Projekt – Atak z trzeciego wymiaru

Skład zespołu:

1. Kamil Krystowski (kierownik, wizualizacja, implementacja algorytmów do części graficznej)
2. Maciej Zajdel (implementacja algorytmów, autor dokumentacji)
3. Kacper Szymczyk (implementacja algorytmów, autor dokumentacji)

Implementacje wszystkich algorytmów wykorzystanych w naszym projekcie zostały zrobione w języku JavaScript. Część graficzna została zrobiona przy pomocy PixiJS. Poprawność rozwiązań zostały umieszczone na GitHubie.

Problem I. Ustalić, w jaki sposób są transportowane odcinki z fabryki do miejsca budowy płotu. Możliwie szybko i możliwie małym kosztem zbudować płot.

Specyfikacja problemu:

1. Transport odcinków z fabryki do miejsca budowy płotu:
   - Odcinki płotu są produkowane w fabryce i muszą być dostarczone do miejsca budowy płotu, gdzie zostaną zmontowane.
   - Konieczne jest ustalenie optymalnej trasy transportu odcinków z fabryki do punktów odbioru przy budowie płotu.
2. Budowa optymalnego płotu:
   - Płot musi zostać zbudowany wokół obszaru Krainy, wykorzystując punkty orientacyjne.
   - Konieczne jest zbudowanie płotu z odcinków o minimalnej łącznej długości, aby chronić obszar Krainy.
3. Dobór par tragarzy do wspólnego transportu odcinków:
   - W świecie dwuwymiarowym, gdzie niemożliwe jest przełożenie rąk (z przodu do tyłu oraz odwrotnie), do przeniesienia odcinka potrzebny jest tragarz z rękoma z tyłu oraz drugi z rękoma z przodu.
   - Po znalezieniu par tragarzy, które spełniają wymagania dotyczące rodzaju rąk, można zoptymalizować trasę transportu odcinków, minimalizując koszty i czas transportu oraz zapewniając odpowiednie umiejętności do montażu płotu.

Szczegóły wykorzystanych algorytmów:

1. Wykorzystanie BFS:
   - BFS wykorzystujemy do wyznaczenia najkrótszej ścieżki do każdego punktu odbioru, wskazując jednocześnie optymalną trasę transportu odcinków.
   - Rozpoczynamy od fabryki jako punktu startowego.
   - Przeszukiwanie poziomami: Dla każdego sąsiada (innego punktu), badamy najkrótszą ścieżkę.
   - Używamy kolejki do przechowywania sąsiadów punktów do zbadania. Na każdym poziomie dodajemy wszystkich sąsiadów do kolejki, aby je zbadać w kolejnych iteracjach.

- BFS kontynuuje przeszukiwanie, dopóki nie zostaną spełnione określone warunki zakończenia, takie jak przejście przez wszystkie poziomy grafu lub odnalezienie najkrótszych ścieżek do wszystkich punktów odbioru.
- Zapisywanie znalezionych ścieżek – podczas przeszukiwania zapisujemy najkrótsze ścieżki do każdego punktu odbioru.

2.Wykorzystanie algorytmu Grahama:

- Jeśli konieczne będzie określenie kształtu obszaru, który ma być ogrodzony, algorytm Grahama może zostać użyty do obliczenia otoczki wypukłej punktów orientacyjnych, które mają być połączone odcinkami płotu.
- Otoczka wypukła może być wykorzystana do określenia kształtu obszaru, który ma być ogrodzony, co może wpłynąć na planowanie trasy transportu odcinków z fabryki do miejsca budowy płotu oraz konstrukcję płotu.
- Algorytm składa się z kilku kroków, w których identyfikowane są punkty skrajne, a następnie sortowane względem kąta tworzonego z punktem referencyjnym. Następnie wybierane są punkty, które tworzą otoczkę wypukłą.
- Otoczka wypukła może pomóc w określeniu optymalnej konfiguracji płotu oraz minimalizacji jego długości poprzez eliminację zbędnych detali geometrycznych.

3. Wykorzystanie programowania dynamicznego do doboru par tragarzy:

- Algorytm dynamiczny jest stosowany do doboru tragarzy w pary w taki sposób, aby maksymalizować liczbę par, które mogą współpracować, biorąc pod uwagę ich preferencje.
- Algorytm wykorzystuje tablice dp i backtrack do przechowywania informacji o maksymalnej liczbie par oraz do śledzenia decyzji, które prowadzą do tego rozwiązania.
- Rozważane są dwa typy tragarzy: z rękoma z przodu (type1) i z rękoma z tyłu (type2). Algorytm iteruje przez obie grupy, próbując znaleźć jak najwięcej par, które spełniają warunek zgodności rąk oraz wzajemnej sympatii.

