Тверской Государственный технический университет

Кафедра: Программного обеспечения и вычислительной

техники

Лабораторная работа № 2

Выполнил: Студент второго курса

Группы Б.ПИН.РИС 18.05

Федотов Ярослав Всеволодович

Тверь 2019

**Задача 1. Биноминальные коэффициенты или нахождение числа сочетаний**

**Формулировка задачи:**

1. Реализовать задачу из формулы через вычисление факториала

Алгоритм решения:

Функция fakt(принимаем n)

Объявление целочисленного r

R = 1

Для n от n до 0 c шагом -1

R\*=n

Всё\_для

Return r

Всё\_функция fakt

Функция bci(принимаем n,k)

return fakt(n)/(fakt(k)\*fakt(n-k));

Всё\_функция bci

Основной запуск

Ввод n,k

Печать(bci(n,k))

Текст программы:

def fakt(n):

r = int();

r = 1

for n in range(n,0,-1):

r\*=n;

return r;

def bci(n,k):

return fakt(n)/(fakt(k)\*fakt(n-k));

print('Поиск Биноминальных коэффициентов через формулу факториалов ')

print('Введите число - n ,до которого будет поиск')

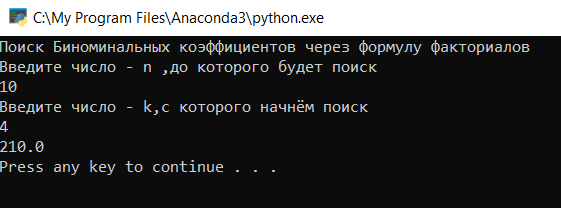
n = int(input())

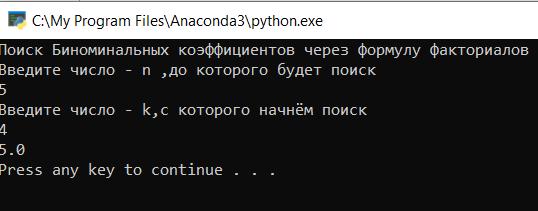
print('Введите число - k,с которого начнём поиск')

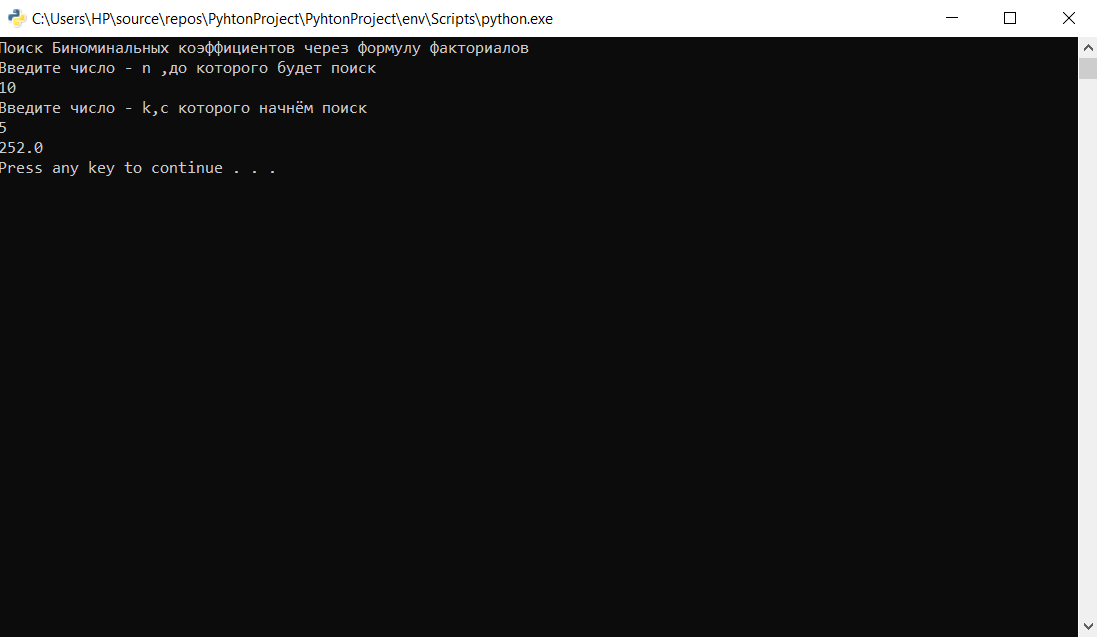
k = int(input())

