1.
Найдите подпоследовательность заданной последовательности такую, чтобы сумма подпоследовательности была как можно больше, а элементы подпоследовательности отсортированы в возрастающем порядке. Эта подпоследовательность не обязательно непрерывна или уникальна. Например, рассмотрим подпоследовательность {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11}. Подпоследовательность, увеличивающая максимальную сумму, равна {8, 12, 14}, сумма которой равна 34.
2.
Самая длинная задача возрастающей подпоследовательности состоит в том, чтобы найти подпоследовательность заданной последовательности, в которой элементы подпоследовательности отсортированы в порядке от низшего к высшему, и в которой подпоследовательность является максимально возможной. Эта подпоследовательность не обязательно непрерывна или уникальна.

Например, самая длинная возрастающая подпоследовательность [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15] равна [0, 2, 6, 9 , 11, 15].

Эта подпоследовательность имеет длину 6; входная последовательность не имеет 7-членных возрастающих подпоследовательностей. Самая длинная возрастающая подпоследовательность в этом примере не уникальна.

Например, [0, 4, 6, 9, 11, 15] или [0, 4, 6, 9, 13, 15] — это другие возрастающие подпоследовательности равной длины в той же входной последовательности.

3.
Самая длинная общая проблема подстроки — это проблема поиска самой длинной строки (или строк), которая является подстрокой (или подстроками) двух строк.

Задача отличается от задачи нахождения самой длинной общей подпоследовательности (LCS). В отличие от подпоследовательностей, подстроки должны занимать последовательные позиции в исходной строке.

Например, самая длинная общая подстрока строк ABABC, BABCA — это строка BABC, имеющая длину 4. Другими общими подстроками являются ABC, A, AB, B, BA, BC и C.

4.
Кратчайшая общая суперпоследовательность (SCS) находит кратчайшую суперпоследовательность Z заданных последовательностей X и Y, такую, что X и Y являются подпоследовательностями Z.

Например, рассмотрим две следующие последовательности, X и Y:

X: ABCBDAB
Y: BDCABA

Длина SCS 9
SCS: ABCBDCABA, ABDCABDAB и ABDCBDABA.

5.
Проблема с самой длинной повторяющейся последовательностью
Проблема самой длинной повторяющейся подпоследовательности (LRS) заключается в поиске самой длинной подпоследовательности строки, которая встречается не менее двух раз.

Задача отличается от задачи поиска самой длинной повторяющейся подстроки. В отличие от подстрок, подпоследовательности не обязаны занимать последовательные позиции в исходной строке.

Например, рассмотрим последовательность ATACTCGGA.

Длина самой длинной повторяющейся подпоследовательности равна 4
Самая длинная повторяющаяся подпоследовательность — ATCG.

ATACTCGGA
ATACTCGGA

6.
Given an `M × N` binary matrix, find the size of the largest square submatrix of `1's` present.

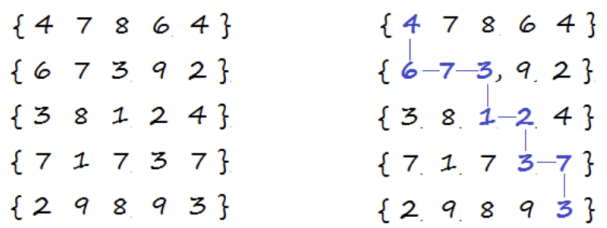
For example, the size of the largest square submatrix of `1's` is `3` in the following matrix:

```
0   0   1   0   1   1
0   1   1   1   0   0
0   0   1   1   1   1
1   1   0   1   1   1
1   1   1   1   1   1
1   1   0   1   1   1
1   0   1   1   1   1
1   1   1   0   1   1
```

7.

Given an `M × N` matrix of integers where each cell has a cost associated with it, find the minimum cost to reach the last cell `(M-1, N-1)` of the matrix from its first cell `(0, 0)`. We can only move one unit right or one unit down from any cell, i.e., from cell `(i, j)`, we can move to `(i, j+1)` or `(i+1, j)`.

