DYNAMIC PROGRAMMING

Longest increasing subsequence

Longest Increasing Subsequence

https://leetcode.com/problems/longest-increasing-subsequence/

https://leetcode.com/problems/largest-divisible-subset/

https://leetcode.com/problems/russian-doll-envelopes/

https://leetcode.com/problems/maximum-length-of-pair-chain/

https://leetcode.com/problems/number-of-longest-increasing-subsequence/

https://leetcode.com/problems/delete-and-earn/

https://leetcode.com/problems/longest-string-chain/

```cpp
class Solution {
private:
    unordered_map<int, int> memo;

    int lengthOfLISRecursive(vector<int>& nums, int prevIndex, int currentIndex) {
        if (currentIndex == nums.size()) {
            return 0;
        }

        string key = to_string(prevIndex) + "_" + to_string(currentIndex);
        if (memo.find(key) != memo.end()) {
            return memo[key];
        }

        int taken = 0;
        if (prevIndex < 0 || nums[currentIndex] > nums[prevIndex]) {
            taken = 1 + lengthOfLISRecursive(nums, currentIndex, currentIndex + 1);
        }

        int notTaken = lengthOfLISRecursive(nums, prevIndex, currentIndex + 1);

        memo[key] = max(taken, notTaken);
        return memo[key];
    }

public:
    int lengthOfLIS(vector<int>& nums) {
        return lengthOfLISRecursive(nums, -1, 0);
    }
};
```