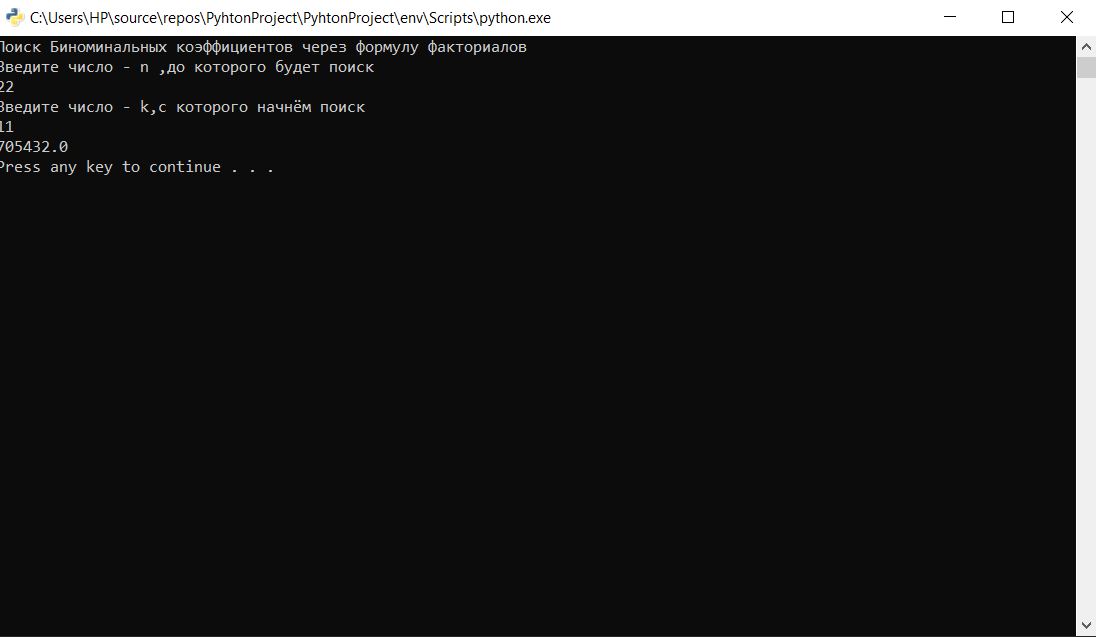
print(bci(n,k))

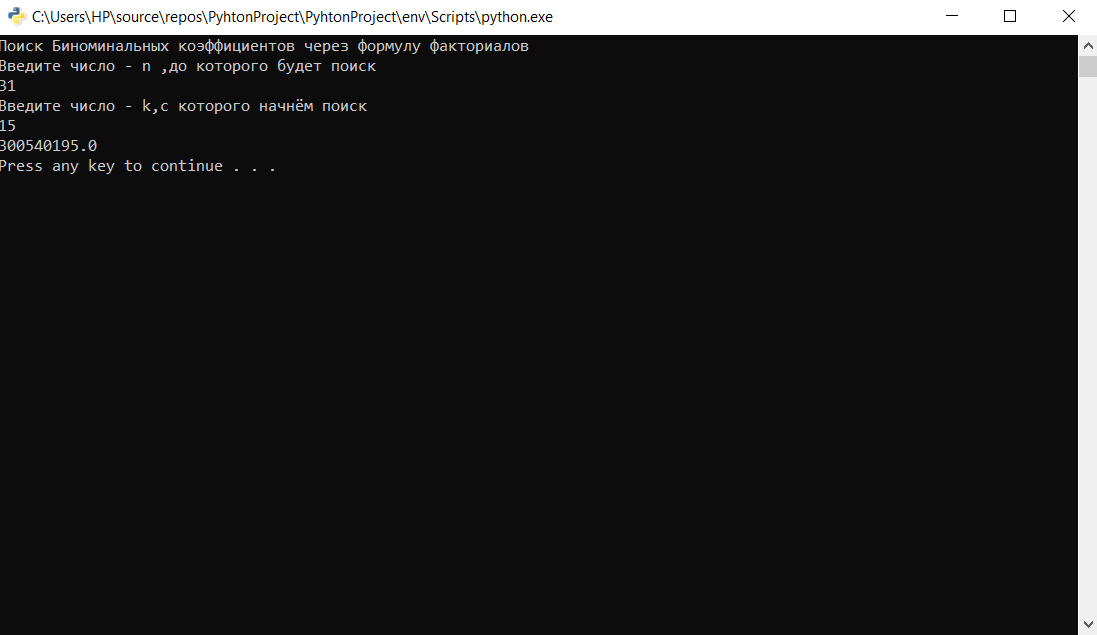
Скриншоты выполнения программы:











Вывод по решению задачи:

Поскольку биномиальные коэффициенты неотрицательные,  
будем использовать в расчетах беззнаковый тип. И в последствии мы   
вызываем функцию bci(10,4) и на вход ей подаём к примеру 10 и 4 — она вернет 210 и это правильное  
значение коэффициента C(10,4), а следовательно задача расчета  
решена.

**Формулировка задачи:**

1. Реализовать представленный рекурсивный алгоритм

Алгоритм решения:

Функция comb(принимаем n,k)

Если(k = 0 или k = n),то

Return 1

Всё\_если

Else

Return Comb(n-1,k)+Comb(n-1,k-1)

Всё\_иначе

Всё\_функция

Основной запуск:

Ввод n,k

Печать(Comb(n,k))

Текст программы:

def Comb(n,k):

if(k == 0 or k == n):

return 1

else:

return Comb(n-1,k)+Comb(n-1,k-1)

print('Поиск Биноминальных коэффициентов через рекурсивный алгоритм ')

print('Введите число - n ,до которого будет поиск')

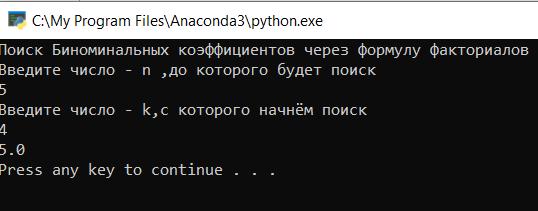
n = int(input())

