Городскаянаучно-практическая конференция школьников

«Содружество»

Секция: ИНФОРМАТИКА

Номинация: исследовательская работа

**Изучение метода динамического программирования**

Автор: Исмагилов Дамир,

10 класс, МБОУ «Лицей №174»,

г. Зеленогорск, Красноярский край

Руководитель: Снегирева В.С.,

учитель МБОУ «Лицей №174»,

г. Зеленогорск, Красноярский край

г. Зеленогорск, 2014/2015 учебный год

**Оглавление**

[ВВЕДЕНИЕ 3](#_Toc251193725)

1 Теоритическая часть 5

[1.1. История](#_Toc251193727) 5

[1.2.Понятие динамического программирования](#_Toc251193728) 5

[1.3. Идея динамического программирования](#_Toc251193728) 5

1.4. Перекрывающиеся подзадачи 6

1.5. Нисходящая динамика 6

1.6. Восходящая динамика 6

2. Практическая часть 7

2.1 Вывод рекуррентных формул 8

2.2 Задачи на поиск оптимального решения. Часть 1 10

2.3 Задачи на вычисление количества решений 11

2.4 Задачи на поиск оптимального решения. Часть 2 13

Заключение15

Приложение16

Список литературы22

**Введение**

Во многих олимпиадных задачах по программированию решение с помощью рекурсии или полного перебора требует выполнения очень большого числа операций. Попытка решить такие задачи, например, полным перебором, приводит к превышению времени выполнения. Однако среди переборных и некоторых других задач можно выделить класс задач, обладающих одним хорошим свойством: имея решения некоторых подзадач (например, для меньшего числа *n*), можно практически без перебора найти решение исходной задачи. Такие задачи решаются методом динамического программирования. Большинство из них часто имеют динамическое, более эффективное решение благодаря тому, что появляется возможность не вычислять многократно одни и те же промежуточные значения. Принцип динамического программирования используется во многих известных алгоритмах и отражает эффективность данного метода над другими решениями.

**Актуальность** моего исследования в том, что знания этого метода позволят решать сложные олимпиадные задачи эффективно по времени и быстро по скорости.

**Проблема:** существует класс задач, в которых лучше использовать метод динамического программирования.

**Цель:** изучение метода динамического программирования

В ходе выполнения работы были поставлены следующие **задачи:**

1. Изучить понятие динамического программирования
2. Изучить класс задач, связанных с использованием этого метода
3. Научиться применять этот метод к олимпиадным задачам

**Объектом** моего исследования будет понятие динамическое программирование.

**Предметом** - олимпиадные задачи, использующие этот метод.

В данной работе была выдвинута следующая **гипотеза:** предположим, что существуют задачи, решение которых методом динамического программирования, осуществляется намного быстрее и эффективнее, чем простое решение.

Методы, которые помогали в решении поставленной задачи:

1. Теоретический
2. Эмпирический
3. Эксперимент**ГЛАВА 1**

**Теоретическая часть**

*История*

Словосочетание «динамическое программирование» впервые было использовано в 1940-х годах Р. Беллманом для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения задачи, «предшествующей» ей. В 1953 г. он уточнил это определение до современного. Первоначально эта область была основана, как системный анализ и инжиниринг, которая была признана IEEE. Вклад Беллмана в динамическое программирование был увековечен в названии уравнения Беллмана, центрального результата теории динамического программирования, который переформулирует оптимизационную задачу в рекурсивной форме.

*Динамическое программирование* - метод решения задач путем составления последовательности из подзадач таким образом, что:

* первый элемент последовательности (возможно несколько элементов) имеет тривиальное решение
* последний элемент этой последовательности - это исходная задача
* каждая задача этой последовательности может быть решена с использованием решения подзадач с меньшими номерами

Другими словами, для решения задачи T методом ДП составляется некоторая последовательность подзадач T1, T2, ... Tn такая, что решение задачи T1 уже имеет решение, T = Tn, и самое главное, что зная решения задач T1, T2, ... Ti-1 можно вывести решение задачи Ti для любого i = 2..n .