Longest common subsequence

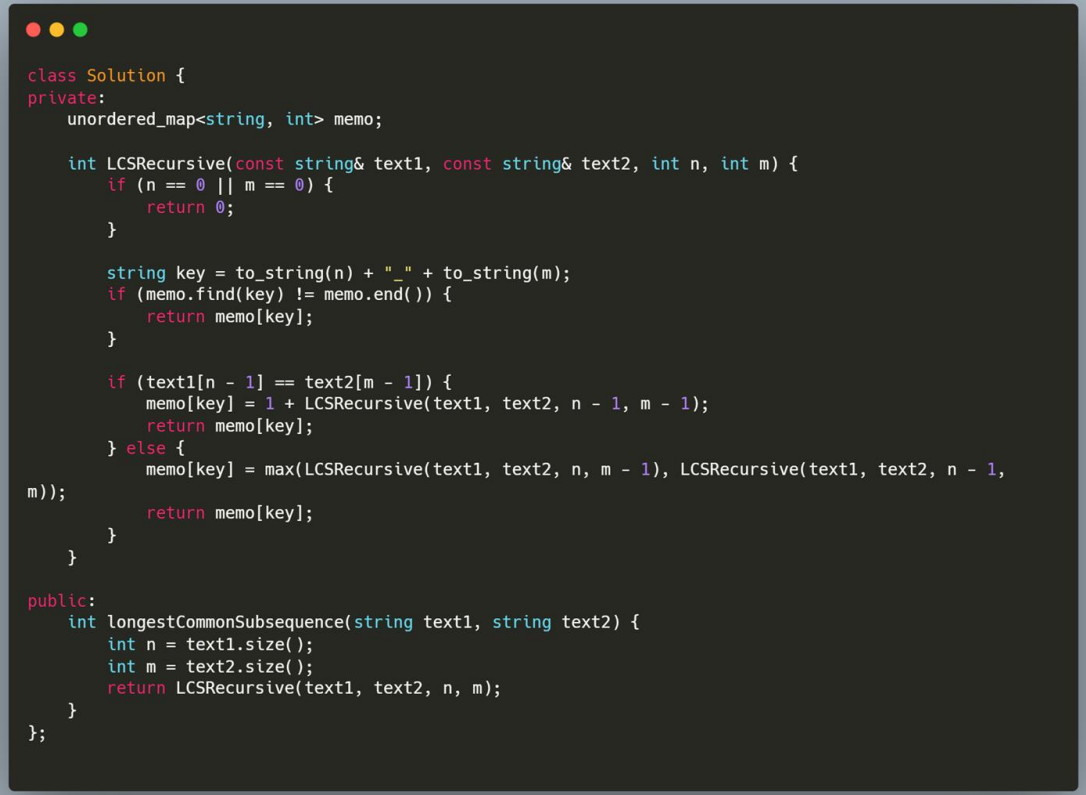
Longest Common Subsequence

https://leetcode.com/problems/longest-common-subsequence/

https://leetcode.com/problems/edit-distance/

https://leetcode.com/problems/distinct-subsequences/

https://leetcode.com/problems/minimum-ascii-delete-sum-for-two-strings/



```cpp
class Solution {
private:
    unordered_map<string, int> memo;

    int LCSRecursive(const string& text1, const string& text2, int n, int m) {
        if (n == 0 || m == 0) {
            return 0;
        }

        string key = to_string(n) + "_" + to_string(m);
        if (memo.find(key) != memo.end()) {
            return memo[key];
        }

        if (text1[n - 1] == text2[m - 1]) {
            memo[key] = 1 + LCSRecursive(text1, text2, n - 1, m - 1);
            return memo[key];
        } else {
            memo[key] = max(LCSRecursive(text1, text2, n, m - 1), LCSRecursive(text1, text2, n - 1,
m));
            return memo[key];
        }
    }

public:
    int longestCommonSubsequence(string text1, string text2) {
        int n = text1.size();
        int m = text2.size();
        return LCSRecursive(text1, text2, n, m);
    }
};
```

Coin change

Coin Change:

https://leetcode.com/problems/coin-change/

https://leetcode.com/problems/coin-change-2/

https://leetcode.com/problems/combination-sum-iv/

https://leetcode.com/problems/perfect-squares/

https://leetcode.com/problems/minimum-cost-for-tickets/

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

class Solution {
private:
    int coinChangeRecursive(const vector<int>& coins, int amount, int n, vector<int>& memo) {
        if (amount == 0) return 0; // Base case: amount is 0
        if (memo[amount] != -1) return memo[amount]; // Check memoization

        int minCoins = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (coins[i] <= amount) {
                int res = coinChangeRecursive(coins, amount - coins[i], n, memo);
                if (res != -1) minCoins = min(minCoins, res + 1);
            }
        }

        memo[amount] = (minCoins == INT_MAX) ? -1 : minCoins; // Save the result in memoization array
        return memo[amount];
    }

public:
    int coinChange(vector<int>& coins, int amount) {
        int n = coins.size();
        if (n == 0) return -1;

        vector<int> memo(amount + 1, -1); // Initialize memoization array
        return coinChangeRecursive(coins, amount, n, memo);
    }
};
```

Matrix multiply

Matrix multiplication:

https://leetcode.com/problems/minimum-score-triangulation-of-polygon/

https://leetcode.com/problems/minimum-cost-tree-from-leaf-values/

https://leetcode.com/problems/burst-balloons/

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

class Solution {
private:
    int minScoreTriangulationRecursive(vector<int>& A, int i, int j, vector<vector<int>>& memo) {
        if (j - i < 2) return 0; // Base case: no triangle can be formed
        if (memo[i][j] != -1) return memo[i][j]; // Check memoization

        int minScore = INT_MAX;
        for (int k = i + 1; k < j; k++) {
            // Calculate score for the triangle formed by points i, k, j and
            // add it to the min score of two sub-polygons (i, k) and (k, j)
            int score = A[i] * A[k] * A[j] +
                        minScoreTriangulationRecursive(A, i, k, memo) +
                        minScoreTriangulationRecursive(A, k, j, memo);
            minScore = min(minScore, score);
        }

        memo[i][j] = minScore; // Save the result in memoization array
        return minScore;
    }

public:
    int minScoreTriangulation(vector<int>& A) {
        int n = A.size();
        vector<vector<int>> memo(n, vector<int>(n, -1)); // Initialize memoization array
        return minScoreTriangulationRecursive(A, 0, n - 1, memo);
    }
};
```