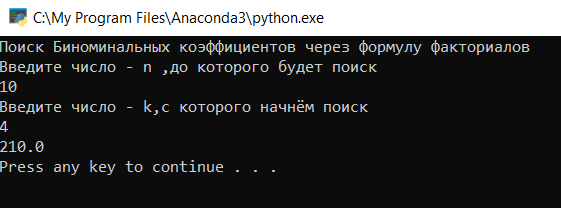
print('Введите число - k,с которого начнём поиск')

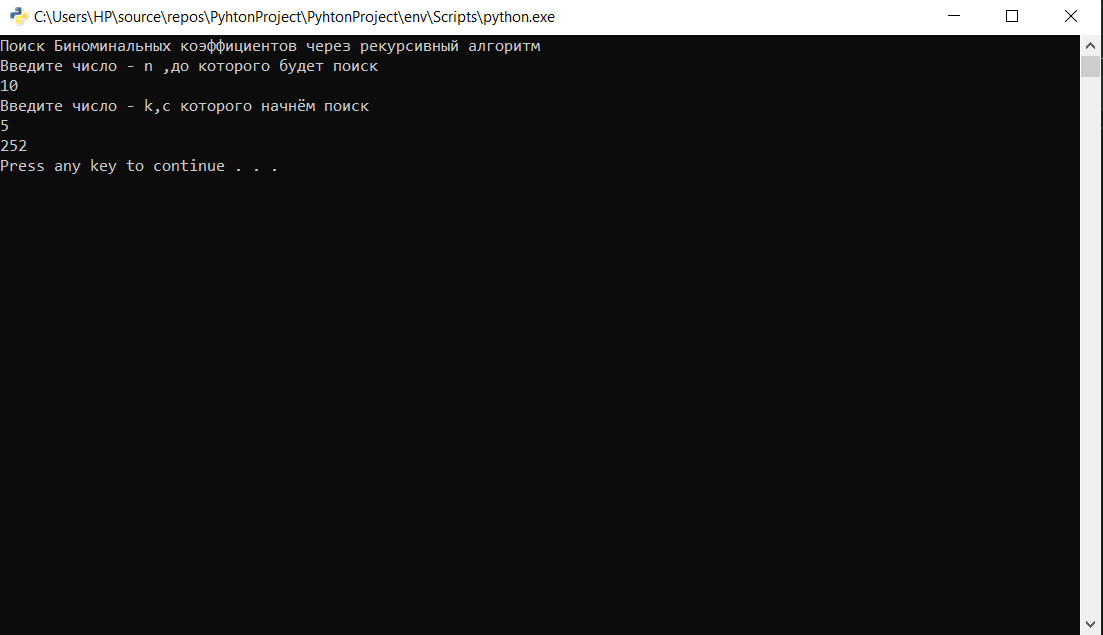
k = int(input())

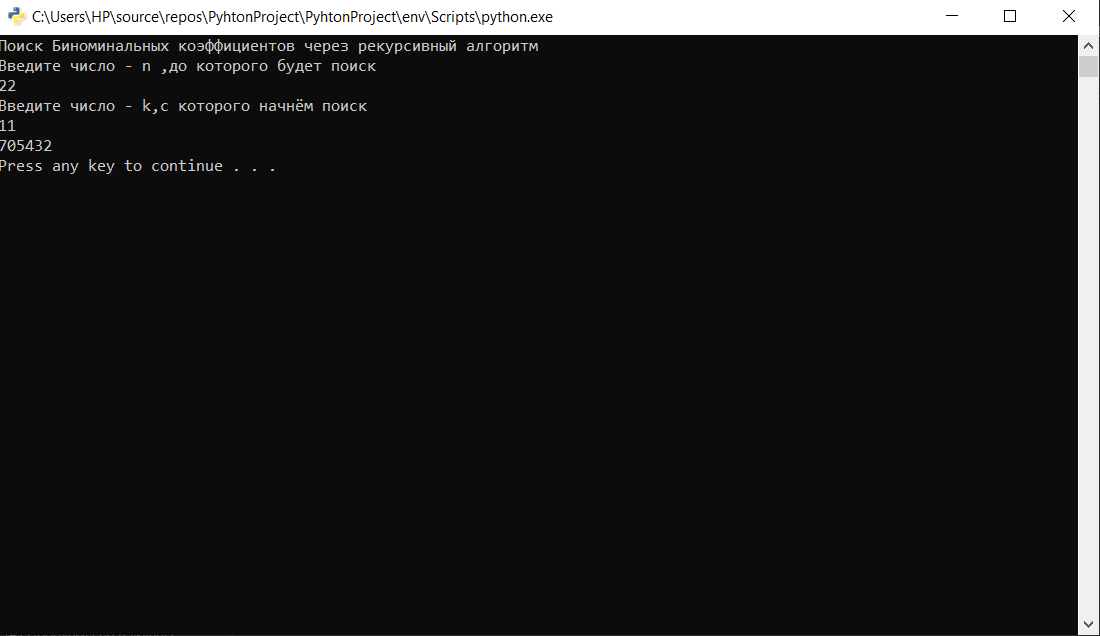
print(Comb(n,k))

