Dawid Urbanowicz 277352

AAL-4-LS

Dokumentacja końcowa

Opis problemu

W układzie współrzędnych XY znajduje się n punktów. Każdy punkt ma przypisaną wagę będącą liczbą naturalną. Należy znaleźć ciąg punktów o następujących właściwościach: (1) każdy wektor pomiędzy kolejnymi punktami z ciągu ma nieujemne składowe, (2) łączna suma wag punktów w ciągu jest maksymalna. Rozważyć też przypadek w którym pierwszy warunek jest rozluźniony w następujący sposób: (1') każdy wektor z wyjątkiem co najwyżej jednego pomiędzy kolejnymi punktami z ciągu ma nieujemne składowe.

Metody rozwiązania

Bruteforce

Spośród wszystkich możliwych podzbiorów zbioru punktów oraz ich permutacji należy znaleźć rozwiązanie spełniające warunki zadania o największej sumie wag.

Złożoność obliczeniowa $T(n)$ tego algorytmu wynosi

$$\sum_{i=1}^{n} \binom{n}{i} * i! = \sum_{i=1}^{n} \frac{n!}{(n-i)*i!} * i! = n! * \sum_{i=1}^{n} \frac{1}{(n-i)!}$$

przy czym sumę z ostatniego równania z dobrym przybliżeniem można sprowadzić do e^1, a więc

$$T(n) = e * n!$$

$$O(n) = n!$$

Dla przypadku (1) oraz (1') można uzyskać optymalne rozwiązanie. Z powodu wysokiej złożoności obliczeniowej metoda sprawdza się jedynie dla bardzo małej ilości punktów. Dzięki wyszukiwaniu optymalnych rozwiązań dla przypadku (1') metoda ta może służyć do sprawdzania jakości nieoptymalnych rozwiązań wyszukiwanych przez kolejne algorytmy.

Programowanie dynamiczne

Punkty należy przeglądać kolumnami od lewej do prawej, a w każdej kolumnie wierszami od dołu do góry. Dla każdego punktu należy znaleźć poprzednie najlepsze rozwiązanie, z którego można dotrzeć do obecnego punktu, czyli $x_1 \leq x_2$ oraz $y_1 \leq y_2$ i do znalezionego rozwiązania dodać obecny punkt. Aby uzyskać rozwiązanie końcowe, należy znaleźć poprzednie najlepsze rozwiązanie, z którego można dotrzeć do prawego górnego rogu macierzy.

Jest n punktów (a więc n iteracji) i dla każdego punktu należy przeszukać wszystkich poprzedników, a więc złożoność tego algorytmu wynosi $O(n) = n^2$.

Dla przypadku (1) metoda daje rozwiązania optymalne. Dla przypadku (1') dwukrotne użycie tego algorytmu z usunięciem punktów wybranych w pierwszym przebiegu daje rozwiązania nieoptymalne, lecz dostatecznie dobre.

Ulepszone programowanie dynamiczne

Ogólna zasada jest taka sama jak w poprzednim algorytmie, jednak należy ustalić dwa porządki: przeglądanie kolumnami od lewej do prawej, a w każdej kolumnie wierszami od dołu do góry oraz przeglądanie wierszami od dołu do góry, a w każdym wierszu kolumnami od lewej do prawej. Przy przeglądaniu punktów w kolejności pierwszej i sprawdzaniu poprzedników w kolejności drugiej nie musimy się martwić o sprawdzenie, czy z poprzedniego punktu można dotrzeć do obecnego, gdyż albo taka operacja jest możliwa, albo dla danego punktu nie zostało jeszcze wyznaczone rozwiązanie. Należy więc znaleźć poprzednie rozwiązanie o największej sumie wag (niewyznaczone rozwiązanie ma sumę wag równą 0).

