Algorytmy generujące kombinacje zbioru

Sprawozdanie z laboratorium 1 – Łukasz Konieczny LK4

Cel ćwiczenia

Podczas zajęć zapoznaliśmy się z dwoma algorytmami, służącymi do generowania kombinacji zbioru, zawierającego liczby z przedziału od 1 do zadanego „n”.

Wstęp teoretyczny

Kombinacja, jest to „k”-elementowy podzbiór „n”-elementowego zbioru, przy czym kolejność ustawienia elementów w niej nie ma znaczenia (tzn. podzbiory „123” i „321” są tą samą kombinacją). Ilość możliwych kombinacji zbioru możemy obliczyć za pomocą dwumianu Newtona:

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

Przykładową 3 elementową kombinacją zbioru „1234567” jest „357”

Algorytmy, które zaimplementowaliśmy, służą do tworzenia kombinacji uporządkowanych w porządku leksykograficznym.

Oznacza to, że kombinacje będą posortowane od najmniejszej do największej, na bazie każdej cyfry w kolejności, np. dla liczb

X = 1457

Y = 1234

Kroki sortowania:

1. Porównujemy cyfry na indeksie 0. W tym wypadku są takie same, więc sprawdzamy kolejną cyfrę.
2. Porównujemy cyfry na indeksie 1. Cyfra 4 jest większa od 2, a zatem liczba Y jest większa. Ustawiamy liczby w kolejności i kontynuujemy proces dla kolejnych liczb (jeśli istnieją, w tym wypadku skończyliśmy)

Opis algorytmów

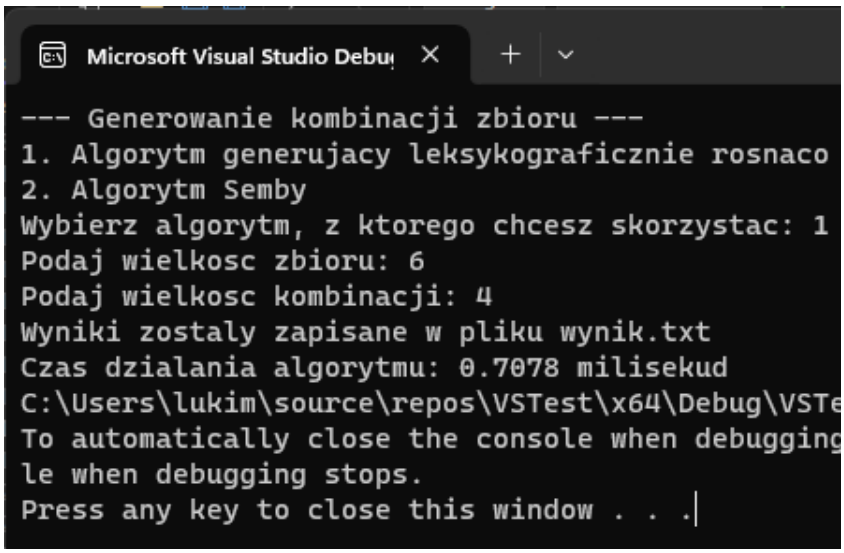
1. Algorytm generujący kombinacje w rosnącym porządku leksykograficznym:

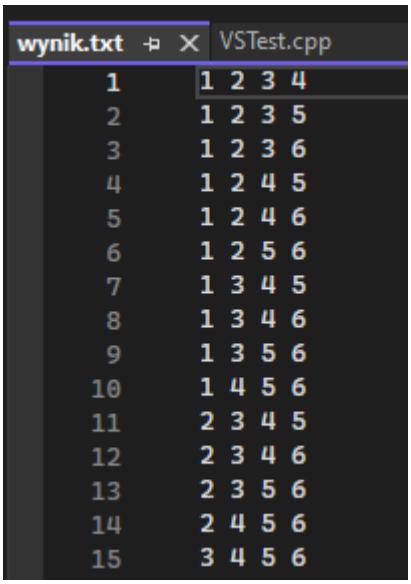
Algorytm ten dla podanych wartości „n” i „k”, gdzie „n” jest wielkością zbioru „od 1 do n”, a „k” ilością elementów w generowanych kombinacjach, tworzy kombinacje, gdzie każda różni się od poprzedniej jedną cyfrą.

