

Муниципальный этап Всероссийской олимпиады
школьников по информатике
в 2014–2015 учебном году
11 класс

Время выполнения задач — 4 часа
Ограничение по времени — 2 секунды на тест
Ограничение по памяти — 64 мегабайта

Муниципальный этап
Всероссийской олимпиады школьников
по информатике

в 2014–2015 учебном году

Разборы решений и идеи тестов

11.1. «Модель Эйфелевой башни». Петя Торопыжкин решил построить модель Эйфелевой башни, для чего ему нужно купить p металлических прутков, стоимостью r рублей каждый. В n -й день от начала покупок ($n \geq 1$) родители выдают ему на карманные расходы $k(n) = an^2 + bn + c$ рублей, из которых он максимальное количество тратит на покупку заготовок, а оставшуюся неистраченную сумму помещает в копилку. Если коэффициенты таковы, что в какой-то день значение $k(n)$ отрицательно, то это означает, что в этот день ему карманных денег не дали, и он ничего не покупает и ничего в копилку не кладёт. Сколько денег окажется в копилке по окончании покупки нужного количества заготовок?

Формат входа: В первой строке через пробел заданы целые числа p и r ($1 \leq p, r \leq 1000$). Во второй строке через пробел заданы три целых числа a, b, c ; $a > 0$, каждое число по модулю не превосходит 