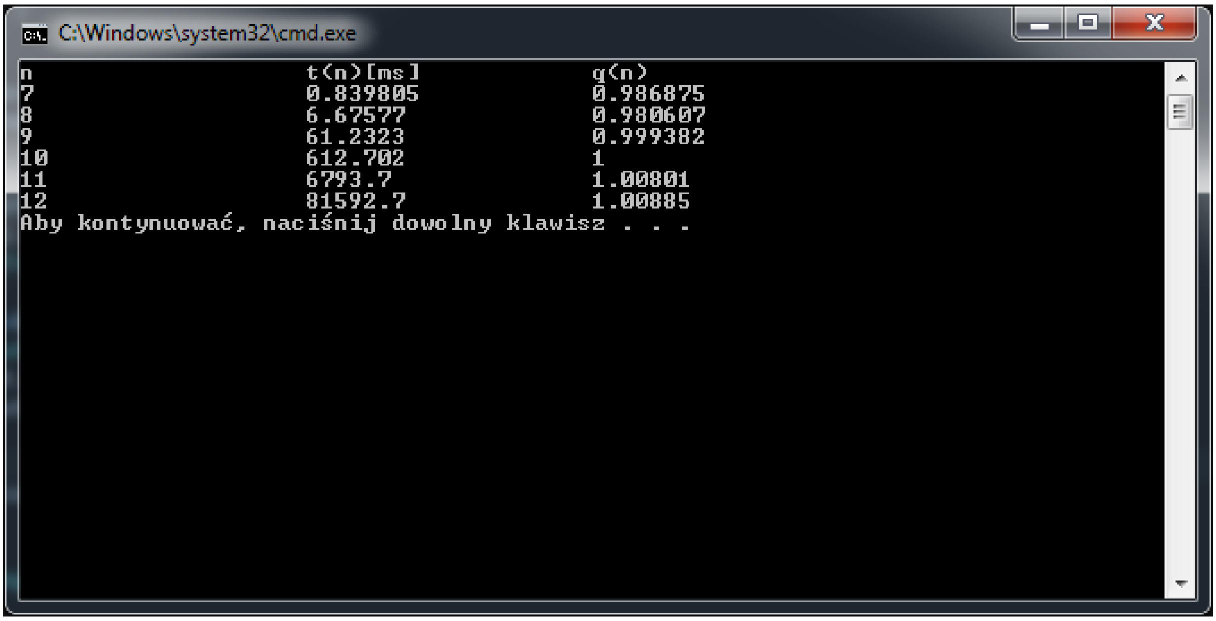
Tak jak w poprzednim algorytmie należy przejść przez n punktów i dla każdego punktu znaleźć poprzednie najlepsze rozwiązanie, jednak dzięki trzymaniu rozwiązań w drzewie przedziałowym można zredukować złożoność do $O(n) = n * logn$.

Podobnie jak w poprzednim algorytmie, dla przypadku (1) metoda daje rozwiązania optymalne, a dla przypadku (1') dwukrotne przejście daje rozwiązania wystarczająco dobre.

