Wyznaczanie cyklu Hamiltona

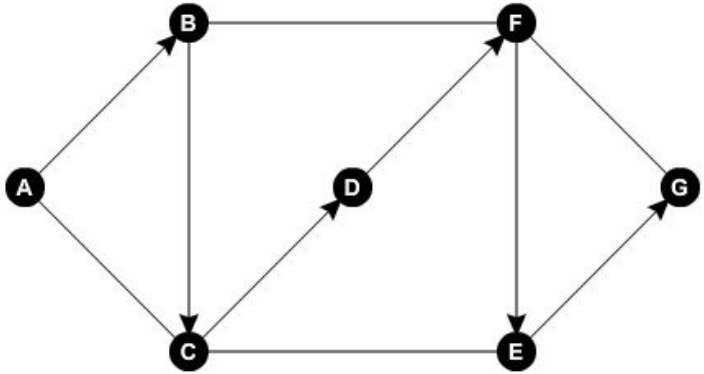
Metody optymalizacji
Temat 3, zadanie 2

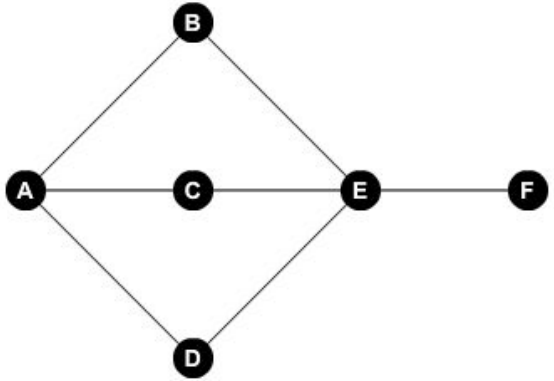
Damian Dyńdo, Michał Zimniak

Cykl Hamiltona

Cyklem Hamiltona w grafie G = <V, E> nazywamy taką ścieżkę, która - z wyjątkiem wierzchołka startowego/końcowego - przechodzi przez każdy wierzchołek dokładnie raz i kończy się w wierzchołku początkowym.

Nie każdy graf posiada taki cykl.





Cykl Hamiltona - problem NP-trudny

Znalezienie cyklu Hamiltona w grafie jest problemem NP-trudnym. To znaczy, że nie istnieje żaden znany nam algorytm potrafiący rozwiązać ten problem w czasie wielomianowym z uwagi na ilość wierzchołków w grafie.

W praktyce oznacza to, że znalezienie cyklu Hamiltona w grafie o bardzo dużym rozmiarze w rozsądnym czasie jest niemożliwe.

Istnieją twierdzenia pozwalające sprawdzić, czy dany graf posiada cykl Hamiltona, bez jego wyznaczania. Niestety są to twierdzenia w “jedną stronę”, zatem nie wykluczają one jego istnienia, a jedynie pomagają sprawdzić, czy takowy cykl musi istnieć.

Ograniczenia dla problemu

Dla problemu znalezienia cyklu Hamiltona w grafie zakładamy, że:

- krawędzie w grafie nie mają wag, lub wszystkie wagi są jednakowe
  - problem znajdujący najkrótszy cykl Hamiltona w grafie to **problem komiwojażera**
- dla uogólnienia, zakładamy że graf posiada krawędzie skierowane
  - jeśli interesuje nas graf nieskierowany, to każdą krawędź nieskierowaną można przedstawić jako parę krawędzi skierowanych

Model matematyczny

Dane wejściowe:

- Graf G = <V, E>
  - V = { v_1, v_2, ..., v_n } - zbiór wierzchołków
  - E = { e_1, e_2, ..., e_n } - zbiór krawędzi
    - $e_k = (v_i, v_j)$ - ścieżka e_k prowadząca z wierzchołka v_i do wierzchołka v_j

Dane wyjściowe:

- Ścieżka s = { s_1, s_2, ..., s_n } taka, że:
  - dla i = 1, 2, ..., n-1 krawędź $(s_i, s_{i+1}) \in E$
  - $(s_n, s_1) \in E$
  - $s_i \neq s_j \Leftrightarrow i \neq j$

Metoda pełnego przeglądu - idea

Metoda pełnego przeglądu polega na sprawdzeniu wszystkich możliwych permutacji zbioru wierzchołków V i dla każdego z nich sprawdzenie, czy taka kolejność odwiedzania wierzchołków jest możliwa (i tworzy cykl).

Takie rozwiązanie daje złożoność czasową O(n!) i jest bardzo wolne nawet dla grafów rzadkich (posiadających małą ilość krawędzi).

Metoda pełnego przeglądu - usprawnienie

Zauważmy, że mając dowolną permutację **π** ciągu wszystkich wierzchołków:

$$\boldsymbol{\pi} = (p_1\ p_2\ p_3\ \ldots\ p_n)$$

To jeśli istnieje $k < n$, takie że $(p_1\ p_2\ \ldots\ p_k)$ nie stanowi ścieżki w grafie, to dowolna permutacja zaczynająca się od $(p_1\ p_2\ \ldots\ p_k)$ **nie** będzie rozwiązaniem tego problemu, tak więc możemy je z góry pominąć.

