A. Тысяча резисторов

Автор задачи: Владимир Мшанецкий
Количество сдавших команд: 3 из 20 (15%)

Задача решается полным перебором. То есть, нужно попробовать взять каждый резистор, переложить его на каждое свободное место и в каждом случае проверить получившееся выражение. Если выражение получилось правильным, то ответ найден, иначе — продолжаем перебор. Но на практике проще реализовать такой перебор, работая не с отдельными резисторами, а с трансформациями целых символов.

При перекладывании одного резистора возможны два отдельных случая: резистор перемещается с одного места на другое внутри одного и того же символа, или резистор забирается из одного символа и добавляется в другой символ. Назовём трансформацию, применяемую к символу в первом случае, трансформацией перемещения, а трансформации, применяемые во втором случае, — трансформацией забирания и добавления соответственно. Путём внимательного изучения картинок из условия (и получения нескольких WA) можно составить полный список всех трансформаций, в результате которых получается корректный символ:

Исходный символ	После перемещения	После забирания	После добавления
0	6, 9		8
1			7
2	3		
3	2, 5		9
4			
5	3		6, 9
6	0, 9	5	8
7		1	
8		0, 6, 9	
9	0, 6	3, 5	8
+	=	-	
-	/		+, =
x		/	
/	-		x
=	+	-	

Трансформации, в результате которых получается некорректный символ, нас, очевидно, не интересуют. Кроме того, можно не рассматривать трансформацию «=» в «+», так как в результате её применения всегда будет получаться выражение, в котором нет знаков равенства, что недопустимо. Трансформации перемещения нужно рассматривать отдельно, так как, например, трансформацию «2» в «3» невозможно разложить на трансформацию забирания и трансформацию добавления, потому что не существует корректного промежуточного символа.

С выражением удобнее всего работать как со строкой, а допустимые трансформации удобно хранить в нескольких (константных) `std::map<char, std::vector<char>>` для каждого типа трансформации. Будем перебирать символы исходного выражения, перебирать и применять к ним трансформации, а уже затем — разбирать получившуюся строку на числа и знаки операций, выполнять эти операции и проверять результат.

Перебор можно реализовать следующим образом. Сначала попробуем использовать только

трансформации перемещения. Для каждого символа исходной строки выполним следующее:

1. запомним текущий символ как исходный
2. для каждой допустимой трансформации перемещения для исходного символа:
    1. заменим текущий символ в строке на полученный в результате трансформации
    2. проверим выражение, которое получилось в изменённой строке
3. восстановим исходный символ

Если в результате какой-то проверки выражение оказалось правильным, следует сразу остановить перебор и вывести ответ.

Далее попробуем использовать в паре трансформации забирания и добавления. Это делается аналогично, только будет четыре вложенных цикла:

1. по символу, к которому мы применяем трансформацию забирания
2. по символу, к которому мы применяем трансформацию добавления
3. по применяемой трансформации забирания (из допустимых для текущего символа из первого цикла)
4. по применяемой трансформации добавления (из допустимых для текущего символа из второго цикла)

Во втором цикле следует пропускать символ, который является текущим в первом цикле (то есть, не следует пытаться добавить резистор в тот же символ, из которого его забрали).

Разбор получившегося выражения с целью проверки можно реализовать несколькими способами. Например, с помощью `std::sscanf(str, "%d%c%d%c%d", ...)`, с помощью `std::istringstream` (путём последовательного считывания `int`, `char`, `int`, `char` и `int`), с помощью регулярных выражений или вручную с помощью нескольких циклов `while` по индексам. Если разобрать выражение не получилось, его можно сразу считать некорректным и продолжать перебор.

