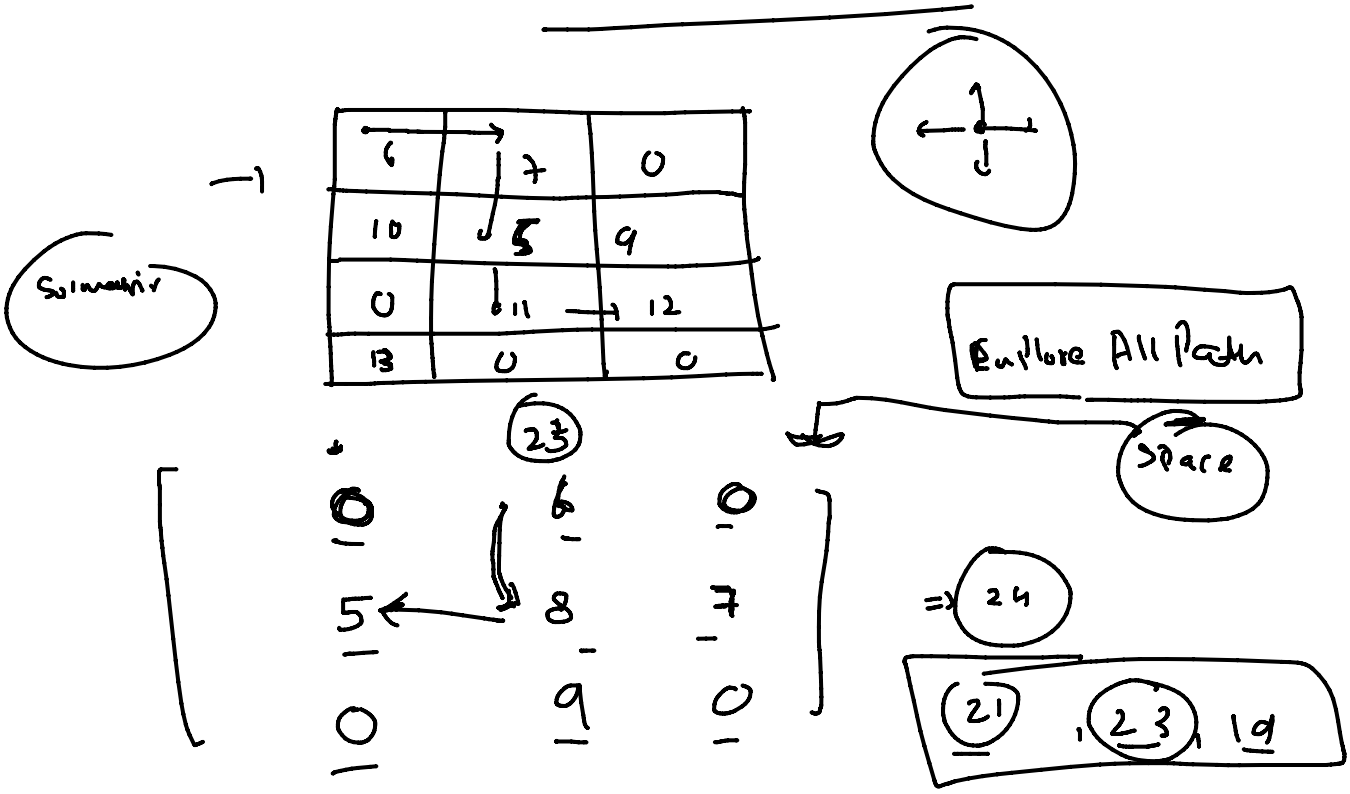
CLASS 72

29 April 2023 12:11

Backtracking

Path with maximum Gold



Code

```
bool visited[16][16]
int getMaximumGold (vector<vector<int>> &a)
{
    int ans = 0;
    int n = a.size();
    int m = a[0].size();
    int i, j;

    for (i=0; i<n; i++)
    {
        for(j=0; j<m; j++)
        {
            if (a[i][j] != 0)
```

```
if (a[i][j] != 0)
{
    memset (visited, false, sizeof(visited));
    ans = max(ans, maxgold (a, i, j));
}
```

```
int maxGold (vector<vector<int>>& grid, int i, int j)
{
    int n = grid.size();
    int m = grid[0].size();

    if ( i<0 or i>=n or j<0 or j>=m or grid[i][j]
        ==0 or visited[i][j] == true)
        return 0;

    int ans = grid[i][j];
    int temp = grid[i][j];
    visited[i][j] = true;

    int op1 = maxgold ( grid, i+1, j );
    int op2 = "  (  ", i-1, j );
    int op3 = "  (  ", i, j+1);
    int op4 = "  (  ", i, j-1);

    ans = max ( ans, op1 + temp);
    ans = max ( ans, op2 + temp);
```

