ShireCraft - Dokumentacja

Wprowadzenie

W odpowiedzi na potrzeby burmistrza Samwise’a powstał projekt „ShireCraft” – narzędzie wspierające zarządzanie zasobami i logistyką produkcji piwa w Shire.

Produkt pozwala m.in. na:

- wygodne wczytywanie, zapisywanie, tworzenie, wizualizację i weryfikację sieci dróg krainy, wykorzystując intuicyjny interfejs graficzny;
- obliczanie i wizualizację najkorzystniejszej trasy dla podanej mapy połączeń, uwzględniając wydajności pól, pojemności browarów, nośności dróg oraz koszty ich potencjalnej naprawy;
- wyszukiwanie na różne sposoby fraz w raportach analizy trasy;
- kompresję i dekompresję plików.
- generację częściowo losowych sieci dróg

Informacje o zespole

Grupa laboratoryjna: **LE** (jedyny zespół w tej grupie)

Skład zespołu i role

Osoba	Rola	Zadania
Dominika Karbowiak	Programistka, testerka, autorka dokumentacji	Implementacja kompresji/dekompresji danych metodą Huffmana i testowanie, eksport danych, testowanie i poprawki modułu wyszukiwania, pomoc przy rozbudowie analizatora przepływu, testy jednostkowe
Mateusz Wójciak	Inżynier modelu grafowego, Programista grafów przepływowych	Implementacja struktur danych i algorytmu maksymalnego przepływu, testowanie i rozszerzenie analizatora o koszty (Min-Cost Max-Flow)
Karol Ławicki	Koordynator zespołu, programista GUI, integrator, autor dokumentacji technicznej	Planowanie i koordynacja prac projektowych, projekt i implementacja interfejsu graficznego, integracja frontendu z backendem, dokumentacja techniczna, pomoc przy algorytmach geometrycznych (m.in. wypukła otoczka), ogólna pomoc w różnych częściach projektu
Jakub Klonowski	Projektant danych wejściowych, specjalista pomocniczy	Projektowanie danych wejściowych, generowanie mapy miasta, integracja danych wejściowych z pozostałymi modułami, implementacja obsługi ćwiartek (algorytmów geometrycznych), budowa drzewa Huffmana
Weronika Gburek	Programistka, testerka, autorka dokumentacji i prezentacji	Implementacja algorytmów wyszukiwania wzorców (naiwny, Rabin-Karp, KMP, Boyer-Moore), import danych, przygotowanie prezentacji i dokumentacji, pomoc przy rozbudowie analizatora przepływu, testy wyszukiwarki

Interpretacja problemu

Obiekty w systemie:

- **Pole uprawne**: punkt na mapie z informacją o ilości produkowanego jęczmienia.
- **Browar**: punkt docelowy dostaw jęczmienia, z określoną maksymalną przepustowością.
- **Karczma**: punkt docelowy dostaw piwa.
- **Skrzyżowanie**: punkt na mapie bez wyszczególnionej funkcjonalności.
- **Droga**: krawędź skierowana grafu z określoną przepustowością oraz kosztem naprawy.
- **Ćwiartka**: wielokąt wypukły, wyznaczający produkcję jęczmienia dla każdego pola w swoim obszarze.
- **Punkt graniczny**: punkt będący wierzchołkiem swojej ćwiartki.

Interpretacja problemu

Założenia:

- Przepływ idzie jednego dnia.
- Jedną drogą mogą być przepuszczane rożne jednostki towaru na raz. Jeżeli towar przechodzi w jednej trasie (ścieżce powiększającej) tą samą drogą wiele razy, wykorzystywany przepływ to maksimum dla wszystkich przejść przez daną drogę (np. gdy zwiększy się po przejściu przez browar).
- Wszystkie browary mają jednolitą wydajność przetwarzania.
- Każdy browar ma ustaloną pojemność. Po jej przekroczeniu browar nie przetwarza więcej jęczmienia, ale towar może być przepuszczany dalej.
- Ilość jęczmienia zależy od położenia pola w odpowiedniej ćwiartce Shire.
- Ćwiartki są ze sobą rozłączne (nie kolidują ze sobą, jedna nie styka się z krawędzią drugiej).
- Drogi mogą wymagać naprawy; koszt naprawy jest brany pod uwagę przy wyznaczaniu tras (dla każdej drogi tylko raz).

Poniższe założenia wynikają ze sposobu implementacji projektu oraz ustaleń zespołu:

- Każde pole musi znajdować się wewnątrz ćwiartki.
- Wszystkie wartości zmiennoprzecinkowe ograniczone są do dwóch miejsc po przecinku.

Zawartość rozwiązania

Język: C++, C# (interfejsy graficzne)
Środowisko: Visual Studio, Unity (GUI), Windows Presentation Foundation (GUI generatora)
Kompatybilność: wyłącznie systemy Windows (10 lub nowszy)

Rozwiązanie składa się z sześciu modułów:

1. Analizator przepływu - program w C++ wyznaczający maksymalny przepływ o minimalnym koszcie na podstawie pliku z danymi wejściowymi.
2. Wyszukiwarka tekstu – program w C++ wyszukujący wystąpienia wzorca w podanym pliku tekstowym.
3. Archiwizator plików - program w C++ umożliwiający kompresję i dekompresję plików.
4. Generator map – program w C++ generujący pliki wejściowe zgodne z założeniami i specyfikacją o wybranym rozmiarze.
5. Interfejs graficzny generatora – aplikacja okienkowa w WPF (C#), pozwalająca na przyjazne dla użytkownika wprowadzanie danych do generatora.
6. Interfejs główny - aplikacja w Unity (C#), zintegrowana z pięcioma ww. modułami, pozwalająca również na wygodne tworzenie i wizualizację danych wejściowych oraz wizualizację danych wyjściowych.

Moduł 1. – analizator przepływu

Analizator wyznacza maksymalny przepływ o minimalnym koszcie, używając algorytmu Edmonds'a-Karp'a oraz Busacker'a-Gowen'a, na podstawie pliku z danymi wejściowymi w formacie wynikającym z treści problemu.

Ścieżka: ShireCraft_Data/StreamingAssets/PathAnalyzer.exe

Aby program zadziałał, wymagany jest plik z danymi wejściowymi zgodnymi ze specyfikacją, w formacie przedstawionym na następnej stronie.

Parametry uruchomienia:
1. Ścieżka do pliku z danymi wejściowymi.
2. Ścieżka do pliku z danymi wyjściowymi (zostanie tam utworzony i zapisany lub nadpisany).

Np.: PathAnalyzer.exe C:\wej.txt C:\wyj.txt

Uwaga:

• Analizator zakłada, że plik z danymi wejściowymi będzie zgodny ze specyfikacją – nie posiada całkowitej weryfikacji przekazywanych danych – zapewnieniem poprawności formatu zajmuje się główny interfejs graficzny.

Moduł 1. – analizator przepływu

Szablon danych wejściowych analizatora

KONWERSJA
konwersja

PUNKTY
id_punktu_1 X_1 Y_1 typ_1 pojemność
[...]

DROGI
id_od_1 id_do_1 $przepływ_1$ $koszt_1$
[...]

CWIARTKI
$wartość\_ćwiartki_1$ x_{11} y_{11} x_{12} y_{12} x_{13} y_{13} [...]
$wartość\_ćwiartki_2$ x_{21} y_{21} x_{22} y_{22} x_{23} y_{23} [...]
[...]