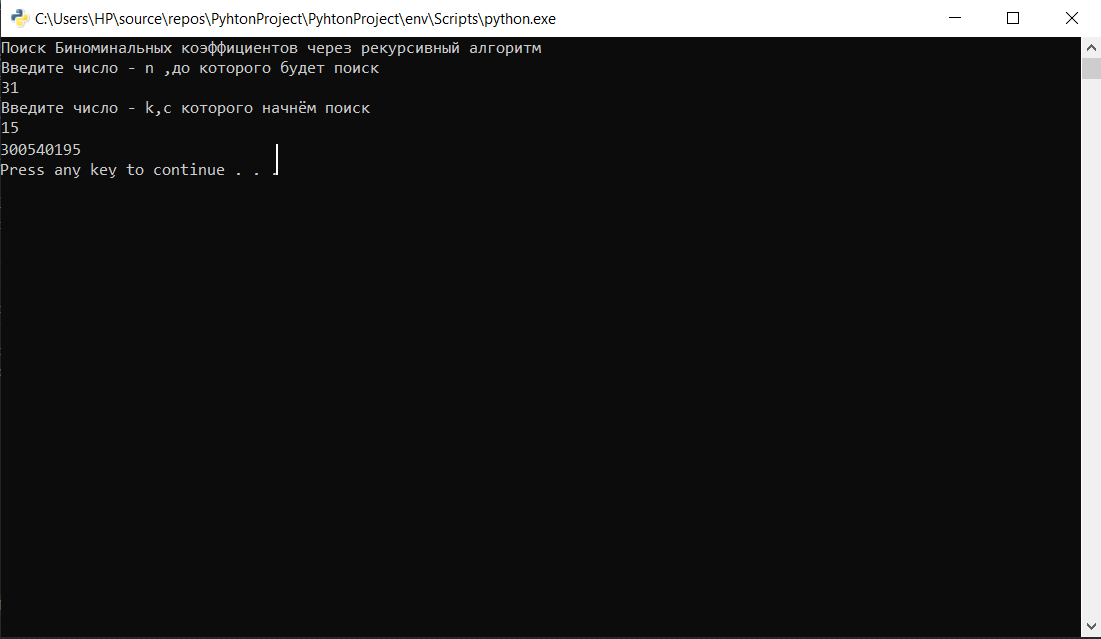
Скриншоты выполнения программы:











Вывод по решению задачи:

Результат решения, данной задачи аналогичен результату представленному выше, следовательно рекурсивный алгоритм работает исправно. И он чем то похож на факториальный алгоритм строкой return Comb(n-1,k)+Comb(n-1,k-1) ,которая похожа на факториал типа 6! – 6 \* 5 \* 4 \* 3 \* 2 \* 1.

**Формулировка задачи:**

1. Реализовать алгоритм с решением каждой подзадачи 1 раз с запоминанием результата

Алгоритм решения:

Функция bcr(принимаем n,k)

Если(k>n/2),то

K=n-k

Если(k=1),то

Return n

Если(k =0),то

Return 1

Return bcr(n-1,k)+bcr(n-1,k-1)

Основной запуск:

Ввод n,k

Печать(bcr(n,k))

Текст программы:

def bcr(n,k):

if (k>n/2): k=n-k;

if (k==1): return n;

if (k==0): return 1;

return bcr(n-1,k)+bcr(n-1,k-1);

print('Поиск Биноминальных коэффициентов с решением каждой подзадачи 1 раз с запоминанием результата ')

print('Введите число - n ,до которого будет поиск')

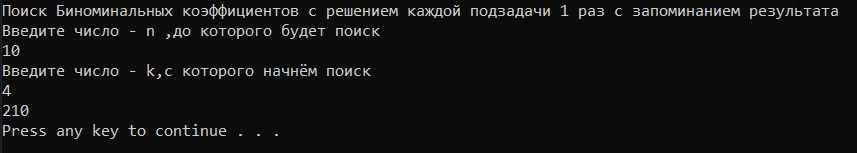
n = int(input())

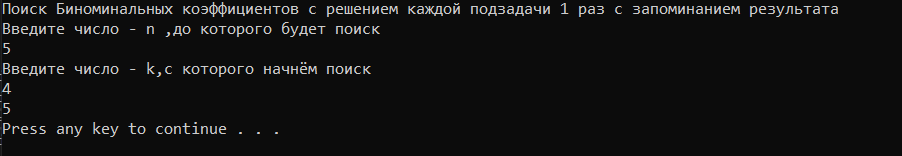
print('Введите число - k,с которого начнём поиск')

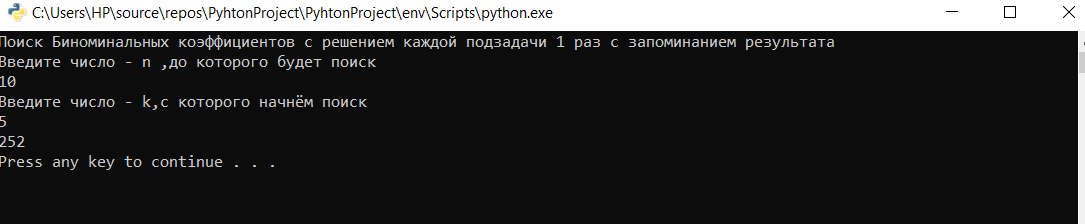
k = int(input())

