

Analiza Algorytmów
Projekt
Dokumentacja Koncowa

Student: Volodymyr Ostruk,

Nr. Albumu: 255356,

Opiekun projektu: Lukasz Skonieczny

Politechnika Warszawska, EiTI, 2016r.

SPIS TREŚCI:

1.	<u>TREŚĆ ZADANIA:</u>	<u>3</u>
2.	<u>OPIS FUNKCJONALNOŚCI I WYMAGAN:</u>	<u>4</u>
3.	<u>SPOSÓB ROZWIĄZANIA PROBLEMUI OSZACOWANIE</u>	<u>5</u>
4.	<u>SPOSÓB AKTYWACJI PROGRAMU</u>	<u>7</u>
5.	<u>TRYBY WYKONANIA</u>	<u>8</u>
6.	<u>PLIKI WE/WY</u>	<u>9</u>
7.	<u>OPIS MODUŁÓW I INTERFEJSÓW</u>	<u>11</u>
8.	<u>TESTOWANIE I POMIAR CZASU</u>	<u>11</u>

1. TREŚĆ ZADANIA:

AAL-12-LS kartony

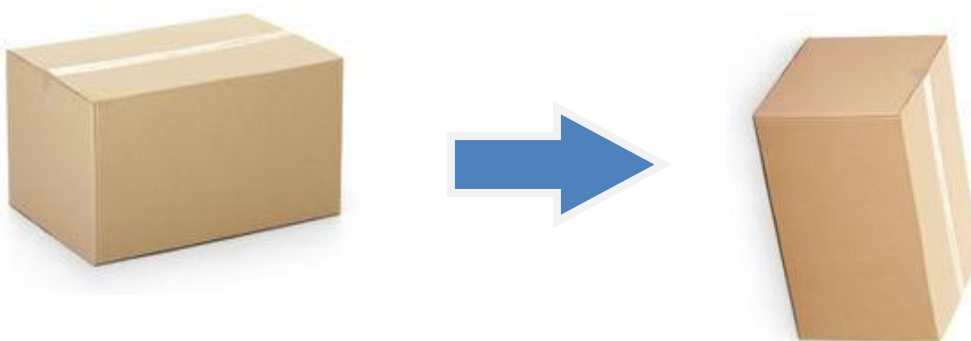
L.Skonieczny@ii.pw.edu.pl

Ortodoksyjny kolekcjoner tekturowych kartonów zaczyna narzekać na brak miejsca do przechowywania swoich cennych zdobyczy. Postanowił oszczędzić miejsce przez wkładanie kartonów jeden w drugi. W trosce o zachowanie dobrego stanu kartonów, umieszcza tylko jeden karton wewnątrz większego, a wolną przestrzeń wypełnia materiałem ochronnym. Tak zabezpieczony karton może następnie umieścić wewnątrz innego większego kartonu, ale nie może umieścić dwóch kartonów obok siebie w jednym kartonie. Dla danego zbioru kartonów należy znaleźć najlepsze upakowanie kartonów, tzn. takie, które zwalnia najwięcej miejsca.

Algorytm znajdowania upakowania kartonów nie jest typowym algorytmem plecakowym, ponieważ istnieje dodatkowe ograniczenie na umieszczanie jednocześnie tylko jednego kartonu wewnątrz drugiego bez możliwości umieszczania kartonów obok siebie. Jest to jednak pewne uproszczenie modelu, co pozwala go rozwiązać w czasie wielomianowym.

