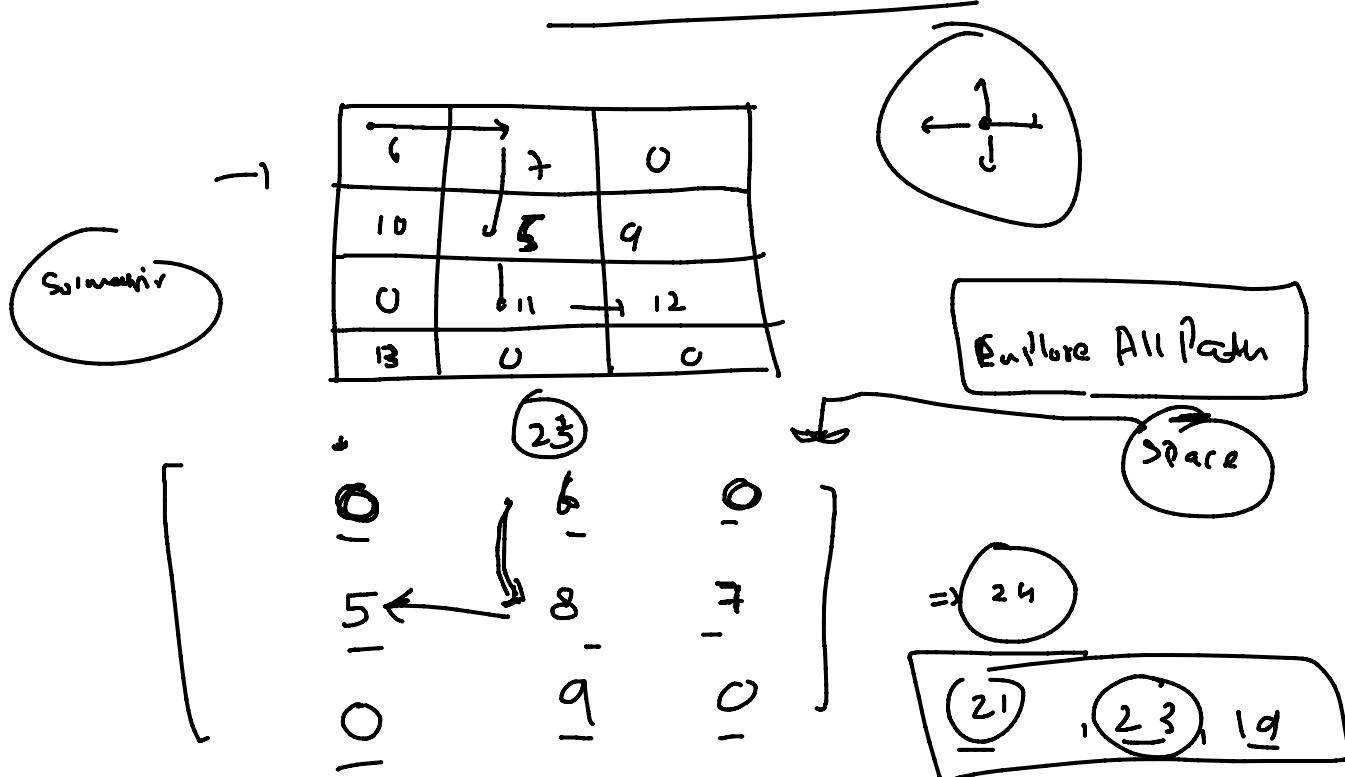


Backtracking

Path with maximum Gold



Code

bool visited[16][16]

int getMaximumGold (vector<vector<int>> a)

```

    int ans = 0;
    int n = a.size(),
        m = a[0].size();

```

int i, j;

for (i=0; i<n; i++)

for (j=0; j<m; j++)

if (a[i][j] == 0)

2 if ($\text{grid}[i][j] = 0$)

{ newset (visited, for, sizeof visited);

Ans = max(Ans, maxgold(a, i, j));

m n n

int maxGold (vector<vector<int>>&grid, int(i), int(j))

2 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)

$\text{ans} = \text{ans} + \text{op}_3 + \text{temp}$
 $\text{"} = \text{"} + \text{op}_4 + \text{"}$