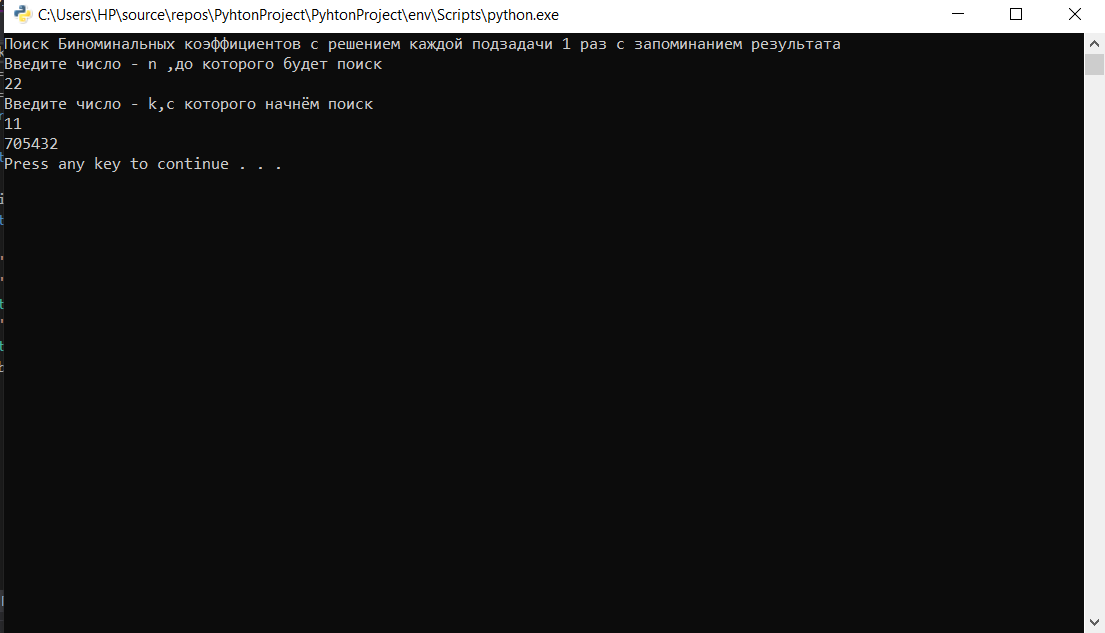
print(bcr(n,k))

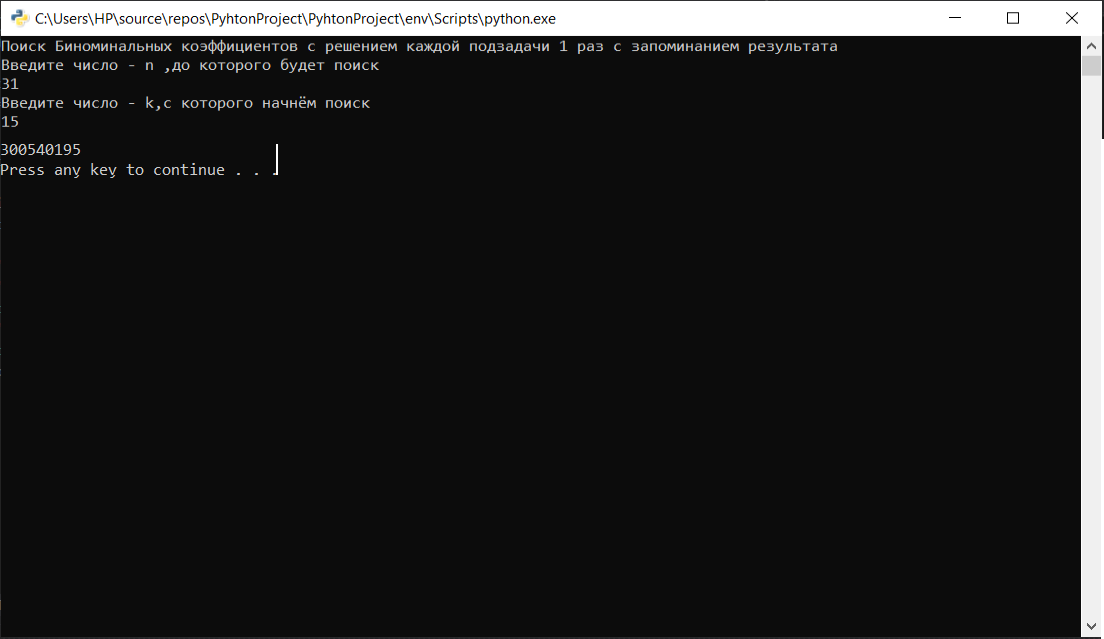
Скриншоты выполнения программы:











Вывод по решению задачи:

Благодаря методу bcr мы не имеем переполнение при вычислении крупных биноминальных коэфициентов так как в этом методе используется только сложение.

**Формулировка задачи:**

1. Реализовать алгоритм с решением данной задачи в матричном представлении и не посредственный вывод из матрицы результата.

