Michał Salamon
218522
https://github.com/218522/PAMSI

Sprawozdanie PAMSI

Wyszukiwanie najkrótszej ścieżki w grafie nieskierowanym, algorytm B&B.

- **Wprowadzenie**

Graf jest bardzo dobrą strukturą do zapisywania zależności pomiędzy np. fizycznymi obiektami. Jednym z przykładów, może być mapa. Wierzchołki grafu odpowiadają miastom umieszczonym na mapie, a krawędzie pomiędzy wierzchołkami odpowiadają drogom łączącym miasta. Dodatkowo krawędzie posiadają swoje „wagi", co może odpowiadać fizycznym odległościom pomiędzy miastami. Mając rozumianą w ten sposób strukturę, brakuje nam tylko algorytmu dzięki, któremu będziemy mogli poznać najkrótszą drogę pomiędzy miastem startowym a końcowym.

Jednym z algorytmów pozwalających wyłonić najkrótszą drogę pomiędzy (w tym przypadku) miastami, jest algorytm Branch and Bound.
Implementację algorytmu można wykonano na dwa sposoby:
- klasyczny B&B pracujący do momentu aż wszystkie rozpatrywane ścieżki będą dłuższe od znalezionej ścieżki do celu, lub rozpatrywane ścieżki zakończą się ślepym zaułkiem, lub zapętlą się.
- B&B zaimplementowany przy pomocy struktury odznaczającej wizytę w danym wierzchołku.

Aby zwiększyć wydajność pracy algorytmu, przy jego implementacji wykorzystałem tablicę odwiedzin (*visited_table[graph_size]).*
W przypadku implementacji algorytmu z odznaczaniem odwiedzonych wierzchołków, po znalezieniu ścieżki o najmniejszej wadze, prowadzącej do wierzchołka, odznaczam wierzchołek jako „sprawdzony" i pomijam go w kolejnych poszukiwaniach. Takie działanie znacznie skraca czas pracy algorytmu, ponieważ nie wykonujemy niepotrzebnych ruchów aby sprawdzać inne ścieżki (zawsze dłuższe) prowadzące do odznaczonego wierzchołka. Przy implementacji użyłem też tablic *final_costs[graph_size]*, przechowującej najmniejsze koszta przedostania się pomiędzy wierzchołkami i *predecessor[graph_size]*, przechowującej „poprzedników" wierzchołka, z którymi ma się najlepsze połączenie. Dzięki zastosowaniu tych dwóch zbiorów, łatwo można odtworzyć najlepszą ścieżkę i koszta jej przebycia.

Graf jest inicjowany krawędziami o losowej wadze z przedziału [1-20], łączącymi losowe wierzchołki z przedziału [0 - *graph_size()]*. Zagęszczenie grafu wynosi około 10%. Szukane połączenie to wierzchołek nr. 0 i wierzchołek nr *graph_size()*/2.

