**Techniki algorytmiczne**



**Problem cyklu Hamiltona w grafie**

**Data wykonania sprawozdania**: 13.05.2021

**Prowadzący**: mgr inż. Martyna Jakubowska

**Wykonawca**: plut. pchor. Adam PARCIAK

# **Opis zadania.**

Problemem, którym będę się zajmował jest problem cyklu Hamiltona w grafie.

# **Teoretyczny opis problemu.**

Cykl Hamiltona jest to cykl w grafie, który zawiera wszystkie wierzchołki grafu, przy czym każdy odwiedzany jest dokładnie jeden raz. Graf, który zawiera cykl Hamiltona jest nazywany grafem hamiltonowskim. Problem cyklu Hamiltona jest szczególnym przypadkiem problemu komiwojażera, uzyskanym poprzez ustalenie odległości między dwoma miastami. Uzyskanym wynikiem jest odpowiedź, czy w danym grafie występuję cykl Hamiltona. Jeśli występuje to dodatkowo wypisuje daną ścieżkę.

Znalezienie cyklu Hamiltona jest bardzo trudne obliczeniowo, ponieważ problem ten zalicza się do tzw. problemów NP. zupełnych, czyli dla dużej liczby wierzchołków jest on praktycznie nierozwiązywalny w sensowym czasie.

Dodatkowo, aby graf mógł posiadać cykl Hamiltona musi być spójny.

**Matematyczny zapis problemu cyklu Hamiltona w grafie:**

*Wejście*: graf nieskierowany , liczba całkowita .

*Wyjście*: TAK jeśli ma cykl Hamiltona, czyli cykl przechodzący przez każdy wierzchołek dokładnie raz, NIE w przeciwnym przypadku.

*Graf wynikowy* oznaczamy w sposób następujący: .

**1 krok redukcji**: wygenerowanie dla każdej krawędzi , fragmentu struktury wynikowej o określonych własnościach .

**2 krok redukcji**: wygenerowanie krawędzi w łączących fragmenty struktury wynikowej o określonych własnościach w tak zwane ścieżki. Dla każdego wierzchołka najpierw porządkujemy dowolnie wszystkie jego sąsiednie wierzchołki, oznaczmy je przez , gdzie jest stopniem wierzchołka . Konstrukcja ścieżki odpowiadające krawędziom incydentym z , dokonuje się przez dołączenie do zbioru krawędzi postaci:

**3 krok redukcji**: ostatnim krokiem jest wygenerowanie wierzchołków-selektorów i połączenie krawędzią każdego wierzchołka z początkiem i końcem ścieżki odpowiadającej wierzchołkowi dla każdego .

Zatem mamy:

# **Opis algorytmu dokładnego służącego do rozwiązania zadanego problemu.**

Algorytm dokładny, który wybrałem jest to **algorytm siłowy (ang. brute force)**. Polega on na wyszukaniu w grafie wszystkich możliwych cykli za pomocą permutacji wierzchołków. Algorytm ten daje nam najlepsze rozwiązanie, lecz niestety wraz ze zwiększaniem liczby wierzchołków staje się on coraz mniej wydajny. Rozpatruje wszystkie możliwe rozwiązania, aż do znalezienia rozwiązania, które jest akceptowalne jako wynik.

# **Opis wskazanego lub zaproponowanego przez siebie algorytmu aproksymacyjnego/heurystycznego służącego do rozwiązania zadanego problemu.**

Algorytm heurystyczny, który wybrałem to **programowanie dynamiczne**. Założycielem jego rozwiązania jest amerykański matematyk Richard Bellman. Opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów. Działa zdecydowanie szybciej, ale wiążę się to równoczesnym zwiększaniem zużycia pamięci.