Алгоритм решения:

Функция Binc (принимаем bcs,n ,k)

Если(k>n)

Return 0

Всё\_если

Если(k>n целая часть на 2)

K = n-k

Всё\_если

Если (k==0)

Return 1

Всё\_если

Если(k == 1)

Return n

Всё\_если

Пока len(bcs) < n-3

Для I от len(bcs) до n-3

R = []

Для j от 2^I Целая часть 2+3 c шагом 1

Добавить элемент в список r(Binc(bcs,i+3,j-1)+Binc(bcs,i+3,j))

Добавить в список bcs (r)

Всё\_для\_j

Всё\_пока len(bcs) < n-3

R = bcs[n-4]

Если (len(r) < k -1)

Для I от len(r) до k-1

Добавить элемент в список r(Binc(bcs,n-1,k-1)+Binc(bcs,n-1,k))

Всё\_для\_i

Всё\_если

Return bcs[n-4][k-2]

Основной запуск()

{

Bcs = []

Из time импортируем time

Start = time()

Печать(Binc(bcs,10,4))

End = time()

Печать(end-start)

Для n от 0 до 16  
 Печать(n)

Всё\_для\_n

Для k от 0 до n+1

Печать(Binc(bcs,n,k))

Всё\_для\_k

}

Текст программы:

def Binc(bcs,n,k):

if (k>n): return 0

if k>n//2: k=n-k

if k==0: return 1

if k==1: return n

while len(bcs)<n-3:

for i in range(len(bcs),n-3):

r=[]

for j in range(2,i//2+3):

r.append(Binc(bcs,i+3,j-1)+Binc(bcs,i+3,j))

bcs.append(r)

r=bcs[n-4]

if len(r)<k-1:

for i in range(len(r),k-1):

r.append(Binc(bcs,n-1,k-1)+Binc(bcs,n-1,k))

return bcs[n-4][k-2]

bcs=[]

from time import time

start = time()

print(Binc(bcs,10,4))

end = time()

print(end-start)

for n in range(0,16):

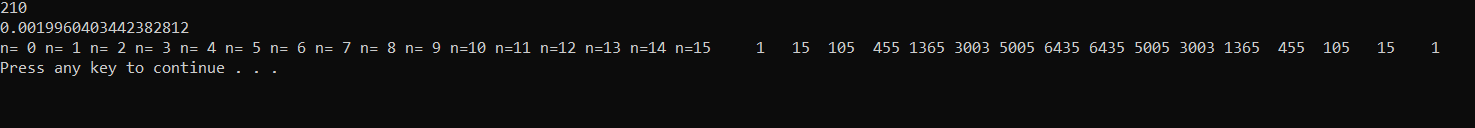
print("n=%2d " % n,end="")

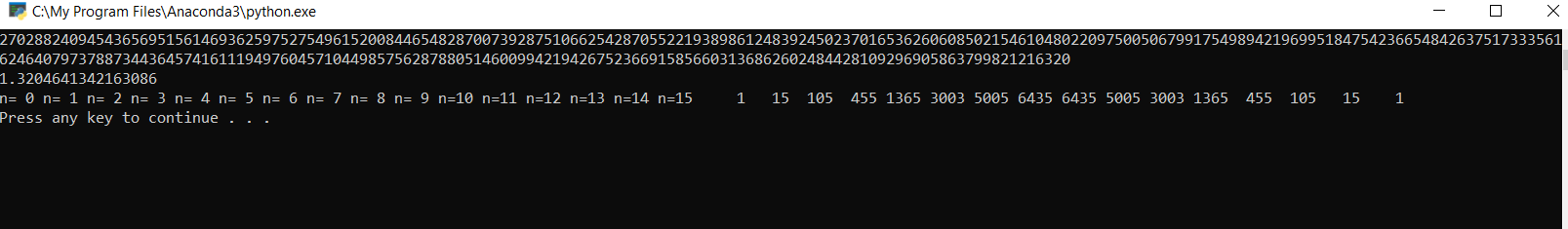
for k in range(0,n+1):

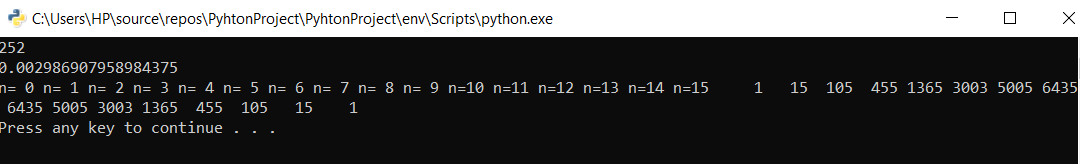
print("%5d" % Binc(bcs,n,k),end="")

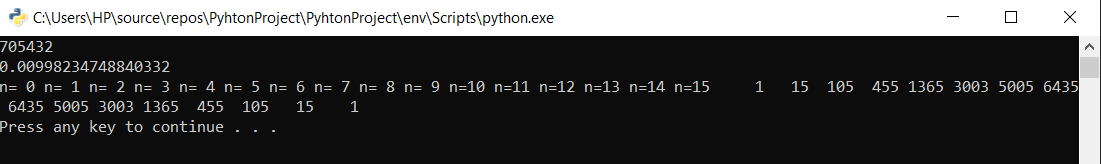
print()