Legenda:
KONWERSJA, PUNKTY, DROGI, CWIARTKI - słowa-znaczniki rozdzielające listy danych;
konwersja – nieujemna wartość zmiennoprzecinkowa; ilość piwa (w tonach), którą otrzymuje się z tony jęczmienia
id_punktu – unikalna wartość całkowita dodatnia, zaczyna się od 1, z każdym następnym wystąpieniem rośnie o 1;
X, Y / x, y - położenie punktu: X / x – na płaszczyźnie poziomej, Y / y – na płaszczyźnie pionowej; typ zmiennoprzecinkowy;
typ - słowo w języku polskim: "pole", "browar", "karczma", "brak" (skrzyżowanie dla danych wejś.);
pojemność - dodatnia liczba zmiennoprzecinkowa; występuje tylko, gdy typ to “browar”; jest określoną ilością jęczmienia, jaką dany browar może przetworzyć;
id_od, id_do – wartości odnoszące się do id_punktu, oznaczające odpowiednio: z którego punktu prowadzi droga oraz do którego punktu prowadzi droga;
przepływ - wartość nieujemna zmiennoprzecinkowa (float), oznacza “ilość zboża oraz piwa, którą można przewieźć” przez daną drogę,

koszt - koszt naprawy drogi (liczba naturalna).
wartość_ćwiartki - “ilości jęczmienia wyrastające na jednym polu” w danej
[...] - “i tak dalej”; indeksy zwiększają się o 1 lub według widocznego wzoru
Uwaga: Wszystkie wartości "zmiennoprzecinkowe" ograniczone są do dwóch miejsc po przecinku.

Moduł 1. – analizator przepływu

Szablon danych wejściowych analizatora

KONWERSJA
konwersja

PUNKTY
id_punktu_1 X_1 Y_1 typ_1 pojemność
[...]

DROGI
id_od_1 id_do_1 $przepływ_1$ $koszt_1$
[...]

CWIARTKI
$wartość\_ćwiartki_1$ x_{11} y_{11} x_{12} y_{12} x_{13} y_{13} [...]
$wartość\_ćwiartki_2$ x_{21} y_{21} x_{22} y_{22} x_{23} y_{23} [...]
[...]

Przykład:

```
KONWERSJA
0,5

PUNKTY
0 0 -2 pole
1 0 2 pole
2 4 -2 browar 30
3 4 2 browar 25
4 8 0 brak
5 12 -2 karczma
6 12 2 karczma

DROGI
0 2 30 4
0 3 25 3
1 2 30 2
1 3 25 1
2 4 17 6
3 4 13 5
4 5 15 8
4 6 20 7

CWIARTKI
30 -1 3 -1 -3 2 -3 2 3
```

Moduł 1. – analizator przepływu

Program przetwarza otrzymane dane wejściowe tak, aby zachować strukturę grafu przepływowego do obliczeń.
Interpretacja mapy w danych wejściowych:

- Każda linijka w części PUNKTY to oddzielny obiekt - wierzchołek grafu o unikalnym ID.
- Każda linijka w części DROGI to oddzielny obiekt - strzałka grafu łącząca dwa wierzchołki.
- Każda linijka w części CWIARTKI to wielokąt wypukły złożony z punktów o współrzędnych podanych parami.

Kiedy analizator zparsuje plik z danymi wejściowymi (przykład na następnej stronie):

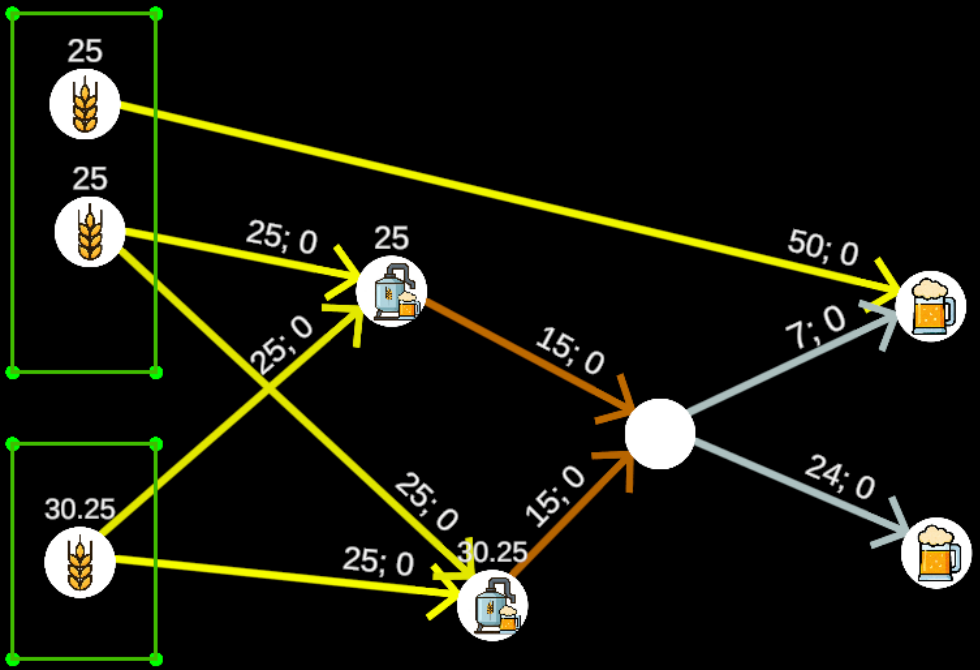
1) Zrobi z nich graf, a następnie zrobi kopię tego grafu.
2) Kopię duplikuje.
3) Doda źródło, które prowadzi z kosztem zerowym do każdego pola z pierwszej części, a wydajności pól przeniesie na przepływ nowo dodanych strzałek.
4) Doda ujście, do którego prowadzą wszystkie karczmy z drugiej części. Przepływy nowych strzałek będą wystarczająco duże, żeby wszystko przepuścić, natomiast koszty będą zerowe.
5) Browary z pierwszej części będą prowadziły do swoich sobowtórów. Pojemność każdego browaru zostanie przeniesiona na nowo utworzone przejście. W ten sposób zapewnimy, że trasa zawsze będzie przebiegać przez browar.

Moduł 1. – analizator przepływu

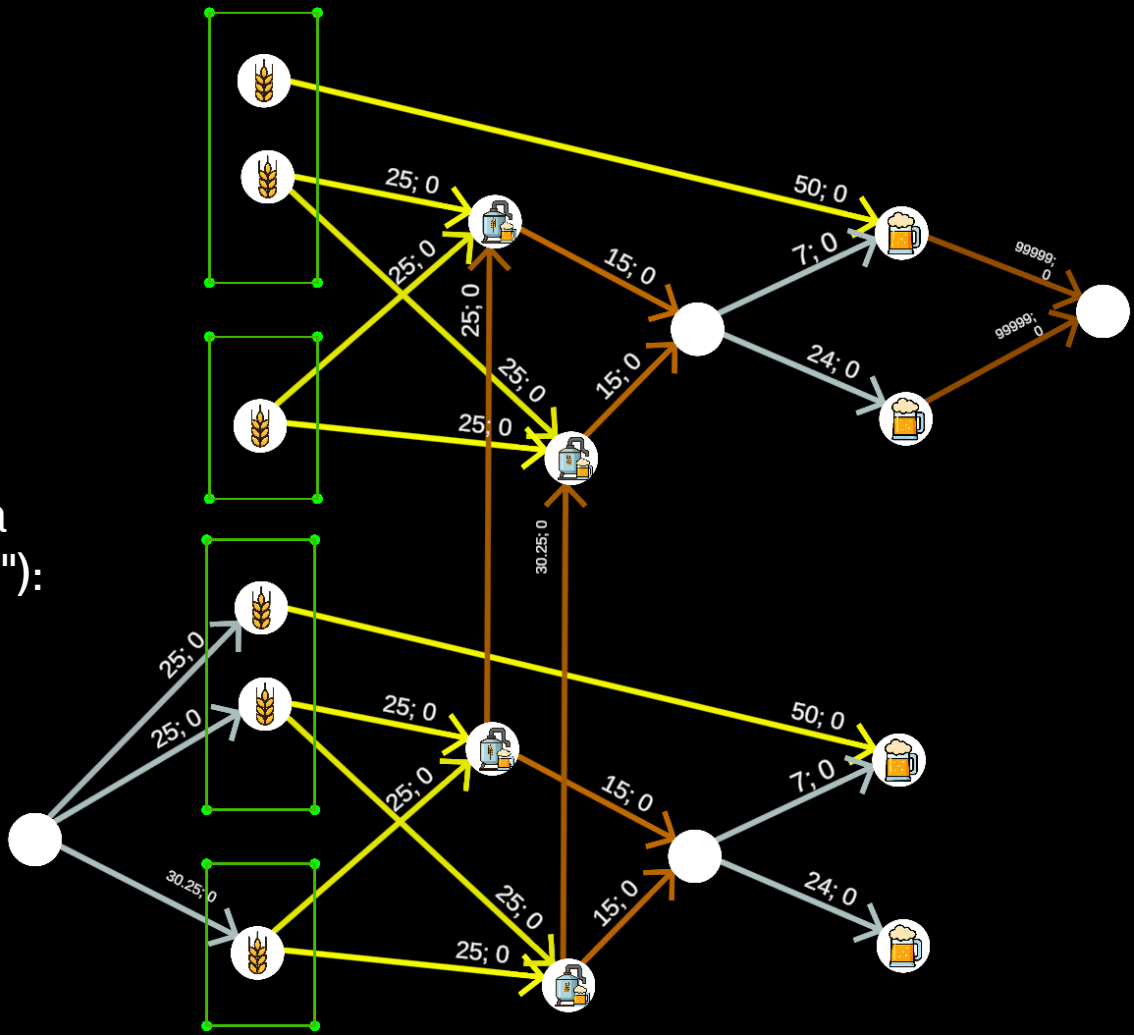
Analizator wykonuje algorytm na maksymalny przepływ na nowym grafie.

