

# Laboratorium Matematyka Dyskretna

## Algorytmy kombinatoryczne

### Założenia

Zakładamy, że obiekty w zbiorze ponumerowane (zakodowane) są liczbami całkowitymi. Problem kodowania jednak pomijamy gdyż nie jest on istotny z punktu widzenia celów laboratorium.

Dostępny zbiór (lub jeśli wymaga tego algorytm zbiory) prezentowane są zatem w następującej postaci

$nazwa\_zbioru = \{1, 2, \dots, n\}$  - zbiór nazwa\_zbioru składa się z  $n$  elementów

### 1. Algorytmy

Algorytmy kombinatoryczne zaprezentowane w niniejszej instrukcji służą generowaniu obiektów kombinatorycznych. Zaprezentowane algorytmy po części działają ogólnie według następującego schematu

- Wypisanie pierwszego obiektu
- Aż do napotkania ostatniego obiektu, powtarzamy
  - Przetwarzany jest bieżący obiekt w celu otrzymania obiektu następnego

Takie podejście do generowania obiektów kombinatorycznych zapewnia oszczędność pamięci, gdyż wystarczy że pamiętamy jeden, bieżąco przetwarzany obiekt.

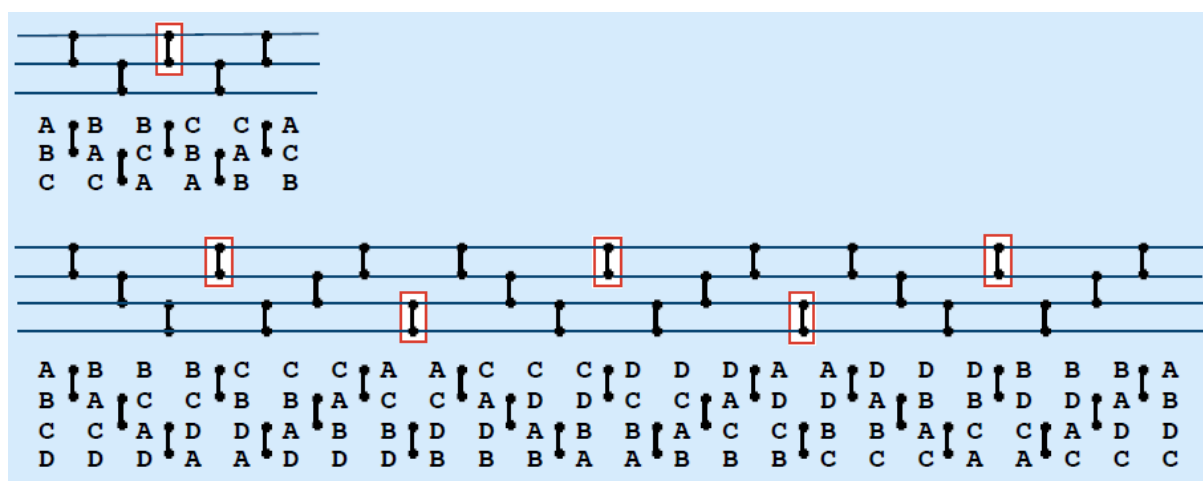
### 2. Generowanie permutacji elementów zbioru

Permutacja określa ustawienie elementów zbioru w określonej kolejności. Dla zbioru  $n$ -elementowego istnieje  $n!$  permutacji.

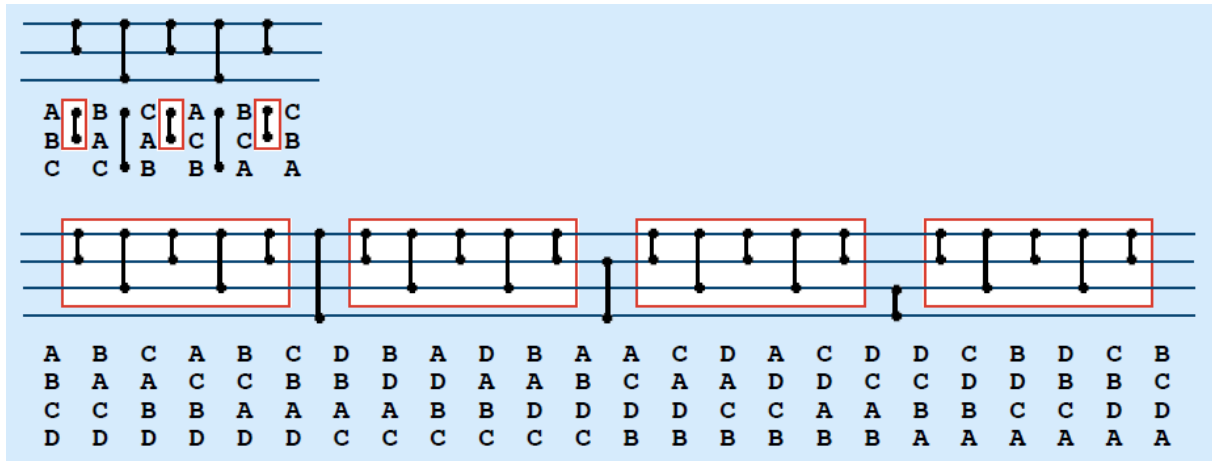
#### Algorytm 1

Algorytm może zostać przedstawiony jako przenoszenie pierwszego elementu na każdą pozycję (tam i z powrotem) każdej permutacji pozostałych elementów.