*Идея динамического программирования:*

*Оптимальная подструктура* в динамическом программировании означает, что оптимальное решение подзадач меньшего размера может быть использовано для решения исходной задачи. В общем случае мы можем решить задачу, в которой присутствует оптимальная подструктура, проделывая следующие три шага.

1. Разбиение задачи на подзадачи меньшего размера.
2. Построение таблицы решений.
3. Решение задачи с помощью построенной таблицы.

Подзадача - та же задача, но

* с меньшим числом параметров
* либо с меньшим значением одного из параметров

Подзадачи решаются делением их на подзадачи ещё меньшего размера и т. д., пока не приходят к тривиальному случаю задачи, решаемой за константное время (ответ можно сказать сразу). К примеру, если нам нужно найти n!, то тривиальной задачей будет 1! = 1 (или 0! = 1).

*Перекрывающиеся подзадачи* в динамическом программировании означают подзадачи, которые используются для решения некоторого количества задач (не одной) большего размера (то есть мы несколько раз проделываем одно и то же).

Многие переборные задачи часто имеют динамическое, более эффективное решение благодаря тому, что появляется возможность не вычислять многократно одни и те же промежуточные значения. Принцип динамического программирования используется во многих известных алгоритмах и отражает эффективность данного метода над другими решениями.

Динамическое программирование обычно придерживается двух подходов к решению задач:

* *Нисходящее динамическое программирование:* задача разбивается на подзадачи меньшего размера, они решаются и затем комбинируются для решения исходной задачи. Используется запоминание для решений часто встречающихся подзадач.
* *Восходящее динамическое программирование:* Все подзадачи, которые впоследствии понадобятся для решения исходной задачи, просчитываются заранее и затем используются для построения решения исходной задачи. Этот способ лучше нисходящего динамического программирования в смысле размера необходимого стека и количества вызова функций, но иногда бывает нелегко заранее выяснить решение каких подзадач нам потребуется в дальнейшем.

**ГЛАВА 2**

**Практическая часть**

**Пример 1**

Задача: Найти количество КN цепочек, состоящих из N нулей и единиц, в которых нет двух стоящих подряд нулей.

Для решения данной задачи методом динамического программирования самое главное вывести рекуррентную формулу, выражающую КN, через решение аналогичных задач меньшей размерности. Рассмотрим цепочку из N бит, первый член которой 1. Поскольку дополнительная 1 не может привести к существованию двух нулей подряд, то подходящих последовательностей длиной N с

единицей вначале существует столько, сколько подходящих последовательностей длины N-1, то есть КN-1. Если же первый символ 0, то вторым обязательно должна быть 1, а остальная цепочка из N-2 битов должна быть «правильной». Поэтому подходящих последовательностей с нулем, вначале существует, столько же, сколько и подходящих последовательностей длины N-2, то есть КN-2.

Теперь давайте рассмотрим простые случаи. Очевидно, что есть две подходящие последовательности длиной 1(0 и 1), то есть К1 = 2. Также существует 3 подходящих последовательности длиной 2(11, 01 и 10), следовательно К2 = 3.

В результате получаем рекуррентную формулу:

KN=KN-1 + KN-2.

Значит, для вычисления очередного числа нам нужно знать значение двух предыдущих.

При решении данной задаче путем перебора всех возможных асимптотика работы 2N, а при решении задачи методом динамического программирования асимптотика работы программы равна N, то есть мы имеем решение за линию.

**Пример 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |

В таблице сразмерамиk \* n, с элементами 1 и 0, найти квадратный блок максимально возможного размера, состоящий из одних единиц. При n = 5, k =6 искомый блок выделен цветом.

Положение любого квадратного блока определяется его размером и нижней правой вершиной. В данном случае максимальный размер блока равен 3, координаты правой нижней вершины квадрата равны b = 4, c = 5.

Пусть F[i][j] – функция значение которой равно размеру максимального квадратного блока правый нижний угол которого расположен в ячейке (i, j). Значение её в 1 строке и в 1 столбце совпадают с элементами m[1][j] и m[i][1].

F[1][j] = m[1][j], F[i][1] = m[i][1]. Допустим, мы находимся в ячейке (i, j) и все предыдущие ячейки таблицы уже заполнены. Тогда значение F[i][j] будет равно 0, только если m[i][j] = 0(ведь квадрат размером 1 мы точно можем составить, несмотря на предыдущие результаты, если m[i][j] != 0).