Znając maksymalny przepływ, tworzy ten nowy graf jeszcze raz, żeby przywrócić wartości sprzed działania algorytmu Edmondsa-Karpa i wykonuje algorytm Busacker'a-Gowen'a (+Dijkstra z potencjałami), żeby znaleźć minimalny koszt i ostateczne rozwiązanie.

Przykład:
Dane wejściowe:



Analizator zamienia na ("nowy graf"):



Moduł 1. – analizator przepływu

Szablon danych wyjściowych (legenda na następnej stronie)

Konwersja:
konwersja ton to ilość piwa, jaką uzyskuje się z tony jęczmienia w każdym z browarów.
Punkty:
Punkt p_1: typ: typ_{p1}; pozycja: x = X_{p1}; y = Y_{p1}<; pojemność: $pojemność_{p1}$ ton>;
[...]

Drogi:
Droga d_1: z typ_{1d1} [id_od_{d1}] na pozycji (X_{1d1}; Y_{1d1}) do typ_{2d1} [id_do_{d1}] na pozycji (X_{2d1}; Y_{2d1}): przepustowość: $przepływ_{d1}$ ton; koszt naprawy: $koszt_{d1}$ srebrnych pensów;
[...]

Ćwiartki:
wartość: $wartość\_ćwiartki_1$; punkty graniczne: (X_{c11}; Y_{c11}), (X_{c12}; Y_{c12}), (X_{c13}; Y_{c13}), [...]
[...]

Rozwiązanie:
Maksymalna ilość piwa, którą można przetransportować: wynik ton; koszt naprawy: koszt srebrnych pensów
Przebieg trasy:
Trasa nr nr_trasy: id_punktu_1, id_punktu_2, [...]
Z typ_1 [id_punktu_1] na pozycji (X_1; Y_1) ile_1 ton $czego_1$ do typ_2 [id_punktu_2] na pozycji (X_2; Y_2);
<$rodzaj\_wartości_1$ typ_1 [id_punktu_1] na pozycji (X_1; Y_1) wynosi $wartość\_punktu_1$;>
<$rodzaj\_wartości_2$ typ_2 [id_punktu_2] na pozycji (X_2; Y_2) wynosi $wartość\_punktu_2$;>
Z typ_2 [id_punktu_2] na pozycji (X_2; Y_2) ile_2 ton $czego_2$ do typ_3 [id_punktu_3] na pozycji (X_3; Y_3);
Z typ_3 [id_punktu_3] na pozycji (X_3; Y_3) ile_3 ton $czego_3$ do typ_4 [id_punktu_4] na pozycji (X_4; Y_4);
[...]

Moduł 1. – analizator przepływu

Legenda:

konwersja - wartość zmiennoprzecinkowa różna od 0; ilość piwa (w tonach), którą otrzymuje się z tony jęczmienia

p, id_punktu, d, nr_trasy - unikalna wartość całkowita dodatnia, zaczyna się od 1, z każdym następnym pojawieniem się rośnie o 1;

X, Y - położenie punktu: X – na płaszczyźnie poziomej, Y – na płaszczyźnie pionowej; typ zmiennoprzecinkowy;

typ - słowo w języku polskim: "pole", "browar", "karczma", "brak" (skrzyżowanie dla danych wejś.) lub “skrzyżowanie” (dla wyjś.);

pojemność - dodatnia liczba zmiennoprzecinkowa; występuje tylko, gdy typ to “browar”; jest określoną ilością jęczmienia, jaką może przetworzyć;

KONWERSJA, PUNKTY, DROGI, CWIARTKI - słowa-znaczniki rozdzielające listy danych;

id_od, id_do – wartości id_punktu oznaczające odpowiednio: z którego punktu prowadzi droga oraz do którego punktu prowadzi droga;

przepływ - wartość nieujemna zmiennoprzecinkowa (float), oznacza “ilość zboża oraz piwa, którą można przewieźć” przez daną drogę,

koszt - koszt naprawy drogi (liczba naturalna).

rodzaj_wartości - jeśli typem punktu jest “pole”, rodzaj wartości będzie słowem “zawartość”; jeśli typem punktu jest “browar”, rodzaj wartości będzie słowem “pojemność”,

wartość_punktu - dodatnia liczba zmiennoprzecinkowa; występuje tylko, gdy punktem jest “pole” lub “browar”; dla pola jest średnią ilością jęczmienia, która wyrasta na jednym poletku; dla browaru - jest określoną ilością jęczmienia, jaką może przetworzyć;

wartość_ćwiartki - “ilości jęczmienia wyrastające na jednym polu” w danej ćwiartce

wynik - wartość “maksymalnej ilości piwa, którą można dostarczyć do karczm w Shire”, jednocześnie dbając o to, żeby “koszt naprawy dróg [...] był możliwie najmniejszy”; obliczona głównym algorytmem

ile - wartość zmiennoprzecinkowa (float) oznaczająca, ile jęczmienia lub piwa zostało przetransportowane daną drogą

czego - słowo “jęczmienia” lub “piwa”, w zależności od tego, czy jęczmień został już przetworzony przez browar

[...] - “i tak dalej”; indeksy zwiększają się o 1 lub według widocznego wzoru

Przykład:

Konwersja:
0,3 ton to ilość piwa, jaką uzyskuje się z tony jęczmienia w każdym z browarów.

Punkty:
Punkt 1: typ: pole; pozycja: x = 0, y = 2; wydajność: 30 ton.
Punkt 2: typ: pole; pozycja: x = 0, y = -2; wydajność: 30 ton.
Punkt 3: typ: browar; pozycja: x = 4, y = 2; pojemność: 25 ton.
Punkt 4: typ: browar; pozycja: x = 4, y = -2; pojemność: 30 ton.
Punkt 5: typ: skrzyżowanie; pozycja: x = 8, y = 0.
Punkt 6: typ: karczma; pozycja: x = 12, y = 2.
Punkt 7: typ: karczma; pozycja: x = 12, y = -2.

Drogi:
Droga 1: z pola [1] na pozycji (0, 2) do browaru [3] na pozycji (4, 2): przepustowość: 25 ton; koszt naprawy: 1 srebrnych pensów.
Droga 2: z pola [1] na pozycji (0, 2) do browaru [4] na pozycji (4, -2): przepustowość: 30 ton; koszt naprawy: 2 srebrnych pensów.
Droga 3: z pola [2] na pozycji (0, -2) do browaru [3] na pozycji (4, 2): przepustowość: 25 ton; koszt naprawy: 3 srebrnych pensów.
Droga 4: z pola [2] na pozycji (0, -2) do browaru [4] na pozycji (4, -2): przepustowość: 30 ton; koszt naprawy: 4 srebrnych pensów.
Droga 5: z browaru [3] na pozycji (4, 2) do skrzyżowania [5] na pozycji (8, 0): przepustowość: 13 ton; koszt naprawy: 5 srebrnych pensów.
Droga 6: z browaru [4] na pozycji (4, -2) do skrzyżowania [5] na pozycji (8, 0): przepustowość: 17 ton; koszt naprawy: 6 srebrnych pensów.
Droga 7: z skrzyżowania [5] na pozycji (8, 0) do karczmy [6] na pozycji (12, 2): przepustowość: 20 ton; koszt naprawy: 7 srebrnych pensów.
Droga 8: z skrzyżowania [5] na pozycji (8, 0) do karczmy [7] na pozycji (12, -2): przepustowość: 15 ton; koszt naprawy: 8 srebrnych pensów.