Algorytm można zilustrować w następujący sposób (Robert Sedgewick, Princeton University, <http://www.cs.princeton.edu/~rs/talks/perms.pdf>):



Algorytm może zostać opisany jako n kolejnych zamian pierwszego elementu poprzedniej permutacji z jej i-tym elementem ( $i = 2, \dots, n-1$ ), następnie następuje zamiana ostatniego elementu z k-tym, gdzie  $k=1$  jeżeli n jest nieparzyste lub  $k = 1, \dots, n-1$  (zmienia się iteracyjnie) jeżeli n jest parzyste. Algorytm można przedstawić na przykładzie (Robert Sedgewick, Princeton University, <http://www.cs.princeton.edu/~rs/talks/perms.pdf>):



### 3. Generowanie wszystkich podzbiorów zbioru n-elementowego $U=\{1,2,..,n\}$

### Algorytm 1

- Pierwszy podzbiór  $A$  jest zbiorem pustym  $A = \emptyset$
- Aby uzyskać następny podzbiór  $A$ 
  - Znajdź największy element z  $U$  nie należący do  $A$  (ozn.  $max$ )
  - Jeśli nie ma takiego elementu to koniec algorytmu ( $U$  jest zbiorem pustym)
  - W przeciwnym przypadku dodaj  $max$  do  $A$  i usuń z  $A$  wszystkie elementy większe od  $max$

Dla zbioru  $U=\{1,2,3\}$  kolejne podzbiory  $A$  będą miały następującą postać:  $\emptyset, \{3\}, \{2\}, \{2,3\}, \{1\}, \{1,3\}, \{1,2\}, \{1,2,3\}$ .

Określmy trójelementowy wektor binarny, który kodować będzie występowanie (lub nie występowanie) w podzbiorze  $A$  konkretnego elementu ze zbioru  $U$  (założmy, że elementy te są uporządkowane malejąco). Przykładowo z podzbiorem  $\{3,2\}$  związany będzie wektor 110, z podzbiorem  $\{1\}$ , wektor 001. Jeśli kody te potraktować jako liczby w zapisie dwójkowym to przedstawiony algorytm generuje liczby od 0 do 7 (ogólnie od 0 do  $2^{|U|}-1$ ).

Czasami ze względu na dziedzinę zastosowań zależy nam na tym aby nowo wygenerowany zbiór różnił się w jak najmniejszym stopniu od poprzedniego (np. wtedy gdy każdorazowo sprawdzamy czy elementy spełniają pewne warunki), zatem dobrze by było wykorzystać do obliczeń częściowe wyniki uzyskane w poprzednim kroku. Wtedy opisany wcześniej algorytm nie jest dobrym rozwiązaniem, gdyż kolejno generowane zbiory mogą „leżeć daleko od siebie”. W takiej sytuacji można wykorzystać następujące podejście:

### Algorytm 2

Metoda opiera się na założeniu że każdy następny podzbiór powstaje z poprzedniego przez dodanie lub odjęcie jednego elementu. Metoda opiera się na założeniu, że jeśli ciąg  $C_1, C_2, \dots$

$C_m$  zawiera wszystkie  $m=2^k$  ciągów binarnych długości  $k$ , przy czym  $C_i$  różni się od  $C_{i+1}$  na dokładnie jednej współrzędnej to ciąg:  $C_10, C_20, \dots, C_m0, C_m1, C_{m-1}1, \dots, C_11$   
 Zawiera wszystkie ciągi binarne długości  $k+1$ , przy czym każde dwa sąsiednie ciągi różnią się na dokładnie jednej współrzędnej. Otrzymany ciąg  $C_1, \dots, C_{2n}$  nazywa się kodem Gray'a rzędu  $n$ .

