Backtracking - Gerando todos os subconjuntos
Professor Wladimir

Backtracking

Subsets

Given an integer array nums of unique elements, return all possible subsets (the power set). The solution set must not contain duplicate subsets. Return the solution in any order.
Example 1:
Input: nums = [1,2,3]
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
Example 2:
Input: nums = [0]
Output: [[],[0]]
Constraints:

- $1 <= nums.length <= 10$
- $-10 <= nums[i] <= 10$
- All the numbers of nums are unique.

Bitmask

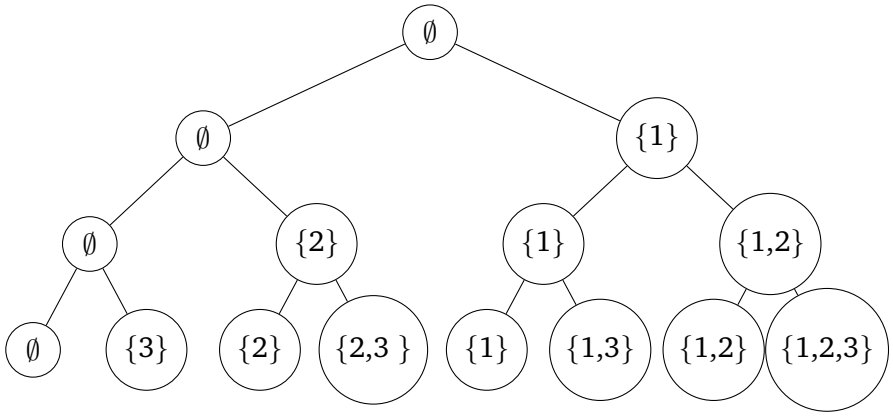
Para saber mais sobre manipulação de bits:
https://cp-algorithms.com/algebra/bit-manipulation.html.

```
class Solution {
public:


    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> ans;

        int n = nums.size();

        for(int mask = 0; mask < 1<<n; mask++){
            vector<int> set;
            for(int j = 0; j < n; j++){
                if( mask & (1 << j) ){
                    set.push_back(nums[j]);
                }
            }
            ans.push_back(set);
        }

        return ans;
    }
};
```

Backtracking

A cada nível da recursão, ele decide se inclui ou não inclui o elemento atual.
A árvore de recursão tem a seguinte estrutura:

- Cada nível da árvore representa uma decisão sobre incluir ou não um elemento.
- A esquerda representa a decisão de não incluir o elemento atual.
- A direita representa a decisão de incluir o elemento atual.

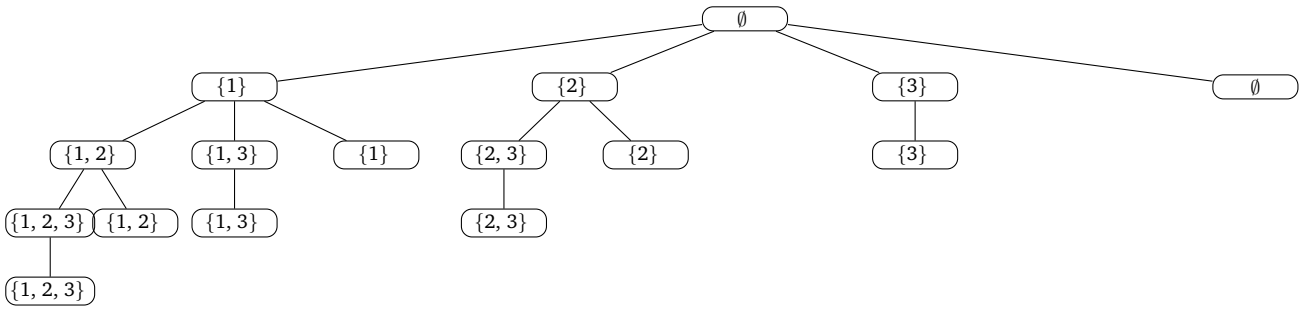



```
class Solution {
public:

    void backtrack(int level,
        vector<int>& nums,
        vector <int> & set,
        vector<vector<int>> & ans
        )
    {
        if(level == nums.size()){
            ans.push_back(set);
        }else{
            backtrack(level+1, nums, set, ans);
            set.push_back(nums[level]);
            backtrack(level+1, nums, set, ans);
            set.pop_back();
        }
    }

    vector<vector<int>> subsets(vector<int>& nums) {
        vector <int> set;
        vector<vector<int>> ans;
        backtrack( 0, nums, set, ans );
        return ans;
    }
};
```

Backtracking2

Em cada nó, a função:

- Decide incluir nums[i], onde i vai de start até nums.size()-1.
- Avança para o próximo índice i+1 com o elemento incluído.
- Ao final do for, adiciona o subconjunto atual (que pode estar vazio ou parcial).




```cpp
class Solution {
public:
    void backtrack2(int start,
        vector<int>& nums,
        vector <int> & set,
        vector<vector<int>> & ans
        )
    {


        for(int i = start; i < nums.size(); i++){
            set.push_back(nums[i]);
            backtrack2(i+1, nums, set, ans);
            set.pop_back();
        }

        ans.push_back(set);


    }
    vector<vector<int>> subsets(vector<int>& nums) {
        vector <int> set;
        vector<vector<int>> ans;
        backtrack2( 0, nums, set, ans );
        return ans;
    }
};
```