Ćwiartki:
wartość: 30; punkty graniczne: (-1; 3), (-1; -3), (2; -3), (2; 3),

Rozwiązanie:
Maksymalna ilość piwa, którą można przetransportować: 27,5 ton; koszt naprawy: 13 srebrnych pensów
Przebieg trasy:
Trasa nr 1:
Z pola [1] na pozycji (0, 2) 30 ton jęczmienia do browaru [4] na pozycji (4, -2);
zawartość pola [1] na pozycji (0, 2) wynosi 0;
pojemność browaru [4] na pozycji (4, -2) wynosi 0;
Z browaru [4] na pozycji (4, -2) 15 ton piwa do skrzyżowania [5] na pozycji (8, 0);
Z skrzyżowania [5] na pozycji (8, 0) 15 ton piwa do karczmy [6] na pozycji (12, 2);
Trasa nr 2:
Z pola [2] na pozycji (0, -2) 25 ton jęczmienia do browaru [3] na pozycji (4, 2);
zawartość pola [2] na pozycji (0, -2) wynosi 0;
pojemność browaru [3] na pozycji (4, 2) wynosi 0;
Z browaru [3] na pozycji (4, 2) 12,5 ton piwa do skrzyżowania [5] na pozycji (8, 0);
Z skrzyżowania [5] na pozycji (8, 0) 12,5 ton piwa do karczmy [7] na pozycji (12, -2);

Moduł 2. – wyszukiwarka tekstu

Ścieżka: ShireCraft_Data/StreamingAssets/StringSearch.exe

Wyszukiwarka tekstu – Program służy do wyszukiwania wzorca tekstowego w pliku tekstowym przy użyciu wybranego algorytmu wyszukiwania. Obsługuje pliki zapisane w kodowaniu UTF-8 oraz poprawnie obsługuje znaki Unicode (np. polskie znaki).

Funkcjonalności:

- Wyszukiwanie tekstu z uwzględnieniem lub ignorowaniem wielkości liter.
- Możliwość wyboru algorytmu wyszukiwania:
  - naiwny
  - rabin-karp
  - KMP (Knuth-Morris-Pratt)
  - Boyer-Moore
- Wypisywanie numeru linii i pozycji w linii, gdzie znaleziono dopasowanie.

Moduł 2. – wyszukiwarka tekstu

Obsługiwane algorytmy wyszukiwania:

- Naiwny - Sprawdza każdy możliwy fragment tekstu porównując z wzorcem znak po znaku.
- Rabin-karp - Oblicza i porównuje hashe fragmentów tekstu i wzorca; porównuje dokładnie tylko przy zgodnych hashach.
- KMP - Używa tablicy LPS by pominąć niektóre porównania, jeśli fragment tekstu już częściowo pasuje.
- Boyer-Moore - Porównuje od końca wzorca. Gdy znajdzie niedopasowanie, używa **heurystyk przesunięcia** (BadSuffix, GoodSuffix), by pominąć jak najwięcej niepotrzebnych porównań.

Struktury danych używane:

•vector – do przechowywania LPS, tablic hashów, pozycji znalezionych wzorców.

•unordered_map – do przechowywania dzieci w węzłach Trie (klucz: znak, wartość: wskaźnik na kolejny węzeł).

•wstring – do obsługi znaków Unicode.

Moduł 2. – wyszukiwarka tekstu

Sposób użycia:

<algorytm> <ścieżka_do_pliku> <wzorzec>

lub z ignorowaniem wielkości liter:

-i <algorytm> <ścieżka_do_pliku> <wzorzec>

Parametry wejściowe:

algorytm – nazwa wybranego algorytmu wyszukiwania.

ścieżka_do_pliku – ścieżka do pliku tekstowego w kodowaniu UTF-8.

wzorzec – szukane słowo lub fragment tekstu.

Wyjście programu:

<nr_linii> <pozycja_w_linii>

Obsługa wielkości liter:

Flaga -i konwertuje tekst do małych liter przed wyszukiwaniem, co pozwala na ignorowanie różnic wielkości liter.

Moduł 2. – wyszukiwarka tekstu

Struktura programu:

1. Wczytywanie argumentów z linii poleceń
    - Obsługa flagi -i (ignorowanie wielkości liter).
    - Weryfikacja poprawności parametrów.
2. Otwieranie pliku
    - Ustawienie lokalizacji LC_ALL dla obsługi znaków narodowych.
    - Dekodowanie pliku jako UTF-8.
3. Wyszukiwanie
    - Dla każdej linii z pliku wywoływany jest odpowiedni algorytm wyszukiwania.
    - Wynikiem są numery linii i kolumny (indeksowane od 1) znalezionego wzorca.
4. Obsługa kodowania
    1. Program poprawnie wyświetla znaki specjalne (np. polskie znaki) dzięki ustawieniu lokalizacji i konwersji UTF-8 → wstring.

Moduł 3. – archiwizator plików

Ścieżka: ShireCraft_Data/StreamingAssets/FileArchiver.exe

Archiwizator plików to program w języku C++, który umożliwia kompresję i dekompresję plików tekstowych przy użyciu algorytmu Huffmana.

Program realizuje kompresję i dekompresję tekstu z plików, korzystając z klasycznego algorytmu Huffmana, który tworzy drzewa kodujące oparte na częstotliwości występowania znaków.

Dane wejściowe przekazywane są przez argumenty wywołania programu:

- -c – tryb kompresji (compress)
- -d – tryb dekompresji (decompress)
- input.* – ścieżka do pliku wejściowego
- output.* – ścieżka do pliku wyjściowego

Moduł 3. – archiwizator plików

W trybie kompresji (-c):

1. Wczytuje cały plik tekstowy do pamięci.
2. Buduje drzewo Huffmana na podstawie częstotliwości znaków.
3. Generuje kody binarne dla każdego znaku.
4. Zapisuje wynik do pliku binarnego, który zawiera:
    - liczbę unikalnych znaków i ich kody binarne.
    - zakodowany ciąg bitów reprezentujący cały tekst.

Plik binarny zawiera nie tylko dane, ale także słownik kodów – co pozwala na dekompresję bez dodatkowych informacji.

Moduł 3. – archiwizator plików

W trybie dekompresji (-d):

1. Odczytuje zakodowany plik binarny.
2. Odtwarza słownik kodów Huffmana.
3. Dekoduje ciąg bitów do oryginalnego tekstu.
4. Zapisuje wynik do pliku tekstowego.

Wynik działania:

Program wypisuje na standardowe wyjście dwa rozmiary (w bajtach):

<size_of_input_file> <size_of_output_file>

Służy to do łatwego porównania rozmiaru pliku przed i po kompresji.

Moduł 4. – generator map

Ścieżka: ShireCraft_Data/StreamingAssets/Resources/Generator/networkGen.exe

Generator map – program w C++ generujący pliki wejściowe zgodne z założeniami i specyfikacją o wybranym rozmiarze.

Parametry wymagane do wygenerowania mapy miasta możemy podać na dwa sposoby:

- Po uruchomieniu bez argumentów z linii poleceń przechodząc przez kolejne kroki ze standardowego wejścia.
- Podając wszystkie wymagane parametry w linii poleceń.

Gdzie kolejnymi argumentami z linii poleceń są:

- Ścieżka do pliku wyjściowego
- Ilość wierzchołków w całej sieci (minimum 6)
- Długość miasta (minimum 0, maksymalna wartość zależna od pozostałych parametrów)
- Szansa na wygenerowanie dróg-skrótów (0 – najmniejsza, 1 - średnia, 2 - największa)
- Liczba pól
- Liczba browarów
- Liczba karczm
- Wartość zero-jedynkowa odpowiadająca dwukierunkowości wygenerowanych dróg
- Minimalna losowa wartość przepustowości dróg
- Maksymalna losowa wartość przepustowości dróg

Moduł 4. – generator map

Generator w pełni weryfikuje poprawność danych wejściowych i w przypadku podania niepoprawnych – zwraca odpowiednią informację i kończy działanie.