Formalny zapis zadania oraz Decyzje projektowe

- Dany jest zbiór N kartonów o kształcie prostopadłościana o wymiarach szerokości X , długości Y oraz wysokości Z .
- Trzeba znaleźć takie umieszczenie kartonów jeden wewnątrz drugiego, żeby wszystkie kartony zajmowały jak najmniej objętości, czyli upakować jak najwięcej kartonów.
- Nie wolno umieszczać kilku kartonów obok siebie - tylko jeden wewnątrz drugiego.
- Kartony w końcu będą zamykane, więc wolno obracać kartony (zmieniać długość na szerokość, albo wysokość) ponieważ kolekcjonera interesuje jak najlepsze upakowanie kartonów:



- Generator kartonów losowych będzie prostym generatorem z użyciem funkcji `rand()`

2. OPIS WYMAGAN I FUNKCJONALNOŚCI

Projekt jest implementowany w standardowym języku C++ 11. Program ma za zadanie wczytać z pliku lub wygenerować losowe dane o dostępnych kartonach, po czym znaleźć najlepsze ich upakowanie zgodnie z zasadami opisanymi w treści zadania.

A. Funkcjonalne

Program powinien umożliwiać 3 rodzaje wykonań:

1. wg danych dostarczonych ze strumienia wejściowego (standardowego lub pliku) dla sekwencji konkretnych problemów; ten tryb pozwala testować poprawności dla małych instancji
2. wg danych generowanych automatycznie (losowo) z ewentualną parametryzacją generacji określaną przez użytkownika; ten tryb także służy do testowania poprawności
3. wykonanie z generacją danych, pomiarem czasu i prezentacją wyników pomiarów.

W każdym projekcie oczekuje się:

- (1) przeprowadzenia analizy złożoności zaproponowanego algorytmu oraz
- (2) wsparcia dla wykonywania eksperymentów z pomiarami czasu dla różnych (wybranych przez użytkownika) rozmiarów problemu. Wynikiem analizy jest oszacowanie asymptoty $O(T(n))$; wsparcie dla eksperymentów pomiarowych polega na możliwości rejestracji ciągu wartości $t(n_1), \dots, t(n_k)$ konkretnych czasów wykonania dla instancji problemu o rozmiarach n_1, \dots, n_k i generacji zestawienia wyników jak w poniższej tabeli. Jeśli ocena teoretyczna $T(n)$ jest zgodna z wynikami pomiarów, to potwierdzenie tego powinno być natychmiast widoczne w tabeli.

B. Niefunkcjonalne wymagania

1. Interfejs programu zgodny z zasadami programów systemu UNIX.
2. Program umożliwia obsługę błędów i w razie ich wystąpienia powiadomi użytkownika o szczegółach.
3. Program zapisuje na bieżąco informacje o każdym wywołaniu i wyniku programu do jednego pliku podsumowującego z nazwa „every_run_result.txt”

3. SPOSÓB ROZWIĄZANIA PROBLEMU, OSZACOWANIE I UZASADNIENIE

Można uznać, że rozwiązanie problemu składa się z trzech etapów.

Etap 1: Wczytywanie.

Podczas wczytywania danych o kartonie długości jego boków są uporządkowywane według reguły: $dlugosc \geq szerokosc \geq wysokosc$

Złożoność: N .

Etap 2: Sortowanie

Sortowanie zbioru kartonów według objętości(malejąco). W wyniku otrzymujemy uporządkowany zbiór kartonów od największego do najmniejszego.

Złożoność: $N \log N$

Etap 3: Pakowanie

Utworz pierwszy stos i umiesc tam największy karton.

Dla każdego kartonu:

----- Dla każdego stosu:

----sprawdz czy karton sie zmiesci na w kartonie umieszczonym na gorze stosu stos.
jesli tak to umiesc na gore stosu. ***Karton się mieści w innym kartonie gdy
każdy jego bok się mieści w odpowiednim boku większego kartona.***

-----Jeśli karton nie został umieszczony na żaden stos, utwórz nowy stos i umieść tam
krton.

Dla każdego kartonu sprawdzamy czy się zmieści tyle razy ile jest stosów. Czyli w najgorszym przypadku, gdy zadny karton sie nie miesci *złożoność: $N \cdot N/2$*

Złożoność dokładna algorytmu: $N \cdot (N/2 + \log N + 1)$

Złożoność asymptotyczna algorytmu: $O(N \cdot N)$

Uzasadnienie optymalności algorytmu

Twierdzenie: algorytm znajduje najlepsze możliwe upakowanie kartonów.

