Transport w Shire

TREŚĆ

1.O PROJEKCIE:

Głównym celem projektu jest optymalizacja transportu piwa w krainie Shire, tak aby każdy mieszkaniec był zadowolony i otrzymał swoją należną porcję trunku. Projekt zakłada stworzenie efektywnej sieci przepływu surowca (jęczmień) od pól uprawnych, przez browary, aż do karczm, minimalizując przy tym koszty oraz kolejno dostarczając zamówienia

Autorzy (Zespół 7): Michał Łepkowski, Maciej Czepiel, Ruslan Semianenka, Danylo Strilchyk.

Prowadzący: T.Grzona

Uczelnia : UMK WMiI

2.Specyfikacja

2.1 System odpowiada za efektywny transport jęczmienia w krainie Shire. Główne cele:

- **Maksymalizacja przepływu** – jak największa ilość piwa powinna trafić do karczm.
- **Minimalizacja kosztów** – przy zachowaniu tego przepływu, należy wybrać drogi o jak najniższym koszcie naprawy.
- **Uwzględnienie ćwiartek geograficznych** – wydajność pól zależy od lokalizacji (A–D).
- **Rozdzielenie zasobów** – każdy węzeł w grafie istnieje jako dwa wierzchołki:
- `(nazwa, jęczmień)` oraz `(nazwa, piwo)`.
- Tylko piwo może dotrzeć do SUPERSINK, co zapewnia poprawność logiczną przepływu.

2.2 System wykorzystuje struktury grafowe i algorytmy przepływu:

- **Algorytm Edmondsa-Karpa** – do obliczenia maksymalnego przepływu.
- **Algorytm Bellmana-Forda** – do optymalizacji kosztu przepływu.
- **Algorytm ray-casting** – do przypisywania pól do ćwiartek (na podstawie współrzędnych).

3. Klasy

3.1 Point

3.1.1 Opis klasy: Klasa Point reprezentuje punkt w przestrzeni dwuwymiarowej (x, y). Używana do zapisu pozycji węzłów na mapie Shire.

3.1.2 Zmienne

Nazwa zmiennej	Typ danych	Opis
x	double	współrzędna X punktu
y	double	współrzędna Y punktu

3.1.3 Funkcje/Metody

Nazwa funcji	Modyfikator	Typ zwracany	Parametry	Opis
Point	-(konstruktor)	-(brak)	double x , double y	Ustawia wartości współrzędnych
ToString	public	String	-	Zwraca punkt jako `(x, y)`

3.2 Node

3.2.1 Opis Klasy: Klasa Node reprezentuje węzeł w grafie (np. pole, browar, karczma, skrzyżowanie, itp.). Przechowuje informacje o typie węzła, jego pozycji, wydajności oraz przypisanej ćwiartce. W identyfikatorze węzła może pojawić się rozszerzenie "jęczmień" lub "piwo", oznaczające zasób transportowany w tej warstwie grafu.

3.2.2 Zmienne

Nazwa zmiennej	Typ danych	Opis
id	String	Unikalny identyfikator węzła
type	NodeType	Typ węzła (POLE, BROWAR, itd.)
wydajnosc	double	Wydajność węzła (np. ilość jęczmienia)
position	Point	Pozycja na mapie
cwiartka	String	Nazwa przypisanej ćwiartki
wspolczynnikWydajnosci	double	Mnożnik wydajności zależny od ćwiartki

3.2.3 Funkcje/Metody

Nazwa Funkcji	Modyfikator	Typ zwracany	Parametry	Opis
Node	-(konstruktor)	-(brak)	String id,NodeType type,double wydajnosc,Point position	Ustawia dane węzła
toString	public	String	-	Zwraca opis tekstowy węzła

3.3 Edge

3.3.1 Opis klasy: Klasa `Edge` reprezentuje skierowaną krawędź w grafie — połączenie między dwoma węzłami z określoną przepustowością, kosztem i aktualnym przepływem.

