Dynamic Programming

1. Longest Palindromic Substring.

Code:

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

string longestPalindrome(const string& s) {
    int n = s.length();
    if (n <= 1) return s;

    int start = 0, maxLength = 1;
    for (int i = 0; i < n; i++) {
        int left = i, right = i;
        while (right < n - 1 && s[right] == s[right + 1]) right++;
        i = right;

        while (left > 0 && right < n - 1 && s[left - 1] == s[right + 1]) {
            left--;
            right++;
        }

        if (right - left + 1 > maxLength) {
            start = left;
            maxLength = right - left + 1;
        }
    }

    return s.substr(start, maxLength);
}

int main() {
    string s = "babad";
    string result = longestPalindrome(s);
    cout << "Longest Palindromic Substring: " << result << endl;
    return 0;
}
```
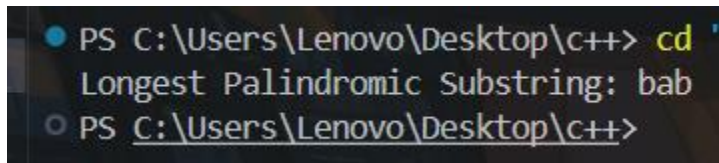
Output:

2. Generate Parentheses

Code:

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

void generateParenthesesHelper(int open, int close, string current, vector<string>& result) {
    if (open == 0 && close == 0) {
        result.push_back(current);
        return;
    }
    if (open > 0) {
        generateParenthesesHelper(open - 1, close, current + "(", result);
    }
    if (close > open) {
        generateParenthesesHelper(open, close - 1, current + ")", result);
    }
}

vector<string> generateParentheses(int n) {
    vector<string> result;
    generateParenthesesHelper(n, n, "", result);
    return result;
}
int main() {
    int n = 3;
    vector<string> result = generateParentheses(n);
    cout << "Generated Parentheses Combinations: [";
    for (size_t i = 0; i < result.size(); i++) {
        cout << "\"" << result[i] << "\"";
        if (i < result.size() - 1) cout << ",";
    }
    cout << "]" << endl;
    return 0;
}
```

Output:

3. Jump Game

Code:

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

bool canJump(const vector<int>& nums) {
    int maxReach = 0;
    int n = nums.size();

    for (int i = 0; i < n; i++) {
        if (i > maxReach) return false;
        maxReach = max(maxReach, i + nums[i]);
    }

    return maxReach >= n - 1;
}

int main() {
    vector<int> nums = {2, 3, 1, 1, 4};
    bool result = canJump(nums);
    cout << (result ? "true" : "false") << endl;
    return 0;
}
```

Output:

4. Minimum Path Sum

Code:

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

```cpp
int minPathSum(vector<vector<int>>& grid) {
    int m = grid.size();
    int n = grid[0].size();

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (i == 0 && j == 0) continue; // Starting point
            if (i == 0) grid[i][j] += grid[i][j - 1]; // First row
            else if (j == 0) grid[i][j] += grid[i - 1][j]; // First column
            else grid[i][j] += min(grid[i - 1][j], grid[i][j - 1]); // General case
        }
    }

    return grid[m - 1][n - 1];
}

int main() {
    vector<vector<int>> grid = {
        {1, 3, 1},
        {1, 5, 1},
        {4, 2, 1}
    };
    int result = minPathSum(grid);
    cout << "Minimum Path Sum: " << result << endl;
    return 0;
}
```
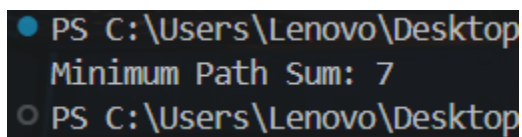
Output:

```
 PS C:\Users\Lenovo\Desktop
  Minimum Path Sum: 7
 PS C:\Users\Lenovo\Desktop
```

5. **Given an integer n, return the least number of perfect square numbers that sum to n.**

Code:

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

