Задача №1

Путь в бинарном дереве — это последовательность узлов, в которой каждая пара соседних узлов в последовательности имеет соединяющее их ребро . Узел может появиться в последовательности не более одного раза . Обратите внимание, что путь не обязательно должен проходить через корень. Сумма путей пути — это сумма значений узлов в пути. Учитывая rootдвоичное дерево, верните максимальную сумму путей любого непустого пути.
Пример 1: Входные данные: root = [1,2,3] Выходные данные: 6 Объяснение: Оптимальный путь — 2 -> 1 -> 3 с суммой путей 2 + 1 + 3 = 6.
Пример 2: Ввод: root = [-10,9,20,null,null,15,7] Выход: 42 Объяснение: Оптимальный путь — 15 -> 20 -> 7 с суммой путей 15 + 20 + 7 = 42.
Ограничения: Количество узлов в дереве находится в диапазоне .[1, 3 * 104] -1000 <= Node.val <= 1000

```
#include <iostream>
#include <algorithm>

using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int maxPathSum(TreeNode* root, int& maxSum) {
    if (root == NULL) {
        return 0;
    }

    int leftSum = maxPathSum(root->left, maxSum);
    int rightSum = maxPathSum(root->right, maxSum);
    int currentSum = root->val + max(0, leftSum) + max(0, rightSum);
    maxSum = max(maxSum, currentSum);
    return root->val + max(0, max(leftSum, rightSum));
}

TreeNode* createTestTree() {
    TreeNode* root = new TreeNode(7);
    root->left = new TreeNode(5);
    root->right = new TreeNode(8);
    return root;
}

int main() {
    TreeNode* root = createTestTree();
    int maxSum = INT_MIN;
    maxPathSum(root, maxSum);
    cout << "Max lenght: " << maxSum << endl;

    return 0;
}
```

```
Max lenght: 20


** Process exited - Return Code: 0 **
```

Задача №2

Вам дан целочисленный массив nums и два целых числа indexDiffи valueDiff.
Найдите пару индексов (i, j)такую, что:
i != j, abs(i - j) <= indexDiff. abs(nums[i] - nums[j]) <= valueDiff, и Возврат,
trueесли такая пара существует или falseнет .
Пример 1:
Ввод:
nums = [1,2,3,1], indexDiff = 3, valueDiff = 0
Выход: true
Объяснение: Мы можем выбрать (i, j) = (0, 3).
Мы удовлетворяем трем условиям:
i != j --> 0 != 3
abs(i - j) <= indexDiff --> abs(0 - 3) <= 3
abs(nums[i] - nums[j]) <= valueDiff --> abs(1 - 1) <= 0

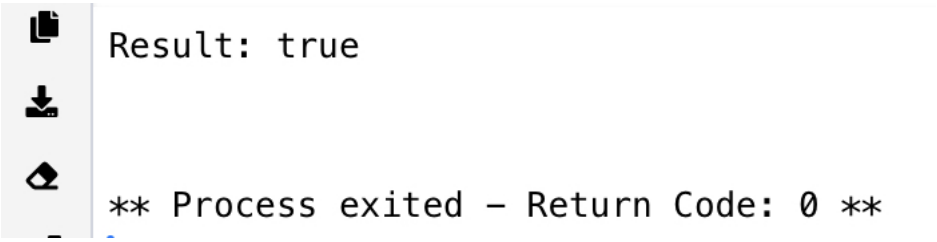
```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

bool findPair(vector<int>& nums, int indexDiff, int valueDiff) {
    for (int i = 0; i < nums.size(); i++) {
        for (int j = i + 1; j < nums.size(); j++) {
            if (abs(i - j) <= indexDiff && abs(nums[i] - nums[j]) <= valueDiff) {
                return true;
            }
        }
    }
    return false;
}

int main() {
    vector<int> nums = {1, 2, 3, 1};
    int indexDiff = 3;
    int valueDiff = 0;

    bool result = findPair(nums, indexDiff, valueDiff);
    cout << boolalpha << "Result: " << result << endl;

    return 0;
}
```



Задача №4

Загадка с n ферзями — это задача о том, как разместить nферзей на n x nшахматной доске так, чтобы никакие два ферзя не атаковали друг друга. Учитывая целое число n, верните все различные решения головоломки с n ферзями . Вы можете вернуть ответ в любом порядке .
Каждое решение содержит отдельную конфигурацию доски для размещения n ферзей, где 'Q'и '.'оба обозначают ферзя и пустое место соответственно.
Пример 1:
Вход: n = 4
Выход: [[".Q..","...Q","Q...","..Q."],["..Q.","Q.. . ","...Q",".Q.."]]
Пояснение: Существует два различных решения головоломки с четырьмя ферзями, как показано выше.

