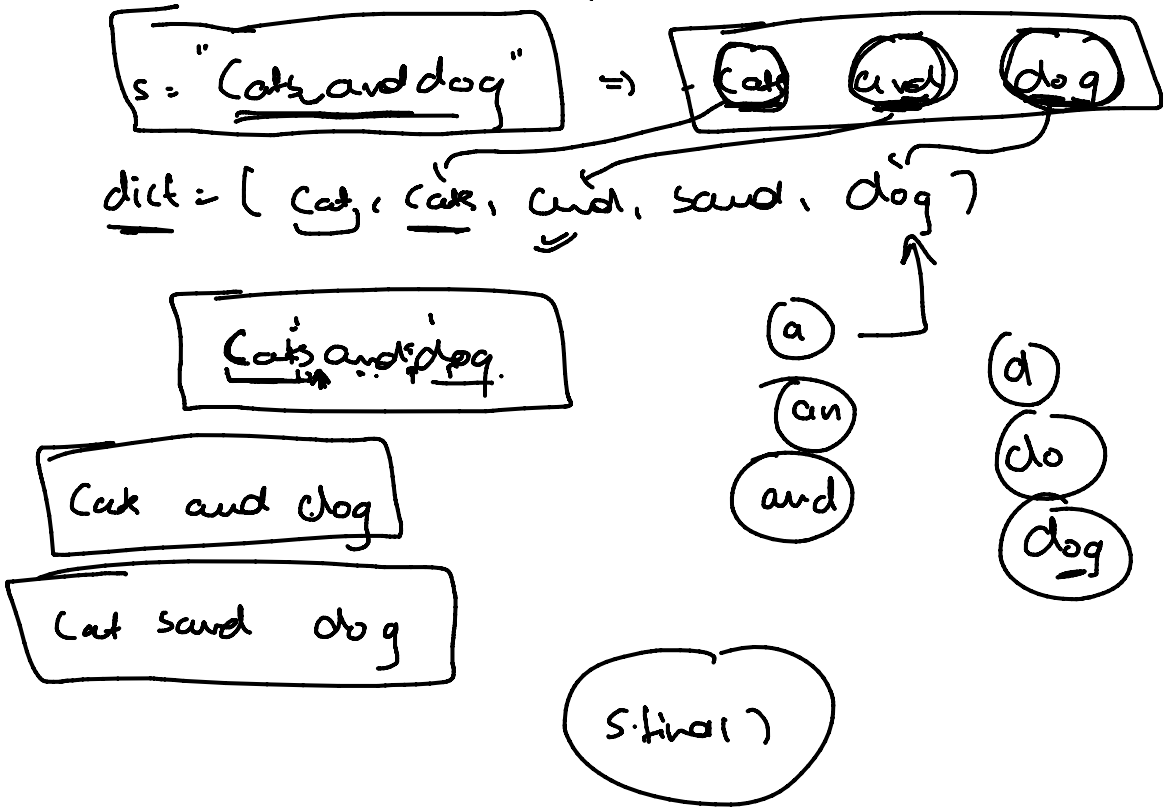
CLASS 78

06 May 2023 14:13

BackTracking

Word Break - 2




```
Vector < string > wordBreak ( string s, vector < string >& wordDict )
{
    set < string > sa;

    for ( auto i : wordDict )
    {   sa.insert (i);
    }

vector < string > ans;
string temp = " ";

word ( s, 0, sa, ans, temp )-   return ans;
```

```
    word(s, 0, sa, ans, temp);   return ans;
}

void word(string s, int idx, set<string>& sa, vector<string>& ans,
          string temp)
{
    int n = s.size();
    if(idx == n)
    {
        ans.push_back(temp);
        return;
    }
    for(int i = idx; i<n; i++)
    {
        string substr = s.substr(idx, i-idx+1);
        string temp2 = temp;
        if(sa.find(substr) != sa.end())
        {
            temp += substr;
            if(i+1 != n)
            {temp += " ";}
            word(s, i+1, sa, ans, temp);
            temp = temp2;
        }
```

