



Fuerza Bruta



Bach. Rodolfo Mercado Gonzales
Programación Competitiva UPC

Espacio de búsqueda

- ❑ Está formado por todos los candidatos a ser solución de un problema.
- ❑ Por ejemplo, si queremos abrir una puerta y no recordamos cuál es su llave, entonces nuestro espacio de búsqueda estaría formado por todo el manajo de llaves.



Fuerza Bruta

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

Fuerza Bruta

En nuestra día a día aplicamos frecuentemente este enfoque:

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

...



Fuerza Bruta

Generalmente se usa fuerza bruta cuando :

- El espacio de búsqueda no es muy grande.
- No existe otro enfoque o algoritmo que aplique al problema.



Ejercicios

- [Codeforces - Expression](#)
- [Codeforces - Beautiful Year](#)
- [Codechef - Chef and Numbers](#)



Desafíos

- [Codeforces – Distinct Digits](#)
- [Codeforces – System of Equation](#)
- [Codeforces - Lucky Division](#)
- [Codeforces - Little Elephant and Magic Square](#)
- [Codechef - Coronavirus Spread](#)
- [Codeforces - Little Dima and Equation](#)
- [SPOJ - Sum to Zero](#)
- [Codechef - Coronavirus Spread 2](#)

CHALLENGE ACCEPTED



Generación de subconjuntos



Subconjuntos

- ❑ Si tenemos un conjunto de n elementos, entonces tenemos 2^n subconjuntos.
- ❑ 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 }

subconjuntos de {1, 3, 8}

Generar todos los subconjuntos

Podemos generar todos los subconjuntos usando operadores bitwise.

```
void subsets(vector<int> &v) {  
    int n = sz(v);  
    for (int mask = 0; mask < (1 << n); ++mask) {  
        for (int i = 0; i < n; ++i) {  
            if((mask >> i) & 1) cout << " " << v[i];  
        }  
        cout << endl;  
    }  
}
```

*complejidad: $O(2^n * n)$*



Ejercicios

- [Codeforces 550B – Preparing Olympiad](#)



Desafíos

- [Codeforces - LLPS](#)
- [Codeforces - Petr and a Combination Lock](#)
- [UVA - Help Dexter](#)

CHALLENGE ACCEPTED



Generación de permutaciones



Permutaciones

- ❑ Una permutación de n elementos es algún reordenamiento que se puede hacer con ellos.
- ❑ Por ejemplo con los elementos $\{a, b, a\}$ podemos generar las siguientes permutaciones:

$\{a, b, a\}$ $\{a, a, b\}$ $\{b, a, a\}$

the word "MISSISSIPPI" was made only to teach permutations and combinations.



Siguiente Permutación Lexicográfica

- ❑ Comparar lexicográficamente dos permutaciones es similar a comparar dos cadenas (elemento por elemento)
- ❑ La siguiente permutación lexicográfica (SPL) de una permutación P es la menor de todas las permutaciones P' tal que $P' > P$.

- La SPL de $\{6, 3, 8\}$ es $\{6, 8, 3\}$.
- No existe SPL de $\{8, 6, 3\}$.



Next Permutation

- ❑ Función incluida en la STL.
- ❑ Devuelve verdadero si existe una SPL.
- ❑ Reordena los elementos y lo transforma en la SPL.
- ❑ Tiene complejidad lineal.

Next Permutation

De esta manera podemos obtener la siguiente permutación lexicográfica de n elementos.

```
void print_next_perm(vector<int> &v) {  
    if (next_permutation(all(v))) {  
        for (int i = 0; i < sz(v); ++i) {  
            cout << " " << v[i];  
        }  
    }  
    else {  
        cout << "There isn't next permutation";  
    }  
}
```

Generar todas las permutaciones

Permutaciones de los elementos $\{1, 2, 3\}$ en orden lexicográfico.

$\{1, 2, 3\}$

$\{1, 3, 2\}$

$\{2, 1, 3\}$

$\{2, 3, 1\}$

$\{3, 1, 2\}$

$\{3, 2, 1\}$

*el número de permutaciones
puede llegar a ser muy grande*



Generar todas las permutaciones

Para generar todas las permutaciones, debemos partir de la menor lexicográfica y de ahí ir generando las siguientes mientras existan.

```
void permutations(vector<int> &v) {  
    sort(all(v));  
    do {  
        for (int i = 0; i < sz(v); ++i) {  
            cout << " " << v[i];  
        }  
        cout << endl;  
    } while(next_permutation(all(v)));  
}
```

*complejidad: $O(n! * n)$*



Ejercicios

- [UVA 146 – ID Codes](#)



Desafíos

- [AtCoder – Average Length](#)
- [Codeforces - Shower Line](#)
- [UVA – Jollo \(ICPC 2010\)](#)

CHALLENGE ACCEPTED



Referencias

- ❑ Antti Laaksonen - Competitive Programmer's Handbook
- ❑ Steven Halim & Felix Halim - Competitive Programming 3



“Start by doing what is necessary, then what is possible, and suddenly you are doing the impossible.”

- Francis of Assisi