Dane:

Dwa kartony: A i B .

Fakty: $A.x \geq A.y \geq A.z$; $B.x \geq B.y \geq B.z$

Definicja: karton B się mieści w kartonie A (karton A jest większy od B), gdy:

$A.x > B.x$ oraz $A.y > B.y$ oraz $A.z > B.z$.

Liczenie objętości: $vol(A) = A.x * A.y * A.z$; $vol(B) = B.x * B.y * B.z$

Algorytm będzie niepoprawny wtedy, gdy karton o większej objętości można umieścić wewnątrz kartonu o mniejszej objętości

Uzasadnienie przez sprzeczność. Niech istnieją taki zbiór kartonów, dla którego istnieje rozwiązanie lepsze niż znalezione przez algorytm. To znaczy, że pewne kartony zostały umieszczone w niepoprawnych stosach. Taki przypadek możliwy tylko wtedy, gdy istnieje taki karton B który się mieści w A ale objętość $vol(B) > vol(A)$ (tylko wtedy algorytm nie wylapie karton w porzadnym miejscu).

Czyli

(1) B się mieści w A : $A.x > B.x \ \&\& \ A.y > B.y \ \&\& \ A.z > B.z$.

(2) Objętość B jest większa od A : $A.x * A.y * A.z < B.x * B.y * B.z$

Z (1) wynika: $(A.x/B.x > 1) \ \&\& \ (A.y/B.y > 1) \ \&\& \ (A.z/B.z > 1)$

Z (2) wynika: $(A.x/B.x) * (A.y/B.y) * (A.z/B.z) < 1$

Mamy sprzeczność!, więc założenia są nieprawdziwe i **program znajduje najlepsze możliwe upakowanie kartonów.**