3.3.2 Zmienne

Nazwa zmiennej	Typ danych	Opis
from	Node	Węzeł początkowy
to	Node	Węzeł końcowy
capacity	double	Maksymalna ilość zasobu do przesłania
cost	double	Koszt przesyłu jednej jednostki zasobu
flow	double	Aktualny przepływ

3.3.3 Funkcje/Metody

Nazwa funkcji	Modyfikator	Typ zwracany	Parametry	Opis
Edge	-(konstruktor)	-(brak)	Node from, Node to, double capacity, double cost	Ustawia dane krawędzi, domyślny `flow = 0`
toString	public	String	-	Zwraca opis tekstowy krawędzi

3.4 ResidualEdge

3.4.1 Opis klasy: Klasa `ResidualEdge`

reprezentuje krawędź rezydualną, używaną podczas minimalizacji kosztów w grafie rezydualnym. Odnosi się do oryginalnej krawędzi `Edge`.

3.4.2 Zmienne

Nazwa zmiennej	**Typ danych**	**Opis**
from	Node	Węzeł początkowy
to	Node	Węzeł końcowy
capacity	double	Przepustowość rezydualna
cost	double	Koszt przesyłu (ujemny dla krawędzi odwrotnej)
originalEdge	Edge	Oryginalna krawędź, której dotyczy ta krawędź rezydualna

3.4.3 Funkcje / Metody

Nazwa funkcji	**modyfikator**	**Typ zwracany**	**Parametry**	**Opis**
ResidualEdge	-(konstruktor)	-(brak)	Node from, Node to, double capacity, double cost, Edge originalEdge	Tworzy krawędź rezydualną na podstawie krawędzi głównej

3.5 Graph

3.5.1 Opis klasy: Klasa `Graph` reprezentuje cały graf transportowy w systemie. Przechowuje węzły, krawędzie, ćwiartki i operacje potrzebne do zarządzania przepływem i strukturą grafu.

3.5.2 Zmienne

Nazwa zmiennej	Typ danych	Opis
nodes	List<Node>	Lista wszystkich węzłów
edges	List<Edge>	Lista wszystkich krawędzi
adjacencyList	Map<Node,List<Edge>>	Lista sąsiedztwa
cwiartki	List<polygon>	Lista ćwiartek (jako wielokąty)
wspolczynnikiCwiartek	Map<Integer,double>	Współczynniki wydajności przypisane do ćwiartek

3.5.3 Funkcje/Metody

Nazwa funkcji	Modyfikator	Typ zwracany	Parametry	Opis
addNode	public	void	Node node	Dodaje węzeł do grafu
addEdge	public	void	Edge edge	Dodaje krawędź do grafu
assignCwiartki	public	void	-	Przypisuje ćwiartki do węzłów na podstawie pozycji
validateCwiartki	public	void	-	Sprawdza, czy każde POLE i BROWAR mają przypisaną ćwiartkę
updateSourceEdgesCapacities	public	void	-	Dostosowuje przepustowości między SUPERSOURCE a polami
isPointInPolygon	public	private	Point point, Polygon polygon	Sprawdza, czy punkt należy do danego wielokąta
printGraph	public	void	-	Wypisuje pełny stan grafu: węzły, krawędzie, ćwiartki

3.6 EdgeUser

3.6.1 Opis klasy: Klasa EdgeUser pełni rolę pomocniczej struktury

danych, wykorzystywanej do przechowywania informacji o wykorzystaniu krawędzi w trakcie działania algorytmów przepływu.

Jest szczególnie przydatna w:

- analizie wyników (np. ile zasobu faktycznie przepłynęło przez daną krawędź),
- wizualizacji przepływu (oddzielenie przepływu jęczmienia od piwa),
- testowaniu poprawności logiki transportowej.

Dzięki przechowywaniu informacji o źródle, celu, ilości przepływu oraz statusie użycia, umożliwia dokładne śledzenie aktywności w grafie.