**Działa w następujących krokach:**

1. Inicjacja macierzy boolowskiej dp [][] o wymiarach 𝑁 ∗ 2^𝑁 gdzie dp [j][i] oznacza czy dany cykl istnieje w podzbiorze czy nie jest reprezentowany przez maskę i, która odwiedza każdy wierzchołek w i raz oraz kończy się w wierzchołku j;
2. Dla przypadku podstawowego zaktualizuj dp [i][1<<1]=true, dla i w zasięgu [0,N-1];
3. Następuje wykonywanie iteracji w zakresie [1, 2^𝑁-1] za pomocą zmiennej i;
4. Jeśli wszystkie wierzchołki z bitami ustawionymi w masce i są zawarte w podzbiorze wykonuj iteracje w zakresie [1,N] używając zmiennej j, która będzie reprezentować końcowy wierzchołek ścieżki hamiltonowskiej bieżącej maski podzbioru i;
5. Jeśli wartość i oraz 2^𝑗 jest prawdziwa to wykonuj iteracje w zakresie [1,N] za pomocą zmiennej k, a jeśli wartość dp [k] [i ^ 2^𝑗] jest prawdziwa, zaznacz, że dp [j][i] jest prawdą i zakończ pętlę, w przeciwnym razie szukaj dalej;
6. Jeśli istnieje ścieżka wypisz „tak”, w przeciwnym razie „nie”.

# **Środowisko do implementacji, format wprowadzanych i wyświetlanych danych, biblioteki do generowania i analizy**

Środowisko, które będę używał do implementacji wybranych algorytmów to **Python**. Dane będą wprowadzane z tablicy.

Biblioteki, które będę wykorzystywał w implementacji to:

* **Time** – umożliwi śledzenie czasu wykonania algorytmu;
* **Tracemalloc**- monitoruje użycie zajętej pamięci;
* **Pandas** – realizuje prace na danych.

# **Teoretyczna pesymistyczna złożoność obliczeniowa.**

* **Algorytm siłowy** – problem znajdowania cyklu Hamiltona w grafie jest NP-zupełny (w dodatku silnie NP-zupełny). Oznacza to, że nie ma algorytmu, który rozwiąże ten problem w czasie wielomianowym. Znajduje wszystkie kombinacje rozwiązania więc złożoność wynosi 𝑶(𝒏!). W złożoności tej n – jest liczbą wierzchołków. Podstawową operacją algorytmu siłowego jest wypisanie wszystkich możliwych kombinacji cyklu Hamiltona, a następnie sprawdzenie czy występuje cykl.
* **Programowanie dynamiczne** – złożoność tego algorytmu wynosi 𝑶(**)**, gdzie **n** - jest liczbą wierzchołków. Pozostałe operacje nie mają wpływu. Podstawową operacją tego algorytmu jest sprawdzenie czy pierwsza ścieżka jest cyklem Hamiltona, jeśli tak to zwróci TAK, jeśli NIE to szuka dalej.

# **Teoretyczna pesymistyczna złożoność pamięciowa.**

* **Algorytm siłowy** – algorytm ten nie potrzebuje dynamicznych struktur danych oprócz wektora rozwiązań i tablic przedmiotów. Cały ten algorytm ma więc złożoność pamięciową rzędu 𝑶(𝒏). Przeszukiwanie istniejących struktur korzysta ze stałej pamięci.
* **Programowanie dynamiczne** – algorytm działa na tablicy o rozmiarze 𝑛 𝑥 𝑛 gdzie n – liczba wierzchołków. Tablica ta jest wykorzystywana przez cały czas działania algorytmu. W związku z tym złożoność pamięciowa tego algorytmu jest rzędu 𝑶(). Dodatkowe instrukcje wykonywane są pomijane.

# **Teoretyczna oczekiwana złożoność obliczeniowa.**

* **Algorytm siłowy** – w tym przypadku występuje taka sama liczba danych, ponieważ algorytm musi przeszukać wszystkie możliwości przejścia cyklu w grafie, więc teoretyczna oczekiwana złożoność obliczeniowa wynosi 𝑶(𝒏!).
* **Programowanie dynamiczne** – w przypadku teoretycznej oczekiwanej złożoności obliczeniowej czas pracy zależy tylko od wielkości danych wejściowych, więc złożoność jest rzędu 𝑶( ), gdzie n jest liczbą wierzchołków.

# **Teoretyczna oczekiwana złożoność pamięciowa.**

* **Algorytm siłowy** – cały ten algorytm ma złożoność pamięciową rzędu 𝑶(𝒏). Przeszukiwanie istniejących struktur korzysta ze stałej pamięci.
* **Programowanie dynamiczne** – w tym przypadku pamięć pracy algorytmu jest zawsze taka sama i zależy tylko od wielkości danych wejściowych. Zatem teoretyczna oczekiwana złożoność pamięciowa wynosi ona 𝑶().

