Algorytm Dijkstry - Znajdowanie najkrótszej drogi w labiryncie.

OMP

Paweł Sawicki

Algorytm Dijkstry, opracowany przez holenderskiego informatyka Edsgera Dijkstrę, służy do znajdowania najkrótszej ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi.

Działanie

Mając dany graf z wyróżnionym wierzchołkiem (źródłem) algorytm znajduje odległości od źródła do wszystkich pozostałych wierzchołków. Łatwo zmodyfikować go tak, aby szukał wyłącznie (najkrótszej) ścieżki do jednego ustalonego wierzchołka, po prostu przerywając działanie w momencie dojścia do wierzchołka docelowego, bądź transponując tablicę incydencji grafu. Algorytm Dijkstry znajduje w grafie wszystkie najkrótsze ścieżki pomiędzy wybranym wierzchołkiem a wszystkimi pozostałymi, przy okazji wyliczając również koszt przejścia każdej z tych ścieżek. Algorytm ten jest przykładem algorytmu zachłannego.

Algorytm

Nazwijmy wierzchołek startowy v0. Niech odległość wierzchołka Y będzie odległością od wierzchołka v0 do wierzchołka Y. Algorytm przydzieli im odległości początkowe, a potem te odległości poprawi.

1. Przydziel każdemu wierzchołkowi odległość(d): d[v0]=0, dla reszty d[vi] = inf.
2. Zaznacz wszystkie wierzchołki jako nieodwiedzone. Ustaw v0 jako aktualny wierzchołek. Stwórz tablice nieodwiedzonych wierzchołków. 3.Dla aktualnego wierzchołka rozważ nieodwiedzonych sąsiadów i porównaj ich wagi. Wybierz najmniejszą. Następnie ustaw wierzchołek z najmniejszą wagą jako aktualny i usuń z wierzchołków nieodwiedzonych.
4. Kiedy zostaną rozważeni wszyscy sąsiedzi wierzchołka, ustaw go jako odwiedzony i usuń z nieodwiedzonych. Odwiedzony wierzchołek nie będzie więcej sprawdzany.
5. Jeżeli wierzchołek docelowy jest ustawiony jako odwiedzony(planując drogę pomiędzy dwoma konkretnymi wierzchołkami) albo jeżeli jego waga wynosi nieskończoność to koniec. Algorytm został zakończony.
6. Wybierz nieodwiedzony wierzchołek, który ma najmniejszą wage i ustaw jako aktualny wierzchołek, a potem wróć do kroku trzeciego.

Rozwiązanie

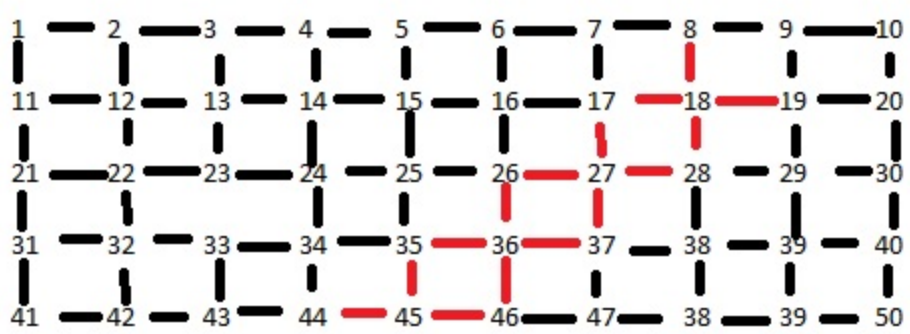
Program sekwencyjny został napisany tak aby wykonywał normalny algorytm dijkstry(odległość od wierzchołka początkowego do każdego wierzchołka w grafie). Jedynie drukowana jest odległość od wybranego wierzchołka v1 do wybranego wierzchołka końcowego.
Program równoległy jest taki sam z wyjątkiem drukowania, tutaj drukowane są dodatkowo wszystkie odległości. Dodatkowo jest wydruk od v1 do vk. Jedynie kroki algorytmu dijkstry są równoległe, wczytywanie z pliku i drukowanie są sekwencyjne.