DZIAŁANIE KROK PO KROKU:

1. Generator zawsze tworzy cztery kwadraty w których umieszcza pewną niewielką losową liczbę punktów o losowych współrzędnych, z których następnie w wyniku działania algorytmu Grahama powstaje wielokąt wypukły. Warto wspomnieć o tym, że rozmiar kwadratów skaluje się wraz z ilością pól.
2. Spośród podanej liczby wierzchołków tworzone są przedziały, z których pierwszy z nich zawsze zawiera id wszystkich pól, a ostatni id wszystkich karczm. Pozostałe przedziały grupują pozostałe wierzchołki w taki sposób, że generowane drogi nigdy nie łączą dwóch wierzchołków z tego samego przedziału. Zadaniem przedziałów jest zbalansowanie mapy miasta w taki sposób, żeby nie powstawał "brzydki" rozkład połączeń (np. drogi 100 wierzchołków prowadzą do jednego, tzw. wąskie gardła) oraz manipulacja długością miasta (mniejsza ilość przedziałów sprawia, że mamy z reguły krótszą drogę z pól do karczm, ale za to więcej możliwości transportu towaru pomiędzy nimi). Rozmiar przedziałów również automatycznie się skaluje. W ten sposób powstałe miasta mają znacznie większy sens logistyczny.
3. Losowane są współrzędne pól w taki sposób, że każde pole należy do jakiejś ćwiartki.
4. Współrzędne pozostałych wierzchołków są ustalane w taki sposób, aby cała mapa była kompaktowa podczas oglądania w GUI i jednocześnie reprezentowała zamysł w jaki sposób generowana jest cała mapa.

Moduł 4. – generator map

5. Browary są losowane spośród wierzchołków które nie są polami lub karczmami.

6. Tworzone są drogi-skróty, czyli drogi prowadzące od wierzchołków x_i do x_j,

gdzie i,j to indeksy przedziałów oraz j>i+1. Ilość dróg-skrótów zależy od odpowiedniego parametru podanego przez użytkownika. Dzięki temu możemy dostosować, czy mapa ma być bardziej poukładana, czy chaotyczna.

7. Tworzone są podstawowe połączenia pomiędzy sąsiednimi przedziałami.

8. Tworzone są korekty połączeń w celu uniknięcia wierzchołków nieosiągalnych w sieci.

9. Uzupełnienie pozostałych informacji wymaganych w danych wejściowych głównego problemu oraz zapis gotowych danych do pliku.

Wydajność generatora zależy głównie od ilości wszystkich wierzchołków w grafie.

PODSUMOWANIE:

Bardzo dużymi zaletami generatora jest prostota użytkowania m.in. za sprawą rekomendowanych wartości poszczególnych parametrów, możliwość dostosowywania generowanych danych do własnych potrzeb jednocześnie zachowując sens logistyczny miasta oraz bardzo wysoka wydajność.

Moduł 5. i 6. – interfejsy graficzne

5. Interfejs graficzny generatora – aplikacja okienkowa w WPF (C#), pozwalająca na przyjazne dla użytkownika wprowadzanie danych do generatora.
Ścieżka: ShireCraft_Data/StreamingAssets/networkGenInterface.exe

Program jest intuicyjny w obsłudze, komunikaty informujące o wymaganych lub niepoprawnych danych wyświetlają się w nim na bieżąco.

6. Interfejs główny - aplikacja w Unity (C#), zintegrowana z pięcioma poprzednimi programami, pozwalająca również na wygodne tworzenie i wizualizację danych wejściowych oraz wizualizację danych wyjściowych.
Ścieżka: ShireCraft.exe

Instrukcja obsługi znajduje się wewnątrz programu. Należy go uruchomić (w razie potrzeby zmienić język w ustawieniach), a następnie przejść do zakładki "Pomoc" w lewym górnym rogu, oznaczonej "?".

Podział problemu na podproblemy

Podproblem (dane i wynik)	Osoby odpowiedzialne	Zastosowane struktury	Zastosowane algorytmy	Złożoność
Sposób reprezentacji danych o polach, ich wydajności, browarach, konwersji jęczmień-piwo, karczmach, drogach i ćwiartkach w komputerze, do wykorzystania przez algorytm.	Mateusz Wójciak, Jakub Klonowski, Karol Ławicki	List, Vector, Map	[nie dotyczy]	[nie dotyczy]
Obliczenia przepływu z konwersją typu (jęczmień → piwo)	Mateusz Wójciak	Macierz sąsiedztwa	Edmondsa-Karpa na maksymalny przepływ w grafie	$O(V * E^2)$ V – liczba wierzchołków w grafie E – liczba krawędzi w grafie
Minimalizacja kosztu naprawy dróg przy zachowaniu maksymalnej ilości przewożonego towaru.	Mateusz Wójciak	Kolejka priorytetowa	Busacker'a-Gowen'a Bellmana-Forda + Dijkstry z potencjałami	$O(V^3) + O(F * V^2)$ V – liczba wierzchołków w grafie F – maksymalny przepływ (liczbowo)
Wyznaczanie przynależności pola do regionu	Jakub Klonowski	List	otoczki wypukłej Grahama	$O(n \log n)$ n – liczba wierzchołków
Kompresja i dekompresja	Dominika Karbowiak, Jakub Klonowski	Drzewo Huffmana (kopiec)	Kodowanie entropijne Huffmana (kompresja + kodowanie)	$O(n \log k)$, ale jeśli alfabet ma ograniczoną liczbę symboli (np. bajty = 256), to można traktować $\log k$ jako stałą i uprościć do $O(n)$. n – liczba znaków w tekście k – liczba różnych znaków (alfabet)
Wyszukiwanie wzorców tekstowych	Weronika Gburek, Dominika Karbowiak	Vector, Unordererd_map, wstring (typ zmiennej)	• naiwny wyszukiwania • Rabina-Karpa • KMP (Knuth–Morris–Pratt) • Boyer-Moore	• $O(n)$ [optym.] / $O(n*m)$ [pesym.] • $O(n+m)$ [amort.] / $O(n*m)$ [pesym.] • $O(n+m)$ [zawsze liniowa] • $O(n + m)$ [śred.] / $O(n*m)$ [pesym.] n – ilość znaków w tekście m - ilość znaków we wzorcu

Dokładność wyników podproblemów

Podproblem	Czy wynik dokładny?	Uzasadnienie
Obliczenia przepływu z konwersją typu (jęczmień → piwo)	✅ Tak	Tworzony jest zduplikowany graf: źródło znajduje się w pierwszej części (prowadzi do pól jęczmienia), a jedyna droga do ujścia prowadzi przez pary browarów (z pierwszej części do odpowiadających im w drugiej części). Dzięki temu wymuszony zostaje przepływ przez co najmniej jeden browar, a wyniki są dokładne. Następnie na tym grafie uruchamiamy algorytm Edmondsa-Karpa.
Wyznaczanie minimalnego kosztu przy zachowaniu maksymalnego przepływu	✅ Tak	Algorytm Busackera-Gowena zapewnia dokładne rozwiązanie problemu Min-Cost Max-Flow.
Kompresja i dekompresja	✅ Tak	Kompresja Huffmana jest bezstratna - dane są odzyskiwane bez błędu
Wyszukiwanie wzorców tekstowych	✅ Tak	Zastosowane algorytmy przeszukiwania (naiwny, KMP, Rabin-Karp, Boyer-Moore) zwracają dokładne dopasowania
Wyznaczanie przynależności pola do regionu	✅ Tak	Algorytm punkt-w-wielokącie działa deterministycznie
Obliczenia przepływu z konwersją typu (jęczmień → piwo)	⚠️ Nie zawsze	W przypadku ułamkowego mnożnika może wystąpić utrata precyzji (ostrzeżenie w GUI)

Ogólny przebieg prac

Prace nad projektem rozpoczęto od dokładnej analizy treści zadania oraz rozpoznania znanych algorytmów przepływu sieciowego. Harmonogram realizacji był elastyczny, dostosowywany do bieżących potrzeb i postępów. Po przetestowaniu kilku podejść zdecydowano się na wykorzystanie algorytmu Dijkstry z potencjałami, klasycznego algorytmu Edmondsa-Karpa oraz metody Busackera-Gowena jako głównego rozwiązania problemu minimalnego kosztu maksymalnego przepływu.

