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.
Celem jest obliczenie sumy odległości pomiędzy wszystkimi węzłami w grafie powstałym po krokach rozbudowy opisanych na wejściu.

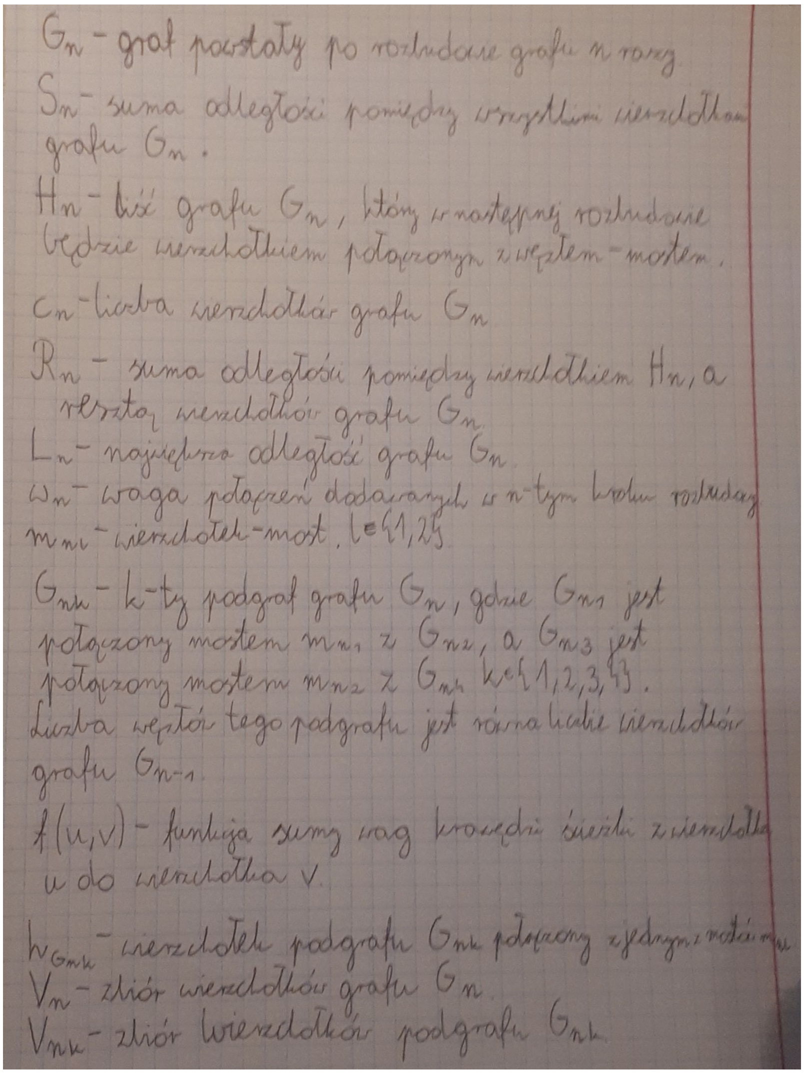
Analiza problemu:
W każdym kroku rozbudowy liczba wierzchołków rośnie czterokrotnie oraz dodawane są dwa nowe wierzchołki, więc po ***n*** krokach liczba wierzchołków będzie wynosiła $|V| = (5/3)*4^n-(2/3)$ (rozwiązanie równania rekurencyjnego).
Liczba krawędzi natomiast jest równa liczbie wierzchołków 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 **$O(n) = (4^n)^2$**.

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

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 obrazach:


G_n - graf powstały po rozbudowie grafu n razy
S_n - suma odległości pomiędzy wszystkimi wierzchołkami grafu G_n.
H_n - liść grafu G_n, który w następnej rozbudowie będzie wierzchołkiem połączonym z węzłem-mostem.
c_n - liczba wierzchołków grafu G_n
R_n - suma odległości pomiędzy wierzchołkiem H_n, a resztą wierzchołków grafu G_n
L_n - największa odległość grafu G_n
w_n - waga połączeń dodawanych w n-tym kroku rozbudowy
m_{nl} - wierzchołek-most, $l \in \{1,2\}$
G_{nk} - k-ty podgraf grafu G_n, gdzie G_{n1} jest połączony mostem m_{n1} z G_{n2}, a G_{n3} jest połączony mostem m_{n2} z G_{n4} $k \in \{1,2,3,4\}$.
Liczba węzłów tego podgrafu jest równa liczbie wierzchołków grafu G_{n-1}
$f(u,v)$ - funkcja sumy wag krawędzi ścieżki z wierzchołka u do wierzchołka v.
$h_{G_{nk}}$ - wierzchołek podgrafu G_{nk} połączony z jednym z mostów m_{nl}
V_n - zbiór wierzchołków grafu G_n
V_{nk} - zbiór wierzchołków podgrafu G_{nk}

