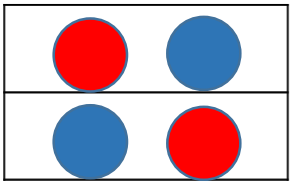
# Permutacije

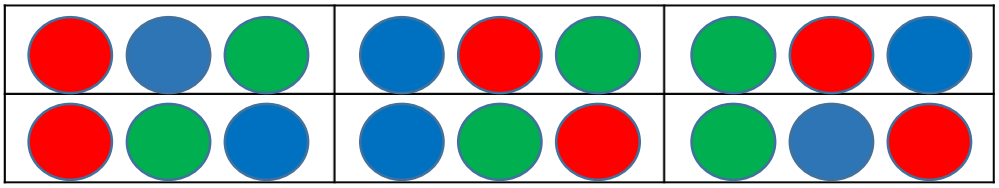
**Zadatak.** U vreći se nalazi $n$ loptica različitih boja. Iz vreće izvlačimo redom jednu po jednu lopticu i stavljamo jednu pored druge. Koliko različitih redosleda boja možemo da dobijemo?

**Primer 1.** Neka je $n = 2$ i neka su loptice plave i crvene boje. Imamo 2 moguća redosleda prikaza na slici 1.



*Slika 1*

**Primer 2.** $n = 3$ i loptice su plave, crvene i zelene boje. Imamo 6 mogućih redosleda prikazih na slici 2.



*Slika 2*

Probaćemo da izračunamo koliko ima različitih redosleda u slučaju proizvoljnog broja $n$. Na prvom mestu u redosledu može da stoji bilo koja od $n$ loptice. Na drugom mestu u redosledu mogu da stoje sve loptice, osim loptice koja stoji na prvom mestu. Možemo zaključiti da postoji ukupno $n \cdot (n-1)$ različitih redosleda loptica na prve 2 pozicije. Koristeći slično razmatranje, na $i$-tom mestu moguće je postaviti $n - i + 1$ lopticu, tj. na mestu $i$ ne mogu da se nađu loptice koje već stoje na nekom od prvih $i - 1$ mesta. Dobijamo da za prvih $i$ mesta postoji $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-i+1)$ različitih redosleda. Kada u formulu ubacimo da je $i = n$ dobijamo da postoji ukupno $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1 = n!$ različitih redosleda.

Sada ćemo dati opis algoritma koji ispisuje sve permutacije. Nama je potrebno da na prvu poziciju u redosledu postavimo neki od $n$ elemenata, te onda da na preostalih $n - 1$ pozicija rasporedimo preostalih $n - 1$ loptica. Ovaj problem možemo rešiti rekurzivnim putem. Na početku napravimo niz $skup$ u kome će se nalaziti svih $n$ elemenata. U prvom pozivu rekurzivne funkcije postavimo jedan od elemenata iz niza $skup$ na prvo mesto, izbacimo taj element iz niza i pozovemo rekurzivno funkciju koja će rasporediti preostalih $n - 1$ elemenata.

Da ne bismo stalno izbacivali elemente iz skupa, mi ćemo napraviti još jedan niz $markirano$, gde će $markirano[i] = true$ označavati da je $i$-ti element u nizu $skup$ izbačen, te ga ne možemo više iskoristiti u daljim rekurzivnim pozivima. Kada se vratimo iz rekurzivnih poziva u funkciju u kojoj smo postavili element $i$, potrebno je postaviti $markirano[i] = false$, kako bismo mogli nadalje da koristimo $i$-ti element.

Sledi primer funkcije koja ispisuje sve permutacije.