<u>Złożoność czasowa i pamięciowa rozwiązania:</u>

- Złożoność czasowa algorytmu BFS wynosi O(V + E), gdzie V oznacza liczbę wierzchołków (punktów) w grafie, a E to liczba krawędzi (połączeń między punktami). Złożoność pamięciowa algorytmu BFS wynosi O(V), gdzie V to liczba wierzchołków (punktów).
- Złożoność czasowa algorytmu Grahama do obliczania otoczki wypukłej wynosi O(n log n), gdzie n to liczba punktów wejściowych. Złożoność ta wynika z sortowania punktów względem kąta, co wymaga czasu O(n log n), oraz z kolejnych operacji wykonywanych na posortowanej liście punktów, które mają złożoność liniową. Złożoność pamięciowa algorytmu Grahama do obliczania otoczki wypukłej wynosi O(n), gdzie n to liczba punktów wejściowych.
- Złożoność czasowa algorytmu dynamicznego do doboru tragarzy wynosi O(m*n), gdzie m to liczba tragarzy z rękoma z przodu, a n to liczba tragarzy z rękoma z tyłu. Złożoność pamięciowa algorytmu dynamicznego do doboru tragarzy wynosi O(m*n), gdzie m to liczba tragarzy z rękoma z przodu, a n to liczba tragarzy z rękoma z tyłu.

Poprawność rozwiązania:

- BFS kończy się, gdy kolejka jest pusta, co oznacza, że wszystkie wierzchołki dostępne z punktu początkowego zostały odwiedzone i przetworzone.
- BFS jest częściowo poprawny, ponieważ zapewnia najkrótszą ścieżkę z punktu początkowego do dowolnego innego wierzchołka w grafie nieskierowanym i ważonym jednostkowo.
- W każdym kroku BFS odwiedza wierzchołki na danym poziomie głębokości przed przejściem do następnego poziomu, co gwarantuje, że najkrótsza ścieżka do każdego wierzchołka jest znaleziona, zanim BFS przejdzie do wierzchołków bardziej oddalonych.
- BFS jest całkowicie poprawny, ponieważ zawsze znajduje najkrótsze ścieżki do wszystkich wierzchołków osiągalnych z punktu początkowego.
- Gwarantowane jest, że każdy wierzchołek odwiedzany przez BFS zostanie odwiedzony dokładnie raz, a ścieżka do tego wierzchołka będzie najkrótsza możliwa, ponieważ BFS eksploruje wierzchołki w porządku rosnącej odległości od punktu początkowego.
- Całkowita poprawność wynika również z faktu, że kolejka BFS zapewnia przetwarzanie wierzchołków w kolejności, w jakiej zostały odkryte.
- Algorytm Grahama kończy się po przetworzeniu wszystkich punktów wejściowych i zbudowaniu otoczki wypukłej.
- Po zakończeniu sortowania punktów według kąta i przetworzeniu ich w porządku, otoczka wypukła jest kompletna.
- Algorytm jest częściowo poprawny, ponieważ poprawnie identyfikuje wszystkie punkty, które mogą należeć do otoczki wypukłej, i sortuje je względem kąta tworzonego z punktem referencyjnym (najczęściej najniższym punktu).
- W każdym kroku budowy otoczki wypukłej sprawdzane jest, czy dodanie nowego punktu tworzy wypukły kształt, co jest kluczowe dla poprawności otoczki.
- Algorytm Grahama jest całkowicie poprawny, ponieważ zawsze znajduje prawidłową otoczkę wypukłą dla danego zestawu punktów.
- Otoczka wypukła jest poprawna, gdy algorytm przeanalizuje wszystkie punkty wejściowe i utworzy minimalny wielokąt obejmujący wszystkie punkty.
- Całkowita poprawność wynika z faktu, że algorytm dokładnie przestrzega procedury dodawania punktów do otoczki tylko wtedy, gdy nie naruszają one wypukłości, co jest gwarantowane przez sprawdzenie kątów i usunięcie punktów tworzących wklęsłość.
- Algorytm kończy się po przetworzeniu wszystkich możliwych kombinacji tragarzy z rękoma z przodu i z tyłu.
- Po zakończeniu przetwarzania, dp[m][n] przechowuje maksymalną liczbę par tragarzy, które można utworzyć.
- Algorytm jest częściowo poprawny, ponieważ prawidłowo identyfikuje wszystkie pary tragarzy, które spełniają warunki zgodności rąk oraz wzajemnej sympatii.
- W każdym kroku algorytmu dynamicznego sprawdzane są możliwe kombinacje par tragarzy i aktualizowane wartości w tablicy dp.
- Algorytm jest całkowicie poprawny, ponieważ zawsze znajduje maksymalną liczbę par tragarzy, które można utworzyć z danej grupy.