3.6.2 Zmienne

Nazwa zmiennej	Typ danych	Opis
from	NodeUser	Węzeł początkowy
to	NodeUser	Wężeł końcowy
capacity_jeczmien	double	Pojemność dla jęczmienia
capacity_piwo	double	Pojemność dla piwa
cost	double	Koszt przesyłu
flow	double	Aktualny przepływ

3.6.3 Funkcje/Metody

Nazwa funkcji	Modyfikator	Typ zwracany	Parametry	Opis
EdgeUser	-(konstruktor)	-(brak)	int from ,int to,int flow,boolean used	Inicjalizuje obiekt z danymi dotyczącymi przeplywu
toString()	public	String	-	Zwraca tekstową reprezentację obiektu

3.7 NodeUser

3.7.1 Opis klasy: Klasa NodeUser przechowuje dane pomocnicze związane z analizą przepływu przez poszczególne węzły. Umożliwia m.in. wizualizację sumy przepływu wchodzącego i wychodzącego oraz śledzenie roli danego węzła w przepływie.

Nazwa zmiennej	**Typ danych**	**Opis**
id	String	Unikalny identyfikator węzła
type	NodeType	Typ logiczny węzła
wydajnosc	double	Bazowa wydajność
position	Point	Pozycja na mapie
cwiartka	String	Nazwa przepisanej ćwiartki
wspolczynnikWydajnosci	double	Mnożnik wydajności wynikający z ćwiartki

3.8 GraphVisualizer

3.8.1 Opis klasy:Klasa GraphVisualizer odpowiada za graficzną prezentację struktury grafu oraz wizualizację przepływu zasobów w systemie.

Jej celem jest dostarczenie intuicyjnego i uproszczonego widoku, który ułatwia:

- analizę działania algorytmów przepływu,
- weryfikację poprawności przepustowości,
- sprawdzenie zgodności między wydajnością a ilością przesyłanego zasobu,
- prezentację wyników końcowych w sposób wizualny.

Wizualizacja ukrywa wewnętrzną reprezentację z podwójnymi węzłami (jęczmień/piwo), skupiając się na czytelnym pokazaniu oryginalnych węzłów i ich powiązań.

3.8.2 Funkcjonalności

- Wyświetlanie oryginalnych węzłów (bez podziału na zasoby)
- Krawędzie oznaczone przepływem:
  - osobno dla **jęczmienia** i **piwa**
- Przepustowość z SUPERSOURCE do pól uwzględnia współczynnik ćwiartki
- Interfejs graficzny oparty na bibliotece **JavaFX**

3.8.3 Metody główne

Nazwa funkcji	Modyfikator	Typ zwracany	Parametry	Opis
paintComponent	public	void	Graphics g	Główna metoda rysująca węzły i krawędzie na panelu
drawNode	private	void	Graphics g,Node	Rysuje pojedynczy węzeł wraz z nazwą
drawEdge	private	void	Graphics g,Edge	Rysuje krawędź i wyświetla przepływ jęczmienia/piwa
setGraphData	public	void	List<Node> List<Edge>	Ustawia dane grafu do wizualizacji

3.9 Main.java

3.9.1 Opis klasy:Klasa Main pełni rolę głównego kontrolera aplikacji. Odpowiada za:

- załadowanie danych wejściowych z pliku graph.json
- zbudowanie struktury grafu
- uruchomienie algorytmów przepływu i minimalizacji kosztu
- uruchomienie graficznej wizualizacji

Chociaż funkcje obliczeniowe i strukturalne znajdują się w innych klasach (Graph, ResidualGraph, itd.), to Main koordynuje przebieg działania programu.