Recursion → Backtracking

DFS

Cross

DFS

Backtracking, DFS, Recursion

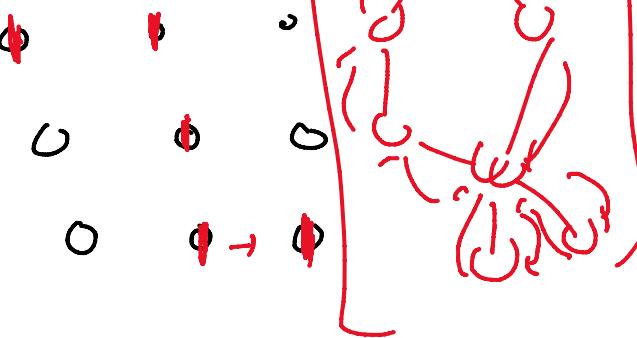
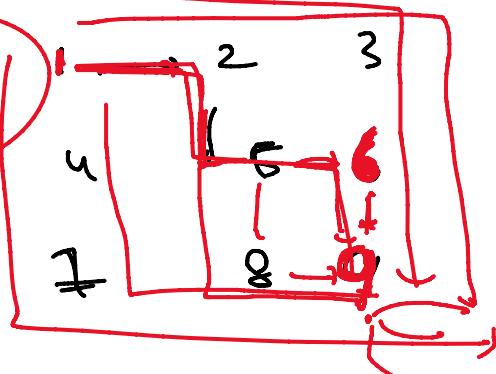
visited(i)(j) = false

check ans;

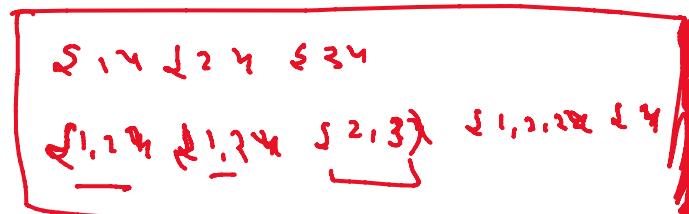
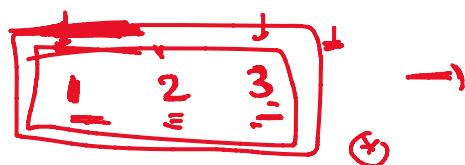
γ

Trees

Recursive Tree



Generate subset



$2^3 = 8$

8 subsets

$\begin{bmatrix} 1 & 2 & 3 \\ - & - & - \end{bmatrix}$

$\{1, 2, 3\}$

$\{1, 3\}$ $\{2, 3\}$

Repetitive

$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$

vector <vector< int>> subset (vector<int> arr)

2. Vector<int> temp

↳ vector<int> temp;
vector<vector<int>> ans;

subset (ans, temp, n, 0);

↑ between ans;

Void subset (vector<vector<int>> &ans, vector<int> &temp,
 vector<int> &n, int &idn)

↳ int n = n.size();
 if (idn == n)
 ↳ ans.push_back(temp);

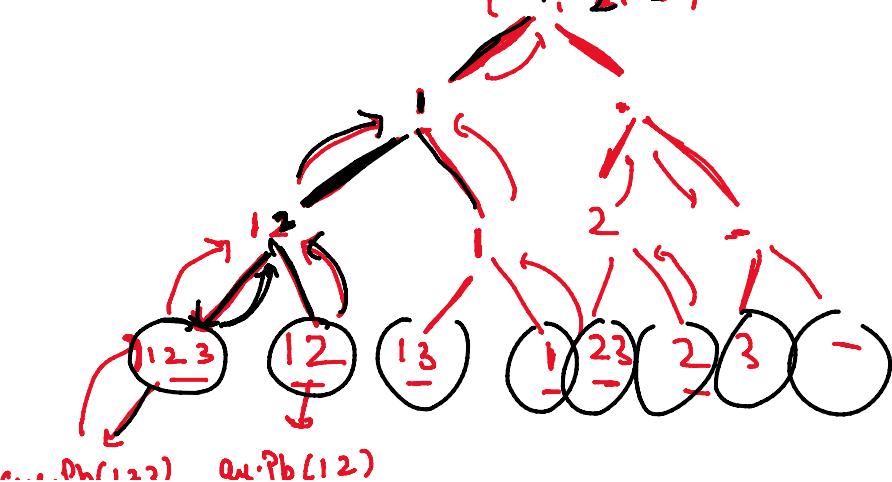
between;

↑ temp.push_back(n[idn]);

subset (ans, temp, n, idn + 1);