- ❖ Poprawność wynika z systematycznego podejścia do sprawdzania i przechowywania maksymalnych wartości liczby par, które można utworzyć dla każdej możliwej kombinacji tragarzy.
- ❖ Całkowita poprawność wynika również z faktu, że algorytm dynamiczny zapewnia, że wszystkie możliwe kombinacje są sprawdzane i optymalizowane, co gwarantuje znalezienie najlepszego możliwego rozwiązania.

Opis implementacji wykorzystanych algorytmów w języku JavaScript:

W algorytmie Grahama używamy tablicy punktów do przechowywania współrzędnych wszystkich punktów wejściowych. Stos używany jest do przechowywania punktów tworzących otoczkę wypukłą. Algorytm zaczyna od znalezienia punktu z najniższą współrzędną y (jeśli kilka punktów ma tę samą współrzędną y, wybierany jest ten z najniższą współrzędną x). Punkt ten jest nazywany pivot. Pozostałe punkty są sortowane względem kąta nachylenia linii tworzonej z pivotem. Sortowanie odbywa się według funkcji compare, która używa funkcji orientation i distanceSquared. Po posortowaniu punktów algorytm zaczyna budowanie otoczki wypukłej. Pierwsze dwa punkty są dodawane do stosu. Następnie algorytm iteruje przez pozostałe punkty, sprawdzając, czy dodanie kolejnego punktu tworzy wypukły kształt. Jeśli nie, usuwa ostatni punkt ze stosu. Stos zawiera punkty otoczki wypukłej w kolejności przeciwnie do ruchu wskazówek zegara.

W algorytmie BFS zaimplementowano kolejkę oraz tablicę odwiedzonych wierzchołków (odwiedzone). Kolejka jest wykorzystywana do dodawania nowych wierzchołków do przetworzenia, natomiast tablica odwiedzone służy do zapamiętania, które wierzchołki już zostały odwiedzone. Algorytm zaczyna od utworzenia kolejki i dodania punktu startowego. Punkt startowy jest także dodawany do zbioru odwiedzonych. BFS działa w pętli, dopóki kolejka nie jest pusta. W każdym kroku wierzchołek jest usuwany z kolejki, a jego sąsiedzi są dodawani do kolejki, jeśli jeszcze nie zostali odwiedzeni. Algorytm sprawdza sąsiadów bieżącego wierzchołka (w górę, w dół, w lewo, w prawo) i dodaje ich do kolejki, jeśli są dostępni i spełniają określone warunki (np. nie są przeszkodą). Algorytm kończy się, gdy kolejka jest pusta, co oznacza, że wszystkie wierzchołki dostępne z punktu startowego zostały odwiedzone.

W algorytmie dynamicznym do doboru par tragarzy używamy tablicy dp do przechowywania maksymalnej liczby par możliwych do utworzenia dla każdej kombinacji liczby tragarzy z dwóch grup: z rękoma z przodu (type1) i z rękoma z tyłu (type2). Tablica dp jest dwuwymiarowa, gdzie dp[i][j] oznacza maksymalną liczbę par możliwą do utworzenia z pierwszych i tragarzy typu type1 i pierwszych j tragarzy typu type2. Algorytm dynamiczny iteruje przez wszystkie możliwe liczby tragarzy w obu grupach, aktualizując tablicę dp na podstawie tego, czy obecni tragarze mogą utworzyć parę. Na końcu, wartość dp[m][n] zawiera maksymalną liczbę par, które można utworzyć z dostępnych tragarzy.