Pomiary czasu wykonania

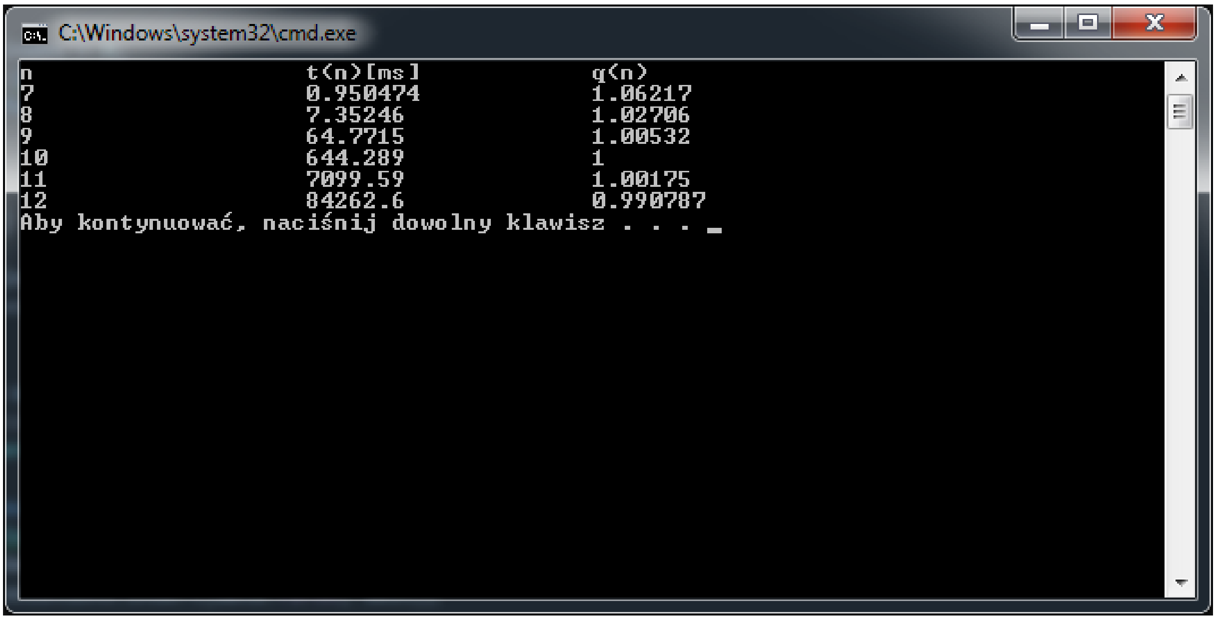
Bruteforce

`--mode=3 --alg=1 --size=7 --inst=10 --iter=6 --step=1`




```
n                    t(n)[ms]             q(n)
7                    0.839805             0.986875
8                    6.67577              0.980607
9                    61.2323              0.999382
10                   612.702              1
11                   6793.7               1.00801
12                   81592.7              1.00885
Aby kontynuować, naciśnij dowolny klawisz . . .
```

