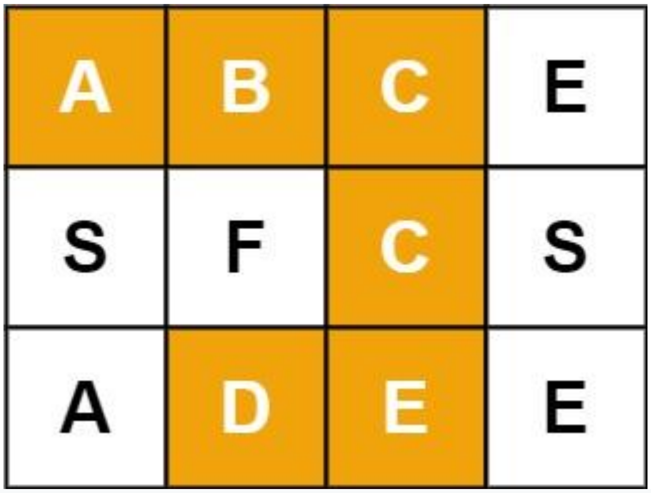
Assignment 7 (Backtracking)

Total 50 points
Each question carries 10 marks

1. Given an `m x n` grid of characters `board` and a string `word`, return `true` *if* `word` *exists in the grid*.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:

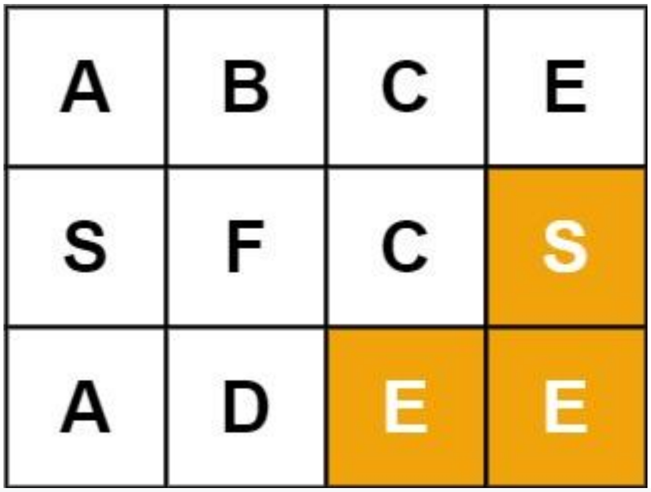


A	B	C	E
S	F	C	S
A	D	E	E

```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word =
"ABCCED"

Output: true
```

Example 2:

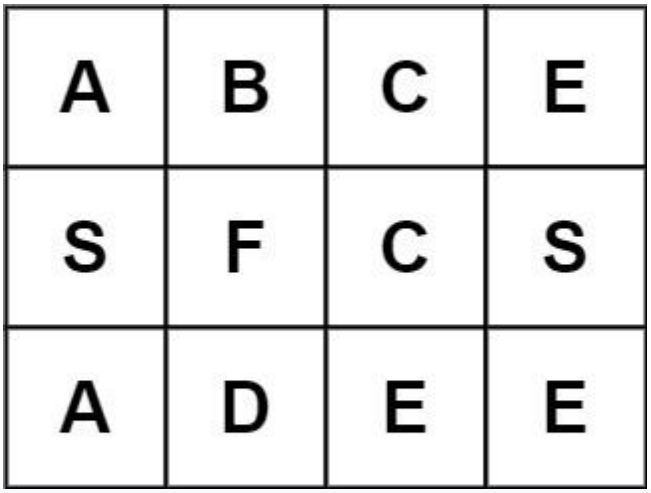


A	B	C	E
S	F	C	S
A	D	E	E

```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"

Output: true
```

Example 3:



A	B	C	E
S	F	C	S
A	D	E	E

```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"

Output: false
```

Constraints:

- `m == board.length`
- `n = board[i].length`
- `1 <= m, n <= 6`
- `1 <= word.length <= 15`
- `board` and `word` consists of only lowercase and uppercase English letters.

2. Suppose you have `n` integers labeled `1` through `n`. A permutation of those `n` integers `perm` (**1-indexed**) is considered a **beautiful arrangement** if for every `i` (`1 <= i <= n`), **either** of the following is true:

- `perm[i]` is divisible by `i`.
- `i` is divisible by `perm[i]`.

Given an integer `n`, return *the **number** of the **beautiful arrangements** that you can construct.*

Example 1:

```
Input: n = 2

Output: 2

Explanation:

The first beautiful arrangement is [1,2]:

    - perm[1] = 1 is divisible by i = 1

    - perm[2] = 2 is divisible by i = 2

The second beautiful arrangement is [2,1]:

    - perm[1] = 2 is divisible by i = 1

    - i = 2 is divisible by perm[2] = 1
```

Example 2:

```
Input: n = 1

Output: 1
```

Constraints:

`1 <= n <= 15`

3. Given a string `s` containing only digits, return all possible valid IP addresses that can be obtained from `s`. You can return them in **any** order.