Первый столбец и первая строчка также будут равны значению массива m, а для остальных значений функции справедливы следующие рекуррентные соотношения:

1. F[i][j] = 0, еслиm[i][j] = 0
2. F[i][j] = min(F[i-1][j], F[i][j-1], F[i-1][j-1]) + 1, приm[i][j] = 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 2 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 2 | 2 | 1 |
| 1 | 1 | 1 | 2 | 3 | 2 |
| 1 | 0 | 1 | 2 | 0 | 1 |

Для приведенного примера справа составлена таблица значений

F[i][j].

Для вывода ответа найдем максимальный элемент в таблице.

Программа для языка С++ приведена в приложении 2.

**Пример 3** Задача о садовнике

Садовник посадил N деревьев в один ряд. После посадки деревьев садовнику нужно их покрасить. В его распоряжении есть краска трех цветов: белая, синяя и оранжевая. Сколько способов покраски деревьев есть у него, если никакие два соседних дерева нельзя красить в одинаковый цвет?

Решение задачи приведено в приложении 7.

**Пример 4**

На прямой дощечке вбиты nгвоздиков. Для каждого гвоздя известна его координата на прямой OX, m[i]. Любые 2 гвоздя можно соединить ниточкой. Надо соединить некоторые пары гвоздиков так, что к каждому гвоздику привязана хотя бы 1 ниточка, а сумма длин всех нитей была минимальной.

Для того чтобы минимизировать результат, будем связывать только соседние гвоздики, для этого отсортируем гвоздики по возрастанию их координат.

Как обычно, начнем с рассмотрения простых случаев. Если гвоздь 1, то длина нити равна L[0] = 0. Если гвоздей 2, то длина нити равна (L[1] = m[1] – m[0]). Если гвоздей 3, то длина нити равна (L[2] = m[2] – m[0]). Если гвоздей 4, то необходимо соединить первые два и последние два (L[3] = L[1] + m[3] – m[0]). Когда у нас пять гвоздей, то мы обязательно связываем ниткой первые два и последние два, а для 3-его гвоздика выбираем минимальную длину ниток, то есть длина нитки равна L[4] = min(L[2] + m[4] – m[3]; L[3] + m[4] – m[2]).

В результате получаем рекуррентную формулу:

L[n] = min (L[n-2] + m[n] – m[n-1]; L[n-3] + m[n] – m[n-2])

Реализация программы на языке С++ приведена в приложении 5.

**Пример 5**

Цистерна объёмом N литров доверху заполнена молоком. У молочника в наличии бидоны объёмом 1, 5 и 6 литров. Он хочет разлить все молоко по бидонам, так чтобы все бидоны оказались доверху заполненными, но сам он не хочет нести большое количество бидонов. Помогите ему узнать какие бидоны нужно использовать, чтобы их количество оказалось минимальным.

Самый простой подход заполнять бидоны максимального объёма (в нашем случае 6 литров), затем меньшего и т.д. Этот способ называется «жадным алгоритмом». Однако он не всегда позволяет находить оптимальное решение. К примеру, для N= 10, жадный алгоритм выдаст результат 6+1+1+1+1 – всего 5 бидонов, тогда же как можно воспользоваться всего двумя бидонами (5 + 5).

Как и для решения любой другой задачи методом динамического программирования, самое главное вывести рекуррентную формулу. Для этого нужно определить оптимальное количество бидонов КN, а потом решить какие бидоны нужно использовать.

Представим, что мы выбираем бидоны постепенно. Тогда последний выбранный бидон может иметь объём, например, 1 литр, в этом случае KN= 1 + Kn-1. Если последний бидон имеет объём 5 литров, то Kn=1+Kn-5. Если 6 литров – KN= 1 + KN-6.

Так как нам нужно минимизировать кол-во бидонов, то KN= 1 + min(KN-1, KN-5, KN-6). Вариант, выбранный при поиске минимума, определяет последний добавленный бидон, то результат этого выбора будем хранить в некотором массиве М. Этот массив будет использован для определения количества выбранных бидонов каждого типа. В качестве начальных значений возьмем К0 = 0 и M0 = 0.