3.9.2 Zmienne

Nazwa zmiennej	Typ danych	Opis
filename	String	Ścieżka do pliku .txt z danymi wejściowymi
graph	Graph	Obiekt grafu zbudowany na podstawie pliku wejściowego

3.9.3 Funkcje

Nazwa funkcji	Modyfikator	Typ zwracany	Parametry	Opis
main	public	void	String[] args	Główna metoda uruchamiająca program
loadGraphFromFile	public	Graph	String filename	Wczytuje plik .txt i przekazuje dane do klasy Graph
run	private	void	Graph g	Uruchamia algorytmy przepływu i minimalizacji kosztu
visualize	private	void	Graph g	Otwiera okno z wizualizacją grafu

3.9.4 Opis funkcji loadGraphFromFile

Metoda loadGraphFromFile odczytuje plik .txt z danymi wejściowymi i przekazuje je do klasy Graph, gdzie tworzona jest struktura grafu.

Plik wejściowy zawiera:

- listę wierzchołków (nodes) z ich typami i współrzędnymi
- listę krawędzi (edges) z przepustowościami i kosztami
- dane o ćwiartkach (regions) oraz ich współczynnikach

Metoda ta odpowiada za:

- inicjalizację obiektu grafu
- ustawienie przepustowości źródła
- przetworzenie danych wejściowych do wewnętrznego modelu

4.Algorytmy

4.1 Algorytm Edmondsa-Karpa

Opis: Algorytm wyznacza maksymalny przepływ w grafie. Bazuje na przeszukiwaniu wszerz (BFS), które znajduje ścieżki powiększające w sieci rezydualnej.

Funkcja

Nazwa funkcji	Modyfikator	Typ zwracany	Parametry	Opis
edmondsKarp	static	double	Graph graph, Node source, Node sink	Zwraca maksymalny przepływ od źródła do ujścia

Kroki działania :

1. Inicjalizacja przepływu = 0
2. Tworzenie sieci rezydualnej
3. BFS w celu znalezienia ścieżki powiększającej
4. Wyznaczenie bottlenecka
5. Aktualizacja przepływów i krawędzi odwrotnych
6. Powtórz, aż brak ścieżek

4.2 Algorytm Belmana-Fordu(minimalizacja kosztu)

Opis: Po wyznaczeniu maksymalnego przepływu optymalizujemy jego koszt. Szukamy cykli ujemnych w grafie rezydualnym i przesuwamy przepływ przez tańsze ścieżki.

Dane dotyczące wykorzystania krawędzi oraz przepływów są zapisywane dodatkowo w strukturach pomocniczych EdgeUser i NodeUser. Umożliwia to późniejszą wizualizację i analizę kosztową przepływu.

Funkcja

Nazwa Funkcji	Modyfikator	Typ zwracany	Parametry	Opis
minimizeCostFlow	static	void	Graph graph, Node source, Node sink	Optymalizuje koszt przepływu bez jego zmniejszania

Kroki działania:

1. Budowa grafu rezydualnego z kosztami (ujemne dla odwrotnych krawędzi)
2. Relaksacja krawędzi przez `|V|-1` iteracji
3. Detekcja cykli ujemnych
4. Przesunięcie przepływu przez te cykle
5. Powtarzaj do braku cykli

5. **Poprawność rozwiązania**

5.1 **Algorytm Edmondsa-Karpa**

a)Kompletność:

Algorytm wykorzystuje przeszukiwanie wszerz (BFS), aby znajdować ścieżki powiększające w sieci rezydualnej. Przepływ zwiększany jest dopóki istnieją możliwe ścieżki.

b)Poprawność:

W każdej iteracji aktualizowany jest przepływ na krawędziach oraz dodawane są krawędzie odwrotne, co pozwala na cofanie przepływu, jeśli znajdą się lepsze ścieżki.

c)Zakończenie:

Algorytm kończy działanie, gdy nie można już znaleźć żadnej ścieżki powiększającej – wtedy uzyskany przepływ jest maksymalny.

5.2 **Algorytm Bellmana-Forda**

a)Kompletność:

Bellman-Ford wykonuje |V| - 1 iteracji relaksacji wszystkich krawędzi. Pozwala to na znalezienie ścieżek o najmniejszym koszcie w grafie rezydualnym.

b)Poprawność:

Algorytm wykrywa cykle ujemnego kosztu, które można wykorzystać do poprawienia istniejącego przepływu, przesuwając go przez tańsze ścieżki.

c)Zakończenie:

Działanie kończy się, gdy nie ma już żadnych cykli o ujemnym koszcie, które mogłyby zoptymalizować koszt całkowity.