Algorytm z powracaniem - idea

Zamiast generować całe permutacje od razu, będziemy budować je w taki sposób, aby z góry odrzucać te permutacje, których początki nie dają rozszerzyć się do poprawnego rozwiązania.

Takie rozwiązanie dalej pesymistycznie jest o złożoności O(n!), jednak w praktyce działa ono znacznie szybciej, szczególnie dla grafów rzadkich, gdyż nigdy nie rozważamy takiego ciągu, który nie może być prefiksem permutacji wynikowej.

Algorytm z powracaniem - pseudokod

Algorytm znajdowania cyklu Hamiltona w grafie G = <V, E>:

1. HamiltonianCycle(k, n) :=
    a. Dla każdego y, takiego, że $(X_{k-1}, y) \in E$
        i. Jeśli k = n oraz $y = X_0$
            1. Wypisz rozwiązanie $(X_0, X_1, ..., X_n)$
        ii. W przeciwnym przypadku, jeśli $y \notin \{ X_0, X_1, ..., X_{k-1} \}$
            1. $X_k = y$
            2. HamiltonianCycle(k + 1)
2. $X_0 = v_0$
3. HamiltonianCycle(1, |V|)

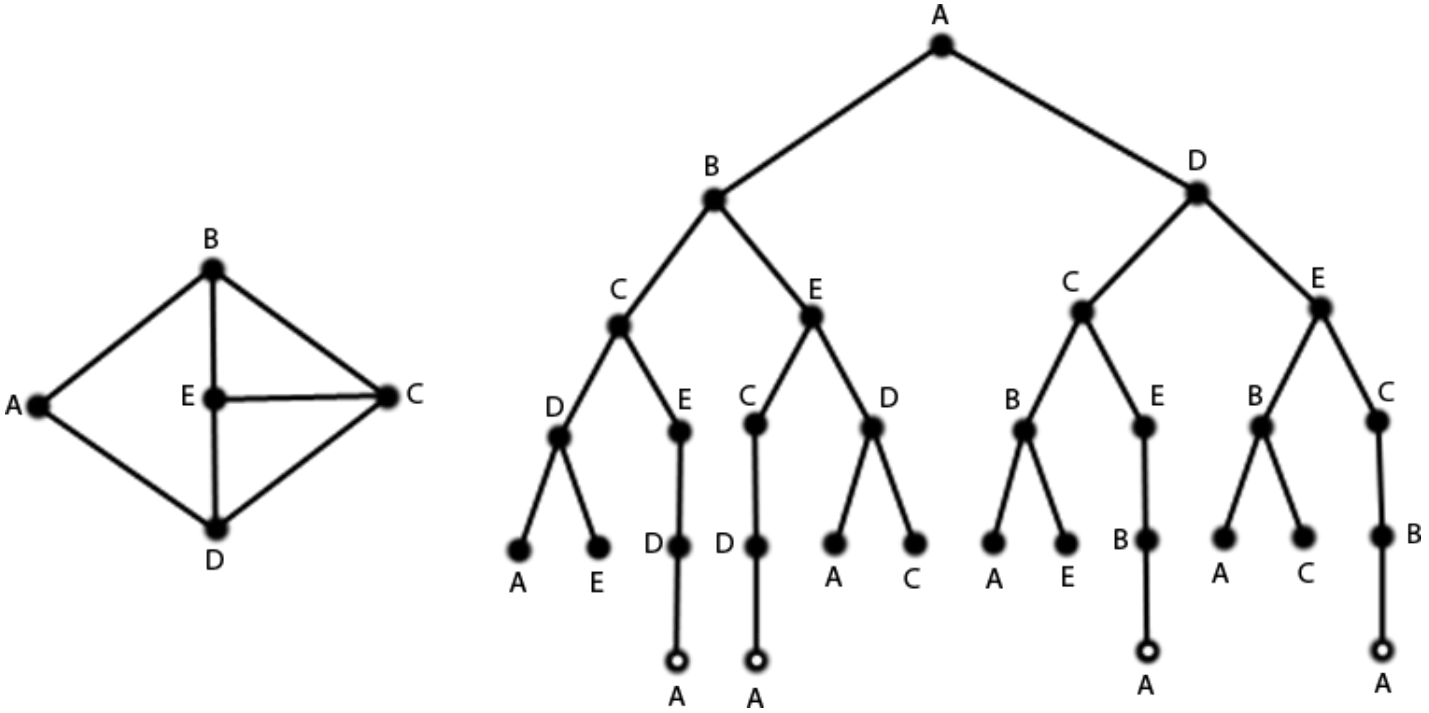
Algorytm z powracaniem - uwagi

- wybór wierzchołka startowego nie ma wpływu na znalezienie wyniku, gdyż w cyklu Hamiltona znajdują się wszystkie wierzchołki w grafie