W trakcie implementacji okazało się, że pierwotny pomysł wymuszania przejścia przez browary był błędny logicznie i trudny do zrealizowania. Ostatecznie przyjęto skuteczniejsze rozwiązanie – każdemu browarowi przyporządkowano jego kopię w duplikacie grafu, a ścieżki przepływu prowadziły przez odpowiednie krawędzie łączące oryginalny graf z jego kopią, co umożliwiło poprawne modelowanie wymaganych zależności.

Interfejs graficzny projektowano od początku równolegle z silnikiem obliczeniowym. Przy jego tworzeniu szczególny nacisk położono na prostotę, intuicyjność obsługi oraz funkcjonalność typową dla edytorów map – umożliwiając łatwe dodawanie i edytowanie obiektów, a także wizualną analizę wyznaczonych tras.

Wnioski końcowe

Zespół projektowy ocenia, że wszystkie kluczowe wymagania zlecenia zostały zrealizowane. Najważniejszymi osiągnięciami są czytelny i wygodny interfejs użytkownika oraz skuteczna implementacja klasycznych algorytmów sieciowych, pozwalająca rozwiązywać złożone problemy przepływu i kosztu.

Nie udało się w pełni rozwiązać problemu związanego z utratą precyzji w przypadku przepływów zmiennoprzecinkowych oraz jednoczesnych niecałkowitych współczynników konwersji jęczmienia na piwo – został on częściowo zaadresowany przez system ostrzegania w interfejsie.

Projekt okazał się wartościowym doświadczeniem edukacyjnym. Zespół rozwinął umiejętności techniczne, nauczył się współpracy w repozytorium oraz wypracował dyscyplinę pracy w zespole rozproszonym.

Testy analizatora

Algorytm Min-Cost Max-Flow, dostosowany do potrzeb burmistrza został przetestowany w różnych warunkach, aby upewnić się co do jego poprawności i efektywności:

- Zweryfikowano zgodność przepływu z ograniczeniami dróg, pól i browarów.
- Sprawdzono, czy całkowita ilość dostarczonego piwa do karczm jest maksymalna w danych warunkach.
- Sprawdzono, czy algorytm znajduje alternatywne trasy o minimalnym koszcie przy zachowaniu tego samego przepływu.
- Porównano wyniki z klasycznym algorytmem Max-Flow (Edmonsa-Karpa, bez uwzględniania kosztów, przy mnożniku 1), aby potwierdzić, że ilość piwa nie uległa zmniejszeniu.

Testy analizatora

Folder z plikami testów

0. Najprostszy graf bez kosztów, przechodzący pole -> browar -> karczma.

Oczekiwany wynik:
przepływ: 1; koszt: 0

Otrzymany wynik się zgadza ✅

Komentarz: analizator poprawnie parsuje plik wejściowy, analizuje dane i aktualizuje przesyłaną wartość po przejściu przez browar

Testy analizatora

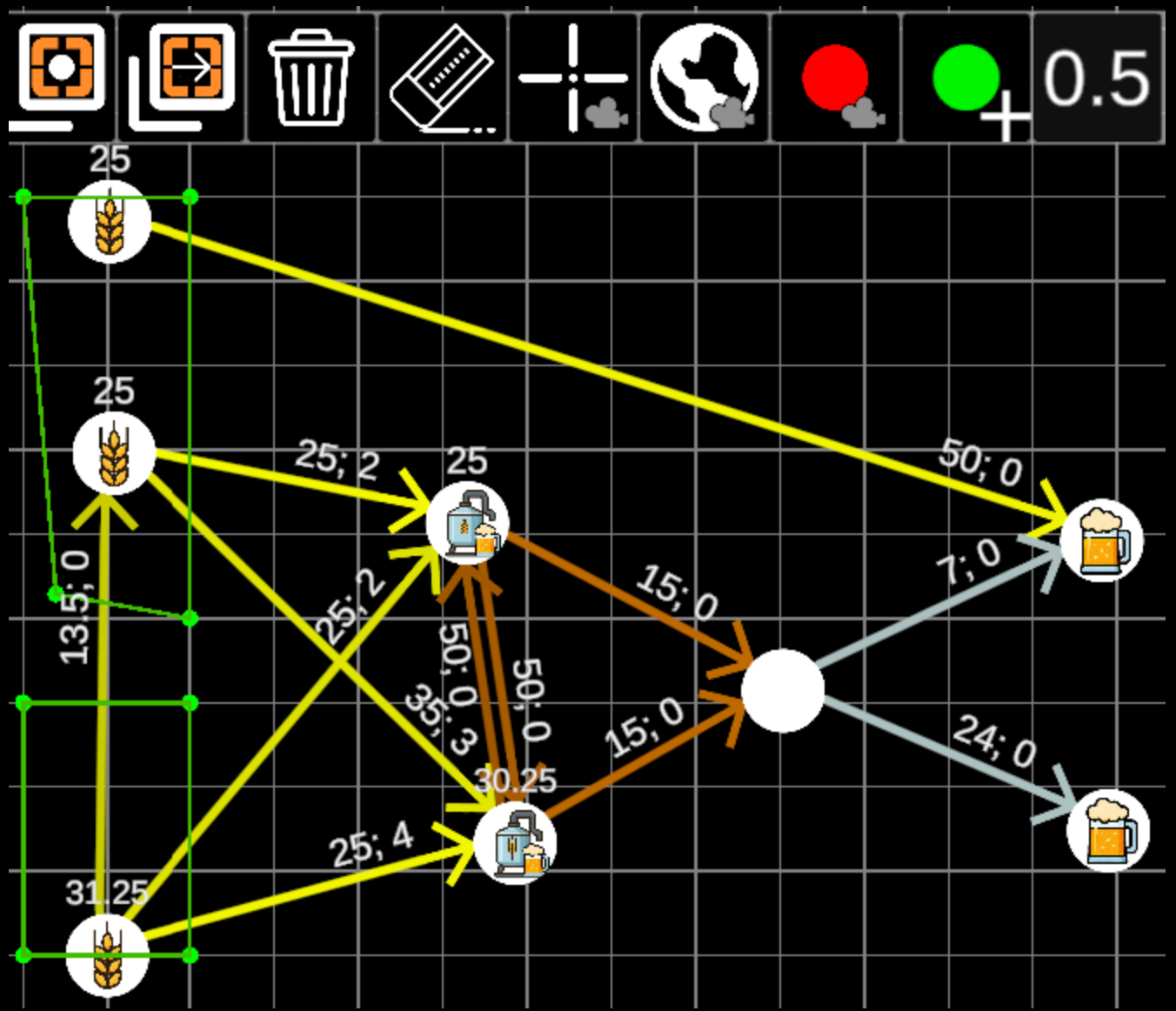
[Folder z plikami testów]

1. Graf z kosztami, z różnorodnymi przejściami.

Oczekiwany wynik:
przepływ: 27.625; koszt: 7

Otrzymany wynik się zgadza ✅

Komentarz: analizator poprawnie przechodzi między polami, między browarami, a także przez browary, które już nie mają pojemności, bez zmieniania przesyłanego towaru. Wykorzystuje całą pojemność każdego browaru, unikając drogi o najwyższym koszcie (=4). Poprawnie nastawia wartości pól w zależności od ćwiartki. Prawidłowo nie uwzględnia w rozwiązaniu górnego pola, które nie ma trasy do browaru.