Problem II. Zapisać opowieść-melodię w maszynie Informatyka, zamieniając wcześniej „poli” na „boli” oraz próbując oszczędzić wykorzystane miejsce. Znaleźć rozwiązanie problemu ewentualnej zamiany innych fragmentów opowieści-melodii, który niepokoi Heretyka oraz Informatyka.

Specyfikacja problemu:

1. Kompresja tekstu:
   - ✓ Zastosowanie algorytmu Huffmana do oszczędnego kodowania opowieści-melodii. Każdej literze przypisany zostaje unikalny ciąg 0-1 długości 5.
   - ✓ Wykorzystywane są najkrótsze kody dla najczęściej występujących liter, co pozwala na oszczędne kodowanie.
2. Zamiana fragmentów tekstu:
   - ✓ Zastosowanie algorytmu Knutha-Morrisa-Pratta (KMP) do szybkiego wyszukiwania wzorców.
   - ✓ Wyszukiwanie wzorca "poli" w skompresowanej opowieści-melodii.
   - ✓ Zamiana każdego znalezionego wystąpienia "poli" na "boli". Zachowanie spójności i sensu opowieści-melodii.

Szczegóły wykorzystanych algorytmów:

1. Wykorzystanie algorytmu Huffmana:
   - Pierwszym krokiem będzie analiza częstości występowania poszczególnych liter w opowieści-melodii.
   - Następnie, na podstawie tej analizy, zostanie zbudowane drzewo Huffmana, gdzie częstość występowania liter będzie określała ich pozycję w drzewie.
   - Litery o większej częstości występowania będą miały krótsze kody, co pozwoli oszczędzić miejsce przy zapisie opowieści-melodii.
   - Ostatecznie, opowieść-melodia zostanie zakodowana zgodnie z wygenerowanym drzewem Huffmana, a jej zapis zostanie umieszczony w maszynie Informatyka.
2. Wykorzystanie algorytmu Knutha-Morrisa-Pratta (KMP):
   - Tworzy się tablicę przejść, która przechowuje informacje o długości najdłuższego wspólnego prefiksu i sufiksu dla każdego prefiksu wzorca.
   - Porównuje się wzorzec z tekstem, wykorzystując tablicę przejść do efektywnego przesuwania wzorca.
   - Po znalezieniu wzorca w tekście, dokonuje się zamiany odpowiednich fragmentów zgodnie z wymaganiami problemu.

Złożoność czasowa i pamięciowa rozwiązania:

Algorytm Huffmana:

- Złożoność czasowa: O(n log n), gdzie n to liczba liter w opowieści-melodii.
- Złożoność pamięciowa: O(n), gdzie n to liczba liter w opowieści-melodii.

Algorytm Knutha-Morrisa-Pratta (KMP):

- Złożoność czasowa: O(n + m), gdzie n to długość opowieści-melodii, a m to długość wzorca ("poli").
- Złożoność pamięciowa: O(m), gdzie m to długość wzorca ("poli").

Poprawność rozwiązania:

- Algorytm Huffmana kończy się, gdy w liście węzłów zostanie tylko jeden węzeł, który jest korzeniem drzewa Huffmana.
- Rekurencyjna funkcja do generowania kodów kończy się, gdy wszystkie liście (symbole) zostaną odwiedzone.
- Algorytm zaczyna od utworzenia węzłów dla każdego symbolu i częstotliwości, a następnie iteracyjnie łączy dwa węzły o najmniejszych częstotliwościach, tworząc nowy węzeł będący ich rodzicem. Ten krok jest powtarzany, aż zostanie tylko jeden węzeł, który staje się korzeniem drzewa Huffmana.
- Po zbudowaniu drzewa Huffmana, algorytm przeszukuje drzewo w sposób rekurencyjny, generując kody Huffmana dla każdego symbolu na podstawie ścieżki od korzenia do liścia (lewo to '0', prawo to '1').
- Drzewo Huffmana jest poprawnie zbudowane, ponieważ w każdym kroku łączone są dwa węzły o najmniejszych częstotliwościach, co gwarantuje minimalną długość kodów wynikowych.
- Kody Huffmana są poprawnie generowane dla każdego symbolu, ponieważ algorytm przeszukuje całe drzewo, przypisując każdemu symbolowi unikalny kod na podstawie jego położenia w drzewie.
- Algorytm KMP kończy się, gdy wszystkie indeksy wzorca zostaną przetworzone.
- Algorytm KMP kończy się, gdy cały tekst zostanie przeszukany.
- Algorytm iteracyjnie oblicza długości najdłuższego prefiksu, który jest jednocześnie sufiksem dla każdego prefiksu wzorca, tworząc tablicę LPS (Longest Prefix Suffix).
- Algorytm przeszukuje tekst za pomocą wzorca, wykorzystując tablicę LPS do optymalizacji przeszukiwania, unikając ponownego przeszukiwania prefiksów wzorca.
- Tablica LPS jest poprawnie obliczona, ponieważ dla każdego prefiksu wzorca algorytm poprawnie identyfikuje długość najdłuższego prefiksu, który jest jednocześnie sufiksem.
- Algorytm poprawnie identyfikuje wszystkie wystąpienia wzorca w tekście i zamienia je na nowy wzorzec, wykorzystując tablicę LPS do optymalizacji przeszukiwania i zamiany.

