Recursion vs. DFS.

# DFS is more general scope. it's one kind of search algorithm.

# can be implemented in either recursive or iterative ways.

DFS:

recall "pre-order" to traverse a tree.

Easy to use recursion.

DFS Common Questions:

#1. Print all subsets of a set.

#2. Print all valid permutations of (). ().().

#3. Combination of Coins

#4. Will be covered later.

Basic Idea: ☆☆☆

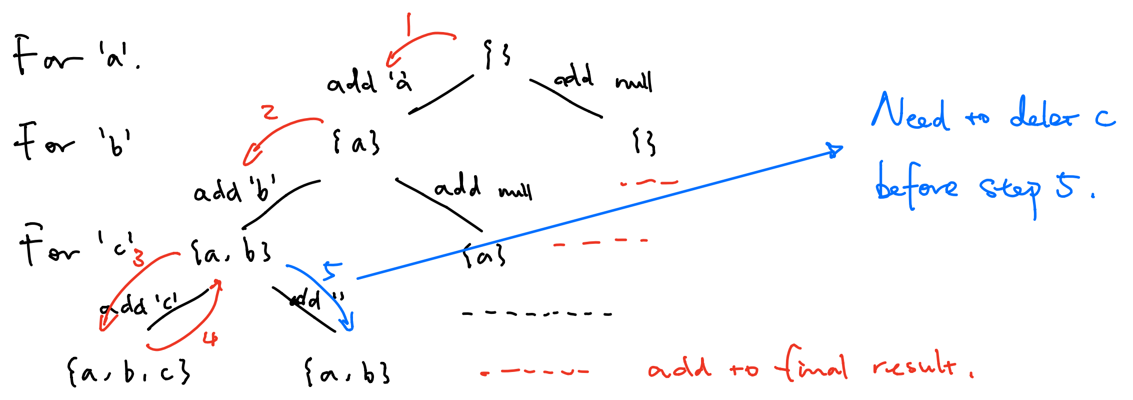
1. How many levels in the recursion tree?
   What does it store on each level?
2. How many diff states should we try to put on each level.

Q1. Print all subsets of a set S.

eg: {a, b, c}.

Basic Idea:

1. How many levels in the recursion tree?.
   3 levels. For each level. add or not add to final set
2. How many diff states should we try to put on each level?
   2 states. add new elements / not add.



inputs : (char[] input, int index, StringBuilder sb).
(current level.

Base Case : if index hits length of input.
add to final result.

Recursion :

sb.append (input [index]) ;
recursion on (index + 1) ;
} add.

☆ sb. deleteCharAt (sb. length() − 1) ;
recursion on (index + 1) ;
} Not add.

Q2. Valid Permutation using the parthesis provided. eg. ()()()
Basic Idea :

1. How many levels ?
   6. each level represents one position.
2. How many states ?
   2. either add "(" or add ")".

Restriction: whenever we add another ")", we make

[illegible]

sure # of "(" added so far > # of ")" so far

inputs:

```
int n — # of pairs of ()
int l — # of left parenthese
int r — # of right parenthes.
StringBuilder.
if ( l== n && r==n). add to rst.
if ( l < n )
    append "(".
    recursion.
    deleteCharAt
if ( r < l ).
    append ")"
    recursion
    deleteCharAt.
```

Time: $O(2^{2n} \cdot n)$.

Space: $O(2n) \rightarrow O(n)$

Q3. All Combinations of Coins. that can sum up to n.

e.g: n = 99. [25, 10, 5, 1]

Soln #1. #1. How many levels?

99: 99/1=99.

#2. How many diff states?

4 states: 25, 10, 5, 1.

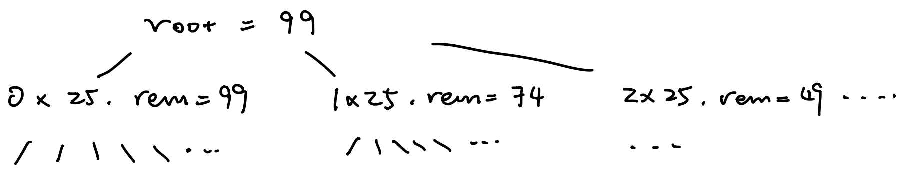
Low efficiency: 4^99.

Soln #2 #1. How many levels?

4: 25. 10. 5. 1.

#2. How many states?

Dynamically changing.



Time: $O(99^4)$.

inputs: money left, index level, int[] sol.

Base Case: if index = coins.length.

sol[index] = moneyleft (last coin is 1).

Recursion: for i = 0 ; i < moneyleft / coins[index].

sol[index] = i.

recursion(moneyleft − i * coins[index], level+1, sol);

Q4. Permutations of a string. (no duplicates).

Basic Idea:

#1. How many levels?

Length of string. Each level represents one position.

#2. How many states?

$n \rightarrow n-1 \rightarrow n-2 \rightarrow \cdots \rightarrow 2 \rightarrow 1$.

Decrease by 1 for each level.

input: char[] s, index.

[0 ... index]. positions that are confirmed.

[index --- n-1]. positions not sure.

Base case : if index hits length of string.
add to rst & return

```
Recursion :  for i = index ; i < length ; i++.
                 swap ( i , index )
                 recursion ( s , index+1 );
                 swap ( i , index ).
```

Time : $O(n! \cdot n)$

Space : $O(n)$

Notes : Whenever every permutation contains all elements in the initial input, then consider swapping.

#1. Subset : order NOT matter. States constant

#2. () : order NOT matter. States constant. Cut subtree

#3. 99 cents : order NOT matter. States dynamically change

#4. Permutation : order MATTER. Swap & swap.