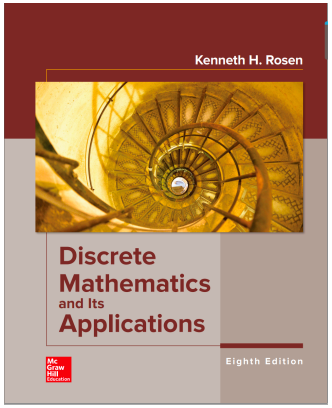
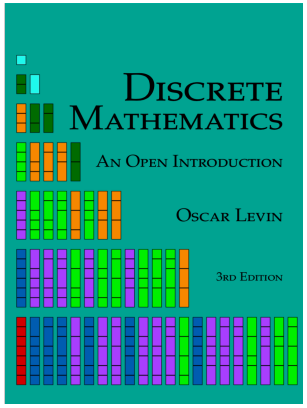
Vietnam National University of HCMC
International University
School of Computer Science and Engineering




# Session 9

## Counting (part 3), October, 2022

**Assoc. Prof. Dr. Nguyen Van Sinh**

**nvsinh@hcmiu.edu.vn**

# Introduction

- Part 1: The basic rules of counting
  - Product rules
  - Sum rule
  - The Inclusion-Exclusion rule
  - The Pigeonhole Principle
- Part 2: Permutations and Combinations
  - Permutation
  - Combination
  - Formulas of combinations
  - The extended combination principles
- Part 3: Enumeration

# Part 3: Enumeration

A. **Recursive method**

B. **Loop method**

# Recursive method

- General rules
- Problem: eight queens
- Enumerate the simple combination

# General rule

In order to enumerate all cases $S=s_1s_2 .. s_k$

The algorithm suppose having the sub-case $s_1s_2..s_{i-1}$. In the tried step of finding $s_i$, Try(i), we loop for all values of j that are voted for $s_i$ and perform the following steps:

# General rule

For each j (voted for $s_i$), execute four steps:

1) $s_i = j$;
2) <*change the status*>;
3) If (i == k) Print(S); else Try(i+1);
4) <*return the old state*>;

The main function call *Try(1)*

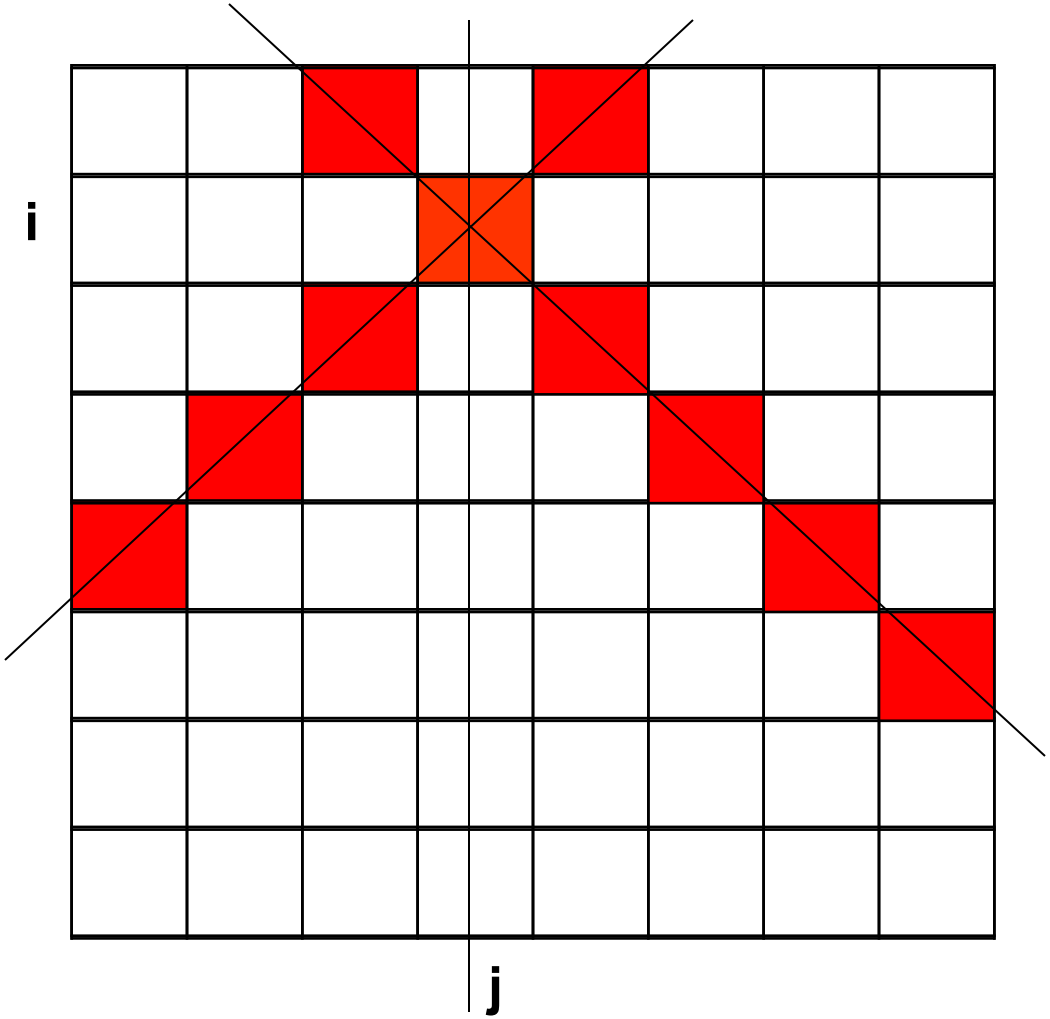
# General rule

*Note*: in some cases, there is no step *2)* and *4)*

The above enumerated algorithm, the case of simple combination are always assumed that X = {1..n}.