Matrix 2d array

Matrix/2D Array:

https://leetcode.com/problems/matrix-block-sum/

https://leetcode.com/problems/range-sum-query-2d-immutable/

https://leetcode.com/problems/dungeon-game/

https://leetcode.com/problems/triangle/

https://leetcode.com/problems/maximal-square/

https://leetcode.com/problems/minimum-falling-path-sum/

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

class Solution {
    int m, n, K;
    vector<vector<int>> mat;
    vector<vector<vector<vector<int>>>> memo;

    // Recursive function to compute the block sum for a given cell (i, j)
    int blockSum(int i, int j, int r1, int c1) {
        if (r1 > min(m - 1, i + K) || c1 > min(n - 1, j + K)) return 0; // Base condition
        if (memo[i][j][r1][c1] != -1) return memo[i][j][r1][c1]; // Check memoization

        // Compute the block sum recursively
        int sum = mat[r1][c1] +
                  blockSum(i, j, r1 + 1, c1) +
                  blockSum(i, j, r1, c1 + 1) -
                  blockSum(i, j, r1 + 1, c1 + 1);

        memo[i][j][r1][c1] = sum;
        return sum;
    }

public:
    vector<vector<int>> matrixBlockSum(vector<vector<int>>& mat, int K) {
        this->mat = mat;
        this->K = K;
        m = mat.size();
        n = mat[0].size();

        // Initialize the memoization matrix
        memo = vector<vector<vector<vector<int>>>>(m, vector<vector<vector<int>>>(n,
              vector<vector<int>>(m, vector<int>(n, -1))));

        vector<vector<int>> res(m, vector<int>(n, 0));
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                res[i][j] = blockSum(i, j, max(0, i - K), max(0, j - K));
            }
        }

        return res;
    }
};
```

Hash DP

Hash + DP:

https://leetcode.com/problems/target-sum/

https://leetcode.com/problems/longest-arithmetic-sequence/

https://leetcode.com/problems/longest-arithmetic-subsequence-of-given-difference/

https://leetcode.com/problems/maximum-product-of-splitted-binary-tree/

```cpp
class Solution {
private:
    unordered_map<string, int> memo;

    int findWays(vector<int>& nums, int S, int i, int currentSum) {
        if (i == nums.size()) {
            return S == currentSum ? 1 : 0;
        }

        // Create a unique key for memoization
        string key = to_string(i) + "_" + to_string(currentSum);

        // Check if the result for this subproblem is already computed
        if (memo.find(key) != memo.end()) {
            return memo[key];
        }

        // Calculate the number of ways by including the current number
        // once as a positive and once as a negative
        int add = findWays(nums, S, i + 1, currentSum + nums[i]);
        int subtract = findWays(nums, S, i + 1, currentSum - nums[i]);

        // Save the result in memoization map
        memo[key] = add + subtract;

