**ЛАБОРАТОРНАЯ РАБОТА №4.1**

**[ОДНОМЕРНОЕ ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html" \l "id12):**

**ЗАДАЧА О ВЫБОРЕ ТРАЕКТОРИИ**

***Цель*:** научиться решать задачи одномерного динамического программирования

**Отчет по работе** должен содержать **2 файла: 1) код** программы**; 2)** файл в формате **.pdf** со скринами результатов работы (входные данные и полученный результат).

Динамическое программирование — решение сложной задачи разбиением её на более простые подзадачи, при этом каждая подзадача решается только один раз.

Динамическое программирование очень похоже на рекурсию, при этом:

* восходящее динамическое программирование — это, по сути, рекурсия с кешированием;
* нисходящее динамическое программирование — это переформулирование задачи в виде индуктивной последовательности подзадач, от крайнего случая к более сложным.

[***Задача о кузнечике — количество способов***](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html#id14)

Рассмотрим следующую задачу. На числовой прямой сидит кузнечик, который может прыгать вправо на одну или на две единицы. Первоначально кузнечик находится в точке с координатой 1. Определите количество различных маршрутов кузнечика, приводящих его в точку с координатой n.

Обозначим количество маршрутов кузнечика, ведущих в точку с координатой n, как K[n]. Прежде всего, заметим, что существует ровно один маршрут из точки 1 в точку 1 — он не содержит ни одного прыжка. В точку 2 можно прыгнуть единственным способом — из точки 1.

Как вычислить K[n]? В точку кузнечик может попасть двумя способами — из точки при помощи прыжка длиной 2 и из точки прыжком длины 1. То есть число способов попасть в точку n равно сумме числа способов попасть в точку (n-1) и (n-2), что позволяет выписать рекуррентное соотношение: K[n] = K[n-1] + K[n-2].

[**Задание №1**](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html#id15)

Решите задачу о количестве способов достичь точки **n** из точки **1**, если кузнечик умеет прыгать +1, +2 и +3.

[**Задание №2**](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html#id16)

Решите задачу о количестве способов достичь точки **n** из точки **1**, если кузнечик умеет прыгать +1, +2 и \*3.

[***Задача о кузнечике со стоимостями посещения точек***](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html#id17)

Пусть кузнечик прыгает на одну или две точки вперед, а за прыжок в каждую точку необходимо заплатить определенную стоимость, различную для различных точек. Стоимость прыжка в точку **i** задается значением price[i] списка price. Необходимо найти минимальную стоимость маршрута кузнечика из точки **0** в точку **n**.

На этот раз нам необходимо модифицировать определение *целевой функции*. Пусть C[n] — минимальная стоимость пути из 1 в n.

Выведем рекуррентное соотношение для этой функции. Попасть в точку **n** можно последним прыжком из (n-1) или (n-2). Минимальные стоимости этих маршрутов будут равны С[n-1] и С[n-2] соответственно, к ним придется добавить значение price[n] за прыжок в клетку n. Но из двух клеток можно выбрать любую.

Нужно выбрать тот маршрут, который имеет наименьшую стоимость: C[n] = min(C[n-1], C[n-2]) + price[n]

Вычислить значение целевой функции также лучше при помощи динамического программирования, а не рекурсии.

[**Задание №3**](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html#id18)

Напишите функцию calculate\_min\_cost(n,price) вычисления наименьшей стоимости достижения клетки n из клетки 1

[***Восстановление наиболее выгодной траектории***](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html#id19)

Итак, список С содержит минимальные стоимости маршрута для всех точек от **1** до **n**.

Но помимо нахождения наименьшей стоимости маршрута, разумеется, хотелось бы найти и сам маршрут минимальной стоимости. Такая задача называется задачей «восстановления ответа».

Для восстановления ответа будем для каждой точки запоминать номер точки prev[i], из которой кузнечик попал в точку **i**, если он будет передвигаться по пути минимальной стоимости. То есть prev[i] — это точка, предшествующая точке с номером i на пути минимальной стоимости (также говорят, что Prev — это массив предшественников).

Как определить prev[i]? Если C[i-1] < C[i-2], то кузнечик попал в точку **i** из точки (i-1), поэтому prev[i] = i - 1, иначе prev[i] = i - 2.

Для восстановления пути необходимо начать с точки **n** и переходить от каждой точки к ее предшественнику, пока путь не дойдет до начальной точки с номером **0**. Номера всех вершин будем добавлять в список path. В конце в список path добавляется начальная вершина номер **1**, которая не была обработана в основном цикле, а затем весь список path разворачивается в обратном порядке (т. к. вершины добавляются в обратном порядке, от конечной к начальной).

[**Задание №4**](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab11.html#id20)

Модифицируйте алгоритм вычисления значений целевой функции так, чтобы вычислить значения prev[i], и восстановите траекторию наименьшей стоимости из точки 1 в точку n.