For example,



The highlighted path shows the minimum cost path having a cost of 36.

1. Найти для данной строки длиннейшую подпоследовательность, которая является палиндромом
   Пример: ABBDACAB -> 5 (подпоследовательность BACAB)
1. Дан набор чисел. Необходимо определить, может ли данный набор быть разделен на два поднабора с равной суммой.
   Пример: S = {3, 1, 2, 1, 2, 1} -> {3, 1, 1} и {2, 2, 1} - можно; S = {5, 4} - нельзя
1. Дан набор чисел. Необходимо определить, может ли данный набор быть разделен на три поднабора с равной суммой.
   Пример: S = {7, 5, 4, 8, 3, 1, 2} -> {7, 3}, {5, 4, 1} и {8, 2} - можно
1. Дан набор чисел. Необходимо разбить данный набор на два поднабора так, чтобы разность сумм чисел в данных поднаборах была минимальной.
   Пример: S = {10, 20, 15, 5, 25} -> {10, 25} и {20, 15, 5}
1. Дан набор чисел и число k. необходимо проверить, можно ли взять из набора чисел такой поднабор, что сумма чисел в нем будет равна k.
   Пример: S = {7, 6, 3, 4, 5}, k = 14 -> True, {7, 3, 4}
1. Рюкзак классический. Дан набор предметов (для каждого предмета - вес и ценность) и задан максимальный вес рюкзака w. Необходимо определить максимальную суммарную ценность, которую можно получить при упаковке рюкзака.
   Пример:
   values = [20, 5, 10, 40, 15, 25]
   weights = [1, 2, 3, 8, 7, 4]
   W = 10
   Ответ: value = 60 (предметы 1 и 4, суммарный вес 1+8=9)
1. Дан набор чисел. Необходимо найти длину наибольшей чередующейся последовательности, то есть последовательности, в которой $a_1<a_2>a_3<a_4\ldots$ или аналогично начиная со знака >
   Пример: [8, 9, 6, 4, 5, 7, 3, 2, 4] -> 6 (например, [8, 9, 4, 7, 2, 4])

1. Дана строка и дан набор слов. Необходимо определить, может ли строка быть разбита на слова, содержащиеся в словаре.
   Пример:
   d = ['self', 'th', 'is', 'famous', 'Word', 'break', 'b', 'r', 'e', 'a', 'k', 'br', 'bre', 'brea', 'ak', 'problem']
   s = 'Wordbreakproblem'
   Вывод: True (Word break problem или Word brea k problem или Word br e ak problem или другие)

Можно брать отсюда задачи - https://medium.com/techie-delight/top-50-dynamic-programming-practice-problems-4208fed71aa3

1.
Найдите подпоследовательность заданной последовательности такую, чтобы сумма подпоследовательности была как можно больше, а элементы подпоследовательности отсортированы в возрастающем порядке. Эта подпоследовательность не обязательно непрерывна или уникальна. Например, рассмотрим подпоследовательность {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11}. Подпоследовательность, увеличивающая максимальную сумму, равна {8, 12, 14}, сумма которой равна 34.

Решение:
https://www.techiedelight.com/increasing-subsequence-with-maximum-sum/
Задача «Увеличение максимальной суммы подпоследовательности» (MSIS) — это стандартный вариант задачи «Самая длинная возрастающая подпоследовательность» (LIS). Идея состоит в том, чтобы использовать рекурсию для решения этой проблемы. Для каждого элемента есть две возможности:

- Включить текущий элемент в MSIS, если он больше, чем предыдущий элемент в MSIS, и повторить для остальных элементов.
- Исключить текущий элемент из MSIS и повторить для остальных элементов.

