Fuerza bruta

¿Que es la fuerza bruta?

Fuerza bruta es una técnica que consiste en realizar una búsqueda completa del espacio del problema, ya sea para encontrar una solución óptima o simplemente una solución factible.

Búsqueda completa

Para realizar una búsqueda completa pueden existir distintos métodos, dentro de estos tipos, podemos destacar dos:

- Búsqueda sobre las permutaciones.
- Búsqueda sobre los subconjuntos.

Cabe aclarar que existen más métodos de búsqueda, tales como realizar un simple for para iterar sobre todos los números. Por simplicidad, se explicará solo los mencionados anteriormente.

Permutaciones

Definición: Es una forma de ordenar los elementos de un conjunto de forma específica. Por ejemplo, una permutación del conjunto $\{1, 2, 4, 4, 5\}$:

$$[4, 5, 4, 1, 2]$$

La cantidad de permutaciones posibles con n elementos diferentes es $n!$, lo que crece rápidamente:

n	1	2	3	4	5	6	7	8	9	10	11
n!	1	2	6	24	120	720	5040	40320	362880	3628800	39916800

Permutaciones

Podemos listar todas las permutaciones posibles de forma ordenada, dando prioridad a los índices menores, por ejemplo, estas son todas las permutaciones posibles de largo 3, ordenadas lexicográficamente:

$$[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]$$

Para conseguir imprimir todas las permutaciones, ¿cómo podemos obtener la permutación siguiente?, para esto existe next_permutation en c++, este comando nos permite iterar sobre todas las permutaciones posibles.

next_permutation

Aquí un código que imprime todas las permutaciones posibles de largo 4:

```
vector<int> perm = {1,2,3,4};
do {
    for (auto c : perm)
        cout << c << " ";
    cout << endl;
} while(next_permutation(perm.begin(),perm.end()));
```

Subconjuntos

Un subconjunto es una serie de elementos que pertenecen a un conjunto original, pero no necesariamente conserva todos los elementos, por ejemplo:

$[1, 3, 4]$, $[1, 2, 3, 4, 5]$, $[5]$, $[]$ son subconjuntos de $[1, 2, 3, 4, 5]$.

$[6]$, $[5, 4]$, $[1, 2, 3, 4, 5, 6]$ **NO** son subconjuntos de $[1, 2, 3, 4, 5]$.

La cantidad de subconjuntos diferentes de un conjunto de n elementos son 2^n, lo cual crece exponencialmente:

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**19**	**20**	**21**
2^n	2	4	8	16	32	64	128	256	524288	1048576	2097152

Subconjuntos

Podemos codificar cada subconjunto como una máscara de bits: si el elemento está dentro del conjunto el bit es 1, en caso contrario 0. Ejemplo:

Conjunto original: $[1, 2, 3]$

Máscara	000	001	010	011	100	101	110	111
Subconjunto	[]	$[1]$	$[2]$	$[1, 2]$	$[3]$	$[1, 3]$	$[2, 3]$	$[1, 2, 3]$

Y cada máscara representa un número natural desde 0 hasta $2^n - 1$, entonces para pasar por todos los subsets, simplemente podemos usar un for en c++.