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Solution {
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> results;
        vector<string> board(n, string(n, '.'));
        vector<int> queens(n, -1);
        backtrack(results, board, queens, 0, n);
        return results;
    }

private:
    void backtrack(vector<vector<string>>& results, vector<string>& board, vector<int>& queens, int
row, int n) {
        if (row == n) {
            results.push_back(board);
            return;
        }

        for (int col = 0; col < n; col++) {
            if (isValid(board, queens, row, col, n)) {
                board[row][col] = 'Q';
                queens[row] = col;
                backtrack(results, board, queens, row + 1, n);
                board[row][col] = '.';
                queens[row] = -1;
            }
        }
    }

    bool isValid(vector<string>& board, vector<int>& queens, int row, int col, int n) {
        for (int i = 0; i < row; i++) {
            if (board[i][col] == 'Q') {
                return false;
            }
        }

        for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }

        for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }

        return true;
    }
};

int main() {
    int n = 4;
    Solution solution;
    vector<vector<string>> results = solution.solveNQueens(n);

    cout << "Solving puzzles with " << n << " queens:" << endl;
    for (const vector<string>& result : results) {
        for (const string& row : result) {
            cout << row << endl;
        }
        cout << endl;
    }

    return 0;
}
```

```
Solving puzzles with 4 queens:
.Q..
...Q
Q...
..Q.

..Q.
Q...
...Q
.Q..
```

Задача №5

Учитывая rows x cols двоичный файл matrix, заполненный символами 0's и 1's, найдите самый большой прямоугольник, содержащий только 1's, и верните его площадь .
Пример 1:
Ввод:
матрица = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

```cpp
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

int largestRectangleArea(vector<int>& heights) {
    stack<int> stk;
    heights.push_back(0);
    int maxArea = 0;

    for (int i = 0; i < heights.size(); i++) {
        while (!stk.empty() && heights[i] < heights[stk.top()]) {
            int height = heights[stk.top()];
            stk.pop();
            int width = (stk.empty() ? i : i - stk.top() - 1);
            maxArea = max(maxArea, height * width);
        }
        stk.push(i);
    }

    return maxArea;
}

int maximalRectangle(vector<vector<char>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) {
        return 0;
    }

    int rows = matrix.size();
    int cols = matrix[0].size();
    vector<int> heights(cols, 0);
    int maxArea = 0;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (matrix[i][j] == '1') {
                heights[j]++;
            } else {
                heights[j] = 0;
            }
        }
        maxArea = max(maxArea, largestRectangleArea(heights));
    }

    return maxArea;
}

int main() {
    vector<vector<char>> matrix = {
        {'1', '0', '1', '0', '0'},
        {'1', '0', '1', '1', '1'},
        {'1', '1', '1', '1', '1'},
        {'1', '0', '0', '1', '0'}
    };

    int result = maximalRectangle(matrix);
    cout << "The max area of the rectangle: " << result << endl;

    return 0;
}
```

```
The max area of the rectangle: 6


** Process exited - Return Code: 0 **
```

Задача №6

Вам дан массив prices, в котором prices[i]указана цена данной акции на текущий день.ith
Найдите максимальную прибыль, которую вы можете получить. Вы можете совершить не более двух транзакций .
Примечание. Вы не можете совершать несколько транзакций одновременно (т. е. вы должны продать акции, прежде чем купить их снова).
Пример 1:
Входные данные: цены = [3,3,5,0,0,3,1,4]
Выходные данные: 6
Объяснение: Покупайте в день 4 (цена = 0) и продавайте в день 6 (цена = 3), прибыль = 3- 0 = 3.
Затем купите в день 7 (цена = 1) и продайте в день 8 (цена = 4), прибыль = 4 -1 = 3.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int maxProfit(vector<int>& prices) {
    int buy1 = INT_MAX;
    int buy2 = INT_MAX;
    int sell1 = 0;
    int sell2 = 0;

    for (int price : prices) {
        buy1 = min(buy1, price);
        sell1 = max(sell1, price - buy1);
        buy2 = min(buy2, price - sell1);
        sell2 = max(sell2, price - buy2);
    }

    return sell2;
}

int main() {
    vector<int> prices = {3, 3, 5, 0, 0, 3, 1, 4};
    int result = maxProfit(prices);
    cout << "Max profit: " << result << endl;

    return 0;
}
```

```
Max profit: 6


** Process exited - Return Code: 0 **
```