Наконец, верните максимальную сумму, которую мы получим, включив или исключив текущий элемент. Базовым случаем рекурсии будет отсутствие элементов. Ниже приведена реализация этой идеи на C++, Java и Python:

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

// Function to find the maximum sum of an increasing subsequence
int MSIS(vector<int> const &nums, int i, int prev, int sum)
{
    // base case: nothing is remaining
    if (i == nums.size()) {
```

```
        return sum;
    }

    // case 1: exclude the current element and process the
    // remaining elements
    int excl = MSIS(nums, i + 1, prev, sum);

    // case 2: include the current element if it is greater
    // than the previous element
    int incl = sum;
    if (nums[i] > prev) {
        incl = MSIS(nums, i + 1, nums[i], sum + nums[i]);
    }

    // return the maximum of the above two choices
    return max(incl, excl);
}

int main()
{
    vector<int> nums = { 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11 };

    cout << "The maximum sum of increasing subsequence is " <<
            MSIS(nums, 0, INT_MIN, 0);

    return 0;
}
```

Временная сложность приведенного выше решения экспоненциальна и занимает место в стеке вызовов.

Задача имеет оптимальную подструктуру. Это означает, что проблема может быть разбита на более мелкие, простые «подзадачи», которые затем могут быть разделены на еще более простые, более мелкие подзадачи, пока решение не станет тривиальным. Приведенное выше решение также демонстрирует перекрывающиеся подзадачи. Если мы нарисуем дерево рекурсии решения, мы увидим, что одни и те же подзадачи вычисляются повторно.

Мы знаем, что задачи с оптимальной подструктурой и перекрывающимися подзадачами можно решать с помощью динамического программирования, при котором решения подзадач запоминаются, а не повторно вычисляются. Запоминаемая версия следует нисходящему подходу, поскольку сначала мы разбиваем задачу на подзадачи, а затем вычисляем и сохраняем значения.

Мы также можем решить эту проблему восходящим способом. При подходе «снизу вверх» мы сначала решаем меньшие подзадачи, а затем на их основе решаем более крупные подзадачи. Следующий восходящий подход вычисляет sum[i] для каждого 0 <= i < n, в котором хранится максимальная сумма возрастающей подпоследовательности подмассива nums[0…i], которая

заканчивается на nums[i]. Чтобы вычислить сумму [i], рассмотрите MSIS всех меньших значений i (скажем, j), уже вычисленных, и выберите максимальную сумму [j], где nums [j] меньше, чем nums [i] текущего элемента, и добавьте nums текущего элемента [я] к нему. У него такое же асимптотическое время выполнения, как и у Memoization, но нет накладных расходов на рекурсию.

Ниже приведена реализация этой идеи на C++, Java и Python:

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

// Iterative function to find the maximum sum of an increasing
subsequence
int MSIS(vector<int> const &nums)
{
    // base case
    if (nums.size() == 0) {
        return 0;
    }

    // array to store subproblem solutions. `sum[i]` stores the maximum
    // sum of the increasing subsequence that ends with `nums[i]`
    vector<int> sum(nums.size());

    // base case
    sum[0] = nums[0];

    // start from the second array element
    for (int i = 1; i < nums.size(); i++)
    {
        // do for each element in subarray `nums[0…i-1]`
        for (int j = 0; j < i; j++)
        {
            // find increasing subsequence with the maximum sum that ends
with
            // `nums[j]`, where `nums[j]` is less than the current element
`nums[i]`

            if (sum[i] < sum[j] && nums[i] > nums[j]) {
                sum[i] = sum[j];
            }
        }

        // include `nums[i]` in MSIS
        sum[i] += nums[i];
    }

    // find increasing subsequence with the maximum sum
    int msis = INT_MIN;
    for (int x: sum) {
        msis = max(msis, x);
```

```
    }

    return msis;
}

int main()
{
    vector<int> nums = { 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11 };

    cout << "The maximum sum of increasing subsequence is " <<
MSIS(nums);

    return 0;
}
```

The time complexity of the above solution is $O(n^2)$ and requires $O(n)$ extra space, where n is the size of the given sequence.