Вычисление и проверку выражения можно реализовать с помощью «лесенки» из 9 `if-elseif` по допустимым парам знаков операций. При вычислении выражения следует обратить внимание на два момента: во-первых, в выражении может быть деление на ноль, а во-вторых, хотя оба операнда гарантированно помещаются в 32-битное целое, результат умножения в него может не помещаться, поэтому при умножении следует использовать 64-битные целые. При проверке следует в случае деления не забыть также проверить, что остаток равен нулю.

Если, перебрав все возможные варианты трансформации исходного выражения, мы так и не нашли ответ, то ответа не существует.

B. Законопроект

Автор задачи: Дмитрий Садовый
Количество сдавших команд: 5 из 20 (25%)

Чтобы решить задачу, давайте выпишем несколько наблюдений:

- Каждый депутат получит деньги в сумме A_i, B_i либо C_i.
- Поскольку $C_i \le A_i \le B_i$, то всегда выгодно голосовать под влиянием, так как C_i меньше остальных. Тогда, если в дереве есть вершина i, которая публично выступает, то мы всегда выбираем C_i для всех вершин поддерева вершины i.

Теперь давайте обойдем дерево в глубину и будем для каждой вершины считать два значения:

- $dp(i,0)$ — минимальная сумма денег, которую нужно заплатить, чтобы дерево с корнем в i проголосовало за законопроект, при этом выше по дереву не было публичных выступлений (среди предков вершины i)
- $dp(i,1)$ — минимальная сумма денег, которую нужно заплатить, чтобы дерево с корнем в i проголосовало за законопроект, при этом выше по дереву было публичное выступление

Обозначим множество прямых потомков вершины i как $\text{to}(i)$. Тогда:

$$dp(i,0) = min(private_i, public_i)$$
$$private_i = A_i + \sum_{t \in \text{to}(i)} dp(t,0)$$
$$public_i = B_i + \sum_{t \in \text{to}(i)} dp(t,1)$$

Далее подсчитаем $dp(i,1)$. Тут всё проще — просто суммируем все C_i по всему поддереву:

$$dp(i,1) = C_i + \sum_{t \in \text{to}(i)} dp(t,1)$$

Ответом на задачу будет $dp(1,0)$.

C. Труби

Автор задачі: Іван Фекете
Здало команд: 6 із 20 (30%)

Оберемо якийсь фрагмент із трубою і допустимо, що ми змогли розв'язати задачу для фрагмента, який знаходиться від нього лівіше, і фрагмента, який знаходиться вище. Тоді ми знаємо інформацію про те, скільки і звідки відкритих кінців труб направлені в наш фрагмент зліва і зверху. А ця інформація дозволяє нам однозначно визначити, як має бути повернута труба, або сказати, що не існує повороту труби, який розв'яже задачу.

Тому розв'язок цієї задачі буде виглядати наступним образом. Запустимо обхід по рядках згори донизу, а у кожному рядку — по всіх фрагментах труб зліва направо. Якщо якийсь фрагмент є непустим, то ми спробуємо перебрати поворот цієї труби і обрати такий, що у фрагменті, який вище від нас, фрагменті, який лівіше від нас і в поточному фрагменті не було відкритого кінця труби. Якщо в нас не існує повороту труби, який відповідає цим вимогам, то і відповіді не існує, тому що для попередніх труб ми змогли знайти відповідь, і вона є однозначною. А якщо існує, то ми цей поворот фіксуємо у якості результатного і продовжуємо із наступною клітинкою. Додатково потрібно врахувати те, що в кожному рядку і стовпчику остання труба не має містити відкритих кінців.

Складність: $O(nm)$.