Полученная формула применима для N>= 6. Для меньших N используются только те данные, которые уже есть в таблице. Например

K3= 1 + K2, K5 = 1 + min(K0, K4)

В таблице приведены значения массивов для N = 10

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| K | 0 | 1 | 2 | 3 | 4 | 1 | 1 | 2 | 3 | 4 | 2 |
| M | 0 | 1 | 1 | 1 | 1 | 5 | 6 | 1 | 1 | 1 | 5 |

Как по массиву М определить оптимальный набор бидонов? Пусть для примера N= 10. Из массива М находим, что последний добавленный бидон имеет объём 5л. Остаток равен 10 – 5 = 5л, в элементе М[5] тоже записано значение 5, значит, второй бидон также имеет объём 5л. Остаток равный 0 означает, что мы полностью определили набор бидонов.

Программа для языка C++ приведена в приложении 1.

**Пример 6**

У исполнителя Вычислитель две команды:

1. Прибавь 1
2. Умножь на 4

Первая из них увеличивают число на единицу, а вторая умножает его на 4. Программа для Вычислителя – это последовательность команд. Сколько есть программ, которые преобразуют число 1 в 20?

Заметим, что при выполнении любой из команд число строго увеличивается. Начнем с простых случаев вычисления. Очевидно, что для числа 1 существует только одна программа – пустая не содержащая ни одной команды, для числа 2 также есть только одна программа, состоящая только из одной команды сложения. Если через Кn обозначить количество программ существующих для получения числа n из 1, то Kn = K1 = K2 = 1

Теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую Knс предыдущими элементами последовательности K1, K2, …, Kn-1, то есть с решениями таких же задач для меньших n.

Если число nне делится на 4, то оно могло быть получено только последней операцией сложения, то есть Kn= Kn-1. Если же Knделится на 4, то оно могло быть получено как сложением, так и умножением, следовательно, необходимо сложить Kn-1 (количество программ с последней командой сложения) и Kn/4 (количество программ с последней командой умножения). В итоге получаем:

1. Kn= Kn-1, если n не делится на 4
2. Kn= Kn-1 + Kn/4, если nделится на 4

Тогда заполнение таблицы количества программ для составления чисел можно достаточно просто реализовать следующим циклом:

m[1] = 1;

for(inti=2; i<=n; ++i){

m[i] = (i % 4 ? m[i-1] : m[i-1] + m[i/4]);

}

Ответом будет значениеm[n].

**Пример 7** Мячик на лесенке

На вершине лесенки, содержащей Nступенек, находится мячик, который начинает прыгать по ним вниз, к основанию. Мячик может прыгнуть на следующую ступеньку, на ступеньку через одну, через две. (То есть, если мячик лежит на 8-ой ступеньке он может прыгнуть на 7-ую, 6-уюи 5-ую.) Определить число всевозможных «маршрутов» мячика на землю.

Для удобства решения я предлагаю немного переформулировать задачу. Пусть мячик стоит не на вершине лесенки, а у ее основания (на 0-вой ступеньке) и ему нужно достичь вершины этой лесенки.

Заметим, что при любом действии мячика его текущее местоположение строго возрастает. Как и при решении других задач, я предлагаю начать с простых случаев. Очевидно, что если бы была всего одна ступенька, то достичь ее можно всего одним способом. Если таких ступенек будет 2 или 3, то число способов возрастает до 2 и 4 соответственно.

Теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую Knс предыдущими элементами последовательности K1, K2, …, Kn-1, то есть с решениями таких же задач для меньших n. Мячик может прыгнуть на любую ступеньку 3 разными способами (если ее высота больше 3), а это значит, что общее число способов будет равно сумме этих трех предыдущих элементов последовательности.

Kn =Kn-1 +Kn-2 + Kn-3, приn >3

Можно заметить, что данная задача представляет собой переформулированную предыдущую задачу о вычислителе.

Программа на языке С++ представлена в приложении 4.

**Пример 8 Задача** о рюкзаке.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| m[i] | 4 | 5 | 3 | 7 | 6 |
| c[i] | 5 | 7 | 4 | 9 | 8 |

На складе имеется n неделимых предметов. Для каждого предмета известна его масса m[i] и его стоимость c[i]. Необходимо определить максимальную суммарную стоимость предметов, которые можно унести со склада, при условии, что суммарная масса предметов не должна превышать p =15.

