Dokumentacja

Miasto Borgów

Autor:
Artur Wyrozębski

Problem:
Rozbudowywany jest rekurencyjnie acykliczny graf składający się początkowo z jednego wierzchołka. Rozbudowa polega na stworzeniu trzech duplikatów aktualnego stanu grafu i połączeniu tych duplikatów dwoma węzłami oraz pięcioma krawędziami, gdzie nowo dodany węzeł łączy się z drugim nowo dodanym węzłem oraz z dwoma duplikatami grafu. Każda krawędź ma własną wagę, która jest identyczna dla wszystkich nowych krawędzi w danym kroku powiększania grafu. //do poprawy
Celem jest obliczenie sumy odległości pomiędzy wszystkimi węzłami w grafie powstałym po krokach rozbudowy opisanych na wejściu.

Analiza problemu:
W każdym kroku rozbudowy liczba wierzchołków rośnie czterokrotnie oraz dodawane są dwa nowe wierzchołki, więc po ***n*** krokach liczba wierzchołków będzie wynosiła $|V| = (5/3)*4^n-(2/3)$ (rozwiązanie równania rekurencyjnego).
Liczba krawędzi natomiast jest równa liczbie wierzchołków pomniejszonej o jeden ($|E| = |V| - 1$), co wynika z faktu, że graf jest zawsze drzewem.
Zawsze występują maksymalnie trzy krawędzie na wierzchołek, bo nowo dodany wierzchołek w danym kroku zawsze łączy się z liśćmi zduplikowanych grafów oraz z drugim dodanym wierzchołkiem. Liść grafu ma tylko jedną krawędź, a po połączeniu będzie miał dwa krawędzie.
Wagi krawędzi każdego wierzchołka, który ma dwa krawędzie, są różnowartościowe. Inaczej jest w przypadku wierzchołków, które posiadają trzy krawędzie. U nich są to identyczne wartości.
W każdym grafie rozbudowanym jak w opisie problemu występuje symetria „względem środka grafu”.
Po obliczeniu sumy odległości w grafie dla dowolnego wierzchołka i potem dodaniu wierzchołka do tego grafu tak, aby był on liściem połączonym z tym wierzchołkiem, to suma odległości dla niego będzie równa sumie sumy odległości dla sąsiadującego wierzchołka oraz poprzedniej ilości wierzchołków pomnożonej przez wagę krawędzi między dodanym wierzchołkiem, a sąsiadującym wierzchołkiem ($S_{|V|} = S_{|V|-1} + (|V|-1)*W(|V|;|V|-1)$; S – suma odległości do reszty wierzchołków dla danego wierzchołka; |V| - liczba wierzchołków; W – funkcja zwracająca wagę krawędzi między jednym wierzchołkiem a drugim). Wynika to z faktu, że każda ścieżka nowo dodanego wierzchołka-liścia do innych wierzchołków prowadzi przez wierzchołek sąsiadujący.

Pierwsza propozycja rozwiązania problemu:
Najpierw następuje całkowita rozbudowa grafu o określoną liczbę kroków.
Należy następnie ponumerować wszystkie wierzchołki i iterować od wierzchołka o numerze najmniejszym do tego o numerze największym.
W danej iteracji należy obliczyć sumę odległości od wierzchołka którego dotyczy iteracja do wierzchołków o numerach większych poprzez wykorzystanie algorytmu Depth First Search.
Przy przejściu z wierzchołka do następnego, którego się jeszcze nie odwiedziło, należy dodać wagę krawędzi do dotychczasowej odległości, która początkowo wynosi zero dla danej iteracji, a następnie dodać tę odległość do ich sumy, która również początkowo wynosi zero. Dodawanie odległości do sumy należy dokonywać tylko, gdy numer wierzchołka z którego się przeszło jest mniejszy niż numer wierzchołka będącego celem. Jeżeli z danego wierzchołka nie ma krawędzi do tych, których się nie odwiedziło, to należy odjąć wagę połączenia z poprzednim wierzchołkiem od dotychczasowej odległości i cofnąć się do niego.