D. Lift

Автор задачи: Александр Нестерюк
Количество сдавших команд: 13 из 20 (65%)

Задача решается исходя из предположения, что на заданный этаж можно попасть при помощи лестницы, либо спустившись, либо поднявшись с одного из ближайших этажей, на которых имеется остановка лифта. Также возможен случай, когда лифт останавливается на требуемом этаже. При этом ближайший снизу этаж будет $K-((K-1) mod M)$. Соответственно, количество этажей, на которые необходимо поднять мебель будет: $K-(K-((K-1) mod M))$. Для случая, когда лифт остановится на этаже, расположенном выше требуемого, номер его вычисляется следующим образом: $K-((K-1) mod M)+M$. Если полученный таким образом номер этажа превышает количество этажей в здании L, то тогда данный вариант переноса мебели необходимо исключить из рассмотрения, ограничившись только вариантом подъема. Соответственно, количество этажей, на которые необходимо спустить мебель будет: $(K-((K-1) mod M)+M)-K$. Далее, стоимости подъема и спуска одной единицы мебели соответственно будут: $300 \cdot (K-(K-((K-1) mod M)))$ и $150 \cdot ((K-((K-1) mod M)+M)-K)$. Для всей мебели, умножив на T, соответственно, получаем: $T \cdot 300 \cdot (K-(K-((K-1) mod M)))$ и $T \cdot 150 \cdot ((K-((K-1) mod M)+M)-K)$. При желании формулы можно сократить. Из двух полученных вариантов (при условии, что спуск возможен) выбираем минимальный по стоимости и к полученному результату добавляем стоимость вызова бригады N.

E. Число атомов

Автор задачи: Денис Остапенко
Количество сдавших команд: 14 из 20 (70%)

В задаче нужно было подсчитать общее количество атомов в молекуле, заданной химической формулой. По условию данной задачи химическая формула состоит из последовательности одно- или двух-буквенных обозначений атомов, за каждым из которых может следовать число — сколько раз этот атом встречается в молекуле. Если числа нет, то подразумевается 1. Нужно найти сумму всех этих чисел.

Для подсчёта количества атомов нужно обработать строку по символам слева направо. Также понадобится использовать временную переменную `num` для накопления текущего числа. Изначально она будет равна 0. Для каждого символа строки выполним следующее. Если он является цифрой (0-9), то добавим его к текущему числу по формуле `num = num * 10 + (str[i] - '0')`. Если текущий символ является последним или символ, следующий за текущим, является большой буквой (A-Z), то увеличим ответ на `max(num, 1)`, после чего присвоим `num = 0`. Маленькие буквы никак обрабатывать не нужно.

F. Kindergarten

Автор задачи: Леонид Беркович
Количество сдавших команд: 10 из 20 (50%)

Составить новый набор картинок, состоящий из определённого количества картинок, можно в том случае, когда количество возможных наборов такого размера больше количества уже использованных наборов такого размера. Рассмотрим способ подсчёта количества возможных наборов для каждого размера.

Поскольку в наборе можно использовать не более одной картинки из каждой категории, количество картинок в наборе равно количеству разных категорий в этом наборе. Множество всех возможных сочетаний по `cats` ($1 \le cats \le Ncat$) разных категорий — это множество сочетаний из `Ncat` по `cats`. Такие сочетания можно получить последовательным добавлением одной категории. Для каждого из этих сочетаний можно составить некоторое число наборов картинок. Для вычисления я использовал следующую функцию, которая возвращает число наборов картинок для вектора сочетаний `vec`.

```
int multcats(vector <int> vec) {
    int mcats = 1;
    for (int i = 0; i < vec.size(); i++)
        mcats *= cats[vec[i]];
    return mcats;
}
```

Здесь `cats[i]` — это количество разных картинок в категории `i`.

Например, для сочетания из двух категорий $(1,3)$ можно составить `cats[1] * cats[3]` наборов.

Полученные таким образом числа (количество возможных наборов по каждому размеру набора) сохраним в массиве `catscount`.

Количества использованных наборов получим в цикле ввода данных, обрабатывая значения `Npics` в начале каждой строки и складывая суммарные значения по размерам наборов в массив `catsused`.