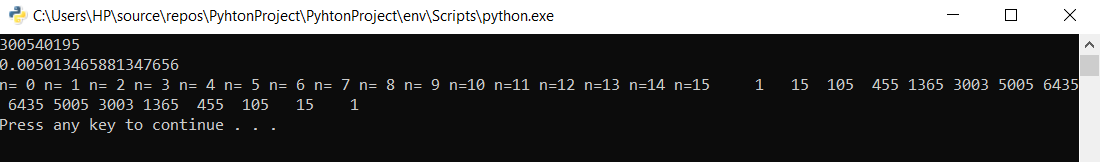
Скриншоты выполнения программы:











Вывод по решению задачи:

Благодаря методу Binc мы можем расчитывать биноминальные коэфициенты в пределах заданной n – количества элементов и k – самого коэфициента заносить все значения в список r и после печатать его на экран.

**Формулировка задачи:**

**Задача 2. Задача о наибольшем квадрате и наибольшем прямоугольнике**

1. Найти наибольший квадрат по алгоритму (реализовать программно).

Алгоритм решения:

Импортируем numpy с переменной np

Функция Solve(принимаем n,m)

A = np.array([[0, 0, 0,0,0], [0, 1, 1,1,1],[1,1,1,1,1],[1,1,1,1,1],[0,1,1,1,1],[1,0,1,1,1]])

B = np.array([[3,4],[2,3]])

Объявление целочисленных переменных I,j,l,k,max

Для i от 1 до n

B[I,1] = A[I,1]

Всё\_для

Для j от 1 до m

B[1,j] = A[1,j]

Всё\_для

Для I от 2 до n

Для J от 2 до m

Если(A[I,j] = 0),то

B[I,j] = a[I,J]

Всё\_если

Иначе:

B[I,j] = min(B[i-1,j],B[I,j-1],b[I-1,J-1])+1

MAX = 0

Для I от 1 до n

Для j от 1 до m

If(B[I,j] > max)

Max = B[I,j]

L = i

K = j

Всё\_для j

Всё\_для i

Всё\_иначе

Текст программы:

import numpy as np

def Solve(n,m):

A = np.array([[0, 0, 0,0,0], [0, 1, 1,1,1],[1,1,1,1,1],[1,1,1,1,1],[0,1,1,1,1],[1,0,1,1,1]])

B = np.array([[3,4],[2,3]])

i = int()

j = int()

l = int()

k = int()

max = int()

for i in range(1,n):

B[i,1] = A[i,1]

for j in range(1,m):

B[1,j] = A[1,j]

for i in range(2,n):

for j in range(2,m):

if (A[i,j] == 0):

B[i,j] = A[i,j]

else:

B[i,j] = min(B[i-1,j],B[i,j-1],B[i-1,j-1])+1

max = 0

for i in range(1,n):

for j in range(1,m):

if(B[i,j] > max):

max = B[i,j]

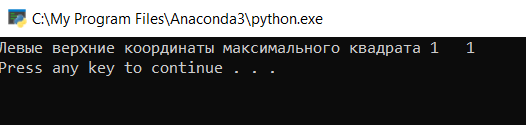
l = i

k = j

print('Левые верхние координаты максимального квадрата', l-max+1, ' ', k-max+1)

Solve(2,2)

Скриншоты выполнения программы:



Вывод по решению задачи:

Исходя из кода задачи, можно сделать вывод о том, что программа ищет максимально больший квадрат(макс 3x3) и после этого фиксирует его положение по формулам l-max+1, k-max+1.

**Формулировка задачи:**

Показать неработоспособность алгоритма 1 при поиске прямоугольника

Алгоритм решения:

Импортируем numpy с переменной np

Функция Solve(принимаем n,m)

A = np.array([[0, 0, 0,0,0], [0, 0, 0,0,0],[0,0,0,0,0],[0,0,0,0,0],[1,1,1,1,1],[1,1,1,1,1]])

B = np.array([[3,4],[2,3]])

Объявление целочисленных переменных I,j,l,k,max

Для i от 1 до n

B[I,1] = A[I,1]

Всё\_для

Для j от 1 до m

B[1,j] = A[1,j]

Всё\_для

Для I от 2 до n

Для J от 2 до m

Если(A[I,j] = 0),то

B[I,j] = a[I,J]

Всё\_если

Иначе:

B[I,j] = min(B[i-1,j],B[I,j-1],b[I-1,J-1])+1

MAX = 0

Для I от 1 до n

Для j от 1 до m

If(B[I,j] > max)

Max = B[I,j]

L = i

K = j

Всё\_для j

Всё\_для i

Всё\_иначе

Текст программы:

import numpy as np

def Solve(n,m):

A = np.array([[0, 0, 0,0,0], [0, 0, 0,0,0],[0,0,0,0,0],[0,0,0,0,0],[1,1,1,1,1],[1,1,1,1,1]])

B = np.array([[3,4],[2,3]])

i = int()

j = int()

l = int()

k = int()

max = int()

for i in range(1,n):

B[i,1] = A[i,1]

for j in range(1,m):