A **valid IP address** consists of exactly four integers, each integer is between `0` and `255`, separated by single dots and cannot have leading zeros. For example, "0.1.2.201" and "192.168.1.1" are **valid** IP addresses and "0.011.255.245", "192.168.1.312" and "192.168@1.1" are **invalid** IP addresses.

Example 1:

```
Input: s = "25525511135"

Output: ["255.255.11.135","255.255.111.35"]
```

Example 2:

```
Input: s = "0000"

Output: ["0.0.0.0"]
```

Example 3:

```
Input: s = "1111"

Output: ["1.1.1.1"]
```

Example 4:

```
Input: s = "010010"

Output: ["0.10.0.10","0.100.1.0"]
```

Example 5:

```
Input: s = "101023"

Output: ["1.0.10.23","1.0.102.3","10.1.0.23","10.10.2.3","101.0.2.3"]
```

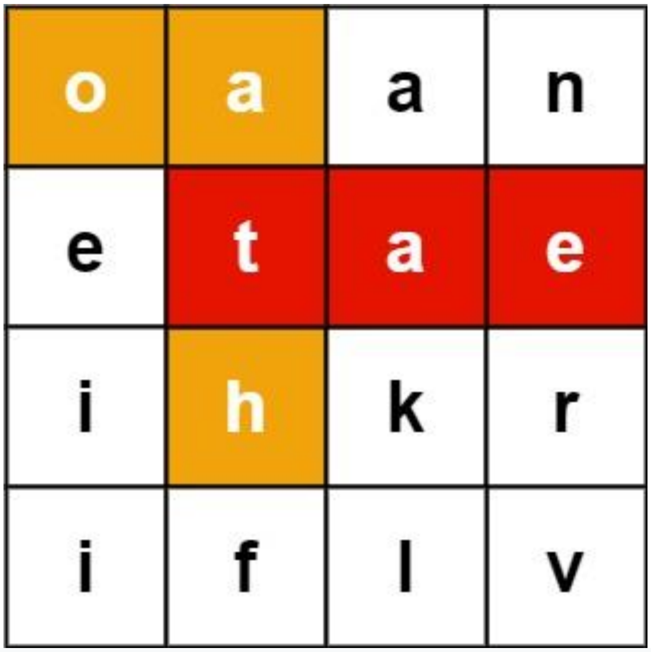
Constraints:

- `0 <= s.length <= 3000`
- `s` consists of digits only.

4. Given an `m x n` `board` of characters and a list of strings `words`, return *all words on the board*.

Each word must be constructed from letters of sequentially adjacent cells, where **adjacent cells** are horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

Example 1:

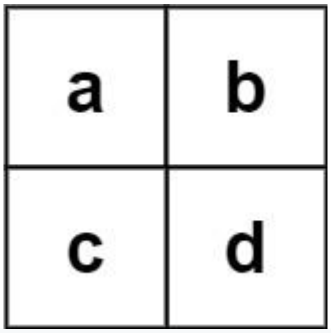


o	a	a	n
e	t	a	e
i	h	k	r
i	f	l	v

Input: board =
[["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],["i","f","l","v"]], words =
["oath","pea","eat","rain"]

Output: ["eat","oath"]

Example 2:



a	b
c	d

Input: board = [["a","b"],["c","d"]], words = ["abcb"]

Output: []

Constraints:

- m == board.length
- n == board[i].length
- 1 <= m, n <= 12
- board[i][j] is a lowercase English letter.
- 1 <= words.length <= 3 * 10^4
- 1 <= words[i].length <= 10

- `words[i]` consists of lowercase English letters.
- All the strings of `words` are unique.

5. You are given an array of strings `arr`. A string `s` is formed by the **concatenation** of a **subsequence** of `arr` that has **unique characters**.

Return *the **maximum** possible length* of `s`.

A **subsequence** is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

```
Input: arr = ["un","iq","ue"]

Output: 4

Explanation: All the valid concatenations are:

- ""

- "un"

- "iq"

- "ue"

- "uniq" ("un" + "iq")

- "ique" ("iq" + "ue")

Maximum length is 4.
```

Example 2:

```
Input: arr = ["cha","r","act","ers"]

Output: 6

Explanation: Possible longest valid concatenations are "chaers" ("cha" + "ers") and
"acters" ("act" + "ers").
```

Example 3:

```
Input: arr = ["abcdefghijklmnopqrstuvwxyz"]

Output: 26

Explanation: The only string in arr has all 26 characters.
```

Example 4:

```
Input: arr = ["aa","bb"]

Output: 0

Explanation: Both strings in arr do not have unique characters, thus there are no
valid concatenations.
```

Constraints:

- `1 <= arr.length <= 16`
- `1 <= arr[i].length <= 26`
- `arr[i]` contains only lowercase English letters.