Algorytmy Zaawansowane
Etap 3 - Dokumentacja końcowa

Wojciech Kowalik, Konrad Miśkiewicz

1 Wstęp

1.1 Opis zadania

Są 2 kajaki, n osób oraz zbiór par, które chcą razem przepłynąć się kajakiem (jedna osoba może chcieć przepłynąć się kolejno z wieloma osobami). Zastosować algorytm znajdujący skojarzenie doskonałe do wyznaczenia rozkładu jazdy kajaków, tak aby zminimalizować ilość kursów pojedynczych kajaków. Należy przy tym spełnić życzenia wszystkich osób.

1.2 Zmiany w stosunku do pierwotnej wersji

Aby rozwiązać przedstawione powyżej zadanie, konieczna jest odpowiednia konstrukcja grafu, wykonanie dwóch jego przekształceń i znalezienie w wynikowym grafie maksymalnego skojarzenia. W celu znalezienia maksymalnego skojarzenia w grafie zdecydowaliśmy się wykorzystać algorytm Edmondsa. Podczas implementacji aplikacji napotkaliśmy na pewne problemy związane z implementacją tego algorytmu według opisu, który przedstawiony został w poprzedniej wersji dokumentu. Zdecydowaliśmy się więc zmienić pewne szczegóły implementacyjne i wydzielić dodatkową funkcję, służącą znajdowaniu ścieżki powiększającej w grafie. Szczegółowy pseudokod algorytmu ze zmienionymi pewnymi fragmentami w stosunku do pierwotnej jego wersji przedstawiony zostanie w następnym rozdziale.

2 Opis rozwiązania

2.1 Algorytm Edmondsa

Poniżej przedstawiono kompletny pseudokod algorytmu Edmondsa, który został zaimplementowany w naszej aplikacji. Sercem algorytmu jest funkcja znajdująca ścieżkę powiększającą w grafie. Wprowadźmy następujące oznaczenia:

- wierzchołek wolny - wierzchołek, który nie należy do krawędzi skojarzonej
- $root(v)$ - korzeń drzewa zawierającego wierzchołek v

Algorithm 1 Pseudokod algorytmu Edmondsa

G - graf, w którym szukamy maksymalnego skojarzenia
$M = \emptyset$

function FindMaximumMatching(G, M)

$P \leftarrow$FindAygmentingPath(G, M)
if P jest niepustą ścieżką **then**
  return FindMaximumMatching(G, M powiększone wzdłuż ścieżki P)
else
  return M
end if

end function

Algorithm 2 Pseudokod algorytmu znajdowania ścieżki powiększającej w grafie

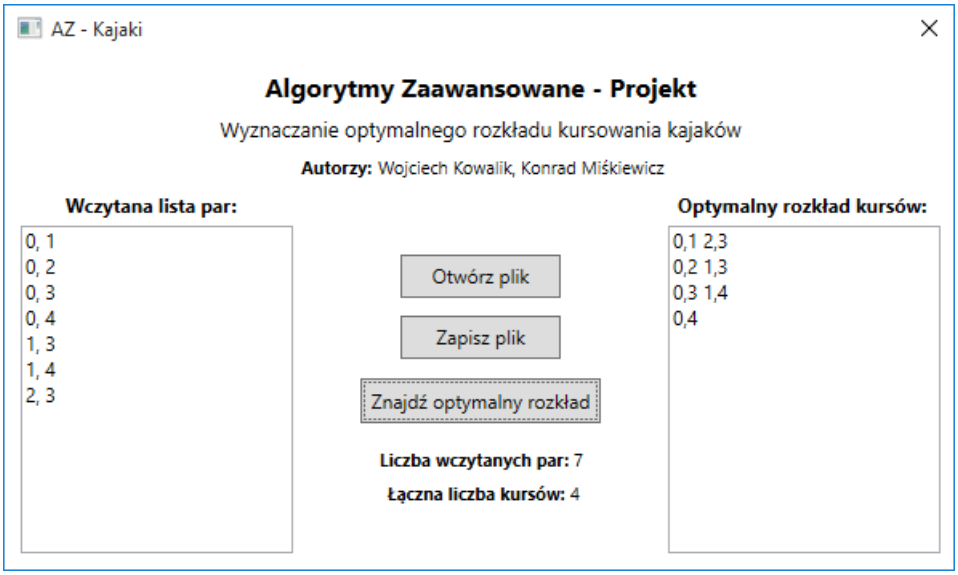
```
function FindAugmentingPatch(G, M)

F ← pusty las
odznacz wszystkie wierzchołki i krawędzie w G
oznacz krawędzie, które należą do M
for each wolnego wierzchołka v
. utwórz drzewo v i dodaj do lasu F
end for each
while istnieje nieoznaczony wierzchołek v w F i odległość (v, root(v)) jest parzysta do
  while istnieje nieoznaczona krawędź e = (v, w) do
    if w nie należy do F then
      x ← wierzchołek skojarzony z w w M
      dodaj krawędzie (v, w) i (w, x) do drzewa zawierającego v
    else
      if odległość (w, root(w)) jest nieparzysta then
        // Nie rób nic.
      else
        if root(v) ≠ root(w) then
          P → ścieżka (root(v) → ... → v) → (w → ... → root(w))
          return P
        else
          B ← cykl uformowany przez krawędź e i krawędzie na ścieżce v → w w F
          G', M' ← nowy graf i skojarzenie powstałe po ściągnięciu cyklu B
          P' ← FindAugmentingPath(G', M')
          P ← ścieżka P' po rozwinięciu cyklu B w grafie G
          return P
        end if
      end if
    end if
    oznacz krawędź e
  end while
  oznacz wierzchołek v
end while
return pusta ścieżka

end function
```