B[1,j] = A[1,j]

for i in range(2,n):

for j in range(2,m):

if (A[i,j] == 0):

B[i,j] = A[i,j]

else:

B[i,j] = min(B[i-1,j],B[i,j-1],B[i-1,j-1])+1

max = 0

for i in range(1,n):

for j in range(1,m):

if(B[i,j] > max):

max = B[i,j]

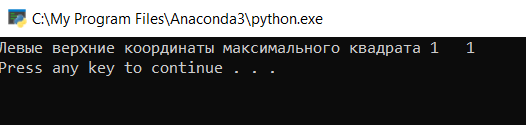
l = i

k = j

print('Левые верхние координаты максимального квадрата', l-max+1, ' ', k-max+1)

Solve(2,2)

Скриншоты выполнения программы:



Вывод по решению задачи:

Исходя из кода задачи, можно сделать вывод, о том, что во-первых при вводе матрицы 000 000 000 000 000 111 111 111 111 прямоугольник будет являться 111 111 111 111 ,но программа всё равно ограничена макс размером 3x3 и следовательно продолжает искать квадраты и находить их координаты по тем же формулам:  
l-max+1, k-max+1.

**Формулировка задачи:**

Реализовать алгоритм поиска наибольшего прямоугольника. Что здесь является подзадачей?

Алгоритм решения:

A = np.array([[0, 0, 0,0,0], [0, 0, 0,0,0],[0,0,0,0,0],[0,0,0,0,0],[1,1,1,1,1],[1,1,1,1,1]])

B = np.array([[3,4],[2,3]])

Объявление целочисленной переменной res

Для I от 1 до n

Для j от 1 до m

T = B[I,j]

K=i

Всё\_для j

Всё\_для i

Пока (k >= 0) и (A[k][j] > 0) и (t(i>res)

Если t\*(i-k+1) > res

Res = t\*(i-k+1)

K = k-1

T = min(t,B[k,j])

Всё\_пока

Печать(res)

Текст программы:

import numpy as np

def Solve(n,m):

A = np.array([[0, 0, 0,0,0], [0, 1, 1,1,1],[1,1,1,1,1],[1,1,1,1,1],[0,1,1,1,1],[1,0,1,1,1]])

B = np.array([[3,4],[2,3]])

res = 0

for i in range(1,n):

for j in range(1,m):

t = B[i,j]

k = i

while (k >= 0) and (A[k][j] > 0) and (t\*i>res):

if t\*(i-k+1) > res:

res = t\*(i-k+1)

k = k-1

t = min(t,B[k,j])

print(res)

Solve(2,2)

Скриншоты выполнения программы:



Вывод по решению задачи:

Исходя из примера из книги Окулова С. Мы имеем наибольший прямоугольник длинной 3, и действительно так оно и есть, существенное отличие данного алгоритма от предыдущего является то, что мы не используем здесь C т.к мы ищем площадь только наибольшего прямоугольника и в цикле While мы также добавляем строку t\*i>res ,чтобы сократить количество вычислении. Этот же кусок кода и является подзадачей данной задачи, он используется для уменьшения времени выполнения задачи:  
while (k >= 1) and (A[k,j]) and (t\*i>res):

if t\*(i-k+1) > res:

res = t\*(i-k+1)

k = k-1

t = min(t,B[k,j])

**Формулировка задачи:**

Реализовать алгоритм поиска наибольшего прямоугольника. Что здесь является подзадачей?

Алгоритм решения:

A = np.array([[0, 0, 0,0,0], [0, 1, 1,1,1],[1,1,1,1,1],[1,1,1,1,1],[0,1,1,1,1],[1,0,1,1,1]])

B = np.array([[3,4],[2,3]])

Объявление целочисленной переменной res

Res = 0

Для I от 1 до n

Для j от 1 до m

T = B[I,j]

K=i

Всё\_для j

Всё\_для i

Пока (k >= 1) и (t\*(i-k+1) > res)

K = k-1

Res = t\*(i-k+1)

T = min(t,B[k,j])

Всё\_пока

Печать(res)

Текст программы:

import numpy as np

def Solve(n,m):

A = np.array([[0, 0, 0,0,0], [0, 1, 1,1,1],[1,1,1,1,1],[1,1,1,1,1],[0,1,1,1,1],[1,0,1,1,1]])

B = np.array([[3,4],[2,3]])

res = 0

for i in range(1,n):

for j in range(1,m):

t = B[i,j]

k = i

while (k >= 1) and (t\*(i-k+1) > res):

k = k-1

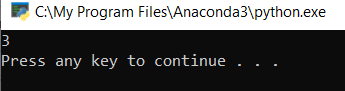
res = t\*(i-k+1)

t = min(t,B[k,j])

print(res)

Solve(2,2)

Скриншоты выполнения программы:



Вывод по решению задачи:

В цикле while убираем A[k,j] Из-за ненадобности так как оно всегда будет true, затем внутри цикла убираем условие если t\*(i-k+1) > res

И вставляем его в заголовок цикла и получаем в итоге оптимизацию цикла:  
while (k >= 1) and (t\*(i-k+1) > res):

k = k-1

res = t\*(i-k+1)

t = min(t,B[k,j])

print(res)