2.
Самая длинная задача возрастающей подпоследовательности состоит в том, чтобы найти подпоследовательность заданной последовательности, в которой элементы подпоследовательности отсортированы в порядке от низшего к высшему, и в которой подпоследовательность является максимально возможной. Эта подпоследовательность не обязательно непрерывна или уникальна.

Например, самая длинная возрастающая подпоследовательность [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15] равна [0, 2, 6, 9 , 11, 15].

Эта подпоследовательность имеет длину 6; входная последовательность не имеет 7-членных возрастающих подпоследовательностей. Самая длинная возрастающая подпоследовательность в этом примере не уникальна.

Например, [0, 4, 6, 9, 11, 15] или [0, 4, 6, 9, 13, 15] — это другие возрастающие подпоследовательности равной длины в той же входной последовательности.

Решение:
https://www.techiedelight.com/longest-increasing-subsequence-using-dynamic-programming/

3.
Самая длинная общая проблема подстроки — это проблема поиска самой длинной строки (или строк), которая является подстрокой (или подстроками) двух строк.

Задача отличается от задачи нахождения самой длинной общей подпоследовательности (LCS). В отличие от подпоследовательностей, подстроки должны занимать последовательные позиции в исходной строке.

Например, самая длинная общая подстрока строк ABABC, BABCA — это строка BABC, имеющая длину 4. Другими общими подстроками являются ABC, A, AB, B, BA, BC и C.

Решение: https://www.techiedelight.com/longest-common-substring-problem/

4.
Кратчайшая общая суперпоследовательность (SCS) находит кратчайшую суперпоследовательность Z заданных последовательностей X и Y, такую, что X и Y являются подпоследовательностями Z.

Например, рассмотрим две следующие последовательности, X и Y:

X: АБВБДАБ
Y: БДКАБА

Длина СКС 9
SCS: ABCBDCABA, ABDCABDAB и ABDCBDABA.

Решение:
https://www.techiedelight.com/shortest-common-supersequence-introduction-scs-length/

5.
Проблема с самой длинной повторяющейся последовательностью
Проблема самой длинной повторяющейся подпоследовательности (LRS) заключается в поиске самой длинной подпоследовательности строки, которая встречается не менее двух раз.

Задача отличается от задачи поиска самой длинной повторяющейся подстроки. В отличие от подстрок, подпоследовательности не обязаны занимать последовательные позиции в исходной строке.

Например, рассмотрим последовательность ATACTCGGA.

Длина самой длинной повторяющейся подпоследовательности равна 4
Самая длинная повторяющаяся подпоследовательность — ATCG.

ATACTCGGA
ATACTCGGA

Решение: https://www.techiedelight.com/longest-repeated-subsequence-problem/

6.

Given an `M × N` binary matrix, find the size of the largest square submatrix of `1's` present.

For example, the size of the largest square submatrix of `1's` is `3` in the following matrix:

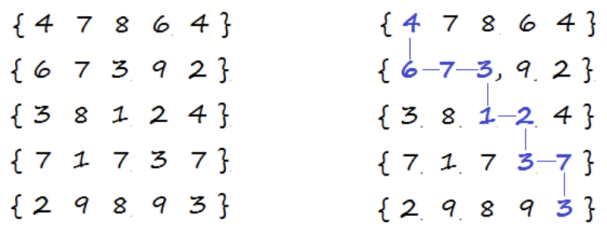
```
0   0   1   0   1   1
0   1   1   1   0   0
0   0   1   1   1   1
1   1   0   1   1   1
1   1   1   1   1   1
1   1   0   1   1   1
1   0   1   1   1   1
1   1   1   0   1   1
```

Решение:
https://www.techiedelight.com/find-size-largest-square-sub-matrix-1s-present-given-binary-matrix/

7.