- algorytm ten ma pesymistyczną złożoność czasową rzędu O(n!), jednak dla grafów rzadkich jest on znacznie szybszy

- w przypadku grafów gęstych, jeśli interesuje nas sam fakt posiadania cyklu Hamiltona, zamiast go znaleźć, można skorzystać z twierdzeń Diraca (1952) lub twierdzenia Orego (1961)

Algorytm z powracaniem - przykład



Implementacja

- Projekt napisany w C++11
- Dostępny na GitHubie
  - Licencja MIT

Najważniejsze klasy projektu:

- Graph
- RandomGraphFactory
- HamiltonianCycleFinder
  - NaiveHamiltonianCycleFinder
  - OptimisedHamiltonianCycleFinder

Sposób uruchomienia

- “Plug&Play”
  - Brak zewnętrznych zależności

- Kompilacja za pomocą (m. in):
  - Microsoft Visual Studio (≥ 2013)
  - GCC (≥ 4.9)
  - Clang (≥ 3.6)

- Wymagania
  - Kompilator zgodny ze standardem C++11

Zestawy danych

- Zestaw danych składa się z pary (ilość_wierzchołków, ilość_krawędzi)
- Klasa RandomGraphFactory pozwala na wygenerowanie losowych danych
- Zalecane są obliczenia dla stosunkowo małych danych testowych
    - Uwaga: obliczenia dla n ≥ 15 wymagają bardzo długiej ilości czasu

Zestawy danych

- Wierzchołków: 10
  - Krawędzi: 24
  - Krawędzi: 45
  - Krawędzi: 70
- Wierzchołków: 13
  - Krawędzi: 30
  - Krawędzi: 50
  - Krawędzi: 80

Graph

- Metody
  - `void addEdge(unsigned int v, unsigned int u)`
  - `void removeEdge(unsigned int v, unsigned int u)`
  - `bool hasEdge(unsigned int v, unsigned int u) const`
  - `std::vector<unsigned int> neighboursOf(unsigned int v) const`

- Pola
  - `const unsigned int vertexCount`
  - `std::vector<std::vector<bool>> edges`

HamiltonianCycleFinder

- Metody
  - `std::vector<unsigned int> findFirstIn(const Graph& G)`
  - `std::vector<std::vector<unsigned int>> findAllIn(const Graph& G)`

- Pola
  - `std::string name`

NaiveHamiltonianCycleFinder

- Implementacja iteracyjna
- Generuje wszystkie permutacje zbioru wierzchołków
- Sprawdza czy któraś nie tworzy poprawnej ścieżki rozszerzalnej do cyklu
  - Jeśli tak, to znaleziono cykl Hamiltona

OptimisedHamiltonianCycleFinder

- Implementacja rekurencyjna
- Algorytm z nawracaniem
- Na k-tym poziomie rekursji próbuje dopasować k-ty wierzchołek budowanej ścieżki tylko z możliwych
    - Jeśli dojdzie do n-tego poziomu, znaleziono cykl Hamiltona

Wyniki testów

- Skupione wokół wydajności
- Porównują algorytmy:
  - naiwny
  - z nawracaniem
- Małe zestawy testowe
  - Problem, nawet dla małych zestawów testowych, wymaga ogromnych zasobów do obliczenia dokładnych wyników.

Wyniki testów

Zestaw testowy		**Algorytm naiwny**		**Algorytm z nawrotami**		**Poprawa**
# wierzchołków	# krawędzi	pierwszy	wszystkie	pierwszy	wszystkie	
10	24	4 ms	12 ms	<1 ms	<1 ms	12x
10	45	<1 ms	18 ms	<1 ms	<1 ms	18x
10	70	121 ms	115 ms	2 ms	2 ms	57x
13	30	1721 ms	3738 ms	<1 ms	17 ms	**220x**
13	50	641 ms	4150 ms	<1 ms	46 ms	90x
13	80	402 ms	4585 ms	<1 ms	298 ms	15x

Testy przeprowadzone na komputerze z procesorem Intel Core i5 3210M (2.5GHz).
Program skompilowany z optymalizacjami (O2).

Wnioski

- Znacznie szybszy od wersji naiwnej
- Im rzadszy graf, tym większy zysk
- Pozwala na obliczenie w sensownym czasie wyniku, dla lekko większych danych wejściowych
  - Nie zmienia złożoności obliczeniowej
  - Dla grafów gęstych nie jest zbyt pomocny
    - Jeśli nam to wystarczy, możliwe że łatwiej będzie udowodnić konieczność istnienia cyklu Hamiltona, aniżeli jego wyznaczenie

Literatura

- W. Lipski, “Kombinatoryka dla informatyków”
- R. J. Wilson, M. Kubale, “Wprowadzenie do teorii grafów”, 1985

Dziękujemy za uwagę!

Opracowanie wraz z projektem dostępne na
`https://github.com/michlord/Metody-optymalizacji`

Przygotowali: Damian Dyńdo, Michał Zimniak