Recursion + Backtrack

ans = ans + "(" + op3 + temp + ");

" = " + "(" + op4 + ")";

visited[i][j] = false

return ans;

Recursion → Backtracking

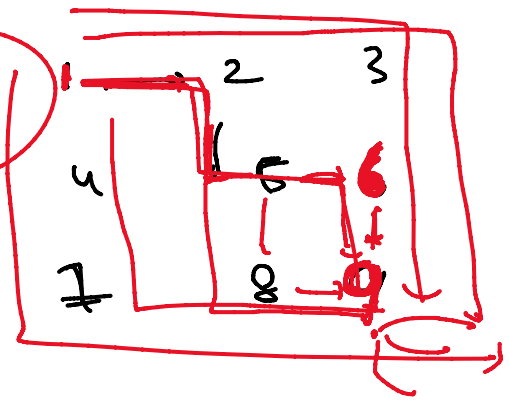
DFS

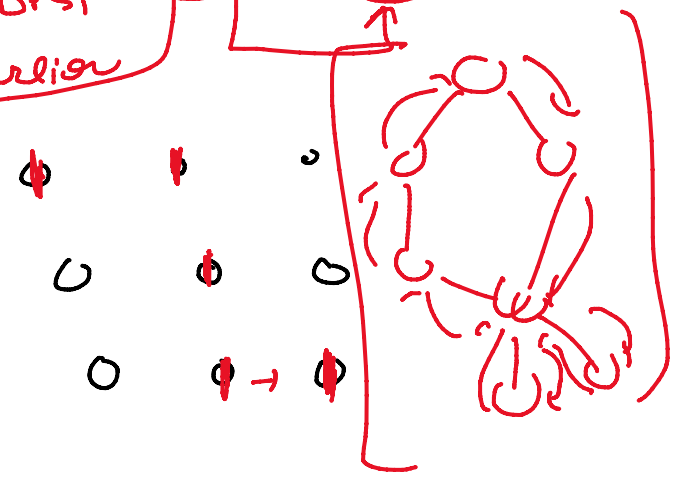
Grid → DFS

Backtracking, DFS, Recursion

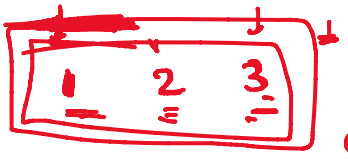
Tree

Recursive Tree





Generate subset



→ {}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}

$2^3 = 8$ subset

[_ , _ , _]