Пусть T[i][j] – функция, значения которой равны суммарной стоимости унесенных предметов, при условии, что выбираются предметы из первых i предметов, j– максимально возможная их масса. Решению задачи соответствует T[5][15]. Для задачи T[n][p] определим подзадачи T[i][j], i–количество начальных предметов, j–максимум возможной массы уносимых предметов. Аргумент i задаёт количество предметов для подзадачи. Найдём рекуррентное соотношение для вычисления функции T:

1. T[0][j] = 0, при j >= 0
2. T[i][0] = 0, при i>= 0

Если i-тый предмет остается на складе, то T[i][j] = T[i-1][j]. Если предмет с номером i уносится со склада, то это уменьшает суммарную массу для i-1 первых предметов на m[i], увеличивая значения решения на c[i]: T[i][j] = T[i-1][j-m[i]]+c[i]. Этот вариант возможен, если m[i] <= j. Из двух вариантов выбираем наибольший. Таким образом, при i>0 и j>0 соотношение имеет вид:

1. T[i][j] = T[i-1][j]
2. T[i][j] = max (T[i-1][j], T[i-1][j-m[i]] + c[i])при m[i]<= j. T[5][15] = 20

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 2 | 0 | 0 | 0 | 0 | 5 | 7 | 7 | 7 | 7 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| 3 | 0 | 0 | 0 | 4 | 5 | 7 | 7 | 9 | 11 | 12 | 12 | 12 | 16 | 16 | 16 | 16 |
| 4 | 0 | 0 | 0 | 4 | 5 | 7 | 7 | 9 | 11 | 12 | 12 | 14 | 16 | 16 | 18 | 20 |
| 5 | 0 | 0 | 0 | 4 | 5 | 7 | 8 | 9 | 11 | 12 | 13 | 15 | 16 | 17 | 19 | 20 |

Построение списка уносимых предметов

Список уносимых предметов будем составлять «с конца». Рассмотрим элементы таблицы функции T[5][15] и T[4][15]. Так как значения обоих этих элементов равны 20, это значит, что можно набрать стоимость 20 ус. ед. без пятого предмета, используя в списке только предметы с номерами 1 – 4, то есть пятый предмет мы не включаем в список. Теперь рассмотрим элементы T[4][15] и T[3][15]. Их значения не равны, 16 и 20. Это означает, что четвертый предмет должен быть в списке уносимых предметов, а масса других предметов списка должна быть не более 15 – m[4] = 15 – 7 = 8(кг). Далее рассмотрим элементы T[3][8] и T[2][8]. Их значения не равны, значит, третий предмет мы также включаем в список, а значение массы других предметов списка не должна быть больше 8 – m[3] = 5 (кг). Теперь рассмотрим элементы T[2][5] и T[1][5]. Их значения не равны, поэтому второй предмет включается в список. Так как его масса равна 5 (кг), то первый предмет не может включаться в список.

Таким образом, в искомый список входят предметы с номерами 2, 3 и 4.

Программа для языка С++ представлена в приложении 3.

**Пример 9** Задача о куче

Из nкамней весом m[i] требуется набрать кучу весом ровно w или, если это невозможно, максимально близкую к w, но меньшую чем w. Все числа целые.

Данная задача похожа на предыдущую, и решается аналогично.

Программа для языка С++ представлена в приложении 6.

**Пример 10** Размен монет

Сколькими различными способами можно выдать сдачу размером Wрублей, если даны монеты стоимость mi[i=1…n]? Для того чтобы сдачу всегда можно было выдать, будем предполагать, что в наборе всегда есть монета достоинством 1 рубль.

Реализация задачи на языке С++ приведена в приложении 8.