In the real case, take elements of X to index for array, and not use S[0]

# Problem: eight queens




There are no two queens that share the same row, column, or diagonal.

# Problem: eight queens

*Declare:*

- The chess table: int S[8];
- The arrays that need to note:

  - column: int a[8];

  - parallel to the main diagonal: int b[15];

  - parallel to the sub diagonal: int c[15];

*Before calling Try(0) need to full-fill a, b, c with value 1* (TRUE)

# Problem: eight queens

```
void Try(int i)
{
  int j;
  for (j=0; j<8; j++)
     if (a[j] && b[i-j+7] && c[i+j])
     {
            S[i]=j;
            a[j]=0; b[i-j+7]=0; c[i+j]=0;
            if (i==7)
                 print( );
            else
                 Try(i+1);
            a[j]=1; b[i-j+7]=1; c[i+j]=1;
     }
}
```

***See: 8queens.cpp***

# Enumerate arrangement with repetition

```
void Try(int i)
{
  int j;
  for (j=1; j<=n; j++){
    S[i]=j;
    if (i==k) print();
    else Try(i+1);
  }
}
```

***See: EnumArrRepeat.c***

# Enumerate arrangement without repetition

*We need an array a to note values j that have been used*

```
void Try(int i)
{  int j;
  for (j=1; j<=n; j++)
    if (a[j])
    {
       S[i]=j;    a[j]=0;
       if (i==k) print( ); else Try(i+1);
       a[j]=1;
    }
}
```

***See: EnumArrNonRepeat.cpp***

# Enumerate combination

*Enumerate the increasing orders. Value j (voted for $s_i$) is: $1+s_{i-1} \le j \le n - k + i$. Let Try(1): array S with S[0]=0.*

```
void Try(int i)
{
  int j;
  for (j=1+S[i-1]; j<=n-k+i; j++){
    S[i]=j;
    if (i==k) print( ); else Try(i+1);
  }
}
```

*See: EnumCom.c*

# Loop method

- General rule
- Enumeration of simple combinations

# General rule

Enumerate all $S=s_1s_2..s_k$ following the lexical ordering, including the following steps:

- Find S smallest; Print(S);
- Repeat until end of S and satisfying the condition:

    Find the next S*; Print(S);*

*or while S is not the largest*:

    Find the next *S; Print(S);*

# General rule

Next step: find the next S depending on the condition that making S change as follows:

- *Find S[i] on the right hand side that can be increased.*
- *Increase S[i] with the smallest increasing value.*
- *Decrease subset S[i+1..k] to smallest.*

*Or you can also find S[i] while as the same time of checking stop condition.*

# Enumerate the repetition arrangement

```
for (i=1; i<=k; i++)
S[i]=1; print(); c=1;
  while (c<n^k)
  {
    i=k; while (S[i]==n) i--;
    S[i]++;
    for (j=i+1; j<=k; j++) S[j]=1;
    print();
  }
```

***See: EnumErrNonRepeatER.cpp***

# Enumerate the combination

*Combination is a non-order set. Enumeration follows increasing order: $s_i<s_{i+1}$. The largest of $s_i$ is n-k+i.*

```
for (i=1; i<=k; i++) S[i]=i; print(); c=1;
  while (c<C(n,k))
  {
    i=k; while (S[i]==n-k+i) i--;
    S[i]++;
    for (j=i+1; j<=k; j++) S[j]=S[j-1]+1;
    print( );
  }
```

# Enumerate the permutation

*The smallest follows increasing order and the largest follows decreasing order:*

```
for (i=1; i<=n; i++) S[i]=i; print( ); c=1;
  while (c<n!)
  {
    i=n-1; while (S[i]>S[i+1]) i--;
    j=n; while (S[j]<S[i]) j--;
    temp=S[i]; S[i]=S[j]; S[j]= temp;
    j=i+1; k=n; while (j<k){
            temp =S[j]; S[j]=S[k]; S[k]= temp; j++; k--;}
    print( );
  }
```

# Enumerate the permutation

*Putting the algorithm of permutation-generation into each step of generation of combination in order to obtain the generation of non-repetition arrangement.*

# Homework (G3)

- Doing the following exercises:
  6, 14, 16, 20, 24, 28, 32, 34 (pages: 435, 436)
- Implement the following exercises using C/C++:
  8 Queens (the size is entered form user: slide 10)
- Enumeration (see algorithms in slides: 11-13)
  - Combination (repeat and not repeat)
  - Arrangement (repeat and not repeat)

**Submit on the BB before 10:00 PM, 26/10/2022**