1.Minimum Path Sum

64. Minimum Path Sum

Solved

Medium | Topics | Companies

Given a `m x n` `grid` filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example 1:

> **Input:** grid = [[1,3,1],[1,5,1],[4,2,1]]
> **Output:** 7
> **Explanation:** Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

Example 2:

> **Input:** grid = [[1,2,3],[4,5,6]]
> **Output:** 12

```cpp
class Solution {
public:
    int minPathSum(std::vector<std::vector<int>>& grid) {
        int a = grid.size();
        int b = grid[0].size();
        for (int i = 0; i < a; ++i) {
            for (int j = 0; j < b; ++j) {
                if (i == 0 && j != 0)
                    grid[i][j] += grid[i][j - 1];
                if (i != 0 && j == 0)
                    grid[i][j] += grid[i - 1][j];
                if (i != 0 && j != 0)
                    grid[i][j] += std::min(grid[i - 1][j], grid[i][j - 1]);
            }
        }
        return grid[a - 1][b - 1];
    }
};
```

Time complexity : O(n^2)

Space complexity : O(1)

2. Validate binary search tree

98. Validate Binary Search Tree

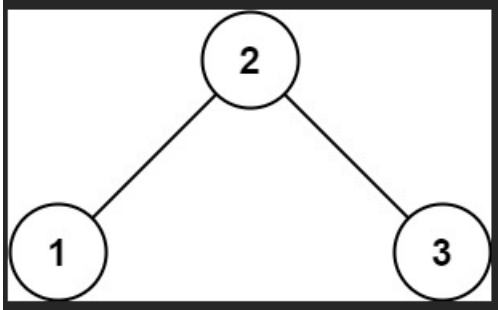
Medium | Topics | Companies

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*

A **valid BST** is defined as follows:

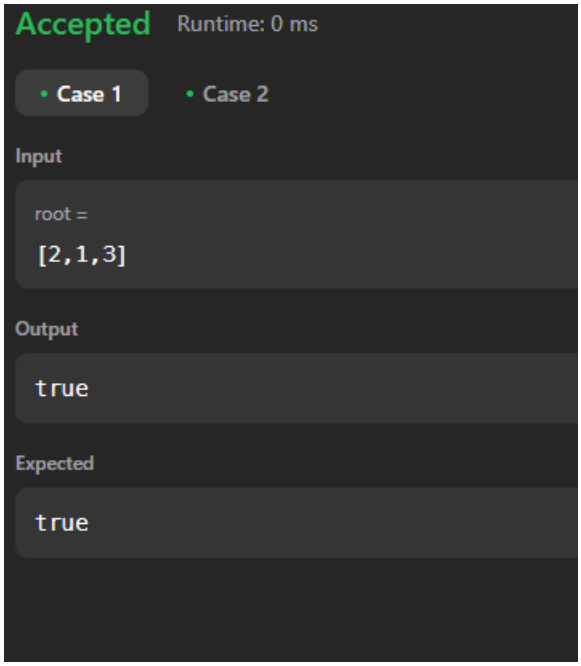
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:




```
Input: root = [2,1,3]
Output: true
```

```cpp
class Solution {
public:
    void helper(TreeNode* root,vector<int>& arr){
        if(!root) return;
        helper(root->left,arr);
        arr.push_back(root->val);
        helper(root->right,arr);
    }
    bool isValidBST(TreeNode* root) {
        vector<int> arr;
        helper(root,arr);
        int n=arr.size();
        for(int i=1;i<n;i++){
            if(arr[i-1]>=arr[i]) return false;
        }
        return true;
    }
};
```



Time complexity : O(n)

Space complexity : O(n)

3. Word ladder

127. Word Ladder

Solved

Hard | Topics | Companies

A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord -> s1 -> s2 -> ... -> sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every `si` for `1 <= i <= k` is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- `sk == endWord`

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return *the **number of words** in the **shortest transformation sequence** from `beginWord` to `endWord`, or `0` if no such sequence exists.*