- **Tabele zawierające wyniki pomiarów**

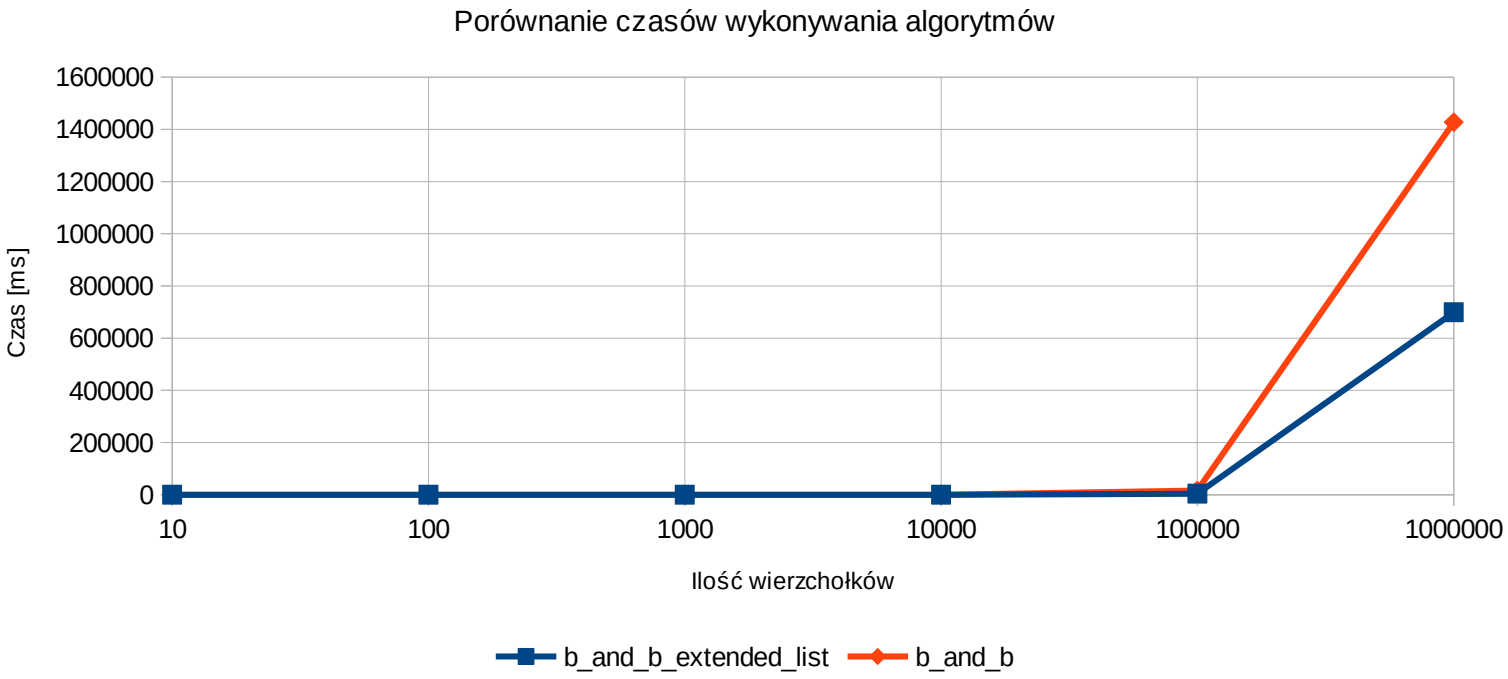
Ilość Wierzchołków	Czas wyszukiwania Najkrótszej drogi [ms]	Extended list Czas wyszukiwania Najkrótszej drogi [ms]
10	0,030	0,023
	0,031	0,020
	0,030	0,020
	0,035	0,023
	0,030	0,025
	0,030	0,030
	0,031	0,020
	0,034	0,020
	0,031	0,021
	0,030	0,024
100	0,234	0,199
	0,318	0,269
	0,314	0,266
	0,319	0,269
	0,313	0,263
	0,314	0,265
	0,316	0,266
	0,316	0,267
	0,231	0,211
	0,315	0,266
1000	1,895	1,018
	1,914	1,023
	1,840	1,020
	1,624	0,627
	1,896	1,026
	1,847	1,011
	1,868	1,025
	1,856	0,959
	1,836	1,052
	1,798	0,983
10000	79,123	38,971
	76,843	37,571
	74,539	38,798
	75,698	35,791
	76,705	39,905
	74,949	36,162
	75,135	36,256
	75,169	38,398
	74,985	36,186
	76,231	35,987
100000	16384,769	4165,350
	16642,194	4192,782
	15697,235	4054,179
	16264,936	4187,342
	15874,458	4250,560
	15978,126	4176,723
	15797,056	4158,087
	16251,714	4032,554
	15934,234	4145,912
	15904,795	4098,183
1000000	1419918,260	700549,614
	1432193,609	698265,191
	1431286,239	698554,335

