Analiza Algorytmów
Dokumentacja wstępna projektu

Numer zadania/Temat: 13/Plan Studiów
Autor: Adrian Nadratowski
Data: 29.11.2019 r

1. **Opis problemu**
   Franek chce ułożyć sobie plan studiów i wybrać wykłady, na które się zapisze. Pomiędzy wykładami występują zależności - niektóre do zapisu wymagają ukończenia innych. Przygotować algorytm, który pokaże w jakiej kolejności powinien zapisywać się na zajęcia Franek tak, aby wszystkie zależności były spełnione.

   Przykładowe zależności dla 5 wykładów:
   1 2
   2 3
   1 3
   1 5
   Prawidłowa odpowiedź:
   1 4 2 5 3

2. **Sposób przechowywania danych**
   Zależności między przedmiotami będą odzwierciedlone przez graf skierowany. Każdy wierzchołek grafu reprezentuje jeden przedmiot. Krawędź od wierzchołka (przedmiotu) A do wierzchołka B oznacza, że przedmiot B wymaga uprzedniego zaliczenia przedmiotu A.
3. **Propozycja algorytmu dla rozwiązania problemu**
   3.1. **Terminologia**
      3.1.1. *indegree* - stopień wierzchołka określający ile krawędzi kończy się w danym wierzchołku
      3.1.2. *plan zajęć* - uporządkowana lista przedmiotów w kolejności, w jakiej powinien się na nie zapisywać
      3.1.3. *korzeń* - wierzchołek o *indegree* = 0 (reprezentuje przedmiot, dla którego nie ma już poprzedników, tzn. wszyscy poprzednicy zostali wpisani do *planu zajęć*)
   3.2. **Algorytm**
      3.2.1. **Budowa grafu**
      Program iteracyjnie zczytuje dane wejściowe. W pierwszej kolejności rozmiar grafu - *n* (liczba przedmiotów), następnie w kolejnych liniach krawędzie grafu w postaci *src dst* (zależności między przedmiotami). Inicjalnie w strukturze przechowującej graf, zapisujemy wszystkie wierzchołki na liście korzeni. Dopiero przy dodawaniu krawędzi program zapisuje w każdej iteracji wierzchołek *dst* na liście sąsiadów wierzchołka *src* oraz inkrementuje *indegree* wierzchołka *dst*. Na

koniec z listy korzeni usuwane są wszystkie wierzchołki o *indegree* > 0. W ten sposób otrzymujemy gotowy do przygotowania *planu zajęć* graf skierowany reprezentujący zależności występujące między przedmiotami.
Szacowana złożoność czasowa algorytmu budowania grafu to O(|V| + |E|)
Uwagi:
- termin lista w kontekście listy korzeni / sasiadów w rzeczywistości oznacza strukturę hashset/hashmapę (unordered_set/unordered_map w C++). Jej zastosowanie znacznie poprawia wydajność działania algorytmu budowania grafu (średni i amortyzowany koszt dostępu do elementów, wstawiania oraz usuwania elementów jest O(1))

3.2.2. Algorytm właściwy

Algorytm rozwiązujący problem ułożenia *planu zajęć* działa w oparciu o listę korzeni. Dopóki, dopóty graf nie jest pusty, algorytm wybiera dowolny element z listy korzeni, wstawia go do planu zajęć, dla każdego sąsiada dekrementuje *indegree*, uzupełnia listę korzeni o tych sąsiadów, dla których *indegree* po dekrementacji jest równe 0 oraz usuwa przetwarzany wierzchołek z listy korzeni (a więc i z grafu) dekrementując jednocześnie rozmiar grafu. Jeśli w którejś iteracji okaże się, że graf jest niepusty, podczas gdy lista korzeni jest pusta, oznacza to, że *planu zajęć* nie da się ułożyć (jest to równoznaczne z wystąpieniem cyklu w grafie, czyli cyklicznej zależności w jakiejś grupie przedmiotów). Złożoność czasowa algorytmu właściwego jest szacowana na O(|V| + |E|).

4. Założenia

- Liczba wierzchołków w grafie jest równa liczbie przedmiotów podanych wraz z zestawem par opisujących występujące pomiędzy przedmiotami zależności
- Algorytm nie gwarantuje identycznego wyniku w przypadku dwóch zestawów danych identycznych pod względem zawartości, ale różniących się kolejnością podawanych par przedmiotów
- Zarówno algorytmy budowania grafu i układania planu zajęć, jak i programy: generujący dane i testujący wydajność rozwiązania zostaną zaimplementowane w C++. Do wizualizacji wyników wykorzystany zostanie język Python.

5. Sposób generowania danych i testowania wydajności

5.1. Generator danych

- Przy realizacji zadania konieczne będzie opracowanie algorytmu generującego losowy graf skierowany. Najprawdopodobniej kryterium koniecznym do spełnienia przez generowane grafy będzie acykliczność. Można to zapewnić przez naiwne generowanie grafu, a następnie sprawdzanie np. algorytmem DFS czy w grafie nie występują cykle lub zadbaniem o acykliczność grafu jeszcze na etapie generowania danych

- Grafy generowane przez program powinny mieć jeden z dwóch stopni gęstości: rzadki/gęsty. Liczba krawędzi będzie co najwyżej n*(n-1)/2, gdzie n to liczba przedmiotów. Na potrzeby analizy złożoności przyjmiemy, że graf rzadki będzie miał liczbę krawędzi wprost proporcjonalną do liczby wierzchołków, podczas gdy graf gęsty będzie miał liczbę krawędzi wprost proporcjonalną do kwadratu liczby wierzchołków. Bazując na tych założeniach będziemy mogli zmierzyć czy przewidywana złożoność czasowa (O(n) dla grafu rzadkiego; O(n^2) dla grafu gęstego) pokrywa się z faktyczną złożonością algorytmu
- Generator będzie zapisywał grafy do plików data${ID}.txt w postaci:
  ${rozmiar grafu}
  ${srcVert dstVert}
  ${srcVert dstVert}
  .
  .
  .
  W takiej też postaci program realizujący algorytm będzie oczekiwał danych wejściowych. Dane będzie można podawać interaktywnie z klawiatury - wielkość grafu, a następnie pary przedmiotów lub z pliku.

5.2. Testowanie wydajności algorytmu

Program testujący będzie uruchamiał algorytm z odpowiednimi danymi wejściowymi i mierzył czasy wykonania. Na podstawie wyników będzie można stworzyć wykresy obrazujące przyrost czasu wykonania algorytmu wraz z przyrostem liczby danych zarówno dla grafów rzadkich, jak i dla grafów gęstych.