree*) to takie drzewo rozpinające, dla którego suma wag krawędzi jest najmniejsza spośród wszystkich możliwych drzew rozpinających danego grafu. W grafach nieważonych, jak te stosowane

przy modelowaniu labiryntów, wszystkie krawędzie są traktowane jako równe, dlatego każde drzewo rozpinające o minimalnej liczbie krawędzi spełnia warunki *MST*. Przykładowe minimalne drzewo rozpinające dla grafu z równoważnymi krawędziami pokazuje Rysunek 6.



(a) Ścieżka będąca minimalnym drzewem rozpinającym (3 kroki).



(b) Ścieżka nie stanowiąca minimalnego drzewa rozpinającego (4 kroki).

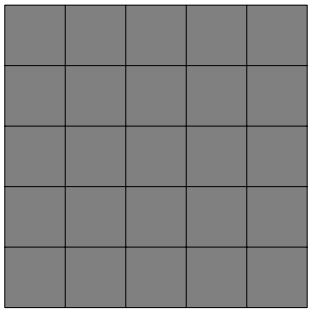
Rysunek 6: Ścieżki spełniające oraz nie spełniające założeń *MST* (pogrubione linie).

Algorytmy generujące labirynty sprowadzają się właśnie do wyznaczenia wspomnianego drzewa rozpinającego. Uzyskane drzewo określa, które krawędzie (ścieżki) są dostępne, a które stanowią ściany labiryntu.

2.1 Algorytm Prima

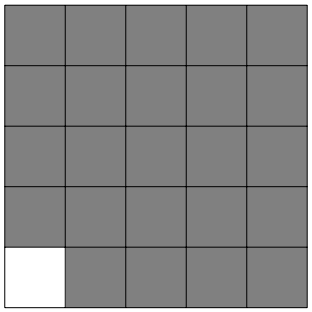
Algorytm Prima [2] w kontekście generowania labiryntu działa poprzez stopniowe budowanie połączeń między komórkami planszy. W każdej iteracji wybierana jest losowa krawędź prowadząca z odwiedzonej komórki do jednej z sąsiednich, jeszcze nieodwiedzonych. Wybrana komórka zostaje następnie połączona z dotychczasowym obszarem labiryntu. Proces ten powtarzany jest aż do momentu, gdy wszystkie komórki zostaną połączone, tworząc spójną strukturę bez cykli. Szczegółowy przebieg algorytmu wygląda następująco:

1. Na początku tworzona jest plansza, w której wszystkie komórki są oznaczone jako ściany. Rysunek 7 przedstawia początkowy układ planszy w całości wypełnionej ścianami.



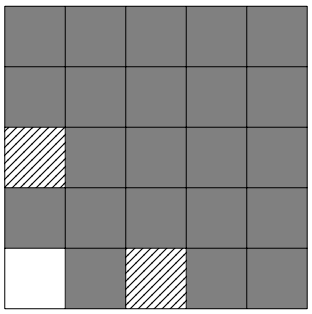
Rysunek 7: Cała plansza stanowi ściany.

2. Następnie wybierane jest losowe pole, które zostaje oznaczone jako przejście. Rysunek 8 przedstawia wybrane losowo pole startowe.



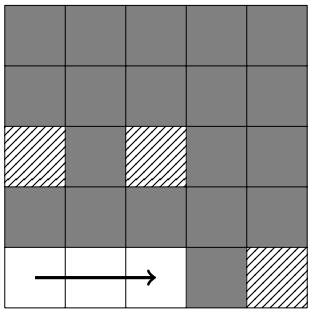
Rysunek 8: Losowe pole startowe.

3. Do zbioru krawędzi dodawane są sąsiednie komórki, do których można przejść bezpośrednio z pola startowego. Za sąsiednie uznaje się komórki oddalone o jedno pole w pionie lub poziomie. Ilustrację tego etapu przedstawiono na Rysunku 9.



Rysunek 9: Wybór sąsiednich pól.

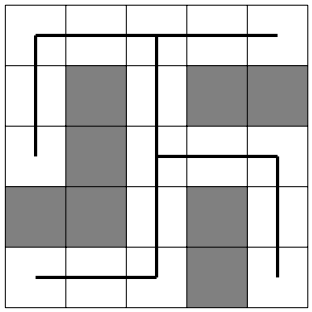
4. Losowana jest jedna krawędź ze zbioru potencjalnych przejść. Jeśli prowadzi ona do nieodwiedzonego pola, tworzy się przejście między bieżącym polem a nowym (usuwana jest ściana między nimi), a nowe pole zostaje oznaczone jako przejście. Następnie do zbioru krawędzi dodawani są sąsiedzi nowo odwiedzonego pola. Ilustrację tego etapu przedstawiono na Rysunku 10.