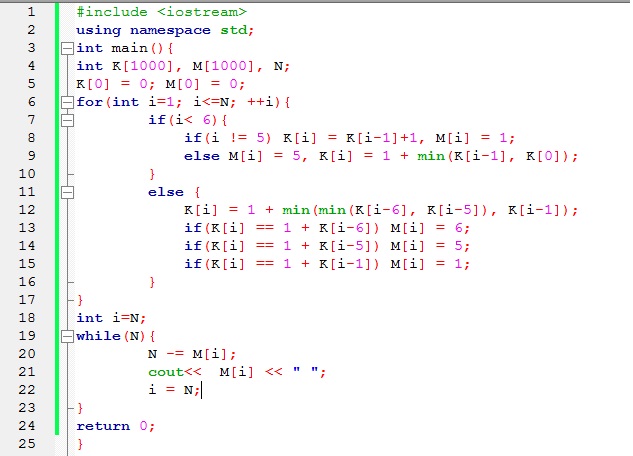
**Заключение**

На примере рассмотренных в работе задач, мы можем убедиться, что решение задач методом динамического программирования действительно существенно быстрее, чем решение с помощью перебора вариантов. Помимо этого, в плане реализации метод динамического программирования достаточно прост, так как он не требует особых знаний в программировании.

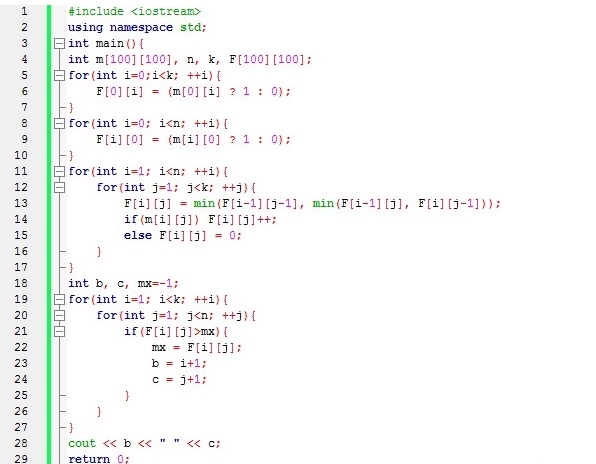
Поставленная мною цель достигнута, задачи решены. Гипотеза о том, что существуют задачи, решение которых методом динамического программирования, осуществляется намного быстрее и эффективнее, чем простое решение, подтвердилась.