{1,2}

{1,3}

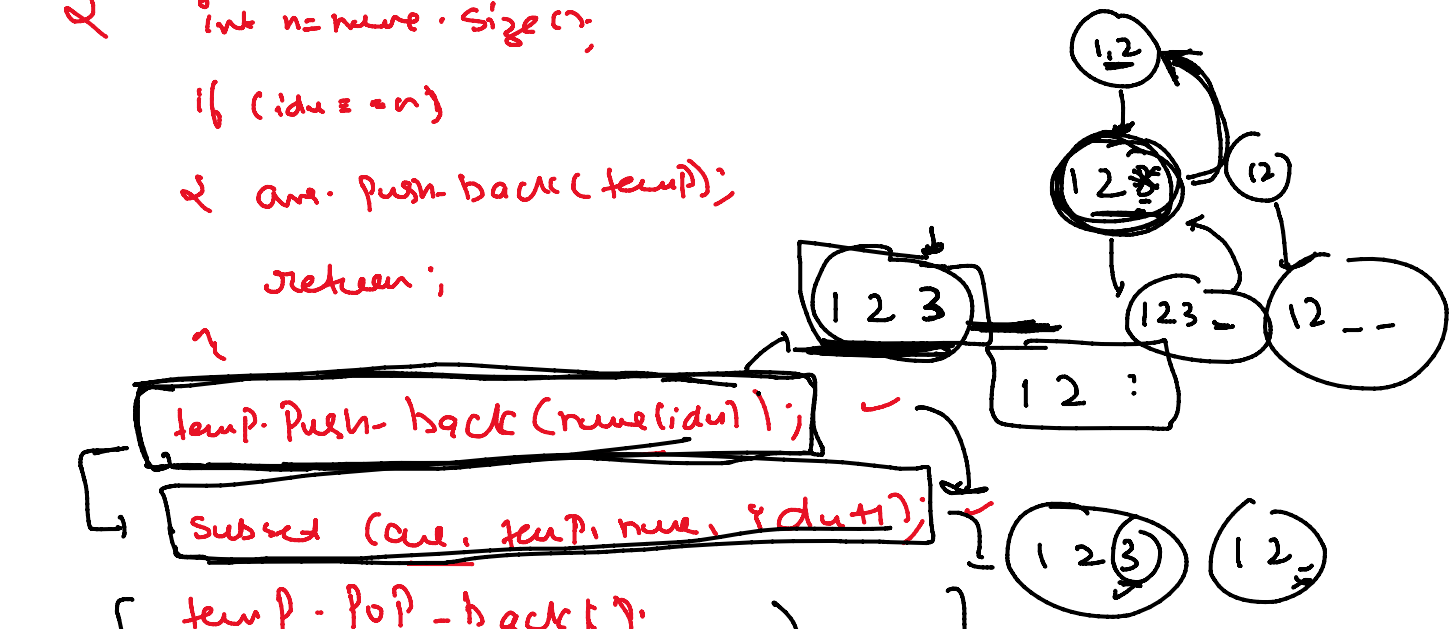
{2,3}

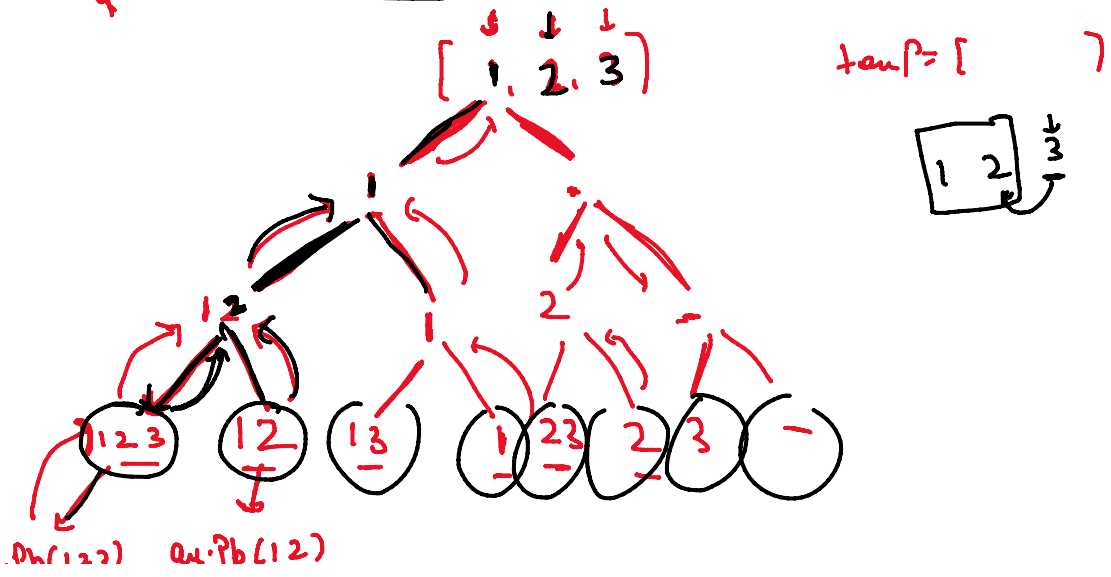
Repeatitive

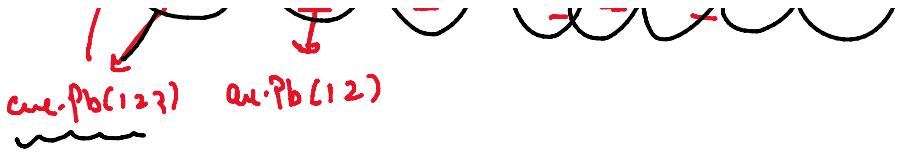
Vector<vector<int>> subset(vector<int>& nums)

{ vector<int> temp;