Рассмотрим возможность составления нового набора картинок в порядке возрастания количества используемых категорий. Для этого достаточно в цикле последовательно сравнивать число возможных наборов картинок с числом использованных наборов (для одной категории, для двух и так далее). Если на какой-то итерации цикла число возможных наборов окажется больше числа использованных, то ответ найден и можно выходить из цикла. Если после окончания цикла ответ так и не был найден, то ответ — -1.

G. Три белых коня

Автор задачи: Денис Остапенко
Количество сдавших команд: 14 из 20 (70%)

Подсчитать количество клеток, находящихся под атакой коней, можно следующим образом. Переберём все 64 клетки шахматного поля (на самом деле, будем перебирать просто координаты этих клеток, строку и столбец) и для каждой клетки проверим, что в ней нет коня, но в одной из 8 клеток, из которых можно попасть в данную ходом коня, есть конь. Если это так, то увеличим ответ на 1. Перебрав все клетки, выведем ответ.

Рассмотрим подробно одну из возможных реализаций данного алгоритма. Для начала следует считать координаты коней и преобразовать их из строк вида «b4» в пары чисел, такие как $(2,4)$ или $(1,3)$, если считать от 0. Это можно сделать с помощью вычитания кодов символов. Например, если координата была прочитана в строку `s`, то `s[0] - 'a'` будет номером столбца, а `s[1] - '1'` будет номером строки (оба номера — считая от 0). Можно сохранить координаты коней в таком виде в массив пар целых (`std::vector<std::pair<int, int>>`) или просто в 6 переменных (если, конечно, вы хотите дальше копипастить кучу проверок).

Далее следует в двух вложенных циклах перебрать координаты (r,c) всех клеток поля от $(0,0)$ до $(7,7)$. Для каждой клетки сначала проверим, что в ней нет коня (то есть, в массиве с координатами коней нет такой координаты). Затем вычислим координаты восьми клеток, из которых конь может пойти в данную:

1. $(r-2,c-1)$
2. $(r-2,c+1)$
3. $(r+2,c-1)$
4. $(r+2,c+1)$
5. $(r-1,c-2)$
6. $(r+1,c-2)$
7. $(r-1,c+2)$
8. $(r+1,c+2)$

Для каждой из вычисленных координат проверим что в соответствующей клетке есть конь (то есть, что координата встречается в массиве с координатами коней). Если это так, то увеличим ответ на 1.

Обработку 8 клеток можно, опять таки, реализовать с помощью Control+C, Control+V, или сделав константный массив разностей координат и перебрав его элементы в цикле:

```
const std::vector<std::pair<int, int>> moves = {
    { -2, -1 },
    { -2, +1 },
    ...,
    { -1, +2 },
    { +1, +2 }
};
```

Альтернативное решение

Для подсчёта клеток, находящихся под атакой, можно для каждого коня вычислить координаты всех клеток, которые он атакует, и добавить их все в `std::set<std::pair<int, int>>`. Добавлять следует только координаты, не вышедшие за границы поля. Затем следует удалить из получившегося множества координаты самих коней. Размер полученного после этого множества будет ответом.

H. Обхід міст

Автор задачі: Іван Фекете
Здало команд: 10 із 20 (50%)

Шлях у графі, який обходить всі вершини, і в кожну з них заходить не більше 1 разу, в теорії графів називається гамільтоновим. В загальному випадку задача про кількість гамільтонових шляхів у графі є NP-повною і може бути розв'язана лише повним перебором цих шляхів. Але відповідь на дану задачу можна знайти швидше. Ключем до розв'язку даної задачі є те, що потрібно вивести не саму кількість шляхів, а її залишок від ділення на 2, тобто сказати, парне це число чи непарне. Для цього достатньо згадати просту властивість неорієнтованого графа, а саме те, що якщо існує шлях з вершини u до вершини v, то існує і зворотній шлях з вершини v у вершину u. Отже, для кожного шляху існує йому протилежний шлях, що означає, що кількість таких шляхів є парною. Єдиний виняток із цієї ситуації — це ситуація, коли в графі всього 1 вершина, тоді шлях складається з однієї вершини, і вищенаведені міркування для нього не є справедливими, тому для цього випадку відповідь — 1.

