71. Simplify Path

C++ solution:

```
class Solution {
public:
    string simplifyPath(string path) {

        string result;

        deque<string> directories;
        string current_directory_name;

        int n = path.size(), count = 0;

        for (int i = 0; i < n; ++i)
        {
            if (path[i] == '/')
            {
                while (i < n - 1 && path[i + 1] == '/')
                {
                    ++i;
                }

                if (i == n - 1)
                {
                    break;
                }
            }
            else if (path[i] == '.')
            {
                count = 0;
                current_directory_name = "/";
                while (i < n && path[i] != '/')
                {
                    current_directory_name.insert(current_directory_name.end(), path[i
]);
                    ++count;
                    ++i;
                }

                if (count == 2 && path[i - 1] == '.')
                {
                    if (!directories.empty() && directories.back() != "/..")
                    {
                        directories.pop_back();
                    }
                    else
```

```
                    {
                        directories.insert(directories.end(), "/..");
                    }
                }
                else if (count == 1)
                {
                    continue;
                }
                else
                {
                    directories.insert(directories.end(), current_directory_name);
                }
            }
            else
            {
                current_directory_name = "/";
                while (i < n && path[i] != '/' && path[i] != '.')
                {
                    current_directory_name.insert(current_directory_name.end(), path[i
]);
                    ++i;
                }
                --i;
                directories.insert(directories.end(), current_directory_name);
            }
        }

        for (string directory : directories)
        {
            if (directory != "/..")
            {
                result += directory;
            }
        }

        if (result == "")
        {
            result += "/";
        }

        return result;
    }
};
```

[72. Edit Distance]

C++ solution:

```
class Solution {
public:
    int minDistance(string word1, string word2) {

        int n1 = word1.length(), n2 = word2.length();

        if (n1 == 0)
        {
            return n2;
        }

        if (n2 == 0)
        {
            return n1;
        }

        int minEditDistance[n1 + 1][n2 + 1];

        for (int i = 0; i <= n1; ++i)
        {
            minEditDistance[i][0] = i;
        }

        for (int i = 1; i <= n2; ++i)
        {
            minEditDistance[0][i] = i;
        }

        for (int i = 1; i <= n1; ++i)
        {
            for (int j = 1; j <= n2; ++j)
            {
                if (word1[i - 1] == word2[j - 1])
                {
                    minEditDistance[i][j] = minEditDistance[i - 1][j - 1];
                }
                else
                {

                    minEditDistance[i][j] = min({ minEditDistance[i - 1][j - 1] + 1, m
inEditDistance[i - 1][j] + 1, minEditDistance[i][j - 1] + 1 });
                }
            }
        }

        return minEditDistance[n1][n2];
    }
};
```

[73. Set Matrix Zeroes]

C++ solution:

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {

        int r = matrix.size(), c = matrix[0].size(), i = 0, j = 0;
        bool firstRowHasZero = false, firstColHasZero = false;

        for (i = 0; i < r; ++i)
        {
            for (j = 0; j < c; ++j)
            {
                if (matrix[i][j] == 0)
                {
                    if (i == 0)
                    {
                        firstRowHasZero = true;
                    }
                    if (j == 0)
                    {
                        firstColHasZero = true;
                    }
                    matrix[i][0] = matrix[0][j] = 0;
                }
            }
        }

        for (i = 1; i < r; ++i)
        {
            for (j = 1; j < c; ++j)
            {
                if (matrix[i][0] == 0 || matrix[0][j] == 0)
                {
                    matrix[i][j] = 0;
                }
            }
        }

        if (firstRowHasZero)
        {
            for (j = 0; j < c; ++j)
            {
                matrix[0][j] = 0;
            }
        }
```

```
        if (firstColHasZero)
        {
            for (i = 0; i < r; ++i)
            {
                matrix[i][0] = 0;
            }
        }

    }
};
```

74. Search a 2D Matrix

C++ solution:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int r = matrix.size();
        if (r == 0)
        {
            return false;
        }

        int c = matrix[0].size(), left = 0, right = r * c - 1, mid = 0, value = 0;

        while (left <= right)
        {
            mid = (left + right) / 2;

            value = matrix[mid / c][mid % c];

            if (value == target)
            {
                return true;
            }

            if (value < target)
            {
                left = mid + 1;
            }
            else
            {
                right = mid - 1;
            }
        }

        return false;
    }
};
```

[75. Sort Colors]

C++ solution:

```
class Solution {
public:
    void sortColors(vector<int>& nums) {

        int n = nums.size(), lastZero = -1, firstTwo = n;

        while (lastZero < n - 1 && nums[lastZero + 1] == 0)
        {
            ++lastZero;
        }

        while (firstTwo > 0 && nums[firstTwo - 1] == 2)
        {
            --firstTwo;
        }

        int i = lastZero + 1;

        while (i < firstTwo)
        {
            if (nums[i] == 0)
            {
                if (i > lastZero + 1)
                {
                    nums[i] = nums[lastZero + 1];
                    nums[lastZero + 1] = 0;
                }
                ++i;
                ++lastZero;
            }
            else if (nums[i] == 2)
            {
                if (i < firstTwo - 1)
                {
                    nums[i] = nums[firstTwo - 1];
                    nums[firstTwo - 1] = 2;
                }
                --firstTwo;
            }
            else
            {
                ++i;
            }
        }
    }
};
```

