**Задача**

Даны N прямоугольников с вершинами в целочисленных координатах и M точек. Требуется как можно быстрее выдавать ответ на вопрос «Скольким прямоугольникам принадлежит точка x ,y?». Если точка находится на верхней или правой границе прямоугольника, считается, что она не принадлежит ему.

**Цель**

Реализовать 3 алгоритма: перебор, карта со сжатыми координатами, дерево отрезков. Определить, при каком объеме начальных данных и точек наиболее эффективен каждый из алгоритмов.

**Описание алгоритмов**

1. Полный перебор

Сложность алгоритма - О(N\*M), предварительная обработка данных отсутствует

Алгоритм состоит в переборе прямоугольников для каждой точки и определение принадлежности. Реализация: файл bruteforce.py

def contains(rect, p):  
 return rect.first.x <= p.x < rect.second.x and rect.first.y <= p.y < rect.second.y  
  
  
def bruteforce\_algo(points, rectangles):  
 result = []  
 for point in points:  
 k = 0  
 for rect in rectangles:  
 if contains(rect, point):  
 k += 1  
 result.append(k)  
 return result

1. Алгоритм на карте со сжатыми координатами

Сложность подготовки - O(N^3), сложность алгоритма - O(M\*logN)

Для каждой точки определяется положение на карте. Для этого требуется найти индексы координат точки. Ответ для каждой точки будет находиться в ячейке под полученными индексами. Реализация: файл map.py

def map\_algo(points, map, compressX, compressY):  
 answer = [0 for \_ in range(len(points))]  
  
 for i in range(len(points)):  
 posX = binsearch(compressX, points[i].x)  
 posY = binsearch(compressY, points[i].y)  
  
 if posX == -1 or posY == -1:  
 answer[i] = 0  
 else:  
 answer[i] = map[posY][posX]  
 return answer

1. Алгоритм на персистентном дереве отрезков

Подготовка требует O(N\*logN), алгоритм - O(M\*logN)

Для выдачи ответа нужно найти нужный корень для данной точке в массиве roots бинарным поиском нужного индекса в массиве compressY и осуществить спуск по дереву до нужного листа, считая в процессе количество встретившихся прямоугольников. Реализация - файл tree.py

def getAnswer(node, target):  
 if node is not None:  
 mid = (node.leftInd + node.rightInd) // 2  
 if target < mid:  
 return node.value + getAnswer(node.left, target)  
 else:  
 return node.value + getAnswer(node.right, target)  
 return 0  
  
  
def segtree(points, compressX, compressY, roots):  
 answer = [0 for \_ in range(len(points))]  
 if not roots:  
 return answer  
 for i in range(len(points)):  
 posX = binsearch(compressX, points[i].x)  
 posY = binsearch(compressY, points[i].y)  
  
 if posX == -1 or posY == -1:  
 answer[i] = 0  
 else:  
 answer[i] = getAnswer(roots[posX], posY)  
 return answer

В файле help.py определены классы Point, Rectangle, Node, Event, а также вспомогательные функции contains, binsearch, compress (для алгоритма 2 и 3)

**Тестирование и выводы**

Для тестирования были использованы сгенерированные наборы данных:

- набор вложенных друг в друга прямоугольников, имеющих координаты с шагом больше 1: {(10\*i, 10\*i), (10\*(2\*N-i), 10\*(2\*N-i))}

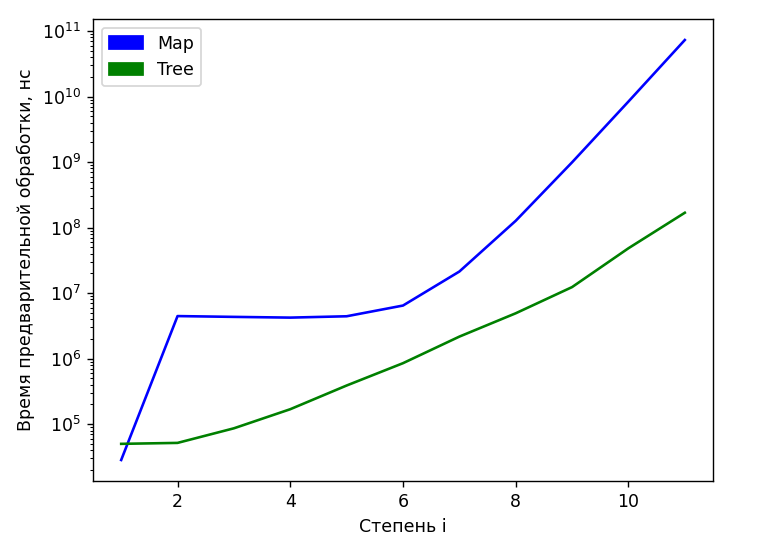
- неслучайный набор координат точек распределенных равномерно по ненулевому пересечению прямоугольников - хэш-функции от i: (p\*i)^31%(20\*N), p - большое простое, разное для x и y.