$$\sum_{v\in V_{nk}} f(v, h_{G_{nk}}) = R_{n-1} \quad k=\{1,2,3,4\}$$

$$S_n = 4\cdot S_{n-1} + \sum_{\substack{v\in V_{nk}\\ k\in\{1,2,3,4\}}} f(v, m_{n1}) + \sum_{\substack{v\in V_{nk}\\ k\in\{1,2,3,4\}}} f(v, m_{n2}) + f(m_{n1}, m_{n2}) +$$

$$+ \sum_{v\in V_{n1}} \sum_{\substack{u\in V_{nk}\\ k\in\{2,3,4\}}} f(v,u) + \sum_{v\in V_{n2}} \sum_{\substack{u\in V_{nk}\\ k\in\{3,4\}}} f(v,u) + \sum_{v\in V_{n3}} \sum_{u\in V_{n4}} f(u,v)$$

$$\sum_{v\in V_{n1}} f(v, m_1) = \sum_{v\in V_{n2}} f(v, m_1) = \sum_{v\in V_{n3}} f(v, m_2) = \sum_{v\in V_{n4}} f(v, m_2)$$

$$\sum_{v\in V_{n1}} f(v, m_2) = \sum_{v\in V_{n2}} f(v, m_2) = \sum_{v\in V_{n3}} f(v, m_1) = \sum_{v\in V_{n4}} f(v, m_1)$$

$$\sum_{v\in V_{n1}} f(v, m_1) = \sum_{v\in V_{n1}} f(v, h_{G_{n1}}) + c_{n-1}\cdot \omega_n = R_{n-1} + c_{n-1}\cdot\omega_n$$

$$\sum_{v\in V_{n1}} f(v, m_2) = \sum_{v\in V_{n1}} f(v, h_{G_{n1}}) + 2\cdot c_{n-1}\cdot \omega_n = R_{n-1} + 2\, c_{n-1}\cdot\omega_n$$

$$\sum_{v\in V_{n1}} \sum_{u\in V_{n2}} f(u,v) = \sum_{v\in V_{n3}} \sum_{u\in V_{n4}} f(u,v) = c_{n-1}\cdot\Big[\Big(\sum_{v\in V_{n1}} f(v, h_{G_{n1}})\Big) +$$

$$+ c_{n-1}\cdot\omega_n + c_{n-1}\cdot\omega_n\Big] + c_{n-1}\cdot\sum_{v\in V_{n2}} f(v, h_{G_{n2}}) =$$

$$= 2\, c_{n-1} R_{n-1} + 2\cdot c_{n-1}^2\cdot\omega_n$$

$$\sum_{v\in V_{n1}} \sum_{u\in V_{n3}} f(u,v) = \sum_{v\in V_{n1}} \sum_{u\in V_{n4}} f(u,v) = \sum_{v\in V_{n2}} \sum_{u\in V_{n3}} f(u,v) = \sum_{v\in V_{n2}} \sum_{v\in V_{n4}} f(u,v) =$$

$$= c_{n-1}\cdot\Big[\Big(\sum_{v\in V_{n1}} f(v, h_{G_{n1}})\Big) + 3\cdot c_{n-1}\cdot\omega_n\Big] + c_{n-1}\cdot\sum_{v\in V_{n3}} f(v, h_{G_{n3}}) =$$

$$= 2\cdot c_{n-1}\cdot R_{n-1} + 3\cdot c_{n-1}^2\cdot\omega_n$$

$$S_n = 4\cdot S_{n-1} + 4\cdot(R_{n-1} + \omega_n\cdot c_{n-1}) + 4\cdot(R_{n-1} + 2\,\omega_n\cdot c_{n-1}) +$$