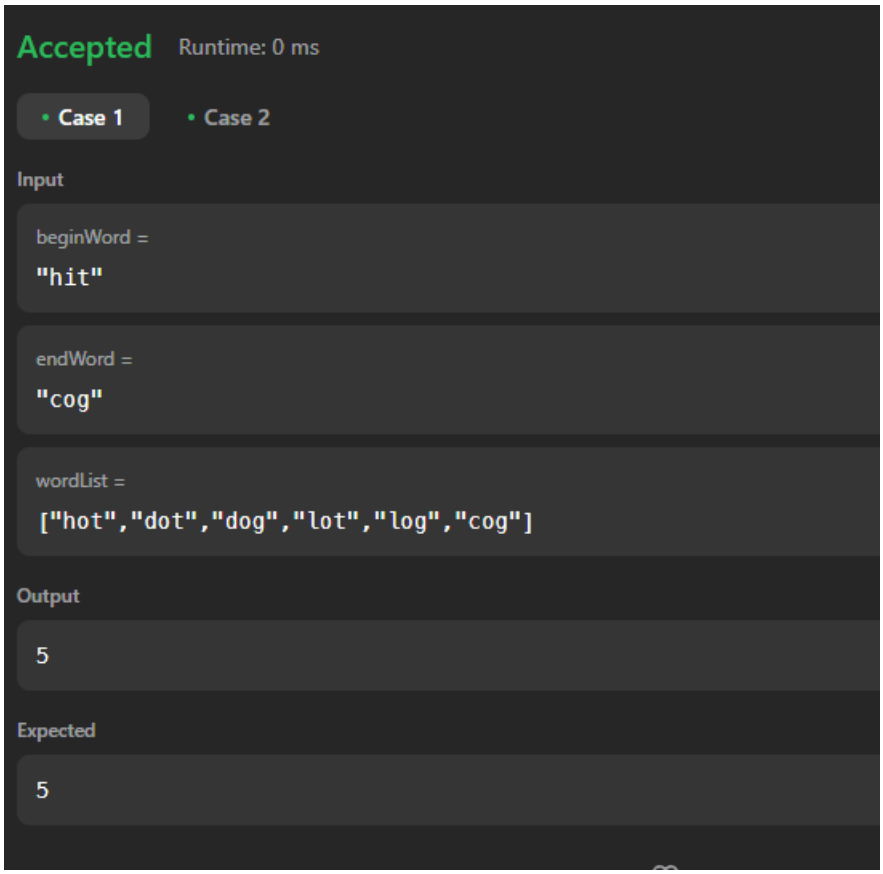
Example 1:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog",
which is 5 words long.
```

Example 2:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation
sequence.
```

```cpp
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        int n=wordList.size();
        set<string> map(wordList.begin(),wordList.end());
        if (map.find(endWord) == map.end()) return 0;
        bool flag=false;
        for(auto i:wordList){
            if(i==endWord) flag=true;
        }
        if(!flag) return 0;
        queue<pair<string,long long>> q;
        q.push({beginWord,1});
        while(!q.empty()){
            auto p=q.front();
            q.pop();
            if(p.first==endWord) return p.second;
            for(int i=0;i<p.first.size();i++){
                char change=p.first[i];
                for(char c='a';c<='z';c++){
                    p.first[i]=c;
                    if(map.find(p.first)!=map.end()){
                        q.push({p.first,p.second+1});
                        map.erase(p.first);
                    }
                }
                p.first[i]=change;
            }
        }
        return 0;
    }
};
```



Time complexity : O(n*m)

Space complexity : O(n)

4. Word ladder II

126. Word Ladder II

Solved

Hard | Topics | Companies

A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord -> s1 -> s2 -> ... -> sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every `si` for `1 <= i <= k` is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- `sk == endWord`

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return *all the* ***shortest transformation sequences*** *from* `beginWord` *to* `endWord`*, or an empty list if no such sequence exists. Each sequence should be returned as a list of the words* `[beginWord, s1, s2, ..., sk]`.