`--mode=3 --alg=4 --size=7 --inst=10 --iter=6 --step=1`

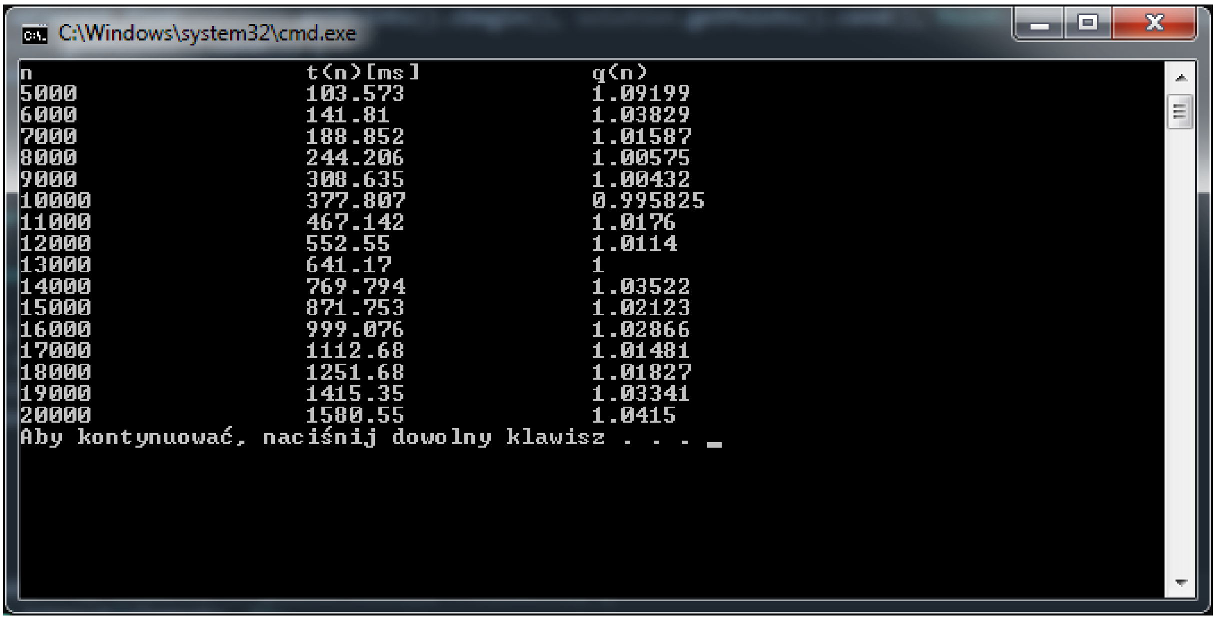



```
n                    t(n)[ms]             q(n)
7                    0.950474             1.06217
8                    7.35246              1.02706
9                    64.7715              1.00532
10                   644.289              1
11                   7099.59              1.00175
12                   84262.6              0.990787
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Pomiar czasu potwierdza poprawne wyznaczenie złożoności obliczeniowej. Z powodu długiego czasu wykonania nie jest możliwe przeprowadzanie testów dla większego rozmiaru problemu.

Programowanie dynamiczne

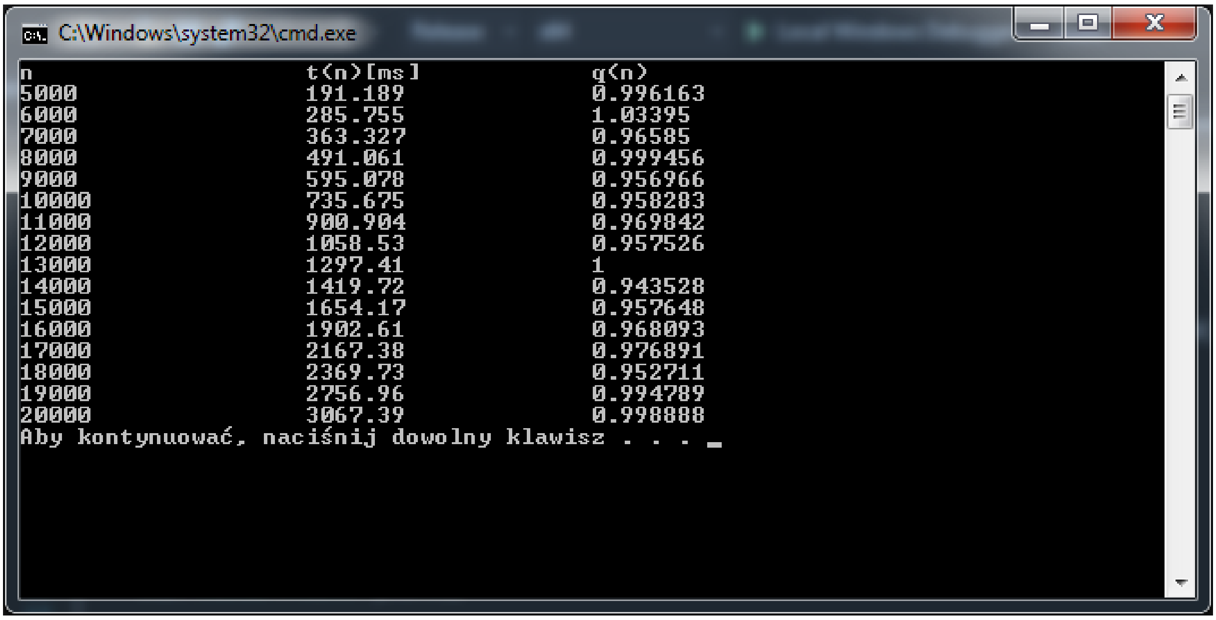
`--mode=3 --alg=2 --size=5000 --inst=10 --iter=16 --step=1000`



n	t(n)[ms]	q(n)
5000	103.573	1.09199
6000	141.81	1.03829
7000	188.852	1.01587
8000	244.206	1.00575
9000	308.635	1.00432
10000	377.807	0.995825
11000	467.142	1.0176
12000	552.55	1.0114
13000	641.17	1
14000	769.794	1.03522
15000	871.753	1.02123
16000	999.076	1.02866
17000	1112.68	1.01481
18000	1251.68	1.01827
19000	1415.35	1.03341
20000	1580.55	1.0415

Aby kontynuować, naciśnij dowolny klawisz . . .

`--mode=3 --alg=5 --size=5000 --inst=10 --iter=16 --step=1000`



n	t(n)[ms]	q(n)
5000	191.189	0.996163
6000	285.755	1.03395
7000	363.327	0.96585
8000	491.061	0.999456
9000	595.078	0.956966
10000	735.675	0.958283
11000	900.904	0.969842
12000	1058.53	0.957526
13000	1297.41	1
14000	1419.72	0.943528
15000	1654.17	0.957648
16000	1902.61	0.968093
17000	2167.38	0.976891
18000	2369.73	0.952711
19000	2756.96	0.994789
20000	3067.39	0.998888

Aby kontynuować, naciśnij dowolny klawisz . . .

Również i dla tego algorytmu teoretyczna złożoność obliczeniowa zgadza się z rzeczywistymi wynikami.

Ulepszone programowanie dynamiczne

--mode=3 --alg=3 --size=10000 --inst=10 --iter=21 --step=1000

C:\Windows\system32\cmd.exe