$$+ 2\cdot(2\, c_{n-1} R_{n-1} + 2\, c_{n-1}^2 \omega_n) + 4\,(2 c_{n-1} R_{n-1} + 3\, c_{n-1}^2\, \omega_n) + \omega_n$$

$$S_n = 4 S_{n-1} + (12\, c_{n-1} + 8) R_{n-1} + (16\, c_{n-1}^2 + 12\, c_{n-1} + 1)\,\omega_n$$

$$R_n = R_{n-1} + f(H_n, m_{n1}) + f(H_n, m_{n2}) + \sum_{\substack{v \in V_{nk} \\ k \in \{1,2,3,4\}}} f(v, H_n)$$

$$f(H_n, m_{n1}) = L_{n-1} + \omega_n \quad f(H_n, m_{n2}) = L_{n-1} + 2\omega_n$$

$$\sum_{v \in V_{n2}} f(v, H_n) = c_{n-1} \cdot f(h_{G_{n2}}, H_n) + \sum_{v \in G_{n2}} f(h_{G_{n2}}, v)$$

$$\sum_{v \in V_{n3}} f(v, H_n) = \sum_{v \in V_{n4}} f(v, H_n) = c_{n-1} \cdot f(h_{G_{n3}}, H_n) + \sum_{v \in G_{n3}} f(h_{G_{n3}}, v)$$

$$f(h_{G_{n2}}, H_n) = L_{n-1} + 2\omega_n$$

$$f(h_{G_{n3}}, H_n) = L_{n-1} + 3\omega_n$$

$$\sum_{v \in V_{n2}} f(v, H_n) = c_{n-1} \cdot (L_{n-1} + 2 \cdot \omega_n) + R_{n-1}$$

$$\sum_{v \in V_{n3}} f(v, H_n) = c_{n-1} \cdot (L_{n-1} + 3 \cdot \omega_n) + R_{n-1}$$

$$R_n = 4R_{n-1} + 2L_{n-1} + 3\omega_n + c_{n-1}(L_{n-1} + 2\omega_n) + c_{n-1}(2L_{n-1} + 6\omega_n)$$

$$R_n = 4R_{n-1} + 2L_{n-1} + 3\omega_n + c_{n-1}(3L_{n-1} + 8\omega_n)$$

$$R_n = 4R_{n-1} + L_{n-1}(3c_{n-1} + 2) + \omega_n(8c_{n-1} + 3)$$

$$L_0 = 0 \quad L_n = 3\omega_n + 2L_{n-1}$$

$$c_0 = 1 \quad R_0 = 0 \quad S_0 = 0$$

$$S_n = \sum_{k=1}^{n} 4^{n-k} \left[(12c_{k-1} + 8)R_{k-1} + (16c_{k-1}^2 + 12c_{k-1} + 1)\omega_k \right]$$

Ten wzór pozwala wyliczać wynik iteracyjnie

Na ostatnim obrazie znajduje się równanie sumy S_n, które pozwala na utworzenie algorytmu iteracyjnego.

<u>Analiza jakości drugiego rozwiązania problemu:</u>

Złożoność obliczeniowa algorytmu rekurencyjnego wynosi **O(n) = n**^{2}. Otrzymuje się taką asymptotykę, jeżeli rozwiąże się równanie rekurencyjne.

Złożoność obliczeniowa algorytmu iteracyjnego wynosi **O(n) = n**, bo występują tylko podstawowe operacje na liczbach w tym algorytmie oraz są one wykonywane w sposób iteracyjny.

Złożoność pamięciowa algorytmu rekurencyjnego wynosi **O(n) = n**. Obliczanie występuje jak w typowym algorytmie rekurencyjnym, czyli należy policzyć najpierw poprzednie instancje, aby wyznaczyć daną instancję problemu. Podczas „schodzenia" instancji jest przydzielana ta sama pamięć stosu dla każdej instancji funkcji.

Złożoność pamięciowa algorytmu iteracyjnego wynosi **O(n) = 1**. Po pierwszej iteracji algorytmu nie ma potrzeby przydzielania dodatkowej pamięci (poza zwiększaniem pamięci zmiennych wynikającym z przetwarzania dużych liczb).