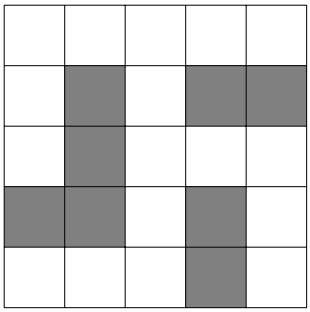
Rysunek 10: Utworzenie krawędzi do sąsiedniego pola.

5. Proces powtarza się, aż zbiór krawędzi stanie się pusty, co oznacza, że wszystkie komórki zostały połączone.

   Na Rysunku 11a przedstawiono wyznaczanie kolejnych krawędzi labiryntu, a na Rysunku 11b końcowy labirynt powstały na podstawie tych krawędzi.



(a) Wyznaczanie kolejnych krawędzi labiryntu.



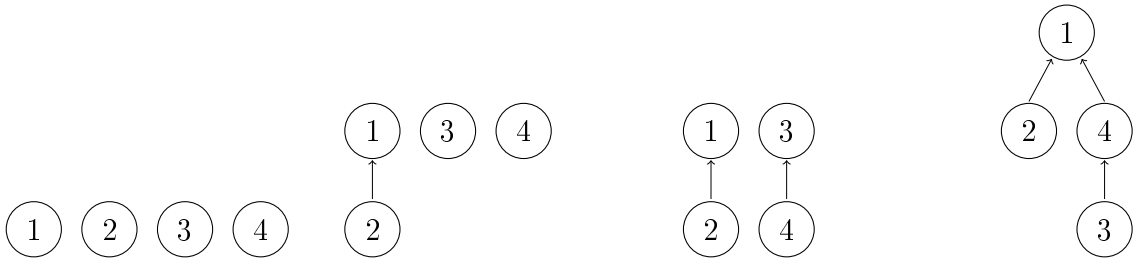
(b) Labirynt powstały na podstawie wyznaczonych krawędzi.

Rysunek 11: Kolejne kroki działania algorytmu.

2.2 Algorytm Kruskala

Algorytm Kruskala [2] do znalezienia minimalnego drzewa rozpinającego wykorzystuje strukturę zbiorów rozłącznych (ang. *Disjoint Set*).

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, *unia*, 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 12.



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

Rysunek 12: Kolejne kroki scalania zbioru

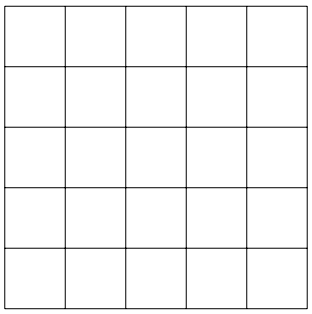
Projekt wykorzystuje zmodyfikowaną wersję algorytmu Kruskala. W klasycznym wariancie algorytm losowo wybiera krawędź, czyli parę sąsiadujących komórek, które mogą zostać połączone ścieżką. Jeśli komórki te należą do różnych zbiorów, są one łączone w jeden zbiór, a między nimi tworzone jest przejście.

W prezentowanej wersji algorytmu zamiast losowo wybierać krawędzie, iteruje się w losowej kolejności po wszystkich komórkach planszy. Dla każdej komórki rozpatrywani są jej sąsiedzi – jeśli należą do innych zbiorów, bieżąca komórka zostaje przekształcona w ścieżkę, a sąsiadujące zbiory zostają połączone.

Kluczową różnicą w porównaniu do klasycznego podejścia jest sposób zakończenia algorytmu. W tradycyjnej wersji wszystkie komórki zostają połączone w jeden wspólny zbiór. W tym przypadku natomiast, ze względu na lokalny charakter działania, istnieje wysokie prawdopodobieństwo, że w wyniku działania algorytmu pozostanie wiele niepołączonych zbiorów. Mimo to, końcowy układ labiryntu wizualnie przypomina ten uzyskany metodą klasyczną.

Poszczególne kroki generowania labiryntu przebiegają następująco:

1. Tworzona jest siatka, w której wszystkie pola są początkowo oznaczone jako przejścia. Warto zaznaczyć, że nie ma znaczenia, czy algorytm rozpoczyna z planszą wypełnioną przejściami, a następnie tworzy ściany, czy odwrotnie — z planszą wypełnioną ścianami, w której tworzone są przejścia. Efekt końcowy pozostaje taki sam. Przykład planszy z samymi przejściami przedstawiono na 13.



Rysunek 13: Cała plansza stanowi przejścia.

2. Każde pole planszy jest osobnym zbiorem w strukturze zbiorów rozłącznych. Na tym etapie żadna komórka nie jest jeszcze połączona z inną, co zostało przedstawione na Rysunku 14.

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 14: Każdy zbiór jest oznaczony unikalną literą.