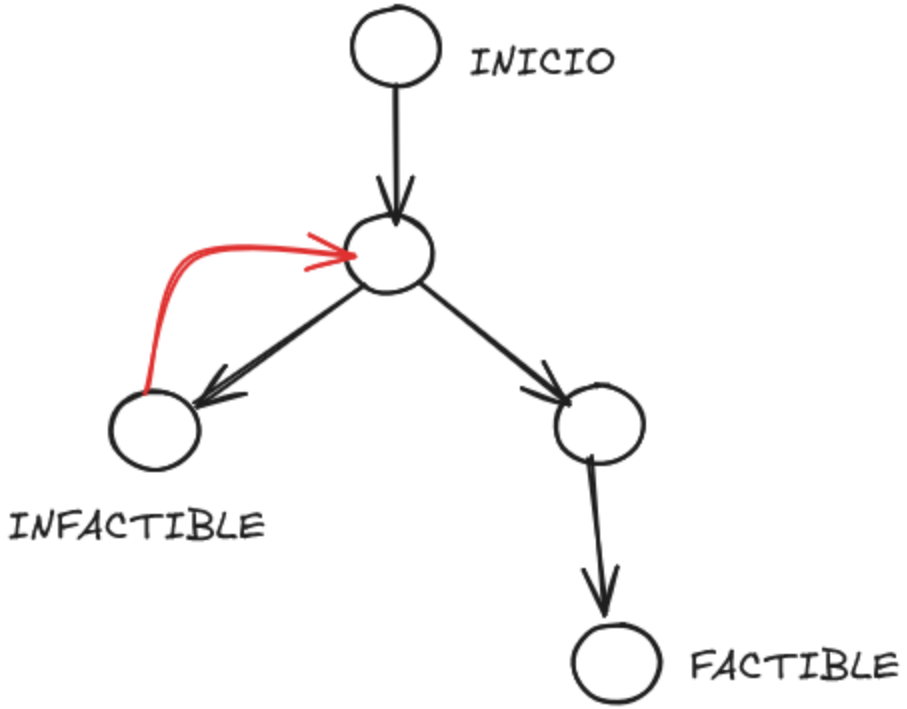
Máscaras

Este código de c++ obtiene la suma de cada subconjunto de $[1, 2, 3]$:

```
int n = 3;
vector<int> arr = {1,2,3};
vector<int> sum(1<<n);
for (int i = 0; i < (1<<n); i++){
    int num = 0;
    for (int j = 0; j < n; j++){
        if (i&(1<<j)){
            num += arr[j];
        }
    }
    sum[i] = num;
}
```

Backtracking

Backtracking es una técnica que consiste en construir la solución paso a paso, y si en algún punto la solución se vuelve infactible, se retorna a un estado anterior, hasta encontrar una solución factible.



Problema Prefiprimos

Enunciado: Se busca contar la cantidad de números de tres dígitos, de forma que cumplen que cada prefijo es también un número primo.

La solución con fuerza bruta, consiste en probar con todos los números posibles de tres dígitos.

```
int cnt = 0;
for (int a = 1; a <= 9; b++){
    for (int b = 0; b <= 9; b++){
        for (int c = 0; c <= 9; c++){
            if (es_primo(a) && es_primo(a+10*b) && es_primo(a+10*b+100*c)){
                cnt++;
            }
        }
    }
}
```

Problema Prefiprimos

La solución con backtracking, en lugar de probar todas las posibilidades, si en un paso se encuentra con un estado inválido regresa al dígito anterior, lo que reduce significativamente el tiempo de búsqueda.

```
int cnt = 0;
for (int a = 1; a <= 9; a++){
    if (!es_primo(a)) continue;
    for (int b = 0; b <= 9; b++){
        if (!es_primo(a+10*b)) continue;
        for (int c = 0; c <= 9; c++){
            if (!es_primo(a+10*b+100*c)) continue;
            cnt++;
        }
    }
}
```

CSES - Chessboard and Queens

La tarea es colocar 8 reinas en un tablero de ajedrez de forma que las reinas no se ataquen entre sí. Además, existen cuadrados ocupados, y no se puede colocar una reina en esos espacios.

Una solución con fuerza bruta probaría todos los casos posibles, sin embargo, podemos encontrar una solución más rápida con backtracking.

CSES - Chessboard and Queens

```cpp
#include<bits/stdc++.h>
using namespace std;
string a[8];
bool col[8],row1[15],row2[15];
int res = 0;
void backtrack(int i){
    if(i == 8){
        res++;
        return;
    }
    for(int j = 0; j < 8; j++){
        if(col[j] || row1[i-j+8-1] || row2[i+j] || a[i][j] == '*') continue;
        col[j] = row1[i-j+8-1] = row2[i+j] = true;
        backtrack(i+1);
        col[j] = row1[i-j+8-1] = row2[i+j] = false;
    }
}
int main(){
    for (int i = 0; i < 8; i++) cin >> a[i];
    backtrack(0);
    cout << res << endl;
    return 0;
}
```