**Приложения**

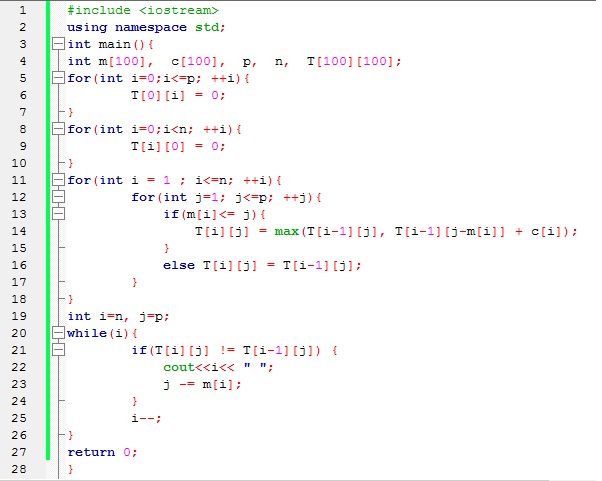
**Приложение 1**

****

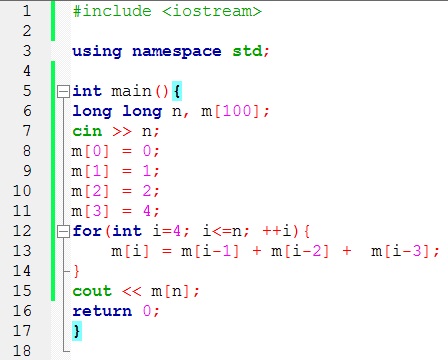
**Приложение 2**

****

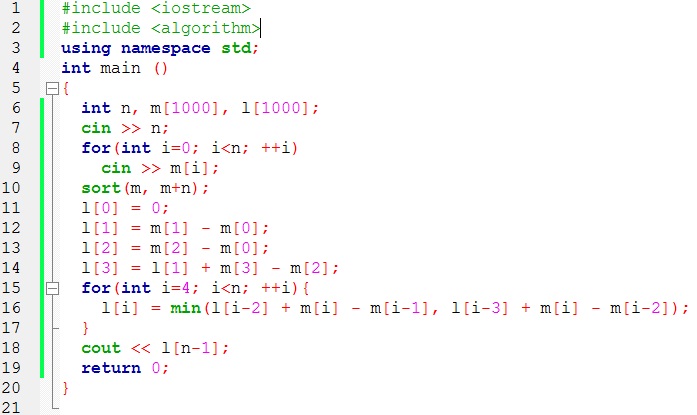
**Приложение 3**



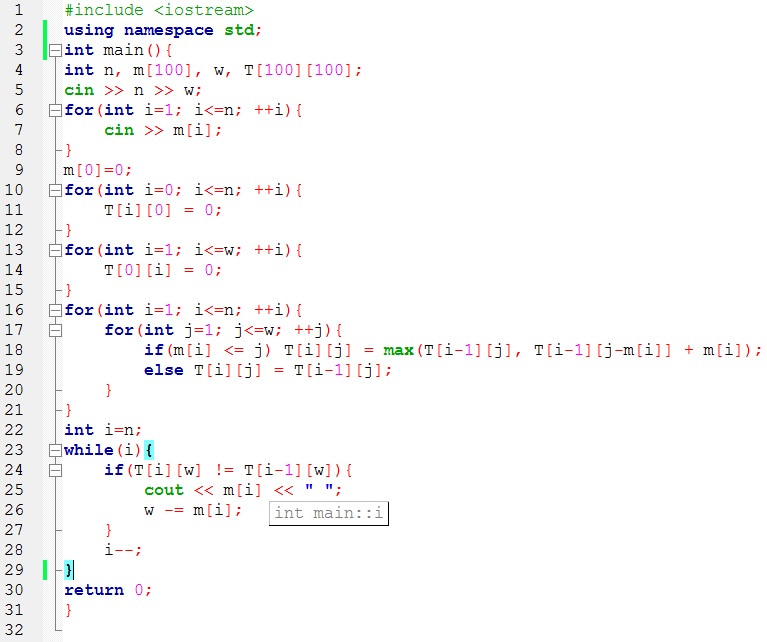
**Приложение 4**

****

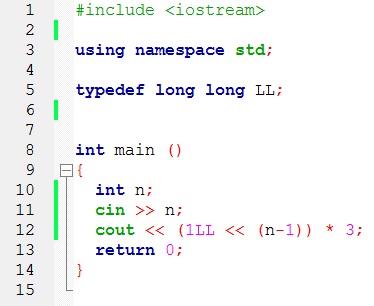
**Приложение 5**

****

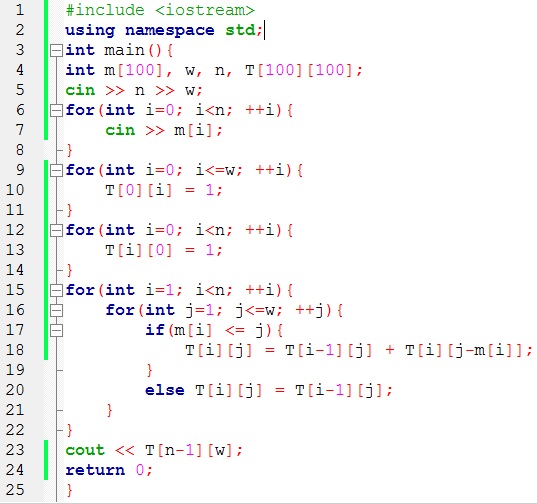
**Приложение 6**



**Приложение 7**

****

**Приложение 8**

**Список литературы:**

1. Информатика. Углубленный уровень. Для 11 класса К.Ю. Поляков, Е.А.Еремин
2. C++. Священные знания. С.С.Дьюхерт
3. Справочник спортивного программиста. М.Д. Кормышов, В.А. Демиденко
4. Решение олимпиадных задач по программированию. С.Н. Беляев
5. Олимпиадные задачи по программированию Ф.В.Меньшиков
6. Информатика и ИКТ Н.Д.Угринович
7. <http://acmp.ru/>
8. <http://codeforces.ru/>
9. <http://dp.acmp.ru/>
10. http://informatics.mccme.ru/