Implementacja wykorzystanych algorytmów w języku JavaScript:

Do implementacji algorytmu Huffmana wykorzystujemy drzewo binarne, które reprezentuje drzewo kodowania. Struktura danych Wezel reprezentuje węzeł drzewa, a klasa Huffman odpowiada za budowę tego drzewa i generowanie kodów dla poszczególnych symboli. Struktura danych Wezel zawiera informacje o symbolu, częstotliwości wystąpienia, oraz wskaźniki na lewe i prawe dziecko.

Klasa Huffman odpowiada za budowanie drzewa kodowania. Wykorzystuje tablicę do przechowywania węzłów. Podczas budowy drzewa wykorzystuje się kolejki priorytetowe, np. tablicę posortowaną wg częstotliwości symboli.

Do implementacji algorytmu Knutha-Morrisa-Pratta wykorzystujemy tablicę przejść (lps - Longest Prefix Suffix), która przechowuje długość najdłuższego wspólnego prefiksu i sufiksu

dla każdego prefiksu wzorca. Wykorzystujemy również dwie zmienne i i j, aby poruszać się po tekście i wzorcu. Algorytm KMP nie korzysta z dodatkowych struktur danych.
Tablica przejść lps przechowuje informacje o długości najdłuższego wspólnego prefiksu i sufiksu dla każdego prefiksu wzorca, jest obliczana przy użyciu funkcji pomocniczej obliczanieTablicyPrzejsc.
Zmienne i odpowiada za indeks w tekście, j odpowiada za indeks w wzorcu

Problem III. Ustalić jak najszybciej grafik pracy strażników i jak najmniejszą liczbę odsłuchań melodii dla każdego strażnika.

Specyfikacja problemu:

- ✓ Tragarze mają być dobierani w pary na podstawie ich energii.
- ✓ W każdym dniu należy wybrać tragarzy z największym poziomem energii.
- ✓ Tragarze przemieszczają się zgodnie z ruchem wskazówek zegara pomiędzy punktami orientacyjnymi o losowej jasności.
- ✓ Jeśli tragarz osiągnie punkt jaśniejszy niż poprzedni, musi się zatrzymać, aby odsłuchać melodię.
- ✓ Celem jest minimalizacja liczby odsłuchań melodii przez tragarzy.

Szczegóły wykorzystanych algorytmów:

1. Wykorzystanie algorytmu zachłannego:
   - Wybór tragarza: Sortowanie tragarzy na podstawie poziomu energii i wybór tego z najwyższym poziomem.
   - Wybór punktów zatrzymania: Tragarze są zatrzymywani w jaśniejszych punktach, aby odsłuchać melodię.

Złożoność czasowa i pamięciowa rozwiązania:

- Złożoność czasowa funkcji wynosi O(n log n), gdzie n to liczba tragarzy.
- Złożoność pamięciowa wynosi O(n), gdzie n to liczba tragarzy.

Poprawność rozwiązania:

- ❖ Algorytm kończy swoje działanie, gdy wszyscy tragarze zostaną przydzieleni do pracy jako strażnicy, wszystkie trasy strażników zostaną wyznaczone, biorąc pod uwagę punkty jasności, zostanie minimalizowana liczba odsłuchań melodii, co jest głównym celem problemu.
- ❖ Tragarze są sortowani według poziomu energii w kolejności malejącej, co gwarantuje, że zawsze wybierany jest tragarz z największą energią. Ta operacja jest poprawna, ponieważ wykorzystuje metodę sort, która prawidłowo porównuje wartości energii.
- ❖ Po posortowaniu tragarzy, ci z najwyższą energią są wybierani do pracy jako strażnicy. To zapewnia, że wybierani są ci tragarze, którzy mają najwięcej energii na początku, co minimalizuje potrzebę odpoczynku.
- ❖ Tragarze zatrzymują się w punktach o jaśniejszej jasności, aby odsłuchać melodię. Ta operacja jest poprawna, ponieważ punkty jasności są porównywane zgodnie z założeniami problemu.