```
n                    t(n)[ms]             q(n)
10000                42.1505              0.836135
11000                47.4711              0.847302
12000                53.2494              0.863161
13000                59.3162              0.880043
14000                65.6862              0.897917
15000                71.7448              0.908786
16000                80.9657              0.955072
17000                85.9274              0.948046
18000                93.5406              0.96902
19000                101.548              0.99113
20000                108.41               1
21000                116.71               1.02027
22000                124.622              1.03508
23000                133.248              1.05392
24000                139.545              1.05327
25000                149.841              1.08137
26000                155.402              1.0742
27000                164.278              1.08946
28000                174.191              1.10999
29000                181.861              1.11508
30000                191.391              1.13066
Aby kontynuować, naciśnij dowolny klawisz . . .
```

--mode=3 --alg=6 --size=10000 --inst=10 --iter=21 --step=1000

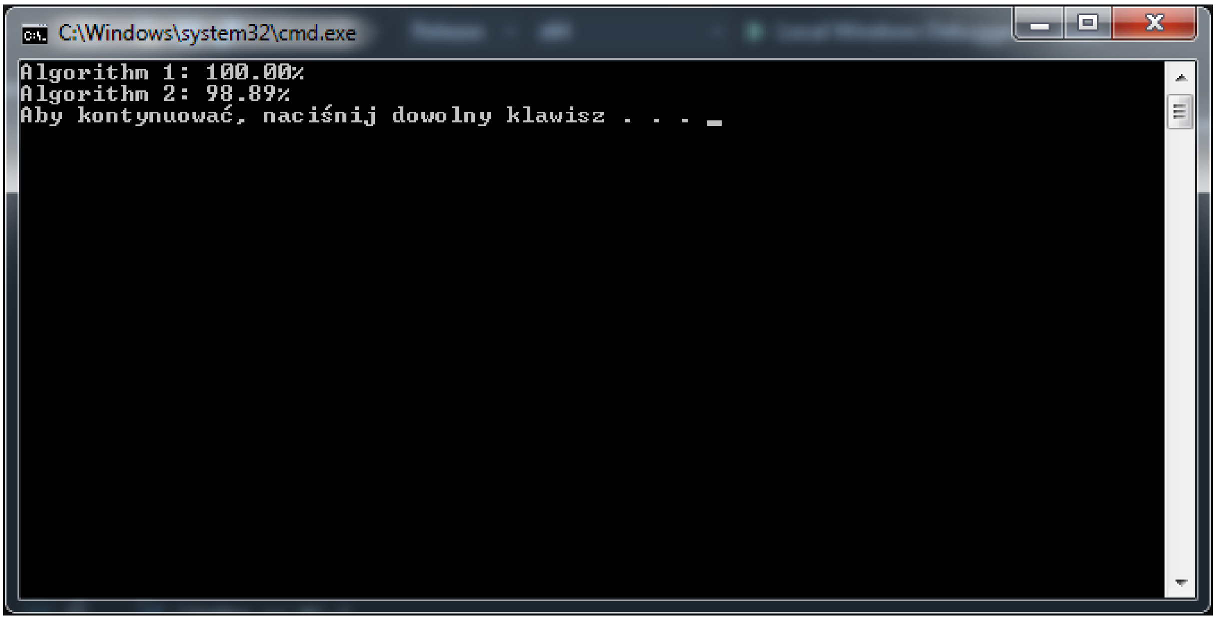
C:\Windows\system32\cmd.exe