Testy analizatora

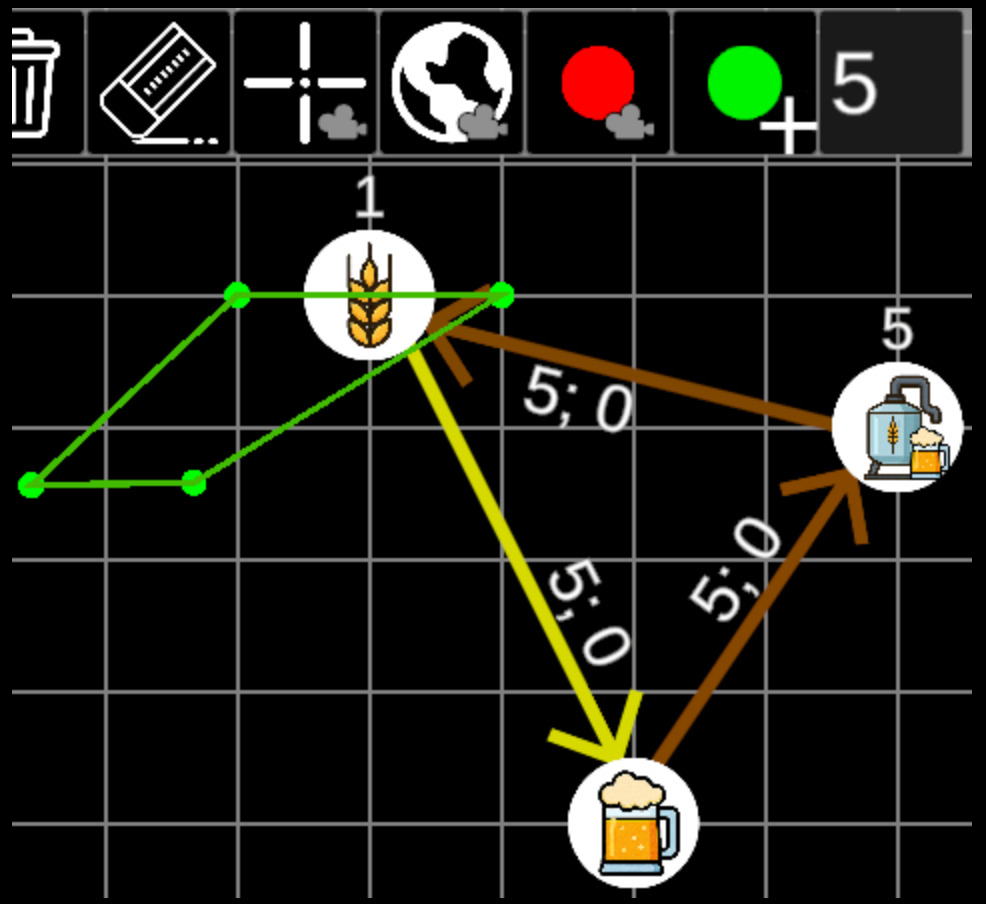
[Folder z plikami testów]

2. Graf-cykl 3-punktowy:
pole1 -> karczma -> browar -> pole1

Oczekiwany wynik:
przepływ: 5; koszt: 0

Otrzymany wynik się zgadza ✅

Komentarz: analizator poprawnie wylicza obszar ćwiartki, mimo że punkty graniczne są w miejscach dziesiętnych. Poprawnie nastawia wartości pól w zależności od ćwiartki, nawet na krawędziach. Poprawnie liczy przepływ, mimo że jedna trasa przechodzi dwa razy przez tą samą strzałkę (raz z jęczmieniem, raz z piwem).



Dlaczego wynikiem jest 5: pole przesyła 1 tonę jęczmienia przez dwie drogi do browaru, ale kiedy dociera do browaru, to "wóz wiozący zboże" już nie znajduje się na tamtych drogach, dlatego mają one dalej przepustowość 5. Kolejne 2 drogi do karczmy wykorzystują już cały przepływ.

Testy analizatora

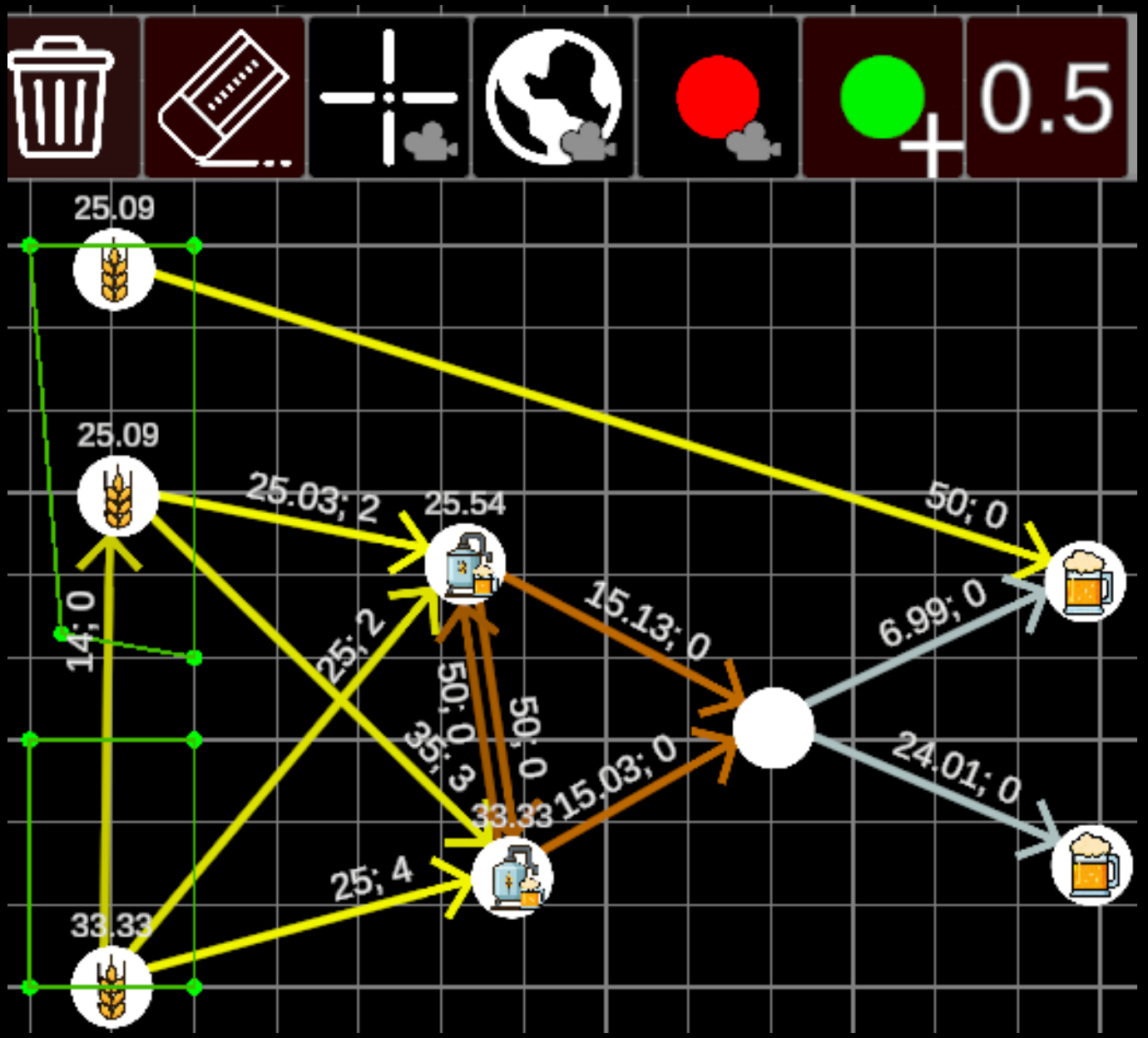
Folder z plikami testów

3. Graf zawiera wartości problematyczne dla komputera (0.01, 0.09, 0.3, 0.33, 0.99)

Oczekiwany wynik:
przepływ: 29.210; koszt: 7

Otrzymany wynik się zgadza ✅

Komentarz: algorytm radzi sobie z liczbami zawierającymi miejsca dziesiętne (dopóki
przepływ drogi * konwersja
nie daje liczby z ponad 3 cyframi po przecinku)