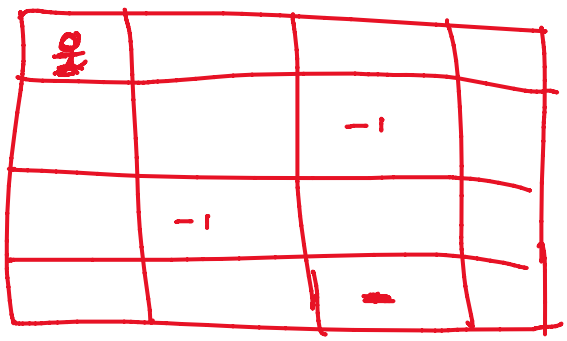
Cute and day

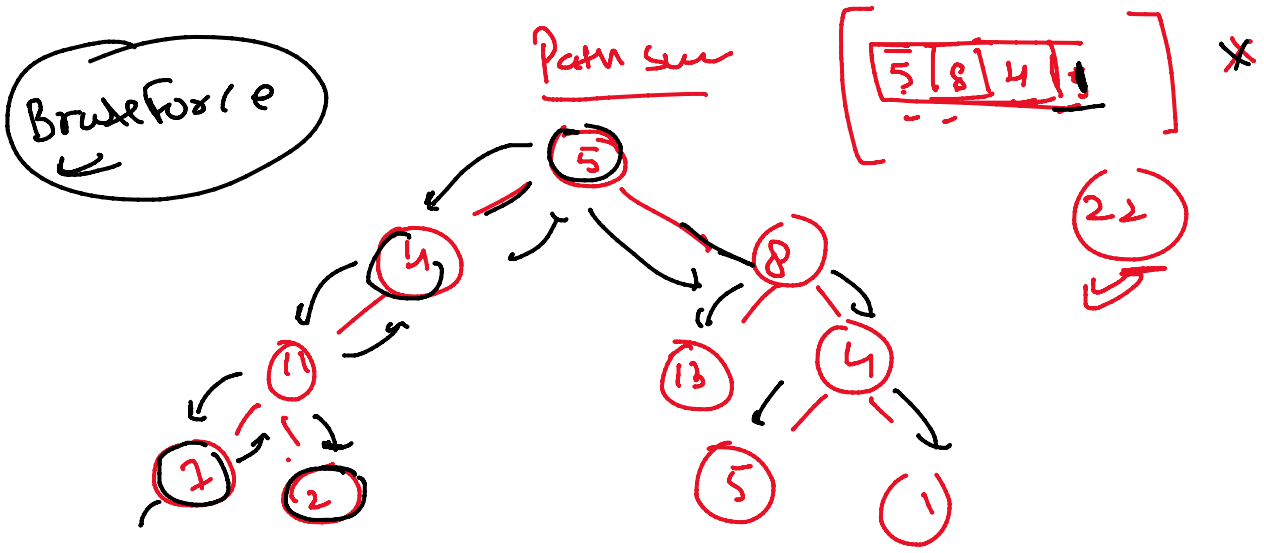
Sorted

Unique Paths 3

```
int uniquePathsIII(vector<vector<int>>& grid) {
    int i, j, k, m, n;
    int spx = -1;
    int spy = -1;
    int epx = -1;
    int epy = -1;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            if (grid[i][j] == 1) {
                spx = i;
                spy = j;
            } else if (grid[i][j] == 2) {
                epx = i;
                epy = j;
            }
        }
    }
    int ans = 0;
    unique(grids, spx, spy, epx, epy, n, m, ans);
    return ans;
}

void unique(vector<vector<int>>& grid, int cx, int cy, int epx,
            int epy, int n, int m, int &ans) {
    if (cx == epx && cy == epy) {
        ans++;
        return;
    }
```

```
    {
        cnt++; return;
    }

    int temp = grid[nx][ny];
    grid[nx][ny] = -1;
    int dx[4] = {-1, 1, 0, 0};
    int dy[4] = {0, 0, -1, 1};

    for (i = 0; i < 4; i++)
    {
        int nx = dx[i] + cx;
        int ny = dy[i] + cy;

        if (nx >= n or nx < 0 or ny >= m or ny < 0 or
            grid[nx][ny] == -1)
            continue;

        unique(grid, nx, ny, ex, ey, n, m, cnt);
    }

    grid[cx][cy] = temp;
}
```






```
vector<vector<int>> PathSum(TreeNode* root, int sum)
{
    if(root == NULL)
    {
        return {};
    }
    Path(root, sum);
    return ans;
}

vector<vector<int>> ans;
vector<int> v;

void Path(TreeNode* root, int sum)
{
    if(root == NULL)
    {
        if(sum == 0)
            ans.pb(v);
        return;
    }
```

