



# Fuerza Bruta



**B.S. Rodolfo Mercado Gonzales**

# Espacio de búsqueda

- ❑ Está formado por todos los candidatos a ser solución de un problema.
- ❑ Por ejemplo, si queremos abrir una puerta, nuestro espacio de búsqueda estaría conformado por el manajo de llaves.

# Fuerza bruta / Búsqueda exhaustiva

- ❑ Enfoque para resolver un problema analizando todo el espacio de búsqueda hasta encontrar la solución requerida.
- ❑ Podemos realizar un “prune”, es decir omitir algunas partes del espacio de búsqueda, si es posible determinar que éstas no tendrán la solución requerida.

# Fuerza bruta

Generalmente se hace fuerza bruta cuando :

- ❑ El espacio de búsqueda es pequeño.
- ❑ No existe otro enfoque o algoritmo que aplique al problema.

# Fuerza bruta

En nuestro día a día lo aplicamos :

- Cuando no sabemos el password de una computadora.
- Cuando no sabemos cual es la llave de una puerta.
- Cuando llenamos un sudoku.
- Cuando armamos nuestro horario en la universidad.
- ...

# Problemas

Codeforces 122A - Lucky Division

Codeforces 479A - Expression

Codeforces 460B – Little Dima and Equation

Codechef – Chef and Numbers

# Permutación de un Arreglo

Una permutación de un arreglo es un cambio en el orden sus elementos.

$\{ 1, 2, 1 \}$

$\{ 1, 1, 2 \}$

$\{ 2, 1, 1 \}$

# Siguiente Permutación Lexicográfica

- ❑ Las permutaciones de un arreglo tienen la misma cantidad de elementos.
- ❑ Comparar lexicográficamente dos arreglos es similar a comparar dos cadenas (elementos por elemento)
- ❑ La siguiente permutación lexicográfica de una permutación  $P$  es la menor de todas las permutaciones  $P'$  tal que  $P' > P$ .



# Siguiente Permutación Lexicográfica

Permutaciones del arreglo {1, 2, 3} en orden lexicográfico.

{1, 2, 3}

{1, 3, 2}

{2, 1, 3}

{2, 3, 1}

{3, 1, 2}

{3, 2, 1}

# Next Permutation

- ❑ Función incluida en la STL.
- ❑ Devuelve verdad si existe una siguiente permutación lexicográfica del arreglo.
- ❑ Reordena el arreglo y lo transforma en la siguiente permutación lexicográfica.

# Next Permutation

De esta manera podemos obtener la siguiente permutación lexicográfica del arreglo

```
int A[] = { 5, 6, 7};  
bool existe = next_permutation( A, A+3);  
if( existe ){  
    for( int i=0; i<3; ++i ) cout<<" "<<A[ i ];  
}
```

# Generar todas las permutaciones

Entonces para generar todas las permutaciones, debemos partir de la menor lexicográfica y a partir de ahí ir generando la siguiente mientras exista.  $O(n * n!)$

```
int A[] = { 5, 6, 7};
sort( A, A + 3 );
do{
    for( int i=0; i<3; ++i )cout<<" "<<A[ i ];
    cout<<endl;
}while( next_permutation( A, A + 3 ) );
```

# Problemas

UVA 146 – ID Codes

Codeforces 431B – Shower Line

# Máscara de Bits

- ❑ Si tenemos un conjunto de  $n$  elementos, entonces tenemos  $2^n - 1$  subconjunto.
- ❑ Todos los subconjuntos pueden ser representados con los enteros en el rango  $[0, 2^n - 1]$ .

Número	Máscara	Subconjunto
0	000	{ }
1	001	{ 1 }
2	010	{ 3 }
3	011	{ 1, 3 }
4	100	{ 8 }
5	101	{ 1, 8 }
6	110	{ 3, 8 }
7	111	{ 1, 3, 8 }

# Generar todos los subconjuntos

Podemos generar todos los subconjuntos usando los operadores bitwise en  $O(2^n * n)$

```
int n = 3;
for( int mask=0 ; mask< (1<<n); mask++ ){
    cout<<mask<<" :";
    for( int i=0; i<n; ++i ){
        if( (mask>>i) & 1 == 1 ) cout<<" "<<i;
    }
    cout<<endl;
}
```

# Problemas

UVA 12406 – Help Dexter  
Codeforces 202A - LLPS



# Referencias

- ❑ Halim, Steven and Halim, Felix. Competitive Programming 3
- ❑ Topcoder, A Bit of Fun: Fun with Bits

¡ Good luck and have fun !