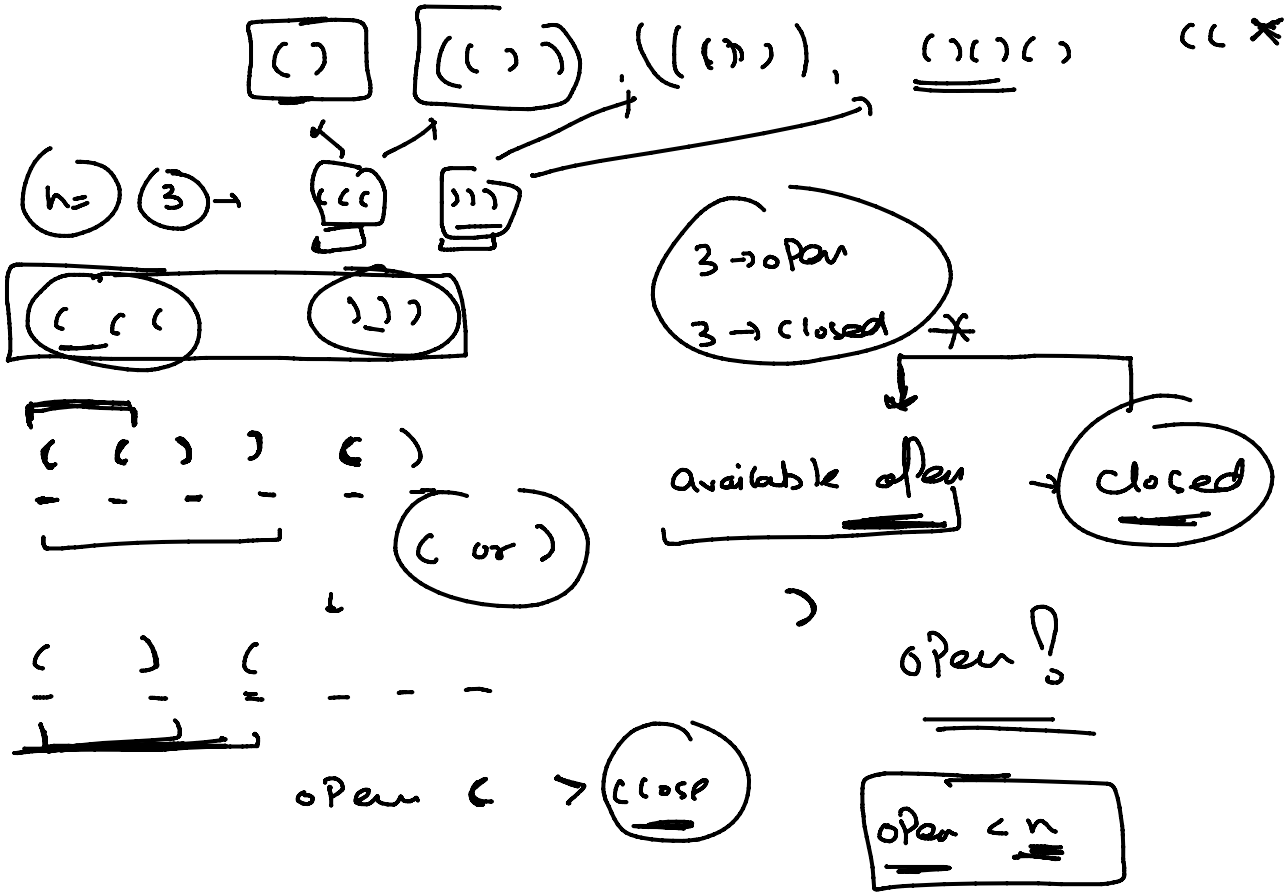
```
{ vector<int> temp;
  vector<vector<int>> ans;
  subset(ans, temp, nums, 0);
  return ans;
}

void subset(vector<vector<int>>& ans, vector<int> temp,
            vector<int> nums, int idx)
{ int n = nums.size();
  if (idx == n)
  { ans.push_back(temp);
    return;
  }
  temp.push_back(nums[idx]);
  subset(ans, temp, nums, idx+1);
  temp.pop_back();
  subset(ans, temp, nums, idx+1);
}
```







Generate Parentheses



close !