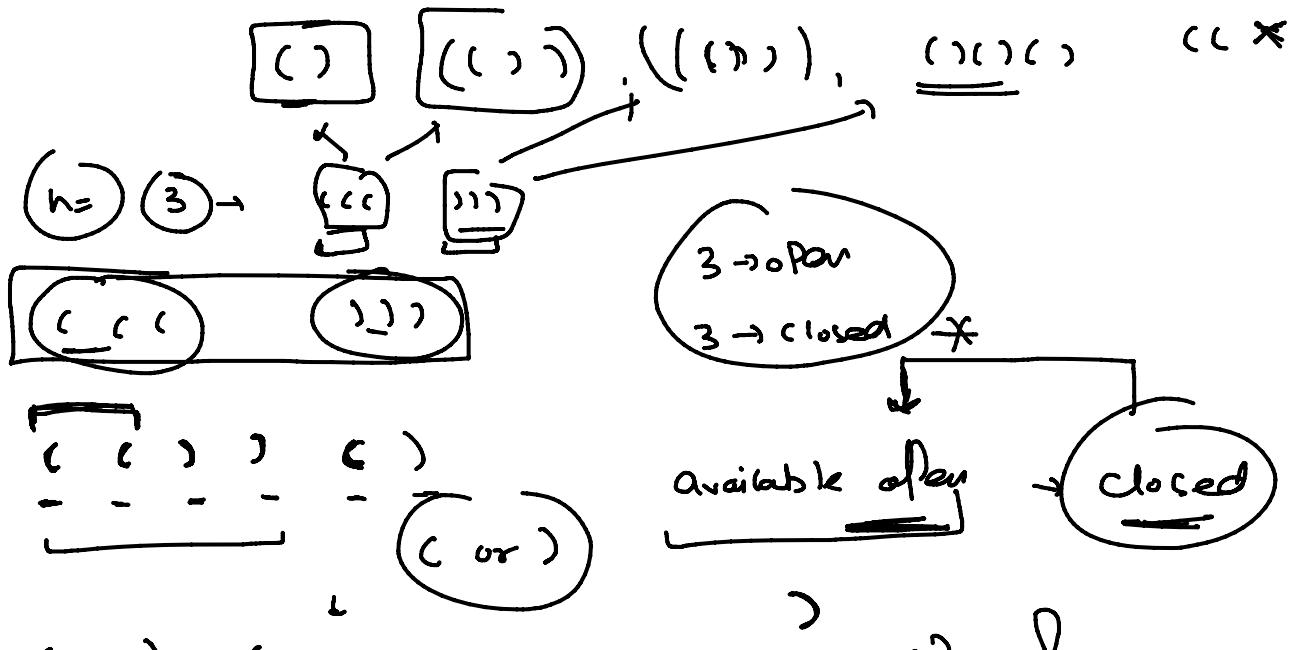
X ↳ temp.pop_back();
 ↳ subset (ans, temp, n, idn + 1);

$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ temp = []



...ans[123] ans[12]

Generate Parentheses



The diagram illustrates a stack frame with the following components:

- Stack Growth:** The stack grows downwards, as indicated by the downward-pointing arrow.
- Current Stack Pointer (CSP):** A circle labeled "CSP" at the bottom of the stack.
- New Frame:** A new frame is created below the CSP, indicated by a bracket labeled "Open".
- Return Address:** A dashed line labeled "RET" points back to the previous frame.
- Local Variables:** A bracket labeled "Locals" groups variables like "a", "b", and "c".
- Function Call:** A bracket labeled "Call" groups the function name "f" and its parameters "x" and "y".

close?

open > close

vector<string> genParentheses(int n) {

int Open = 0;

int done = 0;

```
vector<String> ans;
```

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

`gen(ans, temp, open, close, n);`

reverse ans,

"

void gen (vector<string>&ans, string temp, int open, int close, int n)

{ if (close == n and open == n)

 ans.push_back (temp); reverse (temp);

"

: if (open < n)

 gen (ans, temp + "(", open + 1, close, n)

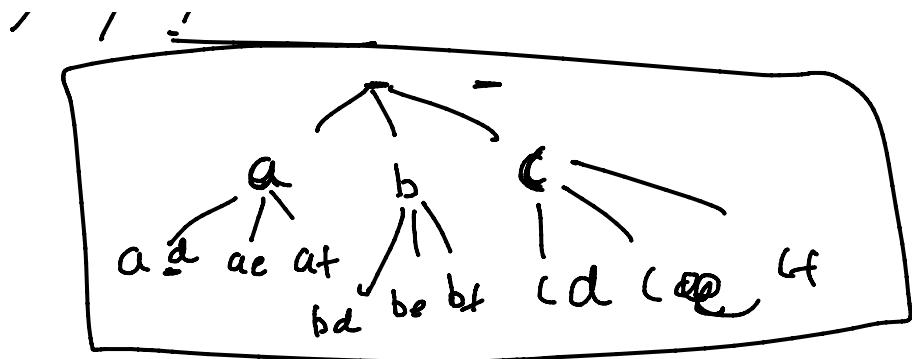
: if (open > close)

 gen (ans, temp + ")". open, close + 1, n)

".

letter combination





String temp = " ";

↓
2 3

vector<string> ans;

if (digit.size() == 0) return ans;

comb(digit, 0, ans, temp);

return ans;

void comb (String digit, int idu, vector<string>& ans, String temp)

{ int n = digit.size();

if (idu == n)

ans.push_back(temp);
return;

for

auto i : ma[digit[idu]])

temp.push_back(i);

comb(digit, idu + 1, ans, temp);

temp.pop_back();

for

a d

ma['2'] =

1	2	3
c	a	b

1 3 = 'def'

temp