// Zapis algorytmu w pseudokodzie

1. Dla $i = 1$ do k wykonaj
   1.1. $K[i] = i; L[i] = n - k + i;$
2. Powtarzaj
   2.1. $i = k;$
   2.2. Jeżeli $K[i] < L[i]$ to
      2.2.1. $K[i] = K[i] + 1;$
      inaczej
      2.2.2. Powtarzaj $i = i - 1$ dopóki $K[i] < L[i];$
      2.2.3. $K[i] = K[i] + 1;$
      2.2.4. Dla $j = i + 1$ do k wykonaj $K[j] = K[j - 1] + 1;$
   2.3. Wyprowadź $K[1] \ldots K[k];$
   dopóki $K[1] = L[1];$

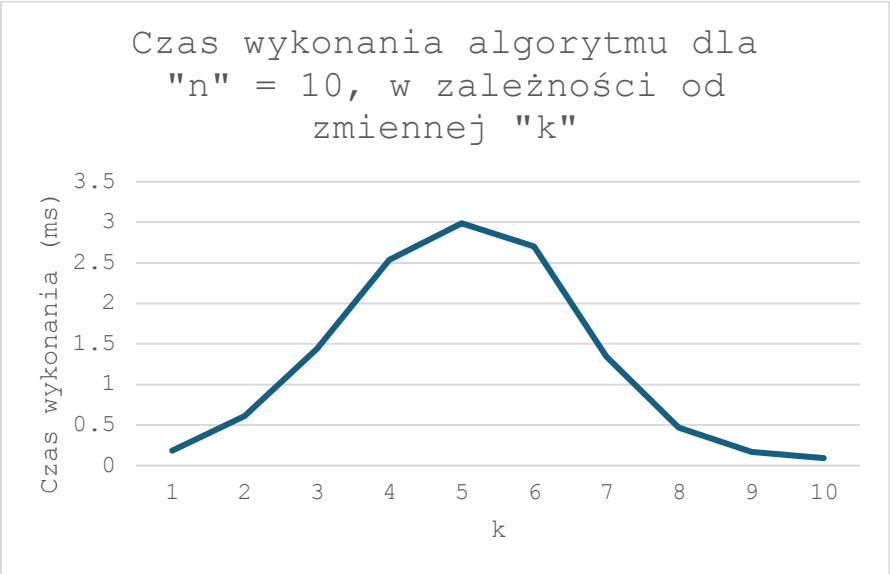
Prezentacja działania mojej implementacji w C++





Szybkość wykonania algorytmu zależy od wielkości zbioru oraz kombinacji, dla danej wielkości „n” czas wykonania jest największy dla „k” najbliższego połowie „n”, i najmniejszy dla „k” = „n”

n	k	Czas pracy algorytmu
5	3	0.19111 ms
7	5	0.202456 ms
10	8	0.476089 ms
13	3	3 ms
20	10	7482 ms



2. Algorytm Semby

Służy on do generowania wszystkich możliwych ciągów liczb rosnących z liczb zawartych w zbiorze od 1 do „n”.

W wyniku jego działania powstaje $2^n - 1$ ciągów, przez co już dla stosunkowo niewielkich zbiorów, zawierających np. 17 elementów, czas jego wykonania jest bardzo duży. Algorytm posiada złożoność czasową wykładniczą.

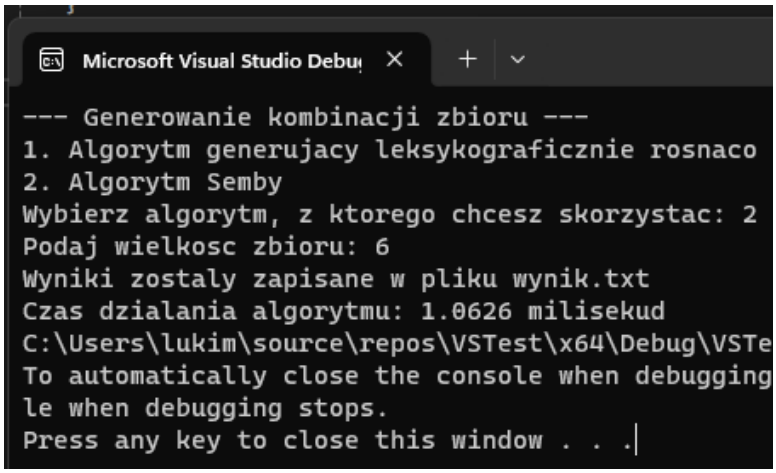
1. $k = 0;\ K[k] = 0;$
2. Powtarzaj
   2.1. Jeżeli $K[k] < n$, to *extend* inaczej *reduce*;
   2.2. Wyprowadź $K[1] \ldots K[k];$
   dopóki $K[1] = n;$

