Fuerza Bruta

B.S. Rodolfo Mercado Gonzales
Universidad Nacional de Ingeniería - FIIS

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 cual es su llave, entonces nuestro espacio de búsqueda estaría formado por el manojo 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 éstas no tendrán la solución requerida.

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.

…

Fuerza bruta

Generalmente se hace fuerza bruta cuando :

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

Problemas

Codeforces 479A - Expression

Codeforces 122A - Lucky Division

Codechef – Chef and Numbers

Codeforces 460B – Little Dima and Equation

Máscara de Bits

- 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 los 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: $O(2^n * n)$

Problemas

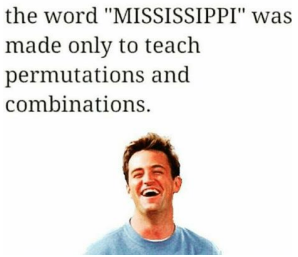
UVA 12406 – Help Dexter

Codeforces 202A - LLPS

Permutaciones

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

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



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 siguiente permutación lexicográfica.
- Reordena los elementos y lo transforma en la siguiente permutación lexicográfica.
- 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: $O(n! * n)$

Problemas

UVA 146 – ID Codes

Codeforces 431B – Shower Line

Referencias

- Steven Halim & Felix Halim - **Competitive Programming 3**
- HackerEarth – **Bit Manipulation**

¡ Good luck and have fun !