Działanie algorytmu można zilustrować w sposób następujący:

Wejście zbiór  $U=\{4,3,2,1\}$

Wyjście podzbiory:

Krok 1:	0000	k=0	
Krok 2:	0000 1000	k=1	$C_1=0, C_2=1$
Krok 3:	0000 1000 1100 0100	k=2	$C_1=00 C_2=10 C_3=11 C_4=01$
Krok 4:	0000 1000 0100 1100 1110 0110 1010 0010	k=3	$C_1=000 C_2=100 C_3=110 C_4=010 C_5=011$ $C_6=111 C_7=101 C_8=001$

Krok 5: daje wygenerowane wszystkie podzbiory

#### 4. Generowanie podzbiorów k-elementowych zbioru n-elementowego $U=\{1,2,\dots,n\}$

Algorytm generujący podzbiory k-elementowe zbioru  $U=\{1,2,\dots,n\}$  bazuje na następujących spostrzeżeniach:

- Zakładamy, że elementy w zbiorze są ponumerowane  $U=\{1,2,\dots,n\}$ , każdemu k-elementowemu podzbiorowi odpowiada wtedy wzajemnie jednoznacznie ciąg rosnący długości  $k$  o elementach z  $X$  (np. podzbiorowi  $\{3, 5, 1\}$  odpowiada ciąg  $\langle 1,3,5 \rangle$ )
- Będziemy generować takie zbiory (te k-elementowe zbiory – utożsamione z ciągami) w porządku leksykograficznym
- W tym celu można zauważyć, że w porządku tym ciągiem bezpośrednio następującym po  $(a_1, \dots, a_k)$  jest ciąg  $(b_1, \dots, b_k) = (a_1, \dots, a_{p-1}, a_p+1, a_p+2, \dots, a_p+k-p+1)$  gdzie  $p = \max\{i: a_i < n-k+1\}$
- i dalej ciągiem bezpośrednio następującym po  $(b_1, \dots, b_k)$  jest ciąg

$$(c_1, \dots, c_k) = (b_1, \dots, b_{p'-1}, b_{p'}+1, b_{p'}+2, \dots, b_{p'}+k-p'+1) \text{ gdzie } p' = \begin{cases} p-1 & \text{if } b_k = n \\ k & \text{if } b_k < n \end{cases}$$

UWAGA: Zakładamy, że ciągi  $(a_i)$  i  $(b_i)$  są różne od ostatniego ciągu w naszym porządku)

Rozumowanie powyższe prowadzi do podania następującego algorytmu

### **Algorytm - Generowanie wszystkich k-elementowych podzbiorów zbioru U w porządku leksykograficznym**

Begin

For  $i:=1$  to  $k$  do  $A[i]=i$  // pierwszy podzbiór

$p=k$

while  $p \geq 1$  do

begin

write( $A[1], \dots, A[k]$ ) // wypisz zbiór

if  $A[k]=n$  then  $p:=p-1$  else  $p:=k$ ;

if  $p \geq 1$  then for  $i:=k$  downto  $p$  do  $A[i]:=A[p]+i-p+1$

end

end;

Przykład działania – podzbiory 4-elementowe zbioru  $\{1,2,\dots,6\}$

(UWAGA: czytaj wierszami)

```
1234 1235 1236 1245 1246 1256 1345 1346 1356 1456 2345
2346 2356 2456 3456
```

### **Zadania do wykonania**

W czasie realizacji zadań mogą Państwo wykorzystać dowolną strukturę danych do przechowywania wygenerowanych obiektów. Na sprawozdaniu należy umieścić skomentowany kod algorytmów oraz czas działania programu.

Programy muszą posiadać możliwość zrzucania wyników to pliku tekstowego. Jeśli wszystkie algorytmy realizuje jeden program to instrukcję obsługi (prostą, program nie powinien mieć GUI, ma być uruchamiany z parametrami). Wszystkie zaimplementowane algorytmy powinny posiadać możliwość mierzenia czasu swojego działania.

1. Zaimplementować algorytm generowania wszystkich permutacji elementów danego zbioru.
2. Zaimplementować algorytm generowania wszystkich podzbiorów danego zbioru.
3. Zaimplementować algorytm generowania zbiorów k-elementowych
4. Przeprowadzić następujące eksperymenty na przykładowych zbiorach danych:
  - a. Wygenerować wszystkie permutacje dla  $n=4$ ,  $n=7$ ,  $n=10$
  - b. Wygenerować wszystkie podzbiory zbiorów  $\{1,2,3,4\}$   $\{1,2,3,4,\dots,20\}$   $\{1,2,\dots,30\}$
  - c. Wygenerować wszystkie podzbiory 5, 6, 7 elementowe zbioru  $\{1,2,3,4,5,6,7,8,\dots,15\}$
5. Porównać czas działania zaimplementowanych algorytmów, na podstawie tego czasu oszacuj złożoność obliczeniową algorytmów (wykres). Jeśli pomiar czasu nie będzie wykazywał różnic proszę proporcjonalnie zwiększać zbiory danych.
6. Podać przykładowe zastosowania zaimplementowanych algorytmów.