5.3 Obsługa ćwiartek

Przypisanie ćwiartek:

Węzły POLE i BROWAR przypisywane są do ćwiartek na podstawie współrzędnych. Algorytm isPointInPolygon *sprawdza, czy dany punkt znajduje się wewnątrz danego wielokąta.*

Modyfikacja wydajności:

Każda ćwiartka ma przypisany współczynnik, który wpływa na efektywną wydajność pola. Przykład: ćwiartka A (×1.5), D (×0.8) itd.

5.4 Weryfikacja poprawności danych wejściowych

System sprawdza, czy wszystkie węzły typu POLE i BROWAR mają przypisaną ćwiartkę.

W przeciwnym razie generowane są komunikaty ostrzegawcze.

5.5 Rozdzielenie zasobów (jęczmień/piwo)

Ograniczenie logiczne:

Dzięki podwojeniu węzłów — (id, jęczmień) i (id, piwo) — system wymusza rozdzielenie transportu surowca i gotowego produktu.

Produkcja tylko w browarach:

Jedynym sposobem przejścia z jęczmienia do piwa są specjalne krawędzie w browarach:(browar, jęczmień) → (browar, piwo) z wydajnością browaru.

SUPERSINK otrzymuje wyłącznie piwo:

Dzięki tej konstrukcji, do ujścia systemu może fizycznie trafić tylko piwo – nigdy surowiec. Zapewnia to pełną poprawność logiczną modelu.

6. Wizualizacja grafu

6.1 Widok uproszczony

System zawiera graficzny komponent odpowiedzialny za przedstawienie grafu przepływu w sposób czytelny i przejrzysty.

Zamiast prezentować szczegółową strukturę z podwójnymi węzłami (np. P1_jęczmień, P1_piwo), interfejs pokazuje tylko oryginalne węzły, które reprezentują rzeczywiste obiekty w świecie Shire: pola, browary, karczmy, skrzyżowania.

Każda krawędź w grafie wizualizacji zawiera:

- informację o ilości przesyłanego **jęczmienia**
- informację o ilości przesyłanego **piwa**

6.2 Elementy wizualizacji

Nadpisy przy krawędziach:

Przy każdej krawędzi pokazana jest liczba jednostek przepływu osobno dla jęczmienia i piwa. Pozwala to szybko sprawdzić, czy przesył odbywa się zgodnie z założeniami.

Oznaczenie browarów:

Nazwa browaru.

Przepustowości z SUPERSOURCE:

Dostosowane do efektywnej wydajności pola, uwzględniając współczynnik ćwiartki. Przepustowość SRC → POLE = bazowa_wydajność × współczynnik_ćwiartki.

Technologia:

Wizualizacja zrealizowana przy użyciu biblioteki JavaFX jako osobna klasa GraphVisualizer, z funkcjami rysującymi krawędzie i węzły na panelu graficznym.

7 .Podsumowanie

7.1 Co osiągnęliśmy:

- Zbudowaliśmy model przepływu surowców i produktu (jęczmień → piwo).
- Zaimplementowaliśmy dwa kluczowe algorytmy:
  - **Edmonds-Karp** — maksymalny przepływ.
  - **Bellman-Ford** — minimalizacja kosztu.
- Ujęliśmy wpływ ćwiartek geograficznych na wydajność pól.
- Opracowaliśmy czytelną wizualizację działania systemu.
- Przetestowaliśmy system na danych z pliku wejściowego.

7.2 Wnioski:

- Separacja zasobów (jęczmień/piwo) pozwala zachować pełną poprawność modelu.
- Współczynniki ćwiartek dodają warstwę realizmu i optymalizacji.
- Model można łatwo rozbudować o inne zasoby, priorytety czy czasy dostawy.