Projekt - AiSD II – Atak z trzeciego wymiaru

Skład zespołu:

1. Kamil Krystowski (kierownik)
2. Maciej Zajdel
3. Kacper Szymczyk

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 algorytmu Dijkstry:
   - Stworzenie grafu reprezentującego sieć dróg łączących fabrykę z punktami odbioru odcinków. Wierzchołki grafu to punkty odbioru, a krawędzie to drogi łączące je z fabryką i między sobą.
   - Oznaczenie wag krawędzi długością odpowiadającą danemu odcinkowi drogi. Uruchomienie algorytmu Dijkstry z fabryki jako punktu początkowego.
   - Algorytm wyznaczy najkrótsze ścieżki do każdego punktu odbioru, wskazując jednocześnie optymalną trasę transportu odcinków.
2. Wykorzystanie algorytmu Kruskala:
   - Stworzenie grafu reprezentującego punkty orientacyjne, które mogą być połączone odcinkami płotu. Wierzchołki grafu to punkty orientacyjne, a krawędzie to potencjalne odcinki płotu łączące je.

- Oznaczenie wag krawędzi długością odpowiadającą odległości między punktami orientacyjnymi.
- Uruchomienie algorytmu Kruskala, który wyznaczy minimalne drzewo rozpinające graf. Drzewo rozpinające określa optymalną konfigurację płotu, minimalizując jego długość.

3.Wykorzystanie DFS/BFS:

- W kontekście doboru par tragarzy, BFS może być wykorzystany do przeszukiwania możliwych par na kolejnych poziomach grafu.
- Rozpoczęcie od jednego tragarza, wybieranie jednego tragarza jako punkt startowy.
- Przeszukiwanie poziomami, rozpoczynanie przeszukiwania od wybranego tragarza. Dla każdego sąsiada (innego tragarza), zbadać możliwe pary na tym samym poziomie w grafie.
- Używanie kolejki do przechowywania sąsiadów tragarzy 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 odpowiedniej liczby par tragarzy.
- Zapisywanie znalezionych par, podczas przeszukiwania zapisujemy znalezione pary tragarzy.

4.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.

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

- Algorytm Dijkstry ma złożoność czasową O(E log V), gdzie E to liczba krawędzi, a V to liczba wierzchołków w grafie.
- Algorytm Kruskala ma złożoność czasową O(E log V)gdzie E to liczba krawędzi, a V to liczba wierzchołków w grafie.
- Złożoność pamięciowa obu algorytmów jest liniowa względem liczby wierzchołków w grafie.
- Jeśli reprezentujemy graf za pomocą listy sąsiedztwa, złożoność pamięciowa BFS będzie wynosić O(n + e), gdzie n to liczba wierzchołków (tragarzy), a e to liczba krawędzi (możliwe połączenia między tragarzami).

- 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.

Implementacja wykorzystanych algorytmów w języku JavaScript:
Algorytm Dijkstry wykorzystuje tablicę do przechowywania grafu w postaci listy sąsiedztwa oraz kolejki priorytetowej do sortowania wierzchołków według ich odległości od wierzchołka startowego.
Algorytm Kruskala wykorzystuje tablicę do przechowywania krawędzi grafu oraz tablice rodziców i rang do reprezentacji zbiorów rozłącznych.
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.

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.
3.Optymalizacja miejsca na maszynie:
   - ✓ Problem plecakowy zakłada wybór odpowiednich fragmentów opowieści-melodii, które mają zostać zapisane w ograniczonej przestrzeni, tak aby maksymalizować liczbę zapisanych fragmentów przy zachowaniu ograniczenia dotyczącego dostępnej przestrzeni.

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.

3.Wykorzystanie algorytmu plecakowego:
   - Selekcja fragmentów opowieści-melodii (przedmiotów) w taki sposób, aby ich łączna wartość (liczba zapisanych fragmentów) była jak największa, przy jednoczesnym zachowaniu ograniczenia dotyczącego dostępnej przestrzeni (maksymalna pojemność maszyny Informatyka).
   - Wartość każdego przedmiotu odpowiada jego znaczeniu dla sensu i spójności opowieści, a waga przedmiotu może być interpretowana jako jego wielkość po skompresowaniu.

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

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").

Algorytm plecakowy:
- Złożoność czasowa: O(nW), gdzie n to liczba fragmentów opowieści-melodii, a W to maksymalna pojemność maszyny Informatyka.
- Złożoność pamięciowa: O(n*W), gdzie n to liczba przedmiotów, a W to pojemność plecaka

<u>Implementacja wykorzystanych algorytmów w języku JavaScript:</u>

1.Algorytm Huffmana:

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.

2. Algorytm KMP:

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 i j:

- 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:

- Strażnicy mają przemierzać płot wokół Krainy, wybierając punkty orientacyjne zgodnie z ruchem wskazówek zegara.
- Każdy strażnik może być wybrany spośród płaszczaków o kolejnych numerach, przy czym wybierany jest ten, który ma najwięcej energii.
- Strażnik po minięciu pewnej liczby punktów orientacyjnych musi się zatrzymać, by rozejrzeć się dookoła. Jednakże może zachować całą energię tylko wtedy, gdy poprzedni punkt zatrzymania był jaśniejszy od aktualnego. W przeciwnym razie traci całą energię i musi odpocząć.
- Celem jest ustalenie grafiku pracy strażników na cały tydzień, minimalizując liczbę odsłuchań melodii przez każdego z nich.

Szczegóły wykorzystanych algorytmów:

1. Wykorzystanie algorytmu zachłannego:
   - wybiera lokalnie najlepszą opcję z nadzieją, że doprowadzi to do globalnie optymalnego rozwiązania
   - wybór strażnika (funkcja wybiera strażnika z najwyższym poziomem energii)
   - wybór punktów zatrzymania (funkcja przydziela strażnikowi punkty zatrzymania zgodnie z ograniczeniem liczby odsłuchań melodii)

Złożoność czasowa i pamięciowa:

- Całkowita złożoność czasowa głównej funkcji wynosi O(d * n), gdzie d to liczba dni, a n to liczba płaszczaków.
- Złożoność pamięciowa wynosi O(d), gdzie d to liczba dni.

Implementacja wykorzystanych algorytmów w języku JavaScript:

Funkcja chooseGuard przeszukuje tablicę energyLevels w celu znalezienia płaszczaka z najwyższym poziomem energii. Oznacza wybranego strażnika jako wykorzystanego, ustawiając jego energię na -1.

Funkcja chooseCheckpoints tworzy tablicę punktów orientacyjnych, w których strażnik musi się zatrzymać. Punkt zatrzymania wybierany jest zgodnie z ograniczeniem melodyTolerance i liczbą punktów orientacyjnych.

Funkcja scheduleGuards generuje grafik pracy strażników na określoną liczbę dni.

Dla każdego dnia wybiera strażnika i przypisuje mu punkty zatrzymania. Wynikowy grafik pracy jest przechowywany w tablicy schedule.

Wyświetlany jest grafik pracy strażników, pokazując który strażnik pracuje którego dnia oraz jego punkty zatrzymania. Kod wykorzystuje tablice jako główną strukturę danych do przechowywania poziomów energii, punktów orientacyjnych oraz grafiku pracy strażników.

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	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
4	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)
5	Optymalizacja pracy strażników	Grafik pracy strażników z minimalną liczbą odsłuchań melodii	Tablice	Algorytm zachłanny: Wybiera lokalnie najlepszą opcję wyboru strażnika z najwyższym poziomem energii i opcje wyboru punktów zatrzymania zgodnie z ograniczeniem liczby odsłuchań melodii.
6	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.