- ## Opis problemu:

Kombinacją k-elementową bez powtórzeń zbioru A składającego się z *n* różnych elementów nazywamy każdy *k*-elementowy podzbiór zbioru A, przy czym 0≤*k*≤*n*. Liczba kombinacji *k*-elementowych bez powtórzeń w zbiorze *n*-elementowym *Ckn* wyraża się następującym wzorem:

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

Kombinacja *k*-elementowa z powtórzeniami w *n*-elementowym zbiorze *A* to każdy *k*-elementowy multizbiór składający się z elementów zbioru *A*, przy czym w odróżnieniu od kombinacji bez powtórzeń elementy mogą się powtarzać. Multizbiór, w którym elementy mogą występować wiele razy, jest rozszerzeniem pojęcia zbioru.

Liczba k-elementowych kombinacji z powtórzeniami w zbiorze *n*-elementowym $\bar{C}_{kn}$ wyraża się następującym wzorem:

$$\bar{C}_n^k = \binom{k+n-1}{k} = \frac{(k+n-1)!}{k!(n-1)!}$$

- ## Opis algorytmu(1):

Generowanie kolejnych **kombinacji bez powtórzeń** :

- Wyszukuję pierwszą (licząc od prawej) pozycję na której mogę zwiększyć wartość, porównując ją do wartości max jaka może wystąpić na danej pozycji,
- zwiększam wartość na znalezionej pozycji o jeden,
- wartości pozycji na prawo zwiększam o jeden w stosunku do sąsiada (index2 = index1 + 1 itd.),
- otrzymuję jedną z kombinacji,
- algorytm wywołuję się rekurencyjnie aż do mementu gdy znajdzie wszystkie możliwe kombinacje.

Kombinacje prezentowane są w porządku leksykograficznym (uporządkowanie słownikowe) – jeżeli przyjęli byśmy, że liczby wchodzące w skład danej kombinacji są kodami znaków, to utworzone w ten sposób słowa uporządkowane byłyby alfabetycznie. Czyli pierwsza kombinacja (dla n = 4 i k = 3) ma postać: 1, 2, 3.

Generowanie **kombinacji z powtórzeniami** odbywa się analogicznie z tą różnicą, że nie ma ograniczenia co do wartości maksymalnej na danej pozycji oraz po zwiększeniu wartości, algorytm zmienia wartość wszystkich wartości na prawo na tą samą wartość. (index1 = index, index2 = index1 itd.). Warto również zauważyć, że w przypadku kombinacji z powtórzeniami k może być większe niż n, co zostało poprawnie zaimplementowane w programie.

- **Fragmenty kodu:**

Kombinacją bez powtórzeń:

```
int comb(int a_pos){
    if(a_pos != -1){
        int max_pos = n-(k-a_pos);
        if(tab[a_pos]<max_pos){
            tab[a_pos]++;

            for(int i=a_pos+1;i<=k; i++)
            {
                tab[i] = tab[i-1]+1;
            }

            a_pos=k;
            show();
            return comb(a_pos);
        }else return comb(a_pos-1);
    }
}
```

Kombinacją z powtórzeniami:

```
int comb(int a_pos){
    if(a_pos != -1){
        int max_pos = n;
        if(tab[a_pos]<max_pos){
            tab[a_pos]++;

            for(int i=a_pos+1;i<=k; i++)
            {
                tab[i] = tab[a_pos];
            }

            a_pos=k;
            show();
            return comb(a_pos);
        }else return comb(a_pos-1);
    }
}
```

(Jak łatwo zauważyć oba algorytmy są bliźniaczo podobne, różnice zostały zaznaczone pokreśleniem)

Przykładowe wywołanie programów:

```
Kombinacje z powtorzeniami
podaj dlogosc ciagu: 3
podaj dlogosc kombinacii: 3
1 1 1
1 1 2
1 1 3
1 2 2
1 2 3
1 3 3
2 2 2
2 2 3
2 3 3
3 3 3
```

```
Kombinacje bez powtorzen
podaj dlogosc ciagu: 5
podaj dlogosc kombinacii: 3
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

- ## Opis algorytmu(2):

  Generowanie wszystkich **kombinacji bez powtórzeń** danego zbioru n-elementowego według algorytmu Semby :
  - Program generuje tablicę n+1 elementową wypełnioną zerami,
  - Funkcja up() sprawdza czy można zwiększyć wartość na kolejnej pozycji o 1 w stosunku do wcześniejszej,
  - W przypadku gdy jest to niemożliwe wywoływana jest funkcja down() która przy spełnionych warunkach dodaje do wartości na danej pozycji 1 oraz zeruje tablicę na prawo od zwiększonego elementu,
  - Przy każdej zmianie wartości, tablica wyświetlana jest przez funkcję show() która listuje ją od elementu 1 do elementu na którym aktualnie pracuje algorytm,
  - Algorytm wywołuje się rekurencyjnie i kończy swoje działanie gdy przejdzie do pozycji 0. Oznacza to, że wszystkie możliwe kombinacje zostały już wyczerpane.
- ## Fragmenty kodu:

```
1  int up(int n_pos){
2      if((tab[n_pos-1]+1 <= n)&&(n_pos <= n)){
3          tab[n_pos] = tab[n_pos-1]+1;
4          show(n_pos);
5          up(n_pos+1);
6      }else down(n_pos-1);
7  }
8
9  int down(int n_pos){
10     if((tab[n_pos] < n)&&(n_pos <= n)){
11         tab[n_pos] = tab[n_pos]+1;
12         tab[n_pos+1] = 0;
13         show(n_pos);
14         up(n_pos+1);
15     }else if(n_pos-1!=0) down(n_pos-1);
16 }
```

Przykładowe wywołanie programów:

```
Kombinacje bez powtorzen (alg. Semby)
podaj dlogosc ciagu: 5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 5
1 2 4
1 2 4 5
1 2 5
1 3
1 3 4
1 3 4 5
1 3 5
1 4
1 4 5
1 5
2
2 3
2 3 4
2 3 4 5
2 3 5
2 4
2 4 5
2 5
3
3 4
3 4 5
3 5
4
4 5
5
```

```
Kombinacje bez powtorzen (alg. Semby)
podaj dlogosc ciagu: 4
1
1 2
1 2 3
1 2 3 4
1 2 4
1 3
1 3 4
1 4
2
2 3
2 3 4
2 4
3
3 4
4
```

- ## Wnioski:

  Programy prawidłowo generują kombinacje z powtórzeniami oraz kombinacje bez powtórzeń. Poprawnie walidują dane wprowadzane przez użytkownika na wejściu. Otrzymane wyniki wyświetla w uporządkowanej formie w konsoli Windows. Spełnia on wszystkie wstępne założenia.

Źródła:

[1]. Wybrane algorytmy tablicowe www.ae.krakow.pl/~lulap/AiSD_2009_02.pdf

[2]. Algorithms: International Symposium SIGAL '90, Tokyo, Japan, August 16-18, 1990.