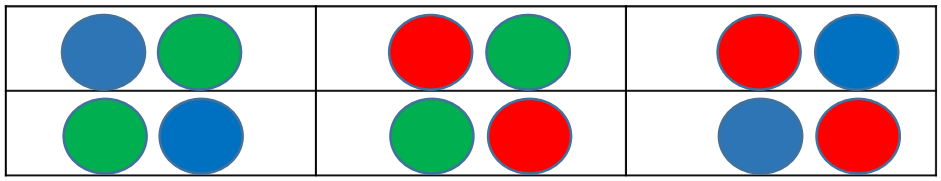
```
============================================================
01     function Permutations( position, ref permutation, ref mark, a, n )
02            if position = n+1 then
03                   print array permutation
04            end if
05
06            for i = 1 to n do
07                   if not mark[i] then
08                          permutation[ position ] = a[i]
09                          mark[i] = true
10                          Permutations( position+1, permutation, mark, a, n )
11                          mark[i] = false
12                   end if
13            end for
14     end function
============================================================
```

## Varijacije

Ponovo ćemo krenuti od zadatka, da bi se dobio osećaj šta se traži.

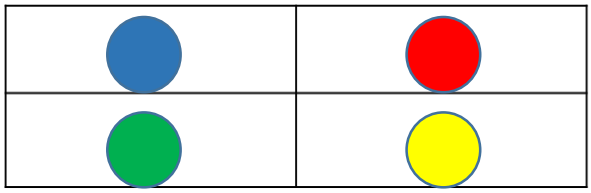
**Zadatak.** U vreći se nalazi $n$ loptica različitih boja. Iz vreće izvlačimo tačno $k$ loptica i stavljamo jednu pored druge. Koliko različitih redosleda boja možemo da dobijemo?

**Primer 1.** $n = 3$ (zelena, plava i crvena) i $k = 2$, na slici 3 možemo videti 6 mogućih rasporeda



*Slika 3*

**Primer 2.** $n = 4$ (zelena, plava, crvena, žuta) i $k = 1$, na slici 4 su prikazana sva 4 moguća rasporeda



*Slika 4*

Ponovo se postavlja pitanje koliko ima mogućih rasporeda za proizvoljne brojeve $n$ i $k$. Kod permutacija smo izveli formulu za broj različitih rasporeda na prvih $i$ pozicija. Ovde je dovoljno da umesto $i$ u formuli ubacimo $k$ i dobićemo broj različitih načina, tj. $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1) = \frac{n!}{(n-k)!}$.

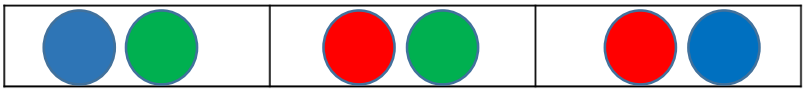
Algoritam za ispisivanje svih varijacija je veoma sličan algoritmu za ispisivanje svih permutacije, sa razlikom da smo kod permutacija išli do dubine $n$ u rekurziji, dok ćemo ovde prekidati rekurziju posle dubine $k$. Pseudo kod je dat u nastavku.

```
============================================================
01     function Variations( position, ref variation, ref mark, a, n, k )
02            if position = k+1 then
03                   print array variation
04            end if
05
06            for i = 1 to n do
07                   if not mark[i] then
08                          variation[ position ] = a[i]
09                          mark[i] = true
10                          Variations( position+1, variation, mark, a, n, k )
11                          mark[i] = false
12                   end if
13            end for
14     end function
============================================================
```

## Kombinacije

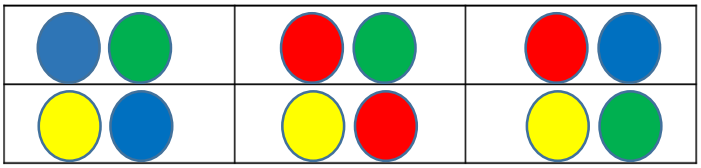
**Zadatak.** U vreći se nalazi $n$ loptica različitih boja. Iz vreće izvlačimo tačno $k$ loptica i stavljamo na gomilu u kojoj redosled loptica nije bitan. Koliko različitih gomila možemo da dobijemo? Dve gomile su različite ukoliko ne sadrže iste loptice.