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 czasu działania algorytmu wykorzystano cProfile, który pozwala na kalibrację profilowania przez co ograniczany jest wpływ profilowania na wyniki czasu wykonywania algorytmu. Przedstawia dodatkowo wpływ czasowy poszczególnych funkcji na działanie algorytmu.
Do testowania pamięci skorzystano z funkcji getsizeof() pakietu sys.
Sposób wyznaczania współczynnika q(n):

$$q(n) = \frac{t(n)\,T(n_{mediana})}{T(n)\,t(n_{mediana})}$$

Wyniki testów:

Czas działania:

Algorytm iteracyjny

	Liczba kroków rozbudowy grafu							Liczba powtórzeń wykonania programu
Algorytm iteracyjny	5	10	15	20	25	30	35	
Średni czas wykonania (w ms)	0,0051	0,0102	0,0158	0,0218	0,0278	0,0338	0,0405	1000000
q(n)	0,9376	0,9385	0,9664	1	1,0202	1,0336	1,0616	Złożoność n

Algorytm rekurencyjny:

	Liczba kroków rozbudowy grafu							Liczba powtórzeń wykonania programu
Algorytm rekurencyjny	5	10	15	20	25	30	35	
Średni czas wykonania (w ms)	0,0226	0,0926	0,221	0,423	0,7	1,08	1,58	1000000
q(n)	0,8548	0,8757	0,9288	1,0000	1,0591	1,1348	1,2197	Złożoność n^2

Niedoszacowanie wynika głównie z konieczności wielu operacji na ogromnych liczbach.

Algorytm oparty na DFS:

	Liczba kroków rozbudowy grafu							Liczba powtórzeń wykonania programu
Algorytm oparty na DFS	1	2	3	4	5	6	7	
Średni czas wykonania (w ms)	0,00032	0,00323	0,0468	0,667	10,43	168	2651	10
q(n)	1,9651	1,2397	1,1226	1	0,9773	0,9839	0,9703	Złożoność $(4^n)^2$

Początkowe przeszacowanie wynika z niskiej wartości kroku, przez co inne czynniki mają duży w tym przypadku wpływ.

Pamięć:

Algorytm iteracyjny:

	Liczba kroków rozbudowy grafu						
Algorytm iteracyjny	1	11	21	31	41	51	Złożoność
Wykorzystanie pamięci (bajty)	136	136	144	156	160	168	
q(n)	0,872	0,872	0,923	1	1,026	1,077	1

Sprawdzano zajmowanie pamięci przez zmienne znajdujące się wewnątrz jednej i jedynej funkcji obliczającej wynik.
Niedoszacowanie wynika z otrzymywania ogromnych wyników których nie jest w stanie pomieścić zmienna o wielkości 8 bajtów i ich wzrost jest kilkunastokrotny względem poprzedniego kroku, więc jest znaczna szansa, że będzie trzeba przydzielić więcej pamięci niż było przydzielone w poprzednim kroku.

Algorytm rekurencyjny:

	Liczba kroków rozbudowy grafu						
Algorytm rekurencyjny	1	11	21	31	41	51	Złożoność
Wykorzystanie pamięci (bajty)	84	948	1908	2952	4096	5336	
q(n)	0,882	0,905	0,954	1	1,049	1,099	n

Sprawdzano zajmowanie pamięci przez zmienne wewnątrz funkcji rekurencyjnej.
Niedoszacowanie wynika z tego samego powodu, co w poprzednim przypadku.

Algorytm oparty na DFS:

	Liczba kroków rozbudowy grafu						
Algorytm oparty na DFS	1	2	3	4	5	6	Złożoność
Wykorzystanie pamięci (bajty)	144	336	1056	3936	15456	61536	
q(n)	2,341	1,366	1,073	1	0,982	0,977	4^n

Sprawdzano zajmowanie pamięci przez tablicę zawierającą wierzchołki grafu po jego rozbudowie do ostatniego kroku.
Początkowe przeszacowanie wynika z niskiej wartości kroku, przez co inne czynniki mają duży wpływ w tym przypadku.