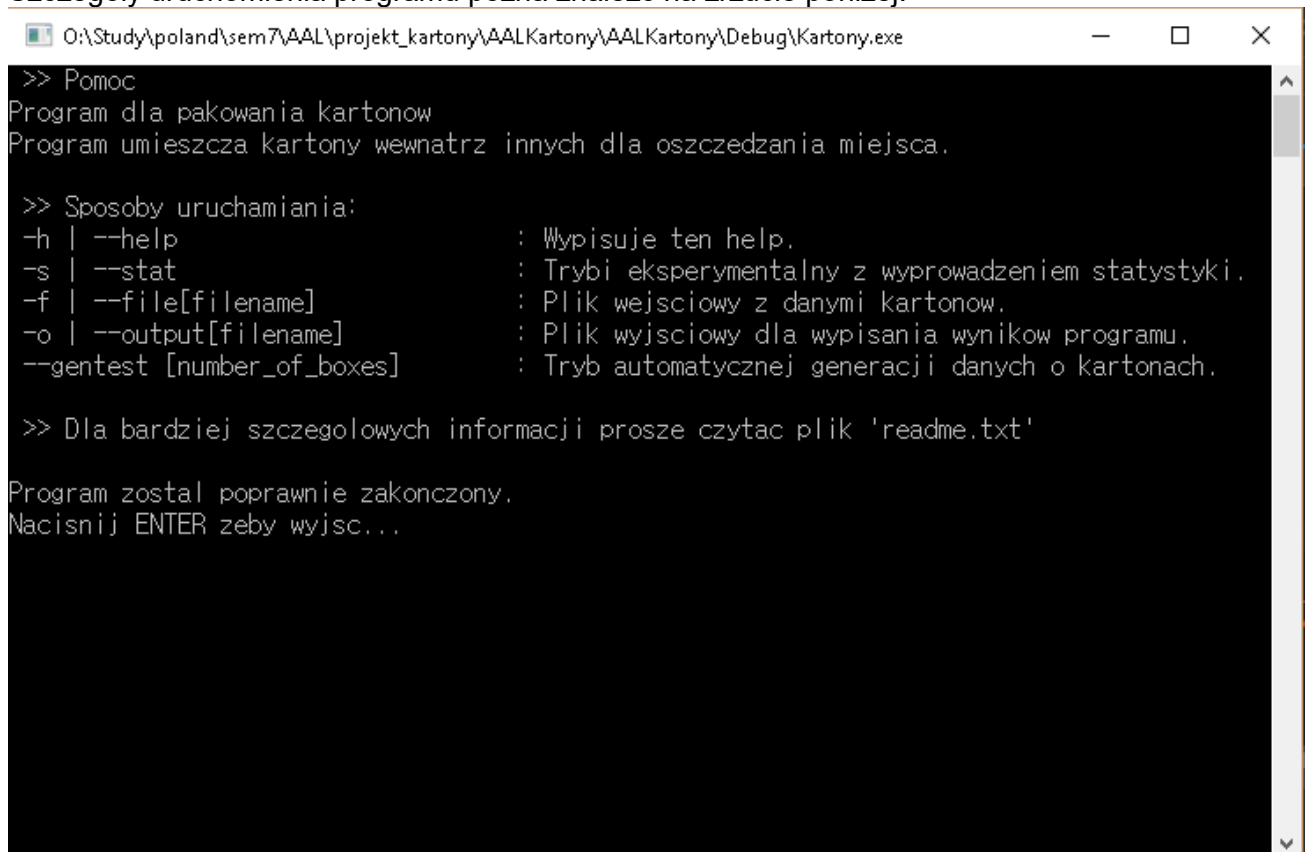
4. SPOSÓB AKTYWACJI PROGRAMU

Wybrany interfejs użytkownika to interfejs tekstowy(interfejs linii poleceń). Program jest w całości zgodny z konwencją systemu UNIX.

W szczególności :

- Nieprawidłowe uruchomienie programu(bez potrzebnych argumentów) powoduje ten sam efekt co i uruchomienie programu z parametrem(-h), czyli wypisanie wszystkich prawidłowych form aktywacji i wskazanie na plik 'readme.txt'.

Szczegóły uruchomienia programu można znaleźć na zrzucie poniżej:



```
>> Pomoc
Program dla pakowania kartonow
Program umieszcza kartony wewnatrz innych dla oszczedzania miejsca.

>> Sposoby uruchamiania:
-h | --help           : Wypisuje ten help.
-s | --stat           : Tryb eksperymentalny z wyprowadzeniem statystyki.
-f | --file[filename] : Plik wejscowy z danymi kartonow.
-o | --output[filename] : Plik wyjsciowy dla wypisania wynikow programu.
--gentest [number_of_boxes] : Tryb automatycznej generacji danych o kartonach.

>> Dla bardziej szczegolowych informacji prosze czytac plik 'readme.txt'

Program zostal poprawnie zakonczony.
Nacisnij ENTER zeby wyjsc...
```

Pakiet przenosny:

Pakiet stworzony byl dla koncowego uzytkownika dla mozliwosci uruchomienia programu bez przegladania zrodel i ich kompilacji.

Pakiet miesci pliki:

Kartony.exe dla uruchomienia programu na Windows (z uzyciem cmd)

KanrtonyLinuxProgram dla uruchomienia na Linux(polecenie ./KartonyLinuxProg..)

5. TRYBY WYKONANIA

Program umożliwia 3 rodzaje wykonania:

1. według danych dostarczonych ze strumienia wejściowego (pliku wejściowego) dla sekwencji konkretnych problemów(testowania poprawności dla małych instancji)

Dla wykonania tego trybu, podczas uruchomienia programu trzeba przekazać argument linii poleceń:

-f nazwa_pliku_z_opisem_kartonów

lub:

--filename nazwa_pliku_z_opisem_kartonów

Program pozwala też na podanie nazwy pliku wyjściowego za pomocą argumentu programu:

-o nazwa_pliku_wyjściowego

lub:

--output nazwa_pliku_wyjściowego

W przypadku nie podania nazwy pliku wyjściowego program użyje domyślnej ścieżki oraz nazwy pliku składającej się z dokładnego czasu wygenerowania pliku.