# **Teoretyczna wrażliwość pesymistyczna.**

* **Algorytm siłowy** – teoretyczna pesymistyczna złożoność obliczeniowa jest rzędu 𝑶(𝒏!). W związku z tym teoretyczna wrażliwość pesymistyczna wynosi ∆𝜶 (𝒏)=(𝒏!).
* **Programowanie dynamiczne** – algorytm zależy od wielkości zadanych wierzchołków. Więc teoretyczna wrażliwość pesymistyczna wynosi ∆𝜶 (𝒏)=( **)−(**𝒏)=𝒏(**−**𝟏).

# **Teoretyczna wrażliwość oczekiwana.**

* **Algorytm siłowy** – w tym przypadku występuje taka sama liczba danych, ponieważ algorytm musi przeszukać wszystkie możliwości przejścia cyklu w grafie, więc teoretyczna wrażliwość oczekiwana wynosi 𝑶(𝒏!).
* **Programowanie dynamiczne** – tak jak w teoretycznej wrażliwości pesymistycznej, zależy od wielkości danych, musi wykonać się (). Więc teoretyczna wrażliwość oczekiwana 𝒏(−𝟏).

# **Teoretyczna dokładność.**

* **Algorytm siłowy** – algorytm ten sprawdza wszystkie kombinacje więc błąd w tym wypadku nie występuje.
* **Programowanie dynamiczne** – w tym algorytmie błąd również nie występuje. Wyniki są w pełni dokładne, ponieważ ideą tego algorytmu nie jest skrócenie kosztów obliczeń dokładności, lecz koszt złożoności pamięciowej.

# **Analiza wyników.**

1. Czasy wykonywania algorytmu siłowego przedstawiono na rysunku poniżej:

Z powyższego wykresu widać, że czas działa algorytmu siłowego zależy od ilości wierzchołków w grafie, co nie jest żadnym zaskoczeniem, ponieważ złożoność wynosi **n!**, więc możliwych kombinacji jest zależnie dużo, co wierzchołków. Dla 8 wierzchołków, algorytm działa jeszcze w sensowym czasie, lecz dla minimum 10 ten czas wynosi już ponad 700 sekund, czyli ponad 11 minut. Gdy wykonywałem algorytm dla 13 wierzchołków, algorytm ten wykonywał się już ponad godzinę i nie znalazł rozwiązania, ponieważ kombinacji było ponad 500 milionów.

1. Ilości pamięci operacyjnej algorytmu siłowego wykonywanego dla n wierzchołków przedstawiono na rysunku poniżej:

Pamięć operacyjna dla algorytmu siłowego jest stała dla zadanych wierzchołków. Jest bardzo niska, nawet dla większej ilości wierzchołków w grafie.

1. Czasy wykonywania programowania dynamicznego przedstawiono na rysunku poniżej:

Czas działania algorytmu w algorytmie wyczerpującym rośnie jeśli liczba wierzchołków jest większa. Jest to spowodowane tym, że ma więcej krawędzi do przejścia.

1. Ilości pamięci operacyjnej dla programowania dynamicznego przedstawiono na rysunku poniżej:

Pamięć operacyjna rośnie wraz z ilością danych. Im więcej wierzchołków tym wykorzystuje więcej wierzchołków. Dla przykładu algorytm programowania dynamicznego podczas szukania cyklu w 20 wierzchołkowym grafie wykorzystał 80 MB pamięci.

1. Porównanie czasów wykonywania obu algorytmów na jednym wykresie:

Na powyższym wykresie widzimy zależności działania obu algorytmów. Widać, że czas działania dla algorytmu siłowego jest o wiele dłuższy, niż dla programowania dynamicznego. Sensowy czas jest do szukania cyklu Hamiltona dla grafu o 8 wierzchołkach.

1. Porównanie ilości pamięci operacyjnej obu algorytmów na jednym wykresie:

Pamięć operacyjna dla algorytmu siłowego jest stała, a dla programowania dynamicznego rośnie wraz z ilością danych.