Окремо слід зазначити, що більшість вхідних даних в цій задачі можна навіть не зчитувати, достатньо зчитати лише перший рядок із кількістю міст. А отже, складність рішення — $O(1)$.

I. Манхэттенский склад

Автор задачи: Кирилл Левенец
Количество сдавших команд: 7 из 20 (35%)

Формула суммы расстояний, значение которой нужно минимизировать, имеет следующий вид:

$$\min_{x_s, y_s} \sum_{i=1}^{N} \left(|x_s - x_i| + |y_s - y_i| \right)$$

Поскольку формула раскладывается на два независимых слагаемых, оптимизировать функцию можно независимо по X и Y. Рассмотрим сумму X-ов (для Y-ков всё будет аналогично).

Отсортируем X_i по возрастанию. Пусть X_s расположено таким образом, что слева от неё k точек, а справа — $(N-k)$. Рассмотрим, как изменится сумма расстояний, если передвинуть X_s на 1 вправо. Она увеличится на k, так как расстояние от каждой из k точек слева увеличится на 1 (если X_s совпадает с некоторым X_i, будем считать, что этот X_i находится слева), и уменьшится на $(N-k)$, так как расстояние до каждой из $(N-k)$ точек справа уменьшится на 1. Таким образом, общая сумма изменится на $k-(N-k)=2k-N$. Из этого видно, что сдвигать X_s вправо выгодно, пока $2k<N$, то есть $k<N/2$. Следовательно, минимум будет при $k=N/2$, то есть при X_s равном медиане X_i.

Более строго можно доказать это следующим образом. Представим значение целевой функции в виде:

$$S_{X_s} = \sum_{i=1}^{k} (X_s - X_i) + \sum_{j=k+1}^{N} (X_j - X_s) = -\sum_{i=1}^{k} X_i + kX_s + \sum_{j=k+1}^{N} X_j - (N-k)X_s =$$

$$= -\sum_{i=1}^{k} X_i + \sum_{j=k+1}^{N} X_j + (2k-N)X_s$$

Вычислим производную по определению:

$$\lim_{\delta \to 0} \frac{S_{X_s+\delta} - S_{X_s}}{\delta} = \frac{-\sum_{i=1}^{k} X_i + \sum_{j=k+1}^{N} X_j + (2k-N)(X_s+\delta) - \left(-\sum_{i=1}^{k} X_i + \sum_{j=k+1}^{N} X_j + (2k-N)X_s\right)}{\delta} =$$

$$= \frac{(2k-N)(X_s+\delta) - (2k-N)X_s}{\delta} = \frac{(2k-N)\delta}{\delta} = 2k-N$$

Знак производной будет вести себя следующим образом:

$$\begin{cases} <0, k<N/2 \\ =0, k=N/2 \\ >0, k>N/2 \end{cases}$$

Следовательно, точкой оптимума будет точка, для которой слева и справа одинаковое количество точек, то есть медиана.

Итоговый алгоритм:

1. сортируем X_i по возрастанию
2. берем центральный элемент массива $X_{N/2}$ как координату склада X_s (если четное число точек — то подойдет любая из двух центральных)
3. аналогично с Y_i и Y_s
4. вычисляем сумму расстояний от каждой точки до склада — это и будет ответ

J. Перепутанные биты

Автор задачи: Денис Остапенко
Количество сдавших команд: 0 из 20 (0%)