Dane wejściowe

Generator po otrzymaniu ilości wierzchołków jakie chcemy mieć w grafie(labiryncie) generuje nam krawędzie pomiędzy wierzchołkami.
./a.out liczba_v v_koncowe ilosc_scian plik_wyjsciowy

```
psawicki@sigma:~/ITHPC$ ./a.out 50 47 3 dane.txt
psawicki@sigma:~/ITHPC$ cat dane.txt
1
47
200
1 2 1
1 0 1
1 11 1
2 3 1
2 1 1
2 12 1
```

Labirynty generowane są w formie:



Plik dane.txt (plik w folderze obok).W miejcach gdzie są czerwone połączenia wagi wynoszą 100, w czarnych 1. Wagi 100 ‘symulują’ ściany labiryntu. Na takim labiryncie wykonano pierwsze 2 testy.

Pomiar Czasu

Każdy test odpalany był 5 razy i do tabelki został wpisany średni wynik.

Test	4 wątki	Sigma (8 wątków)	32 wątki	Xeon Phi 240 wątków
Sekwencyjnie 169 dróg	Time : 0.000408515 Read : 0.000152748 Steps: 0.000035639	Time : 0.001370413 Read : 0.000112501 Steps: 0.000016141	Time : 0.000329656 Read : 0.000144364 Steps: 0.000030662	Time : 0.001110299 Read : 0.0005944 Steps: 0.0000468
OMP 50	4 threads	8 threads	32 threads	240 threads

	Time : 0.019825061 Read : 0.000111322 Steps: 0.019491082	Time : 0.033928341 Read : 0.000099244 Steps: 0.032262697	Time : 0.017841893 Read : 0.000144509 Steps: 0.017456805	Time : 0.278055531 Read : 0.000701645 Steps: 0.250724569
OPENMP 100	4 threads Time : 0.001321087 Read : 0.000245507 Steps: 0.000408094	8 threads Time : 0.00175629700 Read : 0.00012285100 Steps: 0.000389783	32 threads Time : 0.043707225 Read : 0.000116262 Steps: 0.02523804	240 threads Time : 0.273716571 Read : 0.000977124 Steps: 0.271734131
OPENMP 1000	4 threads Time : 0.018177334 Read : 0.002951574 Steps: 0.014092035	8 threads Time : 0.134693921 Read : 0.002354198 Steps: 0.110725065	32 threads Time : 0.028073719 Read : 0.000876282 Steps: 0.009335235	240 threads Time : 0.292792703 Read : 0.008462411 Steps: 0.253203801
OPENMP 10000	4 threads Time : 0.41508565 Read : 0.013204211 Steps: 0.393656946	8 threads Time : 0.193165775 Read : 0.008954283 Steps: 0.160013222	32 threads Time : 0.036099128 Read : 0.000858164 Steps: 0.021397569	240 threads Time : 0.424844348 Read : 0.086286142 Steps: 0.258833177
OPENMP 100000	4 threads Time : 39.7326325639999993 Read : 0.097112304 Steps: 39.5762254970000029	8 threads Time : 12.0788551389999999 Read : 0.09575919 Steps: 11.9192767530000001	32 threads Time : 0.043406191 Read : 0.008435412 Steps: 0.013507856	240 threads Time : 1.8360153440000002 Read : 0.9781656870000001 Steps: 0.417627523

OPENMP 500k	4 threads Time : 83.63712682 Read : 0.189289604 Steps:83.384782 6070000053	8 threads Time : 15.02447536 Read : 0.09248116 Steps: 12.2416325	32 threads Time : 0.593838284 Read : 0.088157858 Steps: 0.461420997	240 threads Time : 1.764742193 Read : 0.901385709 Steps: 0.427203886
OPENMP 1kk	4 threads Time : 141.26182324 Read : 0.241487416 Steps:139.9571	8 threads Time : 21.02447536 Read : 0.15153216 Steps: 19.9012991	32 threads Time : 0.971426213 Read : 0.1371435 Steps: 0.8161478	240 threads Time : 1.8761751 Read : 0.9813547 Steps: 0.443157231