

Recursion - III Complete

Permutation

To print all the permutations of a string.

Idea: for each character $s[i]$ in the given string, we add a character in the ans string and then solve $s.substr(0,i) + s.substr(i+1)$

Sample Input:

ABC

Sample Output:

ABC

ACB

BAC

BCA

CAB

CBA

Time Complexity: $O(N \cdot 2^n)$

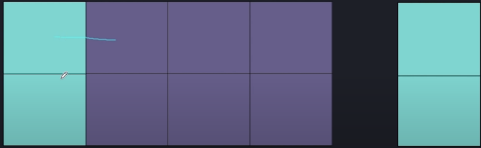
Space Complexity: $O(2^n)$

```
void permutation(string s, string ans) {  
  
    if (s.length() == 0) {  
        cout << ans << endl;  
        return;  
    }  
  
    for (int i = 0; i < s.length(); i++) {  
        char ch = s[i];  
        string ros = s.substr(0, i) + s.substr(i + 1);  
  
        permutation(ros, ans + ch);  
    }  
}
```

Recursion | Last Lecture on Advanced Problems | Lecture 16.4

Tiling problem

Given a "2 x n" board and tiles of size "2 x 1", count the number of ways to tile the given board using these tiles



2 x 4 2 x 1

3:00 / 19:22

If $n = 4$

- 1) - when we place 1 horiz than above it has to be horiz so, remaining two cols - both vertical or both horiz so this leads us to ways - way1 - all horiz
- way 2 - 1 horiz and 2 vert
- 2) - way 3 - all vert.

Note when we place a tile vert it only subtracts one col, but when we place a tile horiz it occupies 2 cols

Tiling problem

Find the number of ways to tile the floor with 1x2 and 1x1 tiles.

Idea: $\text{Tile}[i] = \text{Tile}[i-1] (1 \times 1) + \text{Tile}[i-2] (1 \times 2)$

Time Complexity: $O(2^n)$

Space Complexity: $O(2^n)$ //Memory is used for call stack as well

```
int tilingWays(int n) {
    if (n == 0) {
        return 0;
    }
    if (n == 1) {
        return 1;
    }
    return tilingWays(n - 1) + tilingWays(n - 2);
}
```

Knapsack [IMP]

Given n items, each item has a certain value and weight. We have to maximize the value of the objects we can accommodate in a bag of weight W.

Idea: For each item, we have two choices, either to include it or exclude it.

Time Complexity: $O(2^n)$

Space Complexity: $O(2^n)$ //space for call stack *If n=3*

```
int knapsack(int value[], int wt[], int n, int W) {
    if (n == 0 || W == 0) {
        return 0;
    }
    if (wt[n - 1] > W) {
        return knapsack(value, wt, n - 1, W);
    }
    return max(knapsack(value, wt, n - 1, W - wt[n - 1]) + value[n - 1],
               knapsack(value, wt, n - 1, W));
}
```

Handwritten annotations:

- index 2 → Last Item* (pointing to `wt[n - 1]`)
- 2nd item.* (pointing to `n - 1` in the recursive call)
- index 2* (pointing to `n - 1` in the first recursive call)
- 2* (pointing to `W - wt[n - 1]`)

Friends Pairing Problem

There are n friends, we have to find all the pairings possible. Each person can be paired with only one person or does not pair with anyone.

Idea: we have two options, i 'th friend does not get paired or we have $n-1$ options to pair it with anyone.

Time Complexity: $O(n)$

Space Complexity: $O(n)$

```
int friendsPairing(int n) {  
    if (n == 0 || n == 1 || n == 2) {  
        return n;  
    }  
    return friendsPairing(n - 1) + friendsPairing(n - 2) * (n - 1);  
}
```

Friends pairing problem

Find the number of ways in which n friends can remain single or can be paired up.



If 1 is left out \rightarrow Doing operation on $n-1$.

If 1 is also considered \rightarrow Doing opⁿ on $(n-2)$, but also finding one person from $n-1$, for pairing with 1. \therefore So $\rightarrow (n-2) \times (n-1)$

CountPaths

Find the number of ways to reach e from s.

Idea:

We have 6 ways to go forward (1,2,3,4,5,6).

At the starting point s,

Current answer = countPath(s+1,e) + countPath(s+2,e) + countPath(s+3,e) +
countPath(s+4,e) + countPath(s+5,e) + countPath(s+6,e)

Time Complexity: $O(2^n)$

Space Complexity: $O(2^n)$

```
int countPath(int s, int e) {  
    if (s == e) {  
        return 1;  
    }  
    if (s > e) {  
        return 0;  
    }  
    int count = 0;  
    for (int i = 1; i <= 6; i++) {  
        count += countPath(s + i, e);  
    }  
    return count;  
}
```

0-1 Knapsack Problem

Put n items with given weight and value in a knapsack of capacity W to get the maximum total value in the knapsack



0-1 Knapsack Problem

Put n items with given weight and value in a knapsack of capacity W to get the maximum total value in the knapsack

i	0	1	2
wt[i]	10	20	30
value[i]	100	50	150

W = 50

CountPathMaze

Given a 2D grid, find the number of ways to reach (n-1, n-1).

You can go to (i,j) from (i-1,j) and (i,j-1).

Time Complexity: $O(2^n)$

Space Complexity: $O(2^n)$

```
int countPathMaze(int n, int i, int j) {  
    if (i == n - 1 && j == n - 1) {  
        return 1;  
    }  
    if (i >= n || j >= n) {  
        return 0;  
    }  
  
    return countPathMaze(n, i + 1, j) +  
           countPathMaze(n, i, j + 1);  
}
```