```cpp
int numSquares(int n) {
    vector<int> dp(n + 1, INT_MAX);
    dp[0] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j * j <= i; j++) {
            dp[i] = min(dp[i], dp[i - j * j] + 1);
        }
    }

    return dp[n];
}

int main() {
    int n = 12;    int result = numSquares(n);
    cout << "Least number of perfect squares for " << n << ": " << result << endl;
    return 0;
}
```
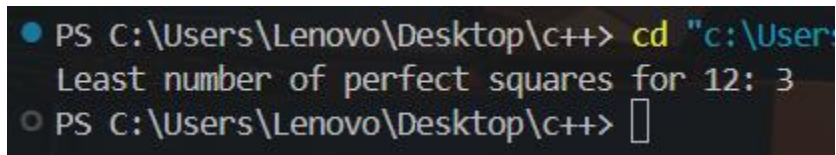
Output:

Backtracking

1. Letter Combinations of a Phone Number
Code:

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

void backtrack(const string& digits, int index, string current, vector<string>& result, const
vector<string>& mapping) {
    if (index == digits.size()) {
        result.push_back(current);
        return;
    }

    int digit = digits[index] - '0';
    for (char c : mapping[digit]) {
        backtrack(digits, index + 1, current + c, result, mapping);
    }
}

vector<string> letterCombinations(string digits) {
    if (digits.empty()) return {};

    vector<string> mapping = {
        "", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"
    };
    vector<string> result;
    backtrack(digits, 0, "", result, mapping);
    return result;
}

int main() {
    string digits = "23";
    vector<string> result = letterCombinations(digits);

    cout << "Letter combinations: [";
    for (size_t i = 0; i < result.size(); i++) {
        cout << "\"" << result[i] << "\"";
        if (i < result.size() - 1) cout << ",";
    }
    cout << "]" << endl;

    return 0;
}
```

Output:

2. Combinations
Code:

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

void backtrack(int start, int k, vector<int>& current, vector<vector<int>>& result) {
    if (current.size() == k) {
        result.push_back(current);
        return;
    }

    for (int i = start; i <= 9; i++) {
        current.push_back(i);
        backtrack(i + 1, k, current, result);
        current.pop_back();
    }
}

vector<vector<int>> combine(int n, int k) {
    vector<vector<int>> result;
    vector<int> current;
    backtrack(1, k, current, result);
    return result;
}

int main() {
    int n = 4, k = 2;
    vector<vector<int>> result = combine(n, k);
    cout << "Combinations: [";
    for (size_t i = 0; i < result.size(); i++) {
        cout << "[";
        for (size_t j = 0; j < result[i].size(); j++) {
            cout << result[i][j];
            if (j < result[i].size() - 1) cout << ",";
        }
        cout << "]";
        if (i < result.size() - 1) cout << ",";
    }
    cout << "]" << endl;
```

```
    return 0;
}
```

Output:

3. Combination Sum
Code:

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

void backtrack(const vector<int>& candidates, int target, int start, vector<int>& current,
vector<vector<int>>& result) {
    if (target == 0) {
        result.push_back(current);
        return;
    }
    for (int i = start; i < candidates.size(); i++) {
        if (candidates[i] > target) continue;
        current.push_back(candidates[i]);
        backtrack(candidates, target - candidates[i], i, current, result);  // Allow reuse of the same
element
        current.pop_back();
    }
}

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> result;
    vector<int> current;
    backtrack(candidates, target, 0, current, result);
    return result;
}

int main() {
    vector<int> candidates = {2, 3, 6, 7};    int target = 7; // Static target
    vector<vector<int>> result = combinationSum(candidates, target);

    cout << "Combinations that sum to " << target << ": [";
    for (size_t i = 0; i < result.size(); i++) {
        cout << "[";
        for (size_t j = 0; j < result[i].size(); j++) {
            cout << result[i][j];
            if (j < result[i].size() - 1) cout << ",";
        }
        cout << "]";
```
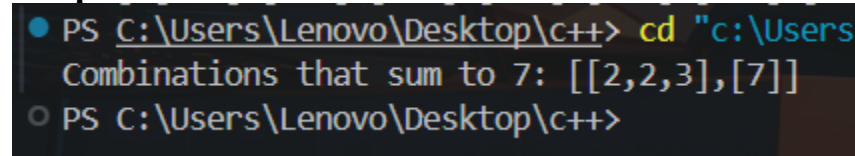
```cpp
        if (i < result.size() - 1) cout << ",";
    }
    cout << "]" << endl;
    return 0;
}
```