2. Według danych generowanych automatycznie (losowo) z ewentualną parametryzacją generacji określaną przez użytkownika; ten tryb także służy do testowania poprawności.

Dla uruchomienia tego trybu służy polecenie(argument programu):

- -gentest ilosc_kartonow_do_wygenerowania

Przy tym trybie działają podobne zasady generowania plików wyjściowych jak i w trybie poprzednim.

3. Wykonanie z generacją danych, pomiarem czasu i prezentacją wyników pomiarów. Ten tryb można uruchomić podając jako argument programu:

- -stat

lub:

-s

Przy uruchomieniu tego trybu za pomocą interfejsu konsoli użytkownik zostanie zapytany o maksymalny rozmiar problemu(maksymalna ilość kartonów do wygenerowania) oraz ilość iteracji problemu(ile instancji problemu musi program wygenerować i obliczyć, żeby wpisać do tabelki statystyki). Program automatycznie generuje tabele wyników i wypisuje ją do konsoli. Przykładowa tabelka wynikowa może wyglądać jak poniżej:


```
O:\Study\poland\sem7\AAL\projekt_kartony\AALKartony\AALKartony\Debug\Kartony.exe
Podaj najwiekszy rozmiar problemu(np. 10000): 10000
Podaj ilosc instancji problemow(np. 20): 12
Proszę czekać, program opracowuje wynik .....
Clocks:1000  Mediana: 5831  C:0.562608

=====|=====|=====|
|  N      | t(n)[ms] | q(n)  |
|=====|=====|=====|
|  833    | 0.022    | 0.619905 |
| 1666    | 0.141    | 0.993244 |
| 2499    | 0.354    | 1.1083   |
| 3332    | 0.628    | 1.10595  |
| 4165    | 0.896    | 1.00987  |
| 4998    | 1.297    | 1.01516  |
| 5831    | 1.739    | 1        |
| 6664    | 2.34     | 1.03023  |
| 7497    | 3.195    | 1.11143  |
| 8330    | 3.869    | 1.09017  |
| 9163    | 5.149    | 1.19904  |
| 9996    | 5.671    | 1.10967  |
|=====|=====|=====|

Program został poprawnie zakończony.
```

6. PLIKI WE/WY

Przykładowy plik **wejściowy** z opisem składni jest podany poniżej:

```
# ===== Plik z danymi o Kartonach =====
# Linije zaczynajace sie od # sa ignorowane
# Format danych: trzy wartosci (szerokosc, dlugosc, wysokosc) rozdzielone spacjami.
# Kolejnosc podanie nie ma zasadniczego znaczenia

5.0 5.0 5.0
4.999 4.999 4.999
4.99 2.01 2.01
5.0 3.0 3.0
5.000 1.54 1.999
4.0 2.0 2.0
1.0 4.0 1.0
10.0 2.0 5.0
9.0 9.0 10.0
```

Przykładowy plik **wyjściowy** z opisem:

```
### PLIK WYNIKOWY Z UPAKOWANIAM I KARTONOW ###
# DATA WYGENEROWANIA : _20-01-2016_22-44-08
# ILOSC WSZYSTKICH KARTONOW PRZED PAKOWANIEM : 9
# CZAS PAKOWANIA : 0.000000 [sec]
# ILOSC PACZEK KARTONOW : 3
# CALKOWITA ZAJETA OBJETOŚĆ PO UPAKOWANIU : 955.000000
# ZAOSZCZĘDZONO MIEJSCA W POROWNANIU Z PRZED : 24.235054 %
```