76. Minimum Window Substring

C++ solution:

```
class Solution {
public:
    string minWindow(string s, string t) {

        int ns = s.length(), nt = t.length();
        unordered_map<char, int> original_char_numbers, current_char_numbers;

        for (char ch : t)
        {
            if (original_char_numbers.find(ch) != original_char_numbers.end())
            {
                ++(original_char_numbers[ch]);
            }
            else
            {
                original_char_numbers.insert(pair<char, int>(ch, 1));
                current_char_numbers.insert(pair<char, int>(ch, 0));
            }
        }

        int count = 0;
        bool valid = false;
        deque<int> current_indexes;
        int start = 0, current_difference = 0, min_difference = ns;

        for (int i = 0; i < ns; ++i)
        {
            if (original_char_numbers.find(s[i]) != original_char_numbers.end())
            {
                if (current_char_numbers[s[i]] < original_char_numbers[s[i]])
                {
                    ++count;
                    if (count == nt)
                    {
                        min_difference = i - current_indexes.front();
                        start = current_indexes.front();
                        valid = true;
                    }
                }
                ++(current_char_numbers[s[i]]);

                current_indexes.push_back(i);

                if (valid)
                {
```

```
                    while (current_char_numbers[s[current_indexes.front()]] > original
_char_numbers[s[current_indexes.front()]])
                    {
                        --(current_char_numbers[s[current_indexes.front()]]);
                        current_indexes.pop_front();
                    }

                    current_difference = current_indexes.back() - current_indexes.fron
t();
                    if (current_difference < min_difference)
                    {
                        min_difference = current_difference;
                        start = current_indexes.front();
                    }
                }
            }
        }

        if (valid)
        {
            return s.substr(start, min_difference + 1);
        }

        return "";
    }
};
```

77. Combinations

C++ solution:

```
class Solution {
public:
    vector<vector<int>> combine(int n, int k) {
        vector<vector<int>> result;
        vector<int> current;
        select(n, 1, k, result, current);
        return result;
    }

    void select(int n, int start, int number, vector<vector<int>>& result, vector<int>
& current)
    {
        if (number == 0)
        {
            result.insert(result.end(), current);
            return;
        }

        if (number > n - start + 1)
        {
            return;
        }

        for (int i = start; i <= n - number + 1; ++i)
        {
            current.insert(current.end(), i);
            select(n, i + 1, number - 1, result, current);
            current.erase(current.begin() + current.size() - 1);
        }
    }
};
```

78. Subsets

C++ solution:

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {

        return subsetsOfRange(nums, 0, nums.size() - 1);
    }

    vector<vector<int>> subsetsOfRange(vector<int>& nums, int start, int end) {
        vector<vector<int>> result;
        vector<int> empty_set;
        result.insert(result.end(), empty_set);

        if (start > end)
        {
            return result;
        }

        if (start == end)
        {
            result.insert(result.end(), vector<int>(1, nums[start]));
            return result;
        }

        int mid = (start + end) / 2;

        vector<vector<int>> result_left = subsetsOfRange(nums, start, mid);
        vector<vector<int>> result_right = subsetsOfRange(nums, mid + 1, end);

        vector<int> vector_current;
        for (auto vector_left : result_left)
        {
            for (auto vector_right : result_right)
            {
                if (vector_left.empty() && vector_right.empty())
                {
                    continue;
                }
                vector_current = vector_left;
                vector_current.insert(vector_current.end(), vector_right.begin(), vect
or_right.end());
                result.insert(result.end(), vector_current);
            }
        }

        return result;
    }
};
```

[79. Word Search]

C++ solution:

```
class Solution {
public:
    bool exist(vector<vector<char>>& board, string word) {

        if (word.empty())
        {
            return true;
        }

        int rows = board.size();
        if (rows == 0)
        {
            return false;
        }

        int cols = board[0].size();
        if (cols == 0)
        {
            return false;
        }

        vector<vector<int>> visited(rows, vector<int>(cols, 0));

        int length = word.size();

        for (int r = 0; r < rows; ++r)
        {
            for (int c = 0; c < cols; ++c)
            {
                if (valid(board, word, visited, rows, cols, length, r, c, 0))
                {
                    return true;
                }
            }
        }

        return false;
    }

    bool valid(vector<vector<char>>& board, string word, vector<vector<int>>& visited,
 int rows, int cols, int length, int r, int c, int start_index)
    {
        if (length == 0 || start_index == length)
        {
```

```
            return true;
        }

        if (board[r][c] != word[start_index])
        {
            return false;
        }

        if (start_index == length - 1)
        {
            return true;
        }

        visited[r][c] = 1;

        for (int i = 0; i < 4; ++i)
        {
            switch (i)
            {
                case 0:
                    --r;
                    break;
                case 1:
                    ++r;
                    break;
                case 2:
                    --c;
                    break;
                case 3:
                    ++c;
                    break;
            }

            if (r == -1 || r == rows || c == -1 || c == cols || visited[r][c])
            {
                switch (i)
                {
                    case 0:
                        ++r;
                        break;
                    case 1:
                        --r;
                        break;
                    case 2:
                        ++c;
                        break;
                    case 3:
                        --c;
```

```
                        break;
                }
                continue;
            }

            if (valid(board, word, visited, rows, cols, length, r, c, start_index + 1)
)
            {
                return true;
            }

            visited[r][c] = 0;
            switch (i)
            {
                case 0:
                    ++r;
                    break;
                case 1:
                    --r;
                    break;
                case 2:
                    ++c;
                    break;
                case 3:
                    --c;
                    break;
            }
        }

        visited[r][c] = 0;

        return false;
    }
};
```

80. Remove Duplicates from Sorted Array II

C++ solution:

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        int n = nums.size();

        if (n < 3)
        {
            return n;
        }

        int current = 1;

        for (int i = 2; i < n; ++i)
        {
            if (nums[i] > nums[current - 1])
            {
                ++current;
                nums[current] = nums[i];
            }
        }

        return current + 1;
    }
};
```