Given an `M × N` matrix of integers where each cell has a cost associated with it, find the minimum cost to reach the last cell `(M–1, N–1)` of the matrix from its first cell `(0, 0)`. We can only move one unit right or one unit down from any cell, i.e., from cell `(i, j)`, we can move to `(i, j+1)` or `(i+1, j)`.

For example,



The highlighted path shows the minimum cost path having a cost of 36.

Решение:
https://www.techiedelight.com/find-minimum-cost-reach-last-cell-matrix-first-cell/

1. Найти для данной строки длиннейшую подпоследовательность, которая является палиндромом
   Пример: ABBDACAB -> 5 (подпоследовательность BACAB)
   Решение:
   https://www.techiedelight.com/longest-palindromic-subsequence-using-dynamic-programming/
1. Дан набор чисел. Необходимо определить, может ли данный набор быть разделен на два поднабора с равной суммой.
   Пример: S = {3, 1, 2, 1, 2, 1} -> {3, 1, 1} и {2, 2, 1} - можно; S = {5, 4} - нельзя
   Решение: https://www.techiedelight.com/partition-problem/
1. Дан набор чисел. Необходимо определить, может ли данный набор быть разделен на три поднабора с равной суммой.
   Пример: S = {7, 5, 4, 8, 3, 1, 2} -> {7, 3}, {5, 4, 1} и {8, 2} - можно
   Решение: https://www.techiedelight.com/3-partition-problem/
1. Дан набор чисел. Необходимо разбить данный набор на два поднабора так, чтобы разность сумм чисел в данных поднаборах была минимальной.
   Пример: S = {10, 20, 15, 5, 25} -> {10, 25} и {20, 15, 5}
   Решение: https://www.techiedelight.com/minimum-sum-partition-problem/
1. Дан набор чисел и число k. необходимо проверить, можно ли взять из набора чисел такой поднабор, что сумма чисел в нем будет равна k.
   Пример: S = {7, 6, 3, 4, 5}, k = 14 -> True, {7, 3, 4}
   Решение: https://www.techiedelight.com/subset-sum-problem/
1. Рюкзак классический. Дан набор предметов (для каждого предмета - вес и ценность) и задан максимальный вес рюкзака w. Необходимо определить максимальную суммарную ценность, которую можно получить при упаковке рюкзака.
   Пример:
   values = [20, 5, 10, 40, 15, 25]
   weights = [1, 2, 3, 8, 7, 4]
   W = 10
   Ответ: value = 60 (предметы 1 и 4, суммарный вес 1+8=9)
   Решение: https://www.techiedelight.com/0-1-knapsack-problem/
   (здесь можно рассмотреть и другие вариации - например, где каждый предмет можно брать неограниченное число раз)
1. Дан набор чисел. Необходимо найти длину наибольшей чередующейся последовательности, то есть последовательности, в которой $a_1<a_2>a_3<a_4\ldots$ или аналогично начиная со знака >
   Пример: [8, 9, 6, 4, 5, 7, 3, 2, 4] -> 6 (например, [8, 9, 4, 7, 2, 4])
   Решение: https://www.techiedelight.com/longest-alternating-subsequence/
   Задание с подвохом - его можно решить и в один проход за линейное время, решение есть в комментариях
1. Дана строка и дан набор слов. Необходимо определить, может ли строка быть разбита на слова, содержащиеся в словаре.
   Пример:
   d = ['self', 'th', 'is', 'famous', 'Word', 'break', 'b', 'r', 'e', 'a', 'k', 'br', 'bre', 'brea', 'ak', 'problem']
   s = ‘Wordbreakproblem’
   Вывод: True (Word break problem или Word brea k problem или Word br e ak problem или другие)
   Решение: https://www.techiedelight.com/word-break-problem/

Если все успеется, то все еще https://medium.com/techie-delight/top-50-dynamic-programming-practice-problems-4208fed71aa3 или посмотреть на оставшиеся с того семинара задачи