        return memo[key];
    }

public:
    int findTargetSumWays(vector<int>& nums, int S) {
        return findWays(nums, S, 0, 0);
    }
};
```

State machine

State machine:

https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/

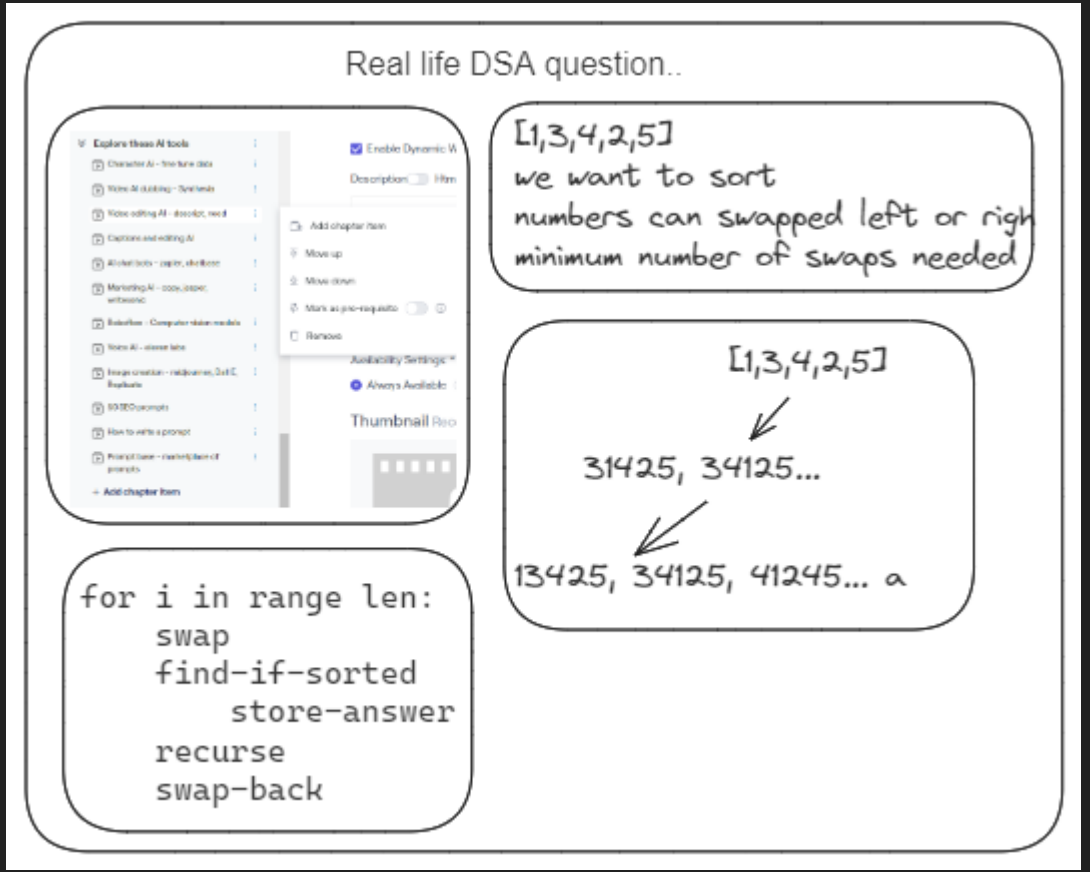
https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/

```cpp
class Solution {
private:
    unordered_map<string, int> memo;

    int maxProfitRecursive(vector<int>& prices, int fee, int i, bool hasStock) {
        if (i == prices.size()) {
            return 0;
        }

        string key = to_string(i) + "_" + to_string(hasStock);
        if (memo.find(key) != memo.end()) {
            return memo[key];
        }

        int doNothing = maxProfitRecursive(prices, fee, i + 1, hasStock);
        int doSomething;

        if (hasStock) {
            // Option to sell the stock
            doSomething = prices[i] - fee + maxProfitRecursive(prices, fee, i + 1, false);
        } else {
            // Option to buy the stock
            doSomething = -prices[i] + maxProfitRecursive(prices, fee, i + 1, true);
        }

        memo[key] = max(doNothing, doSomething);
        return memo[key];
    }

public:
    int maxProfit(vector<int>& prices, int fee) {
        return maxProfitRecursive(prices, fee, 0, false);
    }
};
```

Real life


Real life DSA question..
[1,3,4,2,5]
we want to sort
numbers can swapped left or righ
minimum number of swaps needed
[1,3,4,2,5]
31425, 34125...
13425, 34125, 41245... a
for i in range len:
swap
find-if-sorted
store-answer
recurse
swap-back

Leetcode articles

https://leetcode.com/discuss/general-discussion/665604/Important-and-Useful-links-from-all-over-the-Leetcode

https://leetcode.com/discuss/general-discussion/458695/dynamic-programming-patterns