Analiza jakości pierwszego rozwiązania problemu:
Złożoność czasowa algorytmu DFS jest liniowa względem liczby wierzchołków. Algorytm DFS jest w danej propozycji rozwiązania problemu wykonywany dla każdego wierzchołka. Z tych faktów wynika, że złożoność czasowa algorytmu wynosi $O(k) = k^2$, gdzie k jest liczbą wierzchołków.
Na podstawie treści problemu można zapisać równanie rekurencyjne $k_n = 4*k_{n-1} + 2$ – w tym przypadku n jest numerem kroku rozbudowy, a natomiast k jest liczbą wierzchołków, tym samym uzależniając liczbę wierzchołków od kroku rozbudowy. Rozwiązaniem tego równania rekurencyjnego jest $k_n = (5/3)*4^n-(2/3)$. Z niego wynika liczba wierzchołków po powiększeniu grafu n razy.
Dzięki temu rozwiązaniu można uzależnić złożoność czasową od liczby rozbudowań. Ta złożoność algorytmu wynosi $\mathbf{O(n) = (4^n)^2}$.

Złożoność pamięciowa algorytmu jest uzależniona od ilości wierzchołków w ostatnim kroku rozbudowy. Złożoność pamięciowa wynosi więc $\mathbf{O(n) = 4^n}$.

Druga propozycja rozwiązania problemu:
Algorytm który przedstawię opiera się na równaniach w takim celu, aby uniknąć potrzeby rozbudowy grafu do jego późniejszej analizy. Sama rozbudowa grafu ma złożoność obliczeniową $O(n)= 4^n$, więc by osiągnąć złożoność lepszą niż ta - nie można opierać się na algorytmach przeszukiwania grafu.
Sposób osiągnięcia ostatecznego równania rekurencyjnego obliczającego sumę odległości pomiędzy wszystkimi węzłami w określonym kroku rozbudowy grafu przedstawiam na zamieszczonych zdjęciach:

Analiza jakości drugiego rozwiązania problemu:
Złożoność obliczeniowa algorytmu wynosi **O(n) = n**, bo występują tylko podstawowe operacje na liczbach w tym algorytmie oraz są one wykonywane w sposób iteracyjny.
Złożoność pamięciowa wynosi **O(n) = 1**. Po pierwszej iteracji algorytmu nie ma potrzeby przydzielania programowi dodatkowej pamięci.

Informacje o implementacjach algorytmów:
Implementacje zostały napisane w języku Python 3.7.
Nie korzystano z zewnętrznych bibliotek oprócz narzędzi do profilowania.

Opis sposobu testowania algorytmów:
Do testowania poprawności algorytmu wykorzystuję testy jednostkowe, które mają sprawdzać poprawność działania poszczególnych funkcji algorytmu, grafu oraz rozbudowy grafu.
Do testowania czasu działania algorytmu oraz wymagań pamięciowych korzystam z narzędzi do profilowania. Tymi narzędziami są odpowiednio cProfile oraz memory-profiler.
cProfile pozwala na kalibrację profilowania, przez co ograniczany jest wpływ profilowania na wyniki czasu wykonywania algorytmu. Przedstawia dodatkowo wpływ czasowy poszczególnych funkcji na działanie algorytmu.
Memory-profiler analogicznie przedstawia wpływ pamięciowy poszczególnych funkcji algorytmu.

Wyniki testów:

Średnie czasy wykonania	Liczba kroków rozbudowy grafu						Liczba powtórzeń wykonania programu
Algorytm	1	2	3	4	5	6	
Pierwszy (oparty na DFS)	0.4 ms	3.8 ms	48 ms	770 ms	12 s	204 s	100
Drugi (Rekurencyjny)	0.0037 ms	0.01 ms	0.02 ms	0.0339 ms	0.05 ms	0.07 ms	10000
Drugi (Iteracyjny)	0.003 ms	0.0068 ms	0.0115 ms	0.0171 ms	0.0238 ms	0.0317 ms	10000

Wnioski i obserwacje:

Warto patrzeć na problem od różnych stron.