Algorytmy Optymalizacji Dyskretnej

lista 3

Mateusz Tofil

16 grudnia 2021

1 Opis listy

Lista zaimplementowania w języku `c++`. Zadania dotyczące problemu znajdowania najkrtószych ścieżek w jednym źródłem w sieci $G = (N, A)$ o n wierzchołkach i m łukach z nieujemnymi kosztami.

2 Algorytm Dijkstry - impl. kolejką piorytetowa

2.1 Opis działania

Zaimplementowany przez mnie algorytm wykorzystuje kolejkę piorytetową. Na początku na kolejkę dodajemy wierzchołek startowy. Następnie, będziemy na nią dodawać kolejne wierzchołki, jeżeli aktualnie badania przez nas droga do wierzchołka, jest mniejsza niż dotychczas odkryta. Wteyd aktualizujemy długość najkrótszej ścieżki oraz dodajemy wierzchołek do kolejki piorytetowej. Kontynujemy, do momentu kiedy kolejka nie jest pusta.

2.2 Złożonośc obliczeniowa

Złożność obliczeniowa algorytmy Dijkstry zależy od liczby V wierzchołków i E krawędzi grafu. Złożoność obliczeniowa, algorytmu z wykorzystaniem kolejki piorytetowej wynosi $\mathcal{O}(E \log V)$.

3 Algorytm Dial'a

3.1 Opis działania

Tworzymy $n{*}w{+}1$ pojemników ('bucketów'). Indeks bucketa ('label') to długość scieżki prowadzonej od źródła. Na początku do pierwszego bucketa, o labelu 0, wstawiamy nasze źródło. Ponieważ długość ścieżki jest równa 0. Następnie w pętli, dopóki wszystkie buckety nie będą puste wykonujemy następujące czynności. Idziemy po naszych bucketach, od najmniejszego labela. Gdy napotkamy na niepusty bucket, zatrzymujemy się w nim. Tym sposobem dostatniemy się do wierzchołka/ów które mamy najkrótszą drogę w danym momencie. Ściągamy wierzchołek z bucketa. Dla ściągniętego wierzchołka, rozpatrujemy jego sąsidów. Jeżeli droga do jego sąsiada jest mniejsza niż dotychasz odkryta, wstawiamy wierzchołek do odpiwedniego bucketa (z odpowiednim labelem). W przypadku gdy, w buckecie jest więcej niz jeden wierzchołek, ściagamy z kolejki ten pierwszy element, który wstawiliśmy.

3.2 Złożność obliczeniowa

Algorytm jest efektywny dla małych wag w grafie. Złożoność obliczeniowa takiego algorytmu to $\mathcal{O}(m + n * C)$, gdzie m to liczba krawędzi (łuków), n liczba wierzchołków, C - koszt krawędzi o największym koszcie.

4 Radix Heap

4.1 Opis działania

Algorytm polega na podobnej idei do algorytmu Dial'a, z tą różnicą, że teraz mamy $\lceil \log n * w \rceil$ bucetów. Każdy bucket tym razem, przedział długość, do którego będziemy wstawiać wierzchołki. Długość przedziału dla i-tego bucekta wynosi $[2^{i-1}, 2^i - 1]$

W pętli, tak jak w algorytmie Diala, idzemy po każdym z bucketów aż nie napotkamy na nie pusty bucket. Gdy bucket będzie miał jeden wierzchołek w sobie, postępujemy podobnie jak w Dialu. Natomiast, gdy bucket zawiera więcej niż jeden wierzchołek, musimy zaktualizować label'e bucketów. (czyli dystance). Dystance prezentują się tak, że dla kolejnego $i = 0, 1, 2, 3....$, k-ty bucket, k, jest mniejsze od indeksu bucketu, w którym jest kilka wierzchołków, ma przedziały $[b_{start} + 2^{i-1}, b_{end} + 2^i - 1]$, gdzie b_{start} i b_{end} to przedziały bucketu w którym się zatrzymaliśmy.

4.2 Złożoność obliczeniowa

Zmniejszyła nam się ilość bucketów, co za tym idzie ilość bucketów do odwiedzenia. Zatem złożność obliczeniowa to $\mathcal{O}(m + n * \log{(n * C)})$

5 Wyniki

Zaimplementowane przez mnie algorytmy działają poprawnie dla małych danych. Dla dancyh testowych, które podane były na liście, algorytm Dial nie działa, a jak już działa to strasznie długo czasu mu to zajmuje.

n	m	c	$t_{dijkstra}$ w ms	t_{dial} w ms	t_{radix} w ms	nazwa pliku
1024	4096	1024	11	133761	358	Random4-n.10.0.gr
2048	8192	2048	46	-	1551	Random4-n.11.0.gr
4096	16384	4093	182	-	6622	Random4-n.12.0.gr
8192	32768	8192	729	-	28364	Random4-n.13.0.gr
16384	65536	16384	2965	-	-	Random4-n.14.0.gr

Tablica 1: Porównanie czasu działania algorytmów

n	m	c	$t_{dijkstra}$ w ms	t_{dial} w ms	t_{radix} w ms	nazwa pliku
1024	3968	1023	9	128535	64	Square-n.10.0.gr
2025	7920	2025	36	1006915	166	Square-n.11.0.gr
4096	16128	4096	150	-	581	Square-n.12.0.gr
8190	32938	8190	608	-	1728	Square-n.13.0.gr

Tablica 2: Porównanie czasu działania algorytmów

6 Wnioski

Na podstawie przeprowadzonych eksperymentów, moge stwierdzić że zaimplementowana przez mnie dijsktra jest najbardziej efektywna. Dial działa najgorzej, zajmuje najwięcej pamięci co jest spowodowane trzymaniem tak dużej ilości bucketów.

7 Wnioski - w rzeczywistości

W przypadku, kiedy byśmy mieli największą wagę krawędzi równą 1, okazało by się że algorytm Dial'a działa najszybieciej, pod warunkiem, że zaimplementował bym tak zwaną `cykliczna implementacje algorytmu diala`.