Implementacja wykorzystanych algorytmów w języku JavaScript:

Funkcja generateSchedule sortuje tragarzy na podstawie poziomu energii w kolejności malejącej. Następnie, wybiera tragarzy o największej energii do pracy jako strażnicy. Po dotarciu do punktu jaśniejszego niż poprzedni, strażnik musi się zatrzymać i odsłuchać melodię. W wyniku, liczba odsłuchań melodii jest minimalizowana.

HARMONOGRAM REALIZACJI PROJEKTU

Faza 1: Implementacja algorytmów (marzec, kwiecień)

- ✓ Implementacja algorytmu BFS: 10.03 – 20.03,
- ✓ Implementacja algorytmu Grahama: 15.03 – 30.03,
- ✓ Implementacja algorytmu KMP: 1.04 – 7.04,
- ✓ Implementacja algorytmu Huffmana: 1.04 – 11.04,
- ✓ Implementacja algorytmu dynamicznego: 11.04 – 20.04,
- ✓ Implementacja algorytmu zachłannego: 9.04 – 13.04,
- ✓ Implementacja algorytmu Bresenhama: 9.04 – 29.04,
- ✓ Implementacja algorytmu dynamicznego: 11.04 – 18.04.

Faza 2: Opracowanie grafiki i animacji, dokumentacja (kwiecień, maj, czerwiec)

- ✓ Projektowanie grafiki: 16.04 – 31.05
- ✓ Integracja grafiki z algorytmami: 16.04 – 15.06
- ✓ Rozpoczęcie tworzenia wstępnej dokumentacji: 19.04

Faza 3: Przygotowanie prezentacji i kończenie dokumentacji (czerwiec)

- ✓ Kończenie dokumentacji algorytmów – 17.06
- ✓ Testowanie i poprawki – 1.06 – 17.06