Tabela 1

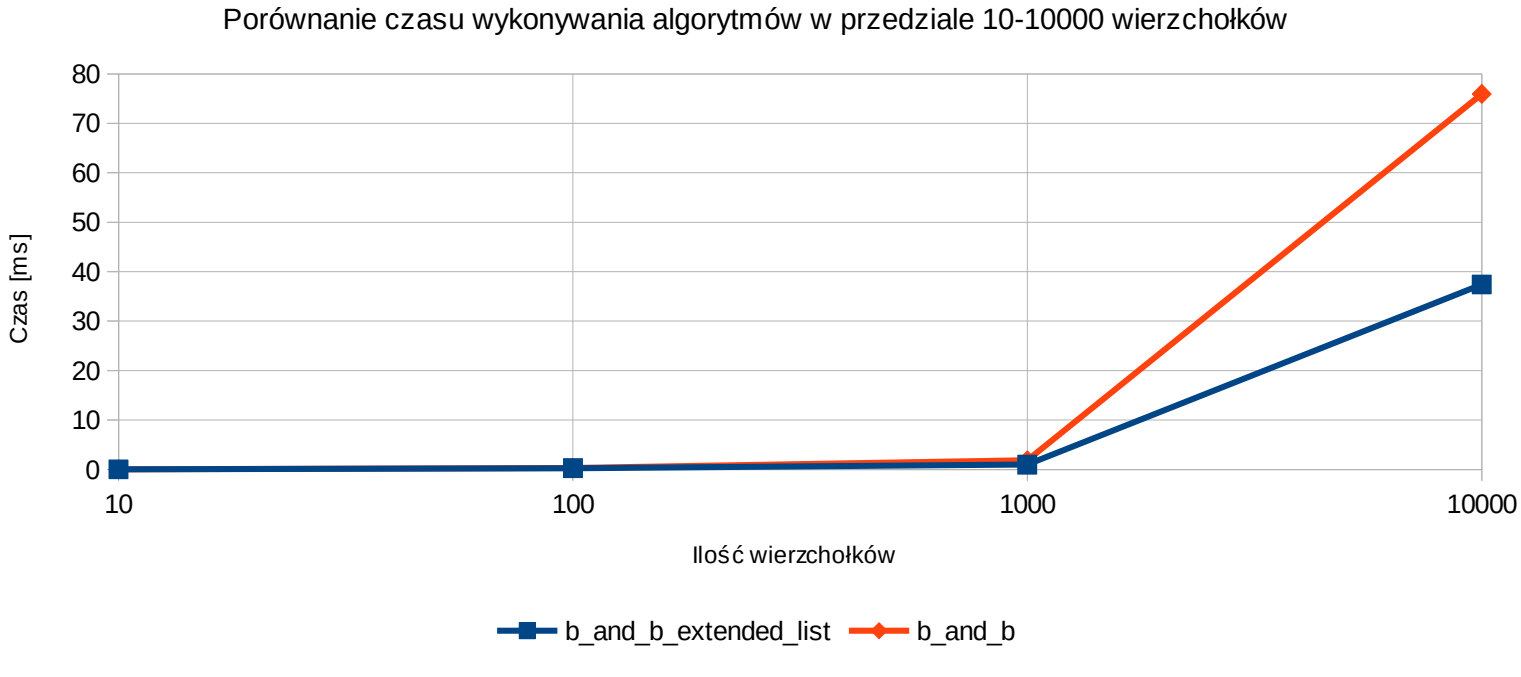
Ilość Wierzchołków	Czas wyszukiwania Najkrótszej drogi [ms]	Extended list Czas wyszukiwania Najkrótszej drogi [ms]
10	0,0312	0,0226
100	0,2990	0,2541
1000	1,8374	0,9744
10000	75,9377	37,4025
100000	16072,9517	4146,1672
1000000	1427799,3693	699123,0467

Tabela 2 – uśrednione wyniki pomiarów

Wykres I



Wykres II



- **Wnioski**

Zgodnie z przewidywaniami algorytm Branch and Bound implementowany przy pomocy struktury zapisującej odwiedzone wierzchołki, okazał się bardziej wydajny. Różnicę w szybkości wykonywania algorytmu widać w *Tabela2, Wykres I* i *Wykres II,* gdzie algorytm zaimplementowany przy uwzględnianiu odwiedzin wykonuje się w znacznie krótszym czasie. Pomijanie ścieżek przechodzących przez odwiedzone już wierzchołki znacznie skraca ilość operacji wykonywanych przez algorytm. Jedynym zauważalnym minusem tego sposobu implementacji jest potrzeba zarezerwowania dodatkowego miejsca w pamięci na kontener przechowujący *n* elementów.