Example 1:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: [["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]
Explanation: There are 2 shortest transformation sequences:
"hit" -> "hot" -> "dot" -> "dog" -> "cog"
"hit" -> "hot" -> "lot" -> "log" -> "cog"
```

Example 2:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: []
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.
```

```cpp
class Solution {
public:
    vector<int>* precursor;
    vector<vector<string>> res;
    bool isANeighbor(string& s1, string& s2) {
        bool hasChanged = false;
        for (int i = 0; i < s1.size(); i++) {
            if (s1[i] != s2[i]) {
                if (hasChanged)
                    return false;
                else
                    hasChanged = true;
            }
        }
        return true;
    }
    void generateRoute(vector<string> right, vector<int>& precursor2, vector<string>& wordList) {
        if (precursor2.size() == 0) {
            res.push_back(right);
            return;
        }
        vector<string> copy;
        for (int i = 0; i < precursor2.size(); i++) {
            copy = right;
            copy.insert(copy.begin(), wordList[precursor2[i]]);
            generateRoute(copy, precursor[precursor2[i]], wordList);
        }

    }
    vector<vector<string>> findLadders(string beginWord, string endWord, vector<string>& wordList) {
        wordList.push_back(beginWord);
        int size = wordList.size();
        vector<int>* neighbors = new vector<int>[size];
        int ewordindex = -1;
        for (int i = 0; i < size; i++) {
            if (wordList[i] == endWord)
                ewordindex = i;
            for (int j = i + 1; j < size; j++) {
                if (isANeighbor(wordList[i],wordList[j])) {
                    neighbors[i].push_back(j);
                    neighbors[j].push_back(i);
```

```
            }
        }
    }
    vector<int> steps(size);
    queue<int> line;
    steps[size - 1] = 1;
    line.push(size - 1);
    precursor = new vector<int>[size];
    while (!line.empty()) {
        int pos = line.front();
        line.pop();
        if (wordList[pos] == endWord)
            break;
        for (int i = 0; i < neighbors[pos].size(); i++) {
            if (steps[neighbors[pos][i]]==0) {
                steps[neighbors[pos][i]] = steps[pos] + 1;
                precursor[neighbors[pos][i]].push_back(pos);
                line.push(neighbors[pos][i]);
            }
            else if (steps[neighbors[pos][i]] == steps[pos] + 1)
                precursor[neighbors[pos][i]].push_back(pos);
        }

    }
    if (ewordindex == -1 || steps[ewordindex] == 0)
        return res;
    vector<string> right{endWord};
    generateRoute(right,precursor[ewordindex] , wordList);
    return res;
}
};
```

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

beginWord =

"hit"

endWord =

"cog"

wordList =

["hot","dot","dog","lot","log","cog"]

Output

[["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]

Expected

[["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]

5. Course schedule

207. Course Schedule

Medium | Topics | Companies | Hint

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have
finished course 1. So it is impossible.
```

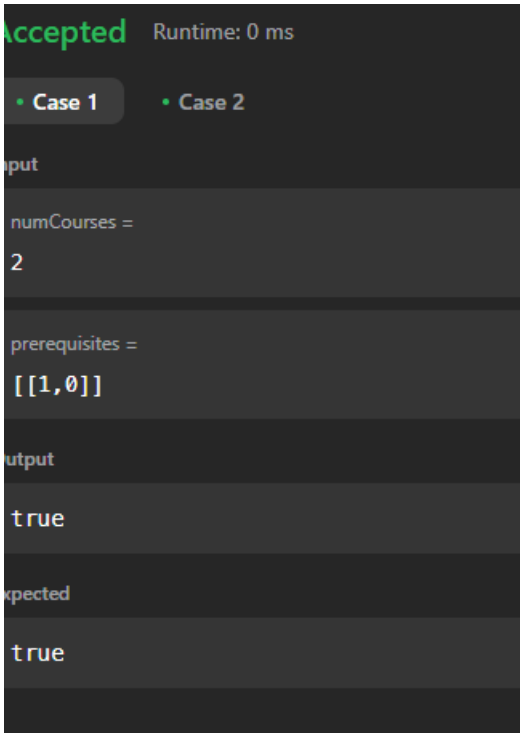
```cpp
class Solution {
public:
    bool canFinish(int n, vector<vector<int>>& prerequisites) {
        vector<int> adj[n];
        vector<int> indegree(n, 0);
        vector<int> ans;

        for(auto x: prerequisites){
            adj[x[0]].push_back(x[1]);
            indegree[x[1]]++;
        }

        queue<int> q;
        for(int i = 0; i < n; i++){
            if(indegree[i] == 0){
                q.push(i);
            }
        }

        while(!q.empty()){
            auto t = q.front();
            ans.push_back(t);
            q.pop();

            for(auto x: adj[t]){
                indegree[x]--;
                if(indegree[x] == 0){
                    q.push(x);
                }
            }
        }
        return ans.size() == n;
    }
};
```


Accepted
Runtime: 0 ms
Case 1
Case 2
Input
numCourses =
2
prerequisites =
[[1,0]]
Output
true
Expected
true