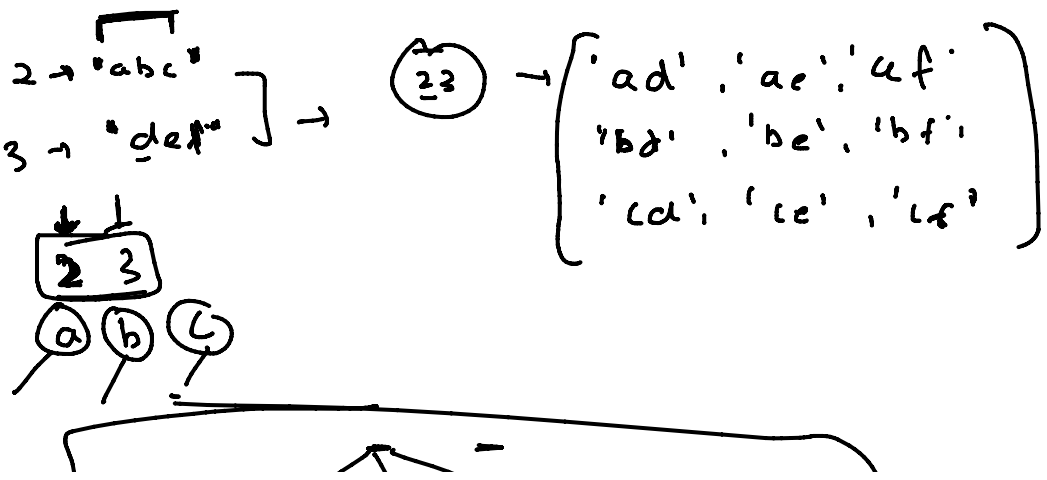
open > close

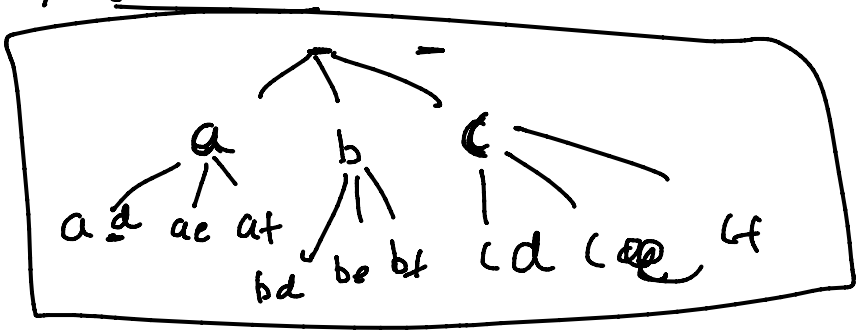
```
vector<string> genParentheses(int n) {

    int oPar = 0;
    int close = 0;
    vector<string> ans;
```

```
vector <string> ans;
string temp = "";
gen(ans, temp, open, close, n);
return ans;
}
void gen(vector<string>&ans, string temp, int open, int close,
         int n)
{
    if(open == n and close == n)
    {
        ans.push_back(temp); return;
    }
    if(open < n)
        gen(ans, temp + "(", open+1, close, n)
    if(open > close)
        gen(ans, temp + ")", open, close+1, n);
}
```

---

letter Combination



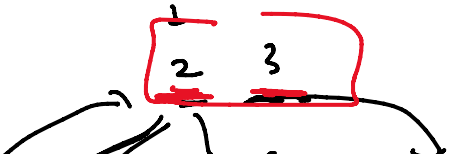


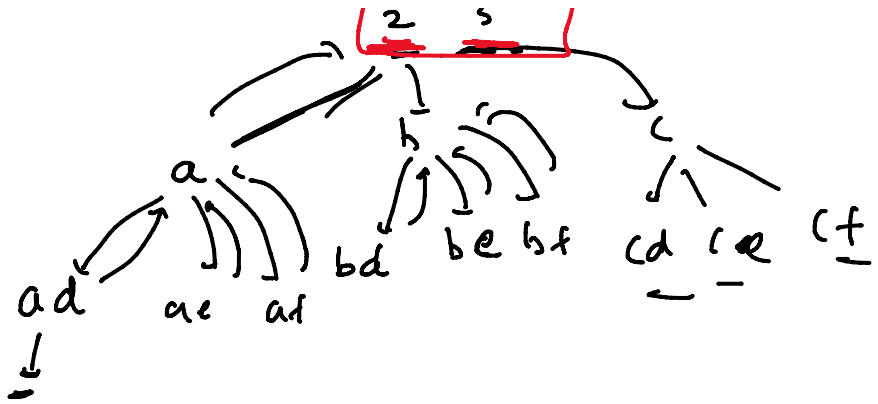

```
string temp = "";
vector<string> ans;
if (digits.size() == 0) return ans;
comb(digits, 0, ans, temp);
return ans;
```

```
void comb(string digits, int idx, vector<string>& ans, string temp)
{
    int n = digits.size();
    if (idx == n)
    {
        ans.push_back(temp);
        return;
    }
    for (auto i : mp[digits[idx]])
    {
        temp.push_back(i);
        comb(digits, idx+1, ans, temp);
        temp.pop_back();
    }
}
```

mp['2'] = "abc"

'3' = 'def'




2
3
a
b
c
ad
ae
af
bd
be
bf
cd
ce
cf