Testy analizatora

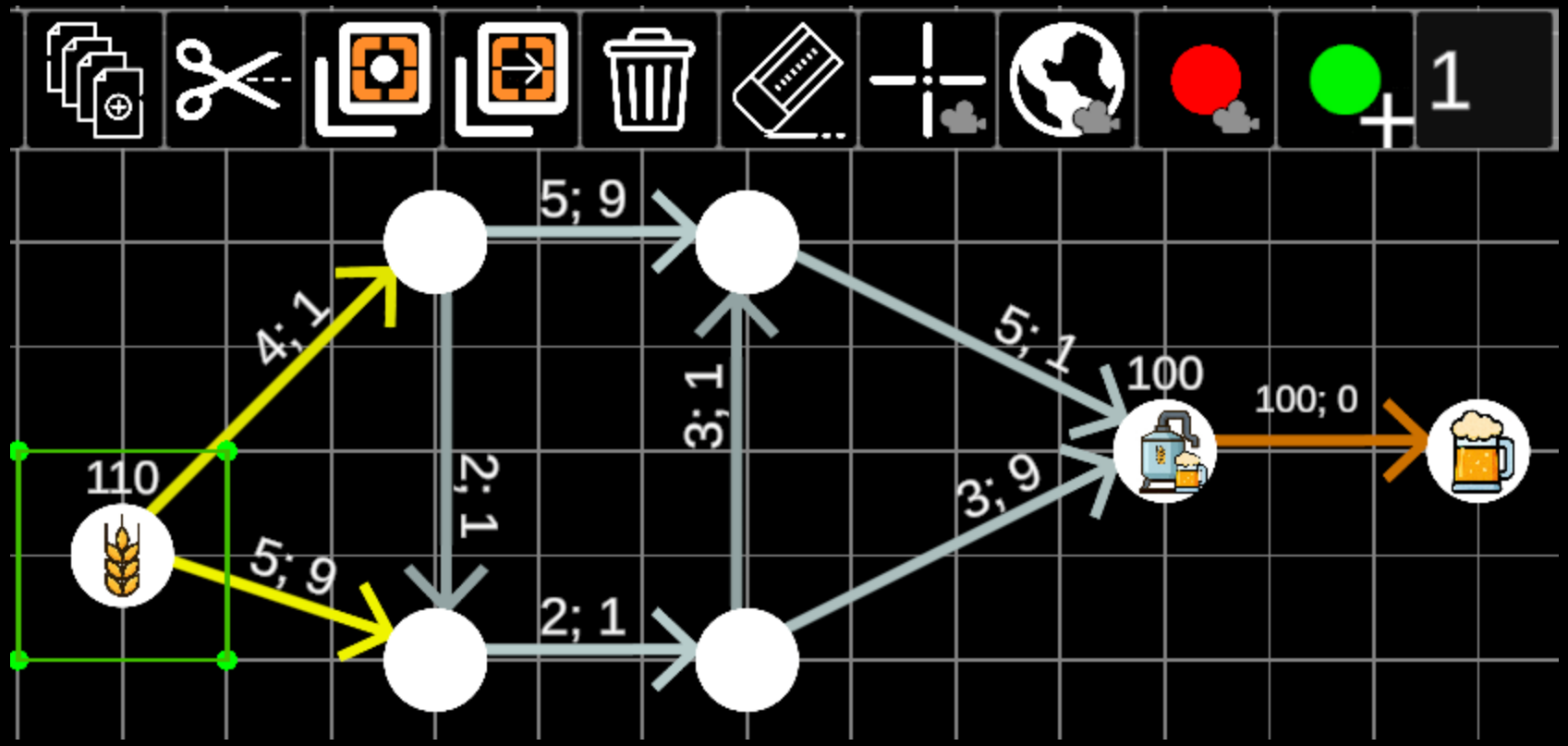
Folder z plikami testów

4. Algorytm Busacker'a Gowen'a znajduje najtańszą trasę z ujemnym kosztem w trakcie analizy.

Oczekiwany wynik:
przepływ: 6; koszt: 32

Otrzymany wynik się zgadza ✅

Komentarz: analizator poprawnie znajduje minimalny koszt dla maksymalnego przepływu mimo, że ścieżka powiększająca cofa się strzałką w przeciwnym kierunku, której na początkowym grafie nie ma



Uwaga:
W wizualizacji świadomie pokazujemy przypadki tzw. „odwrotnych przepływów” (tj. cofnięcia wcześniej przydzielonego transportu), nawet jeśli początkowo nie istnieje droga w przeciwnym kierunku. Zamiast ukrywać ten aspekt algorytmu, GUI pokazuje istniejącą drogę jako „przebiegającą pod prąd”, co wizualnie podkreśla korektę decyzji wynikającą z optymalizacji kosztów. Takie cofnięcia należy interpretować jako zmianę alokacji zasobów w planie, a nie dosłowne cofanie wozów. Co istotne, koszty naprawy dróg nie są cofane – raz poniesione, pozostają w końcowym bilansie, co jest zgodne z rzeczywistością: naprawiona droga nie staje się „nienaprawiona” tylko dlatego, że została potem pominięta.

Testy wyszukiwania

Lp.	Liczba znaków pliku	Liczba znaków wzorca	Algorytm	Czas (ms)
1	5,510,107	23	naiwny	389
	(wzorzec występuje raz w pliku)		Rabin-Karp	378
			KMP	508
			Boyer-Moore	6364
2	1,000,000	8	naiwny	98
	(powtarzający się wzorzec)		Rabin-Karp	74
			KMP	110
			Boyer-Moore	64
3	3,000,000	90	naiwny	5925
	(Długi wzorzec prawie występujący w pewnym miejscu, nie zgadzała się jedynie ostatnia litera)		Rabin-Karp	199
			KMP	268
			Boyer-Moore	291

Testy wyszukiwania

Lp.	Liczba znaków pliku	Liczba znaków wzorca	Algorytm	Czas (ms)
4	60,000,000	20	naiwny	18150
	(Silnie powtarzalny tekst i zawiera wiele fragmentów podobnych do wzorca)		Rabin-Karp	11709
			KMP	5000
			Boyer-Moore	21949
5	1,000,000	7	naiwny	206
	(Wzorzec na końcu pliku)		Rabin-Karp	67
			KMP	88
			Boyer-Moore	91
6	1,000,000	50	naiwny	1110
	(Długi tekst z wieloma powtórzeniami wzorca (aa...aab))		Rabin-Karp	70
			KMP	88
			Boyer-Moore	103

Testy wyszukiwania

Podsumowanie działania algorytmów wyszukiwania wzorców na podstawie testów:

Algorytm naiwny:

Najlepszy przypadek: Krótkie teksty i wzorce, mało dopasowań

Najgorszy przypadek: Teksty i wzorce z dużą liczbą powtarzających się fragmentów wzorca

Rabin-Karp:

Najlepszy przypadek: Gdy trzeba znaleźć wiele wzorców, lub bardzo rzadkie dopasowania

Najgorszy przypadek: Kolizje w funkcji haszującej powodują dużo porównań znak po znaku

KMP:

Najlepszy przypadek: Teksty i wzorce z powtarzającymi się fragmentami wzorca

Najgorszy przypadek: Działa stabilnie, w czasie liniowym O(n)

Boyer-Moore:

Najlepszy przypadek: Długie wzorce, teksty o dużej różnorodności znaków

Najgorszy przypadek: Wzorce krótkie, lub bardzo powtarzające się, wtedy może być wolniejszy

Testy kompresji

Lp.	Rozmiar danych wejściowych (liczba znaków)	Rozmiar przed kompresją (w bajtach)	Rozmiar po kompresji (w bajtach)	Redukcja (%)
1	1012	1076 B	872 B	18.96 %
2	10.345	11.002 B	6752 B	38.65 %
3	20.120	21.401 B	12.916 B	39.64 %
4	40.140	42.690 B	25.529 B	40.23 %
5	80.080	85.171 B	50.703 B	40.50 %
6	123.456	131.313 B	78.049 B	40.56 %
7	1.000.000	1.063.635 B	630.536 B	40.70 %

Testy generatora

Wszystkie testy zostały przeprowadzone dla rekomendowanych przez generator danych wejściowych.

Lp.	Rozmiar danych wejściowych (liczba wierzchołków)	Czy dwukierunkowe połączenia?	Ilość powstałych połączeń	Czas (ms)
1	25	NIE	38	4
2	200	TAK	568	20
3	2 000	NIE	2 834	120
4	20 000	TAK	56 694	1753
5	100 000	NIE	143 216	6093

Nawet największe i najbardziej zaludnione miasta na świecie, takie jak Pekin mają "tylko" około 12 000 skrzyżowań.