- Количество прямоугольников: 2^i; 1 <= i <= 11

- Количество точек: 100000

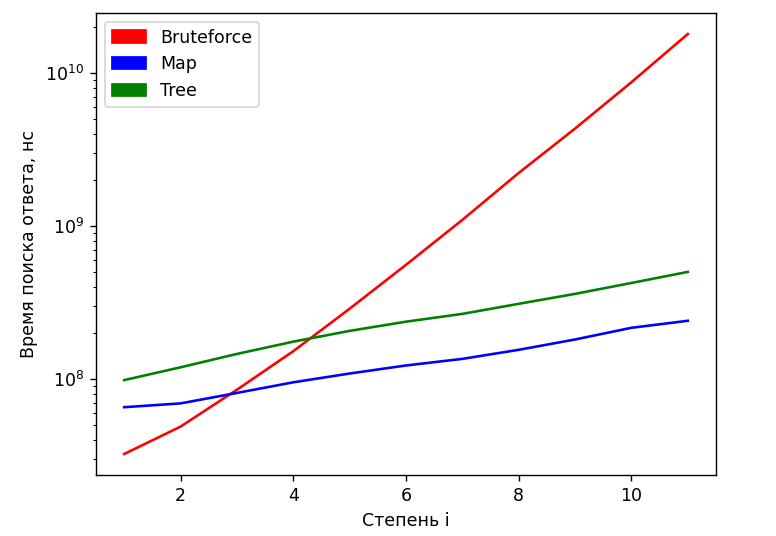
Были проведены сравнения: длительность предварительной обработки данных для двух алгоритмов, длительность работы самого алгоритма и длительность работы алгоритма с учетом предварительной обработки.

1. Время подготовки



Ожидаемо, длительность предварительной обработки O(N^3) для алгоритма на карте превышает длительность обработки O(N\*logN) для дерева в подавляющем большинстве случаев.

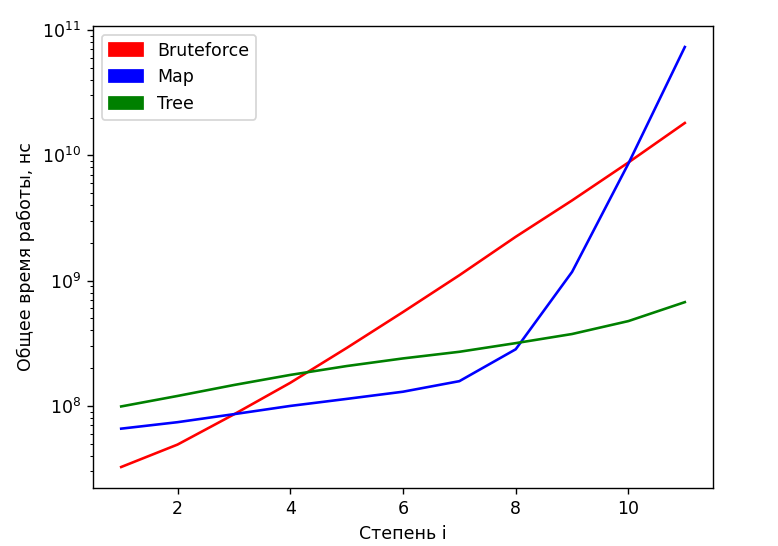
1. Время поиска ответа без учета обработки данных



На небольших входных данных алгоритмы работают за сопоставимое время, однако при увеличении степени i до 5 и более (количество прямоугольников >=32) время работы алгоритма полного перебора увеличивается, а у двух других - сохраняет тенденцию. Несмотря на одинаковую асимптотику O(M \* logN), алгоритм на карте быстрее справляется с задачей на готовых данных, так как алгоритм на дереве сначала использует дважды бинарный поиск для нахождения индексов x и y координат точки в массивах сжатых координат, а затем осуществляет спуск по дереву - константа перед логарифмом оказывается больше, чем у алгоритма на карте.

1. Общее время работы алгоритмов

Для этого измерения складываем числа, полученные в результате измерения времени работы алгоритмов на дереве и карте



Исходя из графика, алгоритм на карте серьёзно уступает двум другим на больших входных данных из-за большой асимптотики времени подготовки, а алгоритм на дереве успешнее всех справляется с поставленной задачей.

Данные для графиков:

'''  
Bruteforce time: [32537700, 49138200, 85565500, 152853200, 289744400, 561060700, 1103021000, 2236086400, 4362518100, 8765302000, 18098143800]  
Map prep time: [28400, 4461800, 4338400, 4228100, 4432400, 6462000, 21348400, 126978900, 991103800, 8384718500, 72856005300]  
Map time: [65803000, 69678900, 81564300, 95711000, 109265800, 123314900, 136176600, 156103100, 182348600, 217454900, 241616500]  
Tree prep time: [50100, 51800, 86500, 169200, 389100, 852800, 2168900, 4923900, 12364000, 48375300, 168489300]  
Tree time: [98944200, 119975900, 146683600, 176508700, 207538400, 238390100, 268080500, 311899400, 362305900, 426821600, 504317400]  
Tree total: [98994300, 120027700, 146770100, 176677900, 207927500, 239242900, 270249400, 316823300, 374669900, 475196900, 672806700]  
Map total: [65831400, 74140700, 85902700, 99939100, 113698200, 129776900, 157525000, 283082000, 1173452400, 8602173400, 73097621800]  
'''

**Вывод**

Исходя из полученных результатов, можно сказать, что полый перебор луяше всего использовать на маленьких данных без затрат дополнительной памяти и времени на подготовку, алгоритм на карте эффективен при малом количестве прямоугольников, а алгоритм на дереве, будучи самым сложным в реализации, стабильно обгоняет остальные на больших объёмах данных.