UPAKOWANIA KARTONOW

ID	DLUGOŚĆ	SZEROKOŚĆ	WYSOKOŚĆ	OBJĘTOŚĆ
Upakowanie 0				
8	10	9	9	810
0	5	5	5	125
1	4.999	4.999	4.999	124.925
2	4.99	2.01	2.01	20.1601
5	4	2	2	16
Upakowanie 1				
7	10	5	2	100
4	5	1.999	1.54	15.3923
6	4	1	1	4
Upakowanie 2				
3	5	3	3	45

7. OPIS MODUŁÓ W I INTERFEJSÓ W

Program jest podzielony na dwie części logiczne: część realizującą struktury danych i algorytmy potrzebne w rozwiązaniu problemu (definicja i implementacja odpowiednich klas) oraz z część realizującą interfejs użytkownika.

A. Część realizująca struktury danych i algorytmy

Klasa **Karton** (pliki Karton.h i Karton.cpp) miesci taki dane o kartonie jak jego uporządkowane wymiary oraz niektóre metody, jak np. porównania kartonów, operator <<.

Klasa **Upakowanie**(pliki Upakowanie.h i Upakowanie.cpp) klasa chroniąca dane o wszystkich kartonach oraz o upakowaniu(stosach) kartonów, oraz metody dla realizacji pakowania(metoda upakuj).

B. Część realizująca interfejs użytkownika

Klasa **UI**(pliki UI.cpp i UI.h) – główna klasa programu, realizująca cały interfejs użytkownika, opracowanie argumentów i porządane tryby działania programu.

Klasa **PisarzPlikow**(pliki cpp i h) – klasa oddzielona od klasy UI. Realizuje zapis wyników do pliku. Oddzielona dla możliwości zmiany sposobu zachowania wyniku bez zmiany interfejsu programu.

Program główny natomiast jest startowany z pliku **main.cpp**

8. PRZYKŁADY TESTOWE

Dla sprawdzenia poprawności programu były przeprowadzone różne typy testów:

1. Testy jednostkowe(w procesie napisania programu, dla podstawowych przypadków)
2. Testy dla przypadków granicznych(wyróżnionych, złośliwych) z wykorzystaniem opcji –f
W szczególności były rozpatrzone przypadki:
 - gdy wszystkie kartony mieszczą się w jednym
 - gdy żaden karton nie miesci się w żadnym innym(z szczegółami: 1 lub 2 boka mniejszych)
 - gdy wszystkie kartony są równe
 - gdy kartony są równej objętości ale z różnymi bokami
3. Testy z generacją losowych danych z wykorzystaniem opcji –gentest n. Ten tryb był pomocniczy dla sprawdzenia zachowywania programu dla dużych instancji problemu(np. powyżej miliona kartonów). Była wykorzystana zwykła generacja za pomocą funkcji rand z ograniczeniem maksymalnej wygenerowanej liczby do 100.
Uzasadnienie wyboru prostej funkcji generacji:

Ponieważ nie istnieje przypadków w jakiś sposób wyróżniających utrudnienie algorytmu, to uznajemy generowanie losowych danych bez dodatkowych ograniczeń za dobre.