В задаче требовалось найти сообщение длиной $N=(A+B)/8$, которое является лексикографически K-ым из всех, удовлетворяющих условию по количеству битов-нулей и битов-единиц. Идея решения заключается в следующем. Представим, что у нас есть выписанная в столбик в лексикографическом порядке последовательность из всех возможных сообщений длиной N с заданным количеством нулей и единиц. Нам нужно, двигаясь от начала последовательности, пропустить $K-1$ сообщение и вывести K-ое. Пусть, мы сможем определить по-отдельности общие количества $n_a, n_b, n_c, \ldots, n_x, n_y, n_z$ сообщений из этой последовательности, которые начинаются с символов «a», «b», «c», …, «x», «y» и «z» соответственно. Это значит, что в нашей последовательности сначала идёт n_a сообщений, начинающихся с «a», затем — n_b сообщений, начинающихся с «b», затем — n_c сообщений, начинающихся с «c», и так далее. Зная это, мы легко можем определить первый символ искомого сообщения. Действительно, найдётся такой символ m_1, что $n_a+n_b+\ldots+n_{m_1-1}\le K-1$, а $n_a+n_b+\ldots+n_{m_1-1}+n_{m_1}>K-1$ (иными словами, мы увеличиваем m_1, пока не получится, что увеличив его ещё раз, мы бы в сумме пропустили больше, чем $K-1$ сообщений). Далее, уже зная первый символ, мы можем свести исходную задачу к задаче поиска сообщения длиной $N-1$. Для этого нужно уменьшить A и B на количество нулей и единиц в m_1 соответственно, а K — на количество уже пропущенных нами сообщений из нашей воображаемой последовательности, то есть на $n_a+n_b+\ldots+n_{m_1-1}$. Продолжая действовать таким образом, мы можем по одному символу найти искомое сообщение. Осталось только найти способ подсчитать числа $n_a, \ldots, n_z$, не генерируя всю последовательность.

Пусть $f(n,b)$ — общее количество сообщений длиной n символов, в которых есть ровно b битов-единиц (и, соответственно, $8n-b$ битов-нулей). Используя эту функцию, мы сможем вычислять $n_x=f(N-1, B-oneBits(x))$. Считаем, что $f(0,0)=1$, $f(0,b)=0$ для всех остальных b и $f(n,b)=0$ для всех $b<0$. Подсчитать все значения $f(1,b)$ для каждого b можно, перебрав все символы от «a» до «z» и подсчитав, сколько из них содержат ровно b единиц. Для всех остальных n значения $f(n,b)$ можно подсчитать, основываясь на значениях для $n-1$:

$$f(n,b)=\sum_{c='a'}^{'z'} f(n-1, b-oneBits(c))$$

Поскольку функция постоянно вызывает себя рекурсивно, её вычисление следует реализовать не как функцию, а как двумерный массив, который следует один раз заполнить с помощью цикла. В целом такой подход к вычислению $f(n,b)$ называют динамическим программированием. Заметим, что $f(n,b)$ никак не зависит от входных данных задачи. Следовательно, её можно заранее подсчитать для всех n от 1 до 12 и b от 0 до $13\cdot 8$ (значения $f(13,b)$ нам никогда не понадобятся).

Построение искомого сообщения с использованием функции $f(n,b)$ выполняется следующим образом. Пусть $k=K-1$. Для каждого i от 1 до N выполним следующее:

1. пусть $m_i='a'$
2. будем увеличивать m_i, пока не дойдём до «z» или не окажется, что $f(N-i, B-oneBits('a'))+f(N-i, B-oneBits('b'))+\ldots+f(N-i, B-oneBits(m_i))>k$
3. выведем m_i в ответ
4. уменьшим k на $f(N-i, B-oneBits('a'))+f(N-i, B-oneBits('b'))+\ldots+f(N-i, B-oneBits(m_i-1))$
5. уменьшим B на $oneBits(m_i)$

K. Обеспокоенные Онагадори

Автор задачи: Максим Молчанов
Количество сдавших команд: 3 из 20 (15%)

Ответ всегда существует при четном N независимо от длин хвостов, потому что тогда можно направить хвосты попарно друг на друга. Ответ будет выглядеть так: RLRLRL...RLRLRL.

