Домашнее задание №11

Томинин Ярослав, 778 группа

5 мая 2018 г.

1.

Алгоритм: при i=1 наш алгоритм записывает первое число в две суммы S_i и S_m. Рассмотрим наш алгоритм на i-ой итеррации. На ней мы уже знаем максимальную сумму, состоящюю из чисел не превосходящих i S_m, и максимальную сумму от j до i числа S_i(j-переменная).(Наши суммы еще хранят начальный и конечный элемент.) Во время i+1 итеррации мы рассматриваем i+1 число: в сумму S_i записываем $max(S_i, a_{i+1})$(если сумма меняется, то мы изменяем начальный и конечный элемент), а в S_m записываем $max(S_i, S_m)$(если сумма меняется, то мы изменяем начальный и конечный элемент). В конце нашего алгоритма в S_m будет лежать max сумма и номера начального и конечного элементов, из которых она состоит.

Корректность: Наш алгоритм на каждом шаге высчитывает max сумму, которую можно составить из элементов не больших, чем i и max сумму, которую можно составить из элементов не больших, чем i и которая содержит i элемент. Докажем это утверждение по индукции.

База: при i=1 наш алгоритм записывает первое число в две суммы и это не противоречит нашему утверждению.

Переход: допустим для i верно, тогда, если S_m поменялась, то она должна содержать i+1 элемент. Расшитаем max сумму, содержащую i+1 элемент: понятно, что S_i для предыдущей итеррации была либо положительна, либо отрицательна. Если она была положительна, то после i+1 итеррации S_i будет содержать все элементы, которые содержала S_i на i итеррации и i+1.(так как, если мы уберем i+1, то это будет max подпоследовательность); если же она была отрицательна, то S_i будет содержать один элемент i+1(в противном случае сумма без i+1 была бы отрицательна, так как она не превосходит S_i на i-ой итеррации). Теперь мы знаем, что S_m должна содержать i+1 элемент, а в S_i у нас уже хранится max такая подпоследовательность.

Следовательно наш инвариант сохраняется и мы действительно получим max сумму.

Сложность: Наш алгоритм на каждой итеррации делает const сравнений, сложений и изменений за O(1). Всего n итерраций. Поэтому сложность O(n).

2.

Алгоритм: посчитаем длину подпоследовательности для первых i членов последовательности x и для первых j членов последовательности y. Обозначим это за s(i,j). Тогда зная s(i-1,j-1),s(i-1,j) и s(i,j-1) мы можем найти s(i,j). Это будет max(s(i-1,j-1)+1,s(i-1,j), s(i,j-1)) при условии, что x[i]=y[j], иначе max(s(i-1,j-1),s(i-1,j), s(i,j-1)). То есть нам достаточно заполнить таблицу m*n.

Корректность: Алгоритм заполняет нулевой столбец и нулевую строку нулями, а потом вычисляет s(i,j). Докажем корректность такого вычисления по индукции.
База: Для s(0,0) условие выполняется.
Переход: Если для предыдущих найдено, то докажем, что s(i,j)=max(s(i-1,j-1)+1,s(i-1,j), s(i,j-1)) при условии, что x[i]=y[j], иначе max(s(i-1,j-1),s(i-1,j), s(i,j-1)). Если два последних числа равны, то наша подпоследовательность может содержать два этих числа(случай s(i-1,j-1)+1), одно из этих чисел или ни одного (случаи s(i-1,j) и s(i,j-1))
Время работы: Алгоритм заполняет таблицу m*n, каждая ячейка заполняется за константу. Поэтому результирующее время работы O(nm).
3.
1. Заполним массив, в которм на i-ом месте будет стоять 1, если можно выдать такую сумму, а 0, если ее нельзя выдать. Сразу ставим 1 на нулевое место. Тогда на i-ой ячейке все предыдущие уже будут известны, рассмотрим ячейки $i - v_1, i - v_2, ..., i - v_n$. Если хотя бы на одной стоит 1, то на нашей ячейке тоже будет стоять 1, иначе 0. Делаем так, пока не доберемся до s.
Корректность: докажем по индукции.
База: для i=0 стоит 1(так как мы можем выдать эту сумму).
Переход: понятно, что i должно содержать одну из n банкнот и если на всех местах $i-v_1, i-v_2, ..., i-v_n$ стоит 0, то это означает, что наша сумма не содержит ни одной из данных банкнот, поэтому мы не можем составить нашу сумму и в ней стоит 0.
Сложность по времени: Алгоритм на каждой ячейке просматривает n ячеек за O(n). Всего s ячеек, поэтому суммарная сложность O(nm).
2.
4.
Алгоритм: Проведем поиск в глубину и рассмотрим все уровни дерева, начиная с самого низкого. В каждой вершине будем записывать два числа: (A_i)минимальное покрытие поддерева этой вершины, (B_i)минимальное покрытие поддерева этой вершины, содержащее эту вершину. Тогда числа, записанные в вершине меньшего уровне будут находиться из ее предков:
A_i=min($A_{i-1} + A'_{i-1} + ...1, B_{i-1} + B'_{i-1}...$), B_i=$A_{i-1} + A'_{i-1} + ...1$(в сумме количество слагаемых равно количеству предков). Так найдем корень и закончим алгоритм.
Корректность: Докажем корректность нахождения чисел в вершине через ее предков.
База: в предках стоит 1,1;
Переход: Допустим в предках найдены числа корректно, докажем, что