3 Testy

Do projektu został dołączony projekt z testami jednostkowymi. Przetestowane zostały wszystkie klasy i funkcje publiczne, takie jak: klasa i funkcje odpowiedzialne za odczytywanie i zapisywanie plików, klasa odpowiedzialna za wykonywanie operacji na grafach oraz klasa zawierająca algorytm wyznaczania rozkładu jazdy kajaków. Wszystkie testy przebiegły pomyślnie.

4 Instrukcja obsługi dla użytkownika

Po uruchomieniu aplikacji na ekranie pojawi się interfejs przedstawiony na poniższym obrazku.



Obrazek 4.1 Interfejs graficzny

Aby rozpocząć korzystanie z aplikacji należy wczytać plik z danymi (przygotowany wg formatu pliku wejściowego). W tym celu użytkownik powinien kliknąć przycisk "**Otwórz plik**". Po załadowaniu danych z pliku w okienku znajdującym się w lewej części interfejsu, zatytułowanym "**Wczytana lista par**" pojawią się wylistowane wszystkie pary osób, które chcą się przepłynąć razem kajakiem. Ponadto zaktualizowana zostanie liczba mówiąca o ilości wczytanych par (centralna część interfejsu).

Aby wyznaczyć rozkład jazdy kajaków dla wczytanych par osób, które razem chcą się przepłynąć użytkownik powinien kliknąć przycisk "**Znajdź optymalny rozkład**". W tym momencie rozpocznie się wykonywanie algorytmu wyznaczającego rozkład dla kajaków. Po zakończeniu jego działania w prawej części interfejsu zatytułowanej "**Optymalny rozkład kursów**" wyświetlona zostanie wyznaczona lista par osób, zgodna z wyznaczonym rozkładem. Zaktualizowana zostanie również liczba kursów w centralnej części interfejsu.

Po wyznaczeniu rozkładu kursów kajaków wynik może zostać zapisany do pliku (zgodnego z formatem wyjściowym opisanym w dokumentacji wstępnej) za pomocą kliknięcia w przycisk "**Zapisz plik**".

5 Podział pracy

Podczas wykonywania projektu wyodrębnione zostały następujące zadania do wykonania:

- parser plików tekstowych - Wojciech Kowalik,
- algorytm wyznaczania dopełnienia grafu - Wojciech Kowalik,
- algorytm wyznaczania grafu krawędziowego - Konrad Miśkiewicz,
- wizualizacja i GUI - Wojciech Kowalik,
- algorytm Edmondsa - Konrad Miśkiewicz/Wojciech Kowalik,
- testy - Konrad Miśkiewicz.