Остались случаи, где N нечетно. Ответ всегда существует, если хотя бы на одной нечетной позиции есть длина хвоста больше единицы. Предположим, мы нашли такую позицию и ее номер k. Тогда слева и справа от этой позиции будут последовательности четной длины. Можно последовательность слева расставить таким же шаблоном, как при четном N, а справа направить хвосты в одном направлении, за исключением позиции $k+1$. Начиная с позиции k ответ будет выглядеть так: RLRRRRRR...RRR. Если $k=N$, можно сделать то же самое, но в обратную сторону: LLL...LLLRL.

Остались случаи с нечетным N и единицами на нечетных местах, то есть последовательности вида 1 a_2 1 a_4 1 ... 1 a_{N-3} 1 a_{N-1} 1. Ответ существует, только если существует хотя бы одна пара i, j такая, что $i<j$, $i+a_i>j$ и $j-a_j<i$. То есть, существует пара таких хвостов, что, если повернуть их в сторону друг друга, они дотягиваются каждый хотя бы на метр дальше, чем друг до друга. Тогда можно направить хвосты следующим образом: i и j в сторону друг друга, то что между ними — в любую сторону, то что слева от i — влево, то что справа от j — вправо.

Поиск описанных выше пар можно реализовать с помощью дерева отрезков со сложностью решения $O(N\log(N))$, но лучше искать их с помощью стека со сложностью $O(N)$, используя идею, схожую с «минимумом в окне». Верхушка стека всегда будет играть роль i из условий выше. Для каждого j от 1 до N сначала будем извлекать из стека верхушку, пока стек не пуст и $i+a_i\le j$, то есть, пока не выполняется первое условие. Потом, если стек не пустой, проверяем, выполняется ли для верхушки второе условие: $j-a_j<i$. Если да, то мы нашли ответ. Если нет, то добавляем j в стек и продолжаем.

Вот пример теста, в котором ответа нет, но увеличение любого числа (кроме 19) хотя бы на единицу даст ответ:

```
19
1 19 1 2 1 4 1 2 1 8 1 2 1 4 1 2 1 16 1
```

L. Watchdog

Автор задачи: Александр Нестерюк
Количество сдавших команд: 0 из 20 (0%)

Задача решается исходя из предположения, что минимальная длина цепи сторожевой собаки при заданных ограничениях будет составлять максимальное из кратчайших расстояний от точки её привязки до вершин заданного многоугольника. При этом кратчайшее расстояние, по которому пройдет цепь от точки привязки до рассматриваемой вершины многоугольника, будет проходить по отрезкам, соединяющим точку привязки и одну из видимых из нее вершин, а также, возможно, одного или нескольких отрезков, соединяющих между собой две вершины при условии, что они взаимно видимы между собой (не закрыты препятствием в виде забора или другой вершины). При этом, исходя из условий задачи, необходимо учитывать только отрезки, находящиеся внутри многоугольника. Также набор данных отрезков будет включать в себя и стороны многоугольника. В результате задача сводится к нахождению минимальных путей на построенном таким образом графе от точки привязки до всех остальных вершин, где вес дуги соответствует расстоянию между вершинами графа. В дальнейшем из всех найденных расстояний выбираем наибольшее.

В начале решения потребуется массив точек — вершин многоугольника `Wi(xi, yi)` размерностью N. Также в этом массиве можно сохранить дополнительно (увеличив его размерность на 1) и точку привязки `Wd(x, y)`. Также вещественный массив `Ri` размерностью N (или $N+1$) для хранения длин путей, который инициализируем максимально большим числом `INF`. Кроме этого понадобится два массива, в которых будем хранить индексы координат отрезков `Ai`, `Bi` из массива `W` максимальной размерностью `int((N + 1) * ((N + 1) / 2.0 - 0.5))`.

На первом этапе проводим заполнение массивов `W`, `R`. Также можно при этом проверить частный случай — нахождение точки привязки на вершине многоугольника. Далее проводим заполнение массивов `A` и `B` элементами `Ai` и `Bi` — индексами координат вершин многоугольника (получаем его стороны). Также необходимо учесть, что последняя вершина соединяется с первой. Далее продолжаем заполнение массивов `A` и `B` индексами координат возможных отрезков. Сначала рассмотрим видимость вершин (каждая из каждой, однако также проверяем отрезки на уникальность, то есть считая отрезки `Ai-Bi` и `Bi-Ai` одним отрезком). Первая проверка — на пересечения со сторонами многоугольника. Для этого воспользуется уравнением прямой:

$$\frac{x-x_1}{x_1-x_2}=\frac{y-y_1}{y_1-y_2}$$

Отсюда вычисляем:

$$den=(y_4-y_3)(x_1-x_2)-(x_4-x_3)(y_1-y_2)$$

Если $den=0$, то тогда отрезки пересекаются, если

$$(x_1y_2-x_2y_1)(x_4-x_3)-(x_3y_4-x_4y_3)(x_2-x_1)=0$$

и $$(x_1y_2-x_2y_1)(y_4-y_3)-(x_3y_4-x_4y_3)(y_2-y_1)=0$$

Иначе — не пересекаются.

Если $den\neq0$, вычисляем:

$$a=(x_4-x_2)(y_4-y_3)-(x_4-x_3)(y_4-y_2)$$
$$b=(x_1-x_2)(y_4-y_2)-(x_4-x_2)(y_1-y_2)$$
$$U_a=a/den$$
$$U_b=b/den$$

Тогда отрезки пересекаются, если $U_a\geq0$, $U_a\leq1$, $U_b\geq0$ и $U_b\leq1$, иначе — не пересекаются.

Все вычисления следует выполнять в целых числах, иначе могут быть проблемы с точностью. Обратите внимание на разрядность, требуемую для вычислений — в соответствии с ограничениями здесь необходимы переменные длиной 64 разряда.

Также перед добавлением отрезка необходимо проверить не проходит ли он за пределами многоугольника полностью. Для этого находим координаты точки средины отрезка: `X = (Wi.x + Wj.x) / 2`, `Y = (Wi.y + Wj.y) / 2`. После этого проводим перпендикуляр и вычисляем количество сторон, с которым он пересечется. Для четного пересечения — отрезок лежит за

пределами многоугольника, иначе — внутри многоугольника.

Аналогичные проверки проводим и для точки привязки Wd. При этом необходимо учесть случай нахождения точки привязки на одной из сторон многоугольника (если использовать один обработчик и обрабатывать её как обычную вершину графа).

На втором этапе вычислений потребуется вещественный массив, состоящий из элементов Ci, размерностью NG, равной количеству получившихся ребер в полученном графе, в котором сохраним вычисленную длину дуг графа: `Ci = sqrt(pow(W[Ai].x - W[Bi].x, 2) + pow(W[Ai].y - W[Bi].y, 2))`.

На третьем этапе проводим поиск наикратчайших путей из точки привязки в каждую из вершин путем заполнения массива R, начиная от точки привязки (`R[N] = 0`, если точка привязки находится в общем массиве W на месте N). Получаем код вида:

```
for(int i = 0, iF = 1; iF && (i < N); i++) {
    iF = 0;
    for(int j = 0; j < NG; j++) {
        if(R[A[j]] < INF) {
            R[B[j]] = min(R[B[j]], R[A[j]] + C[j]);
            iF = 1;
        }
        if(R[B[j]] < INF) {
            R[A[j]] = min(R[A[j]], R[B[j]] + C[j]);
            iF = 1;
        }
    }
}
```

На последнем этапе проводим поиск максимального элемента массива R, значение которого и выводим с заданной точностью.