и в нашей вершине они будут найдены корректно. Если все вершины предков входит в минимальное покрытие, то нашу вершину отмечать не нужно и мы запишем в нее A_i=B_{i-1} + B'_{i-1}..., B_i=A_{i-1} + A'_{i-1} + ...1; в противном случае(когда не все вершины будут входить в минимальное покрытие) выберем A_i=A_{i-1} + A'_{i-1} + ...1, B_i = A_{i-1} + A'_{i-1} + ...1
Это исчерпывает все случаи и мы находим минимальное между этими возможными случаями.
Сложность по времени: На каждой вершине мы высчитываем два числа за O(k-1), где к-степень вершины. Тогда общая сложность алгортима O(E).
5.
1) В каждый момент времени мы можем хранить одну строку и их будет хватать для того, чтобы вычислить следующую, когда она будет заполнена, то мы удаляем левую строку из памяти. Это займет O(n) памяти(можно в каждый момент хранить n+1, но ассимтотика будет той же)
2)Введем новое обозначение: матрица E- это матрица, которая находит минимальное расстояние суффиксов. Тогда возьмем минимальную строку S_1(длина m), разделим ее на пополам и высчитаем m/2 строку матрицы изначальной и m/2 строку матрицы E. Тогда минимальное расстояние будет min(i-ого элемента строки нашей матрицы + N-i элемента строки матрицы E).
3)Введем новое обозначение: матрица E- это матрица, которая находит минимальное расстояние суффиксов. Тогда возьмем минимальную строку S_1(длина m), разделим ее на пополам и высчитаем m/2 строку матрицы изначальной и m/2 строку матрицы E. Тогда минимальное расстояние будет min(i-ого элемента строки нашей матрицы + N-i элемента строки матрицы E). После этого мы уже вызываем рекурсию от левого верхнего и правого нижнего квадратов. Тогда при рекурсивных вызовах наша память будет N+N/2+N/4+...=2N. При этом наша ассимтотика будет такой же, как и прежде, так как мы вычисляем две матрицы за O(nm).

6.
Сделаем задачу следующим образом: рассмотрим две суммы из трех(это будет вершина S/3 и S/3 в нашей плоскости). Возьмем a_1. Оно либо лежит в первой, либо во второй, либо в третьей суммах. В нашей плоскости мы будем вычитать это число по горизонтали и ставить в эту клетку 1, по вертикали и ставить в нее 1 и оставаться на месте. Так же мы будем хранить массив из открытых точек(из которых мы можем "ходить"). На i-ой итеррации мы будем вычитать a_i по горизонтали и вертикали из каждой открытой вершины. Поймем, что вершины не могут закрываться, так как каждый раз мы можем оставаться в ней. Поэтому максимальное

количество открытых вершин будет не больше, чем n^2. Так же каждая открытая вершина будет хранить своего предка. Если этот предок находится слева, то наше число принадлежит первой сумме, если сверху, то второй. Каждый раз мы рассматриваем всевозможные варианты принадлежности числа всем трем сумма, поэтому других ситуаций быть не может. Поэтому если наши числа закончатся и вершина (0,0) не будет открыта, то мы не можем разложить на такие суммы, иначе мы сможем востановит две суммы с помощью предков, а третья выберется остаточным способом.

Сложность по времени: Максимальное количество открытых вершин $(S/3)^2$, всего чисел n, каждый раз мы тратим $O((S/3)^2)$ и так мы повторяем не более n раз. Результирующая сложность $O((S/3)^2 n)$.