**Primer 1.** $n = 3$ (crvena, plava, zelena) i $k = 2$ postoji 3 moguće gomile prikazane na slici 5



*Slika 5*

**Primer 2.** $n = 4$ (crvena, plava, zelena, žuta) i $k = 2$ postoji 6 gomila koje su prikazane na slici 6



*Slika 6*

Primetimo da je redosled loptica nebitan kod kombinacija, te je gomila (plava, crvena) ista kao i gomila (crvena, plava), te možemo zaključiti da za jednu kombinaciju od $k$ loptica, postoji $k!$ varijacija. Iz pomenutog razmatranja možemo zaključiti da je broj kombinacija za proizvoljno $n$ i $k$ jednak $\frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-k+1)}{k\cdot(k-1)\cdot(k-2)\cdot\ldots\cdot2\cdot1} = \frac{n!}{(n-k)!\cdot k!} = \binom{n}{k}$.

Kako redosled elemenata u kombinaciji nije bitan, pri ispisu je dovoljno voditi računa da element sa pozicije $i, i < j$ nikad ne stavimo posle elementa na poziciji $j$. Algoritam za ispisivanje svih kombinacija je veoma sličan prethodno opisanim algoritmima.

```
==========================================================
01     function Combinations( position, start, ref combination, a, n, k )
02            if position = k then
03                   print array combination
04            end if
05
06            if start = n+1 then
07                   return // potrošili smo sve elemente
08            end if
09
10            for i = begin to n do // begin je jedan element odmah posle
prethodno stavljenog
11                   combination[ position ] = a[i]
12                   Combinations( positions+1, i+1, combination, a, n, k )
13            end for
14     end function
==========================================================
```

## Partitivni skup

Neka nam je dat skup $S$. Skup svih podskupova skupa $S$ se zove partitivni skup skupa $S$ i označavaćemo ga sa $P(A)$.

**Primer.** $S = \{1,2,3\}, P(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

Svaki podskup možemo jednoznačno da predstavimo uz pomoć binarnog broja sa $n$ cifara, gde je $n$ broj elemenata skupa. Cifra $i$ u broju je jednaka $1$ ukoliko podskup sadrži $i$-ti element iz skupa, inače je $0$.

**Primer.** $A = \{1,3\} \Leftrightarrow 101, B = \{2\} \Leftrightarrow 010$

Bitwise AND operator je operacija nad domenom $\{0,1\}$ data u tabeli 1.

| $AND$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

*Tabela 1*

Za nenegativne cele brojeve $A$ i $B$, $A\ AND\ B$ se računa na sledeći način: neka su $A = (a_n a_{n-1} \dots a_1)_2$ i $B = (b_n b_{n-1} \dots b_1)_2$ binarni zapisi brojeva $A$ i $B$ (ukoliko se brojevi cifara u binarnom zapisu razlikuju, onda se dodaju vodeće nule onom binarnom zapisu koji sadrži više cifara). Tada je $A\ AND\ B = C = (c_n c_{n-1} \dots c_1)$ gde je $c_i = a_i\ AND\ b_i$.

**Primer.** $20\ AND\ 5 = 4$

$$10100_2 = 20_{10}$$
$$\text{AND } 00101_2 = 5_{10}$$
$$\text{----------}$$
$$00100_2 = 4_{10}$$

Ukoliko imamo skup sa $n$ elemenata, svaki podskup je jedinstveno određen sa binarnim brojem od $n$ cifara, a znamo da takvih brojeva ima $2^n - 1$. Neka nam je dat broj $a$ koji je manji od $2^n$, postavlja se pitanje kako da znamo koji podskup je predstavljen brojem $a$. Element $i$ skupa se nalazi u podskupu određenim brojem $a$ ukoliko važi $a\ AND\ 2^i = 2^i$, ovde proveravamo da li je $i$-ta cifra u binarnom zapisu broja $a$ jednaka $1$.

U nastavku možete videti primer ispisane funkcije koja prolazi kroz sve podskupove skupa od $n$ elemenata.

```
============================================================
01     function PartitivanSkup( a, n )
02            for bitmask = 1 to 2^n-1 do
03                   podskup = empty
04                   for i = 1 to n do
05                          if ( (bitmask AND 2^j) = 2^j ) then
06                                 podskup.add( a[i] )
07                          end if
08                   end for
09
10                   print podskup
11            end for
12     end function
============================================================
```