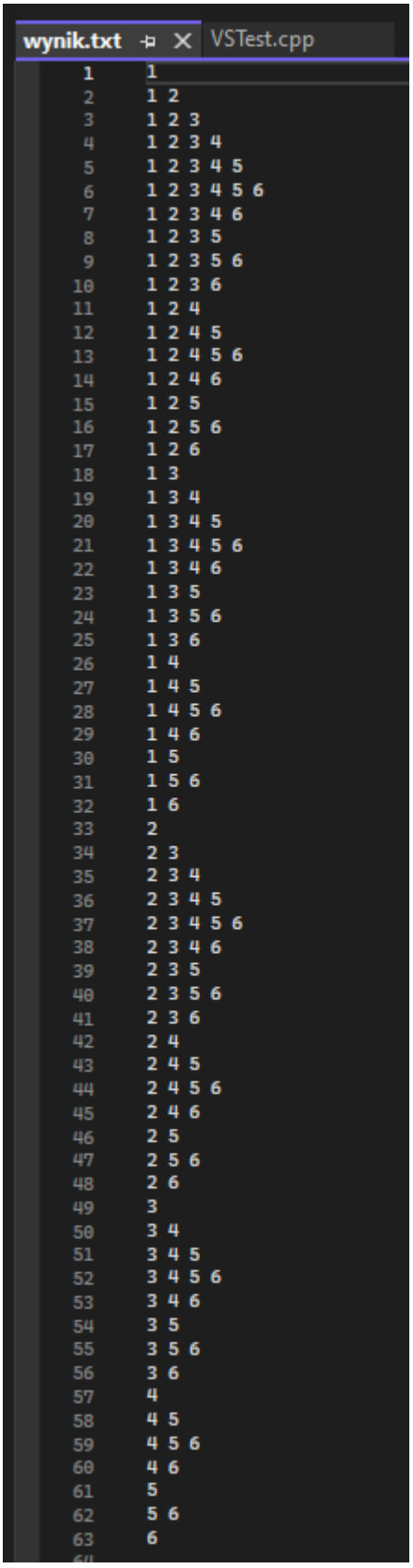
extend $K[k+1] = K[k] + 1; k = k + 1;$

reduce $k = k - 1;\ K[k] = K[k] + 1;$

//Zapis algorytmu w pseudokodzie

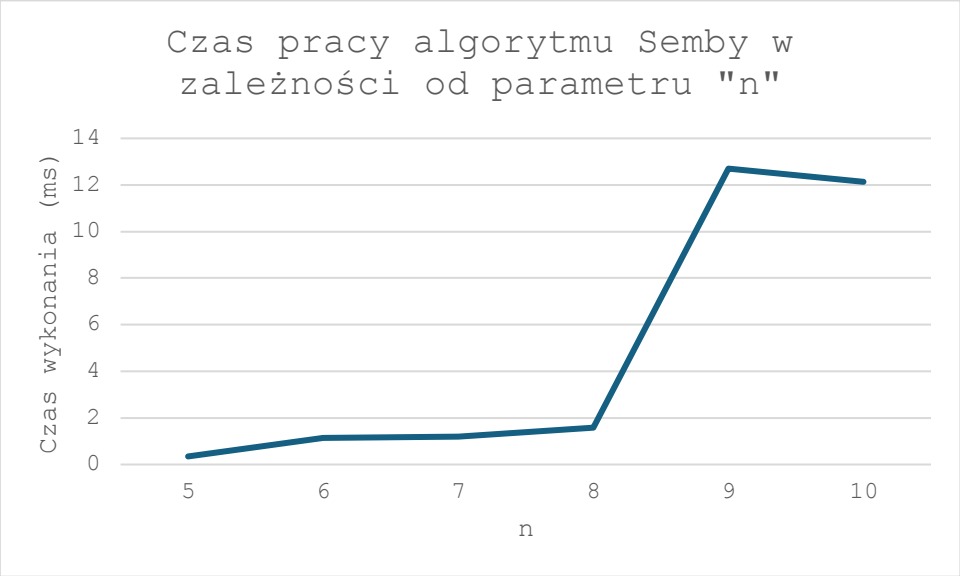
Prezentacja działania mojej implementacji w C++





n	Czas pracy algorytmu
5	0.3467 ms
6	1.1404 ms

7	1.1888 ms
8	1.5879 ms
9	12.702 ms
10	12.124 ms



Wnioski

Przedstawione nam algorytmy pozwalają nam na wygenerowanie kombinacji zbioru od 1 do zadanego „n”, jednak mają zupełnie różne zastosowania.

Pierwszy algorytm pozwoli nam utworzyć wszystkie możliwe „k” elementowe kombinacje zadanego zbioru, natomiast drugi służy do wyznaczenia wszystkich możliwych do utworzenia na bazie zbioru ciągów liczb rosnących.

Oprócz zastosowania, różni je także złożoność obliczeniowa. Szybkość wykonania się algorytmu Semby podyktowana jest wielkością zbioru i posiada on wykładniczą złożoność czasową, natomiast algorytmu pierwszego także wielkością pojedynczej kombinacji.