Output:


```
PS C:\Users\Lenovo\Desktop\c++> cd "c:\Users
Combinations that sum to 7: [[2,2,3],[7]]
PS C:\Users\Lenovo\Desktop\c++>
```

4. Generate Parentheses

Code:

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

void backtrack(int open, int close, string current, vector<string>& result) {
    if (open == 0 && close == 0) {
        result.push_back(current);
        return;
    }
    if (open > 0) {
        backtrack(open - 1, close, current + "(", result);
    }
    if (close > open) {
        backtrack(open, close - 1, current + ")", result);
    }
}

vector<string> generateParentheses(int n) {
    vector<string> result;
    backtrack(n, n, "", result);
    return result;
}

int main() {
    int n = 3; // Static input
    vector<string> result = generateParentheses(n);

    cout << "Generated Parentheses Combinations: [";
    for (size_t i = 0; i < result.size(); i++) {
```

```
        cout << "\"" << result[i] << "\"";
        if (i < result.size() - 1) cout << ",";
    }
    cout << "]" << endl;

    return 0;
}
```
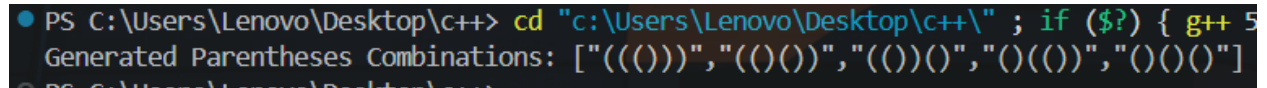
Output:


```
PS C:\Users\Lenovo\Desktop\c++> cd "c:\Users\Lenovo\Desktop\c++\" ; if ($?) { g++ 5
Generated Parentheses Combinations: ["((()))","(()())","(())()","()(())","()()()"]
```

5. Word Search

Code:

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

bool backtrack(vector<vector<char>>& board, string& word, int index, int row, int col) {
    if (index == word.size()) return true;
    if (row < 0 || row >= board.size() || col < 0 || col >= board[0].size() || board[row][col] !=
word[index]) {
        return false;
    }
    char temp = board[row][col];
    board[row][col] = '#';
    bool found = backtrack(board, word, index + 1, row + 1, col) ||
            backtrack(board, word, index + 1, row - 1, col) ||
            backtrack(board, word, index + 1, row, col + 1) ||
            backtrack(board, word, index + 1, row, col - 1);
    board[row][col] = temp;
    return found;
}

bool exist(vector<vector<char>>& board, string word) {
    int m = board.size();
    int n = board[0].size();
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (board[i][j] == word[0] && backtrack(board, word, 0, i, j)) {
                return true;
            }
        }
```

```cpp
        }
    }
    return false;
}

int main() {
    vector<vector<char>> board = {
        {'A', 'B', 'C', 'E'},
        {'S', 'F', 'C', 'S'},
        {'A', 'D', 'E', 'E'}
    };
    string word = "ABCCED";
    bool result = exist(board, word);
    cout << "Word exists in the board: " << (result ? "true" : "false") << endl;
    return 0;
}
```
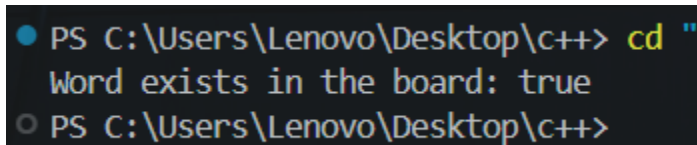
Output: