Dokumentacja projekt 1

Marcin Żyżyński

1 Koncepcja

Jako stan (wierzchołek, koszt) będę traktował wierzchołek wraz z kosztem z jakim do niego przyszedłem. Np. po dotarciu do wierzchołka 3 i zebranym kosztem 9, będę mówił, że jestem w stanie (3,9).
Dla problemu przejścia z wierzchołka p do k z kosztem z, można sformułować problem znalezienia ścieżki w grafie prowadzącej ze stanu (p, koszt(p)) do stanu (k, z).
Ważnym założeniem jest, że jeżeli odwiedzenie wierzchołka spowoduje przekroczenie docelowej sumy, nie ma sensu odwiedzać tego wierzchołka, a więc rozwijać wychodzących z niego stanów. Do przeszukania grafu użyję DFS.
Dla każdego wierzchołka będzie istniało maksymalnie z + 1 stanów.

2 Pseudokod

```
A - lista wierzchołków do odwiedzenia
Dodaj p do A na początek
p.kosztZKtorymPrzyszedlem = p.koszt
Dopóki A nie jest puste:
    Tymczasowy wierzchołek  t = weź z A z początku
    Jeżeli t == k i t.kosztZKtorymPrzyszdlem == z:
        Znaleźliśmy ścieżkę
    Dla każdego sasiada:
        Jeżeli t.kosztZKtorymPrzyszedlem + sasiad.koszt <= z:
            sasiad.kosztZKtorymPrzyszedlem = t.kosztZKtorymPrzyszedlem + sasiad.koszt
            Jeżeli sasiad.stanOdwiedzony[sasiad.kosztZKtorymPrzyszedlem] jest null:
                sasiad.stanOdwiedzony[sasiad.kosztZKtorymPrzyszedlem] =
                (t, t.kosztZKtorymPrzyszedlem)
                Dodaj sasiad do A na początek
```

3 Złożoność obliczeniowa

Ponieważ, każdy wierzchołek rozpatrujemy jako z kopii tego wierzchołka to możemy rozpatrywać graf o z * V wierzchołkach i z * E krawędziach. Złożoność DFS to O(V + E) zatem złożoność obliczeniowa tego algorytmu to O(z * (V + E)).

4 Złożoność pamięciowa

Złożoność pamięciowa z * V.

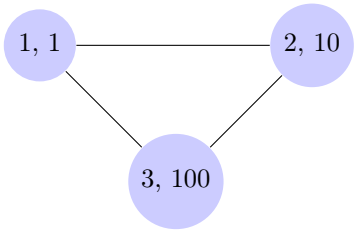
5 Problemy implementacyjne i wnioski

Reprezentacja grafu w ten sprytny sposób z zamienieniem wierzchołka na stany pozwala na zastosowanie prostego algorytmu przeszukiwania.
Zastosowanie stanów zamiast fizycznego duplikowania wierzchołka pozwala zmniejszyć zajętość pamięci. Powoduje to jednak problem z zamazywaniem zmiennych wierzchołka, które trzeba zapamiętywać i odtwarzać ze stosu.
Ważne jest też odcinanie możliwych ścieżek przeszukiwań, gdy przekroczony zostaje poszukiwany koszt.
Problemem jest również odtworzenie ścieżki, którą się przeszło, ale można to zrealizować poprzez zapamiętywanie, zamiast binarnego odwiedzenia, stan który się odwiedziło wcześniej i po skończeniu algorytmu odtworzyć ścieżkę.

6 Przykłady

6.1 Przykład 1

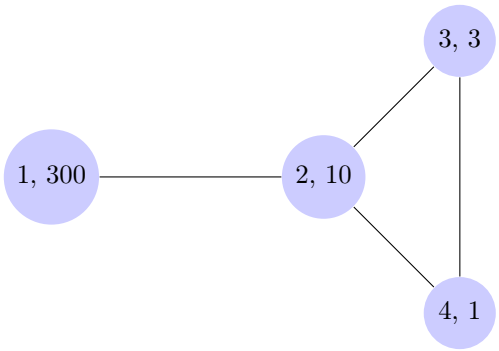
Dla podanego niżej grafu poszukujemy przejścia od 2 do 1 z kosztem 336.