```
n                    t(n)[ms]             q(n)
10000                82.7236              0.824776
11000                93.3125              0.837107
12000                105.679              0.860995
13000                117.781              0.87829
14000                130.465              0.896371
15000                143.29               0.912262
16000                156.43               0.927443
17000                170.041              0.942937
18000                185.011              0.963304
19000                198.617              0.974337
20000                215.694              1
21000                229.379              1.00784
22000                244.987              1.02271
23000                261.637              1.0401
24000                275.873              1.04657
25000                294.2                1.06713
26000                309.674              1.07589
27000                329.367              1.09785
28000                344.892              1.1046
29000                364.136              1.12218
30000                381.213              1.13191
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Dla tego algorytmu q(n) minimalnie rośnie, co oznacza niedoszacowanie. Może to wynikać z wpływu mniej znaczącego czynnika pominiętego w O(n).

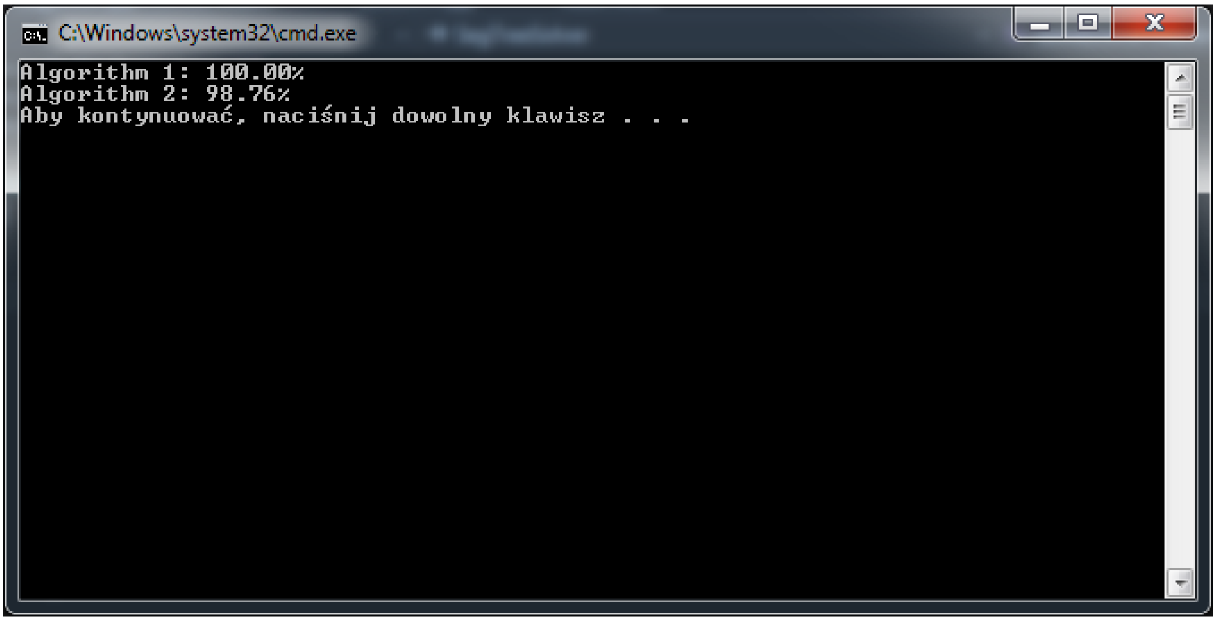
Jakość rozwiązań nieoptymalnych dla rozluźnionego warunku

`--mode=4 --alg=45 --size=6 --inst=20 --iter=5 --step=1`



Algorytm programowania dynamicznego średnio znajduje rozwiązanie o sumie wag równej 98.89% sumy wag rozwiązania optymalnego.

`--mode=4 --alg=46 --size=6 --inst=20 --iter=5 --step=1`



Ulepszony algorytm programowania dynamicznego również daje bardzo dobre wyniki. Niestety z powodu długiego czasu działania algorytmu bruteforce nie jest możliwe przeprowadzanie tych testów na problemach o większym rozmiarze.