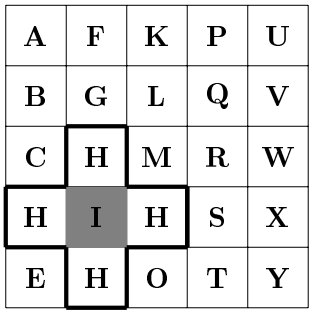
3. Komórki planszy są rozpatrywane w losowej kolejności:

   (a) Dla wybranej komórki określa się bezpośrednich sąsiadów. W tym przypadku są to pola bezpośrednio przyległe w górę, doł, lewo lub prawo - inaczej niż przyjęto w algorytmie Prima. Przykład takiej sytuacji przedstawiono na Rysunku 15.

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**

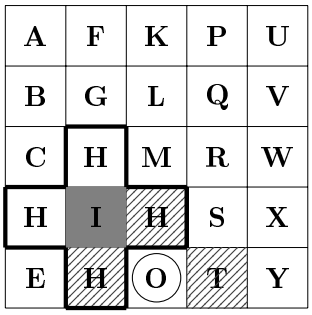
Rysunek 15: Losowa komórka (oznaczona kółkiem) i jej sąsiedzi (zakreskowani).

(b) Jeśli wszyscy sąsiedzi należą do różnych zbiorów to wylosowana komórka staje się ścianą, a jej sąsiedzi są łączeni w jeden zbiór. Schemat łączenia komórek w zbiory pokazano na Rysunku 16.

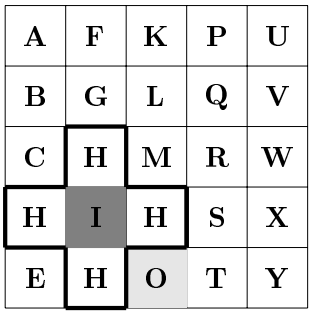


Rysunek 16: Łączenie komórek w zbiory. Reprezentantem zbioru jest pierwszy dodany element (tutaj komórka H), choć może to być dowolna komórka z grupy.

(c) Jeśli chociaż dwaj sąsiedzi należą do tego samego zbioru, komórka nie zostaje oznaczona jako ściana – pozostaje przejściem, co przedstawiono na Rysunku 17.



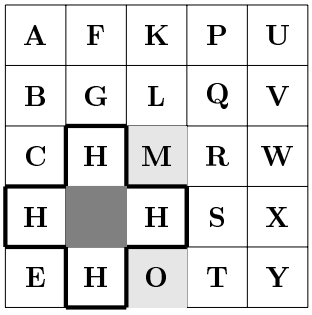
(a) Wylosowanie kolejnej komórki.



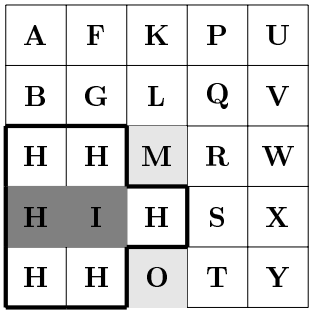
(b) Pozostawienie przejścia.

Rysunek 17: Wylosowana komórka ma już dwóch takich samych sąsiadów — scalanie nie następuje. Przejście poglądowo oznaczono jasnoszarym kolorem, aby zaznaczyć, że zostało odwiedzone.

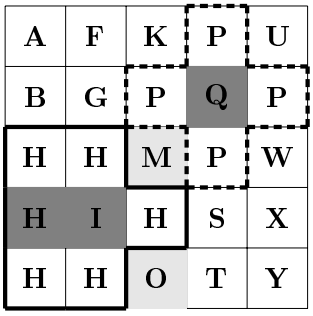
4. Proces powtarza się, aż każda komórka zostanie przetworzona. Przykładowe dalsze kroki algorytmu przedstawia Rysunek 18, a Rysunek 19 ukazuje potencjalny schemat wygenerowanego labiryntu.



(a) Kolejna komórka bez unikalnych sąsiadów.

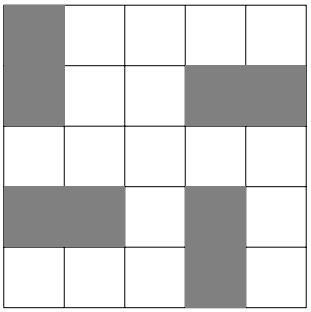


(b) Rozszerzenie połączonego zbioru.



(c) Scalenie kolejnego zbioru.

Rysunek 18: Przykładowe kolejne iteracje algorytmu.



Rysunek 19: Przykładowy wygląd końcowego, wygenerowanego labiryntu.

Literatura

[1] V. K. Balakrishnan. *Schaum's Outline of Theory and Problems of Graph Theory.* McGraw–Hill, New York, nachdr. edition, 2005.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 3rd edition, 2009.