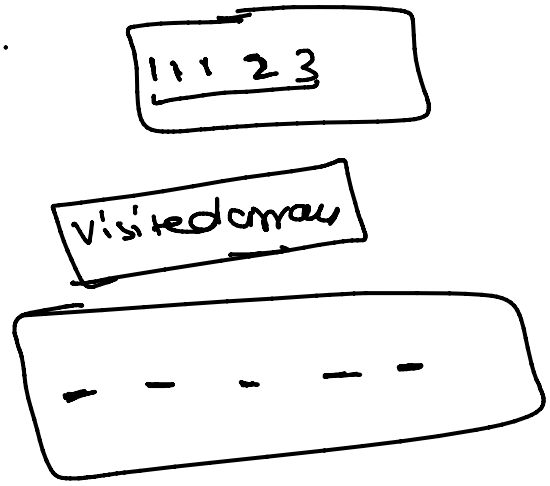
leaf Node

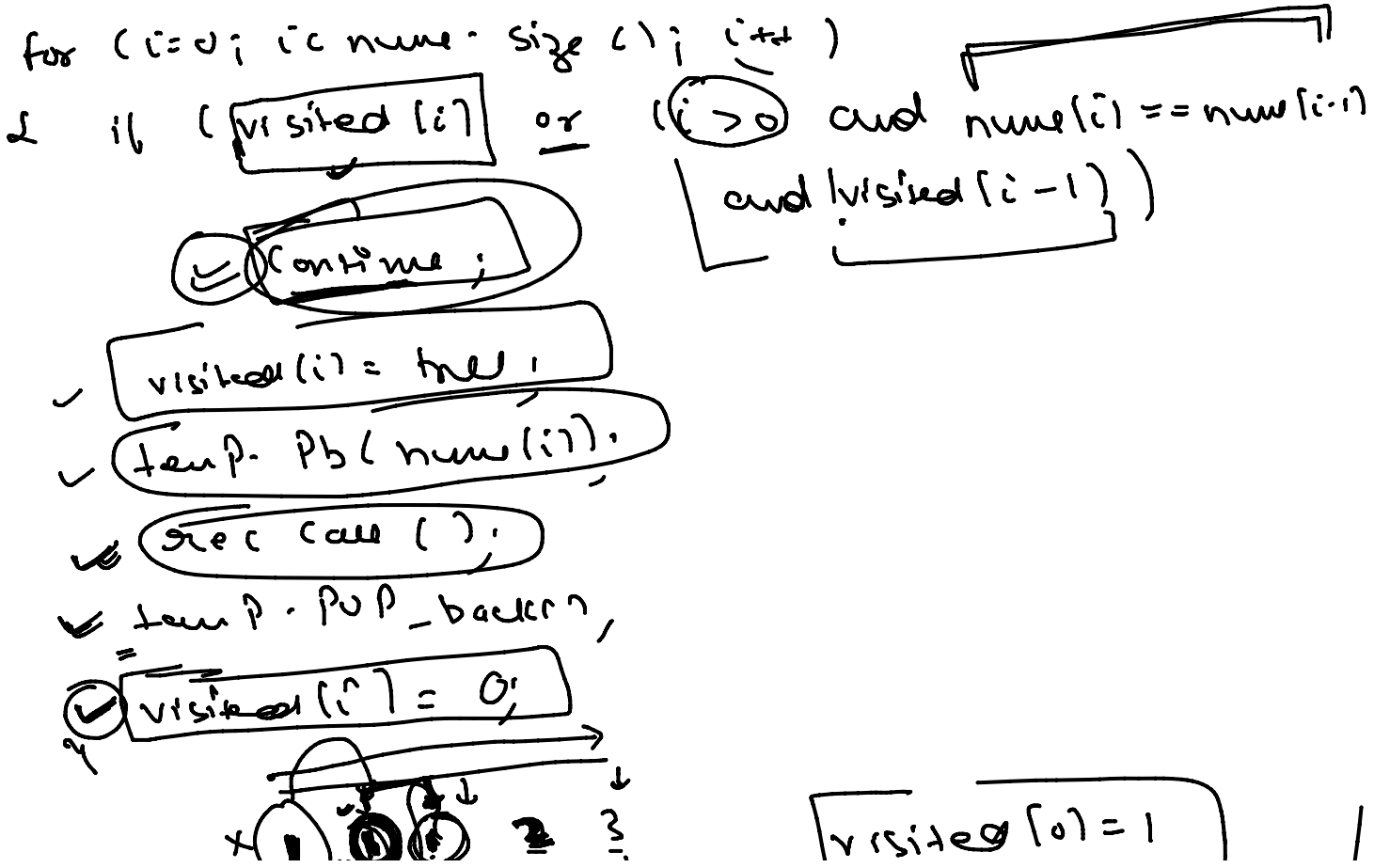
```
        return ;
    }
```

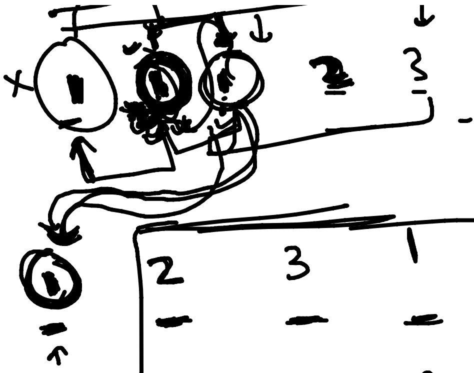
```
v.pb ( root -> val);
if(root -> left == NULL and root -> right != NULL)
{   Path (root -> right, sum = root -> val) ;
    v.pop_back()
    return;
}
if(root -> right == NULL && root -> left != NULL)
{ Path (root -> left, sum -root -val);
    v.pop_back()
    return ;
}
if (root -> right != NULL and root -> left != NULL)
{ Path (root -> left , sum -root -> val )
    Path ( " -> right , "     " );
    v.pop_back()
}
Path( root -> left , sum -root -> val) ;
v.pop_back(),
}
```

```cpp
class Solution {
private:
    void permuteUnique(vector<int>& nums, vector<vector<int>>& output, vector<int> temp, vector<bool>& visited){
        if(temp.size() == nums.size()){
            output.push_back(temp);
            return;
        }
        for(int i=0; i<nums.size(); i++){
            if(visited[i] || i>0 && nums[i] == nums[i-1] && !visited[i-1]) continue;
            visited[i] = true;
            temp.push_back(nums[i]);
            permuteUnique(nums, output, temp, visited);
            temp.pop_back();
            visited[i] = false;
        }
    }
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> output;
        vector<int> temp;
        vector<bool> visited(nums.size(), 0);
        permuteUnique(nums, output, temp, visited);
        return output;
    }
};
```

From <https://leetcode.com/problems/permutations-ii/solutions/3214219/best-c-3-solution-ever-easy-solution-backtracking-one-stop-solution/>







visited[0] = 1

11123

112233

visited[1] = 0

111122

2	3	1
2	1	3
3	2	1
3	1	2

1 1 1 2 3

3 2

2 1 3

3 1

3 1 2

3 2 1