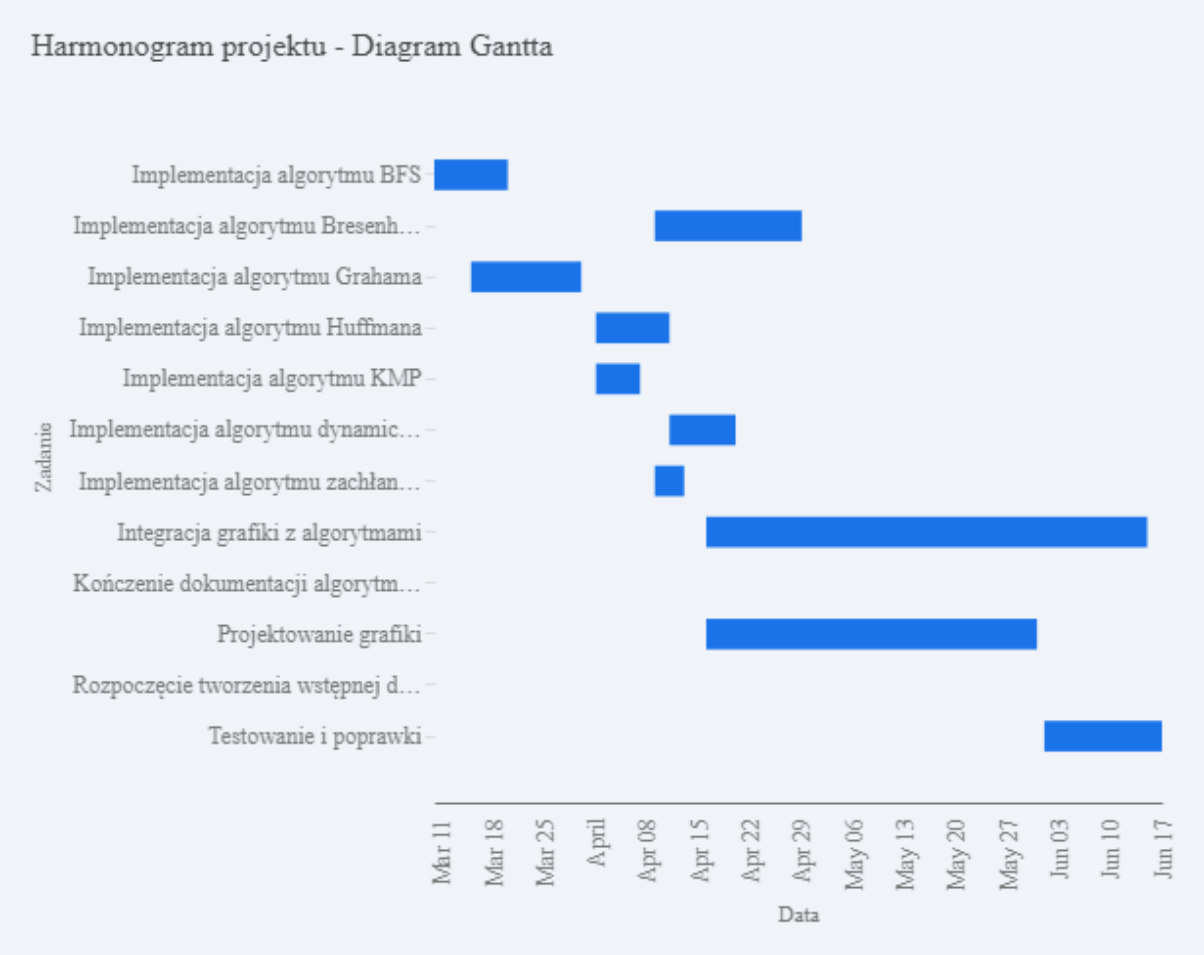

TABELA ALGORYTMÓW

L.p.	***Specyfikacja problemu (dane i wyniki)***	***Do jakich treści w zadaniu odnosi się algorytm***	***Zastosowane struktury danych***	**Informacje o zastosowanym algorytmie**
1	Transport odcinków do budowy płotu	Transport odcinków z fabryki do miejsca budowy płotu	Kolejka, Zbiór, Tablica	Algorytm BFS: Używany do znajdowania najkrótszej ścieżki, wskazując jednocześnie optymalną trasę transportu odcinków.
2	Budowa optymalnego płotu	Minimalny płot wokół Krainy, wykorzystując punkty orientacyjne jako wierzchołki	Stos, Lista	Algorytm Grahama: Zbudowanie płotu z otrzymanej otoczki wypukłej oraz minimalizacja jego długości.
3	Dobór tragarzy w pary	Przydzielanie tragarzy z rękoma z przodu i z tyłu w pary	Tablica dwuwymiarowa	Algorytm dynamiczny: Używany do znajdowania maksymalnej liczby par tragarzy, aktualizując tablicę dp na podstawie możliwych kombinacji tragarzy z dwóch grup, maksymalizując liczbę możliwych do utworzenia par.
4	Oszczędne kodowanie opowieści-melodii	Zapisanie opowieści-melodii na maszynie z ograniczoną pamięcią	Drzewo binarne, Tablice	Algorytm Huffman: Do kompresji danych, zmniejszając liczby bitów potrzebnych do reprezentacji danych przez wykorzystanie zmiennej długości kodów dla różnych znaków.
5	Zamiana fragmentów opowieści-melodii	Identyfikacja i zamiana "poli" na "boli" w opowieści-melodii	Stringi, Tablice znaków	Algorytm Knutha-Morrisa-Pratta (KMP): Do szybkiego wyszukiwania wzorców, wykorzystując tablicę przejść (lps - Longest Prefix Suffix).
6	Optymalizacja pracy strażników	Grafik pracy strażników	Tablice	Algorytm zachłanny: Wybór tragarza z największą energią do pracy jako strażnik, wybierając lokalnie najlepszą opcję w każdym kroku.
7	Wizualizacja ścieżek	Wyznaczanie ścieżek pomiędzy dwoma punktami, zaznaczając piksele, które tworzą najkrótszą trasę na siatce	Tablica punktów, Lista punktów, Współrzędne punktów	Algorytm Bresenhama: Do rysowania linii w grafice komputerowej, pozwala na rysowanie linii między dwoma punktami na siatce pikseli, iterując po punktach na siatce pikseli i

				wybierając kolejne piksele, które najlepiej przybliżają teoretyczną linię łączącą dwa punkty.