Taki koszt możemy uzyskać przechodząć po kolei wierzchołki: 2 1 2 1 2 1 3 1 3 1 3 1

6.2 Przykład 2

Dla podanego niżej grafu poszukujemy przejścia od 1 do 2 z kosztem 335.



Taki koszt możemy uzyskać przechodząć po kolei wierzchołki: 1 2 3 4 2 4 2

7 Badanie wydajności

Testy przeprowadziłem przy użyciu biblioteki jmh.

Dla grafu o 98 wierzchołkach i 322 krawedziach i poszukiwanym koszcie 87700:

```
# Run progress: 0,00% complete, ETA 00:00:30
# Fork: 1 of 1
# Warmup Iteration   1: 2880978615,000 ns/op
# Warmup Iteration   2: 84,877 ns/op
# Warmup Iteration   3: 71,653 ns/op
# Warmup Iteration   4: 64,338 ns/op
# Warmup Iteration   5: 65,833 ns/op
# Warmup Iteration   6: 65,949 ns/op
# Warmup Iteration   7: 64,243 ns/op
# Warmup Iteration   8: 67,198 ns/op
# Warmup Iteration   9: 79,805 ns/op
# Warmup Iteration  10: 69,512 ns/op
Iteration   1: 74,067 ns/op
Iteration   2: 64,838 ns/op
Iteration   3: 63,919 ns/op
Iteration   4: 62,471 ns/op
Iteration   5: 65,412 ns/op
Iteration   6: 74,876 ns/op
Iteration   7: 84,080 ns/op
Iteration   8: 69,971 ns/op
Iteration   9: 62,917 ns/op
Iteration  10: 63,845 ns/op
```

Dla grafu o 8 wierzchołkach i 15 krawędziach i poszukiwanym koszcie 62600:

```
# Run progress: 0,00% complete, ETA 00:00:30
# Fork: 1 of 1
# Warmup Iteration   1: 75,470 ns/op
# Warmup Iteration   2: 51,470 ns/op
# Warmup Iteration   3: 53,789 ns/op
# Warmup Iteration   4: 59,489 ns/op
# Warmup Iteration   5: 56,197 ns/op
# Warmup Iteration   6: 57,838 ns/op
# Warmup Iteration   7: 57,181 ns/op
# Warmup Iteration   8: 64,300 ns/op
# Warmup Iteration   9: 66,562 ns/op
# Warmup Iteration  10: 57,951 ns/op
Iteration   1: 60,226 ns/op
Iteration   2: 62,950 ns/op
Iteration   3: 65,602 ns/op
Iteration   4: 58,891 ns/op
Iteration   5: 58,175 ns/op
Iteration   6: 56,850 ns/op
Iteration   7: 55,862 ns/op
Iteration   8: 58,831 ns/op
Iteration   9: 54,225 ns/op
Iteration  10: 52,665 ns/op
```

Dla grafu o 6 wierzchołkach i 6 krawędziach i poszukiwanym koszcie 33600:

```
# Run progress: 0,00% complete, ETA 00:00:30
# Fork: 1 of 1
# Warmup Iteration   1: 56,469 ns/op
# Warmup Iteration   2: 39,616 ns/op
# Warmup Iteration   3: 38,994 ns/op
# Warmup Iteration   4: 42,688 ns/op
# Warmup Iteration   5: 40,505 ns/op
# Warmup Iteration   6: 44,577 ns/op
# Warmup Iteration   7: 43,849 ns/op
# Warmup Iteration   8: 41,344 ns/op
# Warmup Iteration   9: 40,982 ns/op
# Warmup Iteration  10: 47,434 ns/op
Iteration   1: 41,646 ns/op
Iteration   2: 40,920 ns/op
Iteration   3: 42,798 ns/op
Iteration   4: 43,880 ns/op
Iteration   5: 45,265 ns/op
Iteration   6: 40,472 ns/op
Iteration   7: 40,139 ns/op
Iteration   8: 41,034 ns/op
Iteration   9: 41,645 ns/op
Iteration  10: 47,798 ns/op
```

Próba oceny złożoności na podstawie pomiaru czasu nie udała się. Przeszkodą do tego mogła być zdolność optymalizacyjna jvm. Na pewno można zauważyć, że wzrost rozmiaru problemu nie powoduje gwałtownego skoku czasu wywkonania.