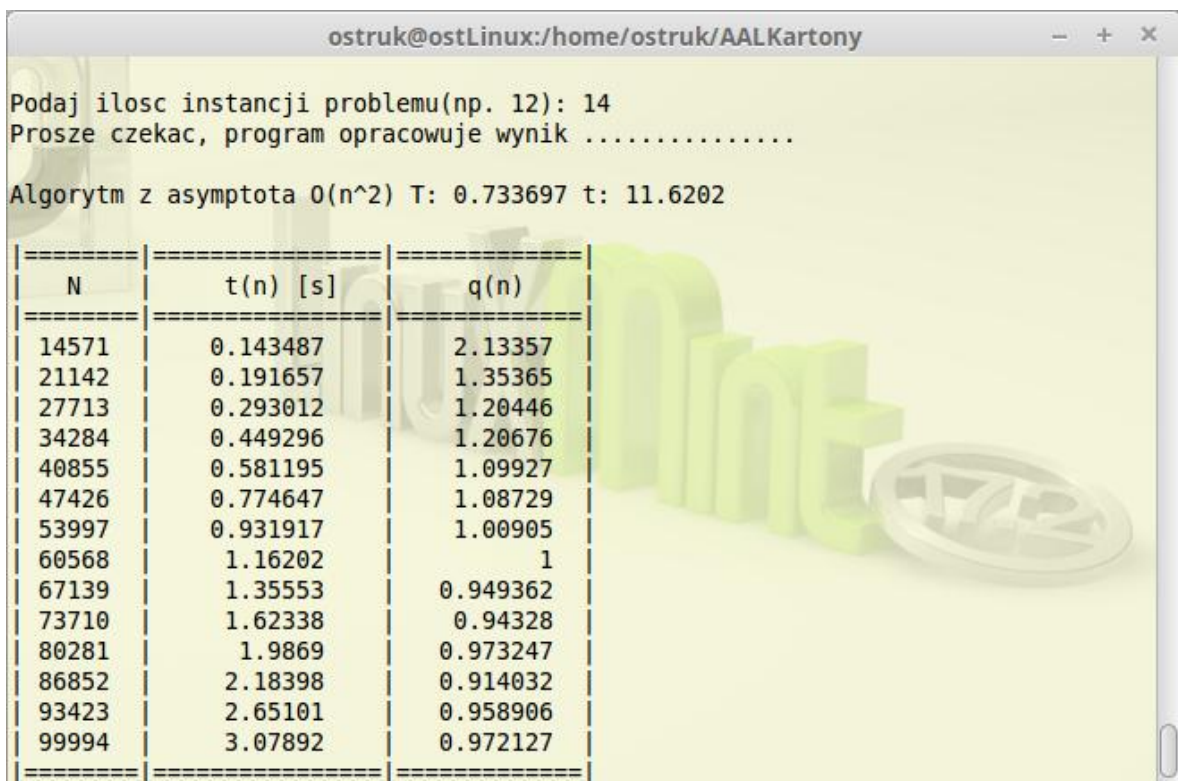
4. Testy czasowe w trybie eksperymentalnym z wykorzystaniem opcji -s. Pomiar czasu odbywa się za pomocą zalecanej funkcji clock().

Przykładowy zrzut dla trybu eksperymentalnego:

Uwaga: w rzeczywistym wykonaniu, przy generacji z wielką ilością kartonów czas działania odbiega od oszacowania $O(N*N)$ i jest mniejszy dlatego, że względna liczba stosów znaczną miarą się zmniejsza w odniesieniu do liczby kartonów i większość wygenerowanych kartonów zostaje umieszczana już w pierwszym największym pudełku. Przykładowy test dla maksymalnej 1000000(miliona) kartonów(12 instancji):

N	t(n) [s]	q(n)
90666	0.602	0.963004
173332	2.19	0.952786
255998	4.713	0.941903
338664	8.142	0.927927
421330	13.711	1.0084
503996	21.467	1.10469
586662	26.353	1.00001
669328	33.451	0.974541
751994	41.937	0.968725
834660	49.868	0.93455
917326	60.035	0.93104
999992	72.409	0.945555

Program został także przetestowany na platformie Linux Mint 17:



N	t(n) [s]	q(n)
14571	0.143487	2.13357
21142	0.191657	1.35365
27713	0.293012	1.20446
34284	0.449296	1.20676
40855	0.581195	1.09927
47426	0.774647	1.08729
53997	0.931917	1.00905
60568	1.16202	1
67139	1.35553	0.949362
73710	1.62338	0.94328
80281	1.9869	0.973247
86852	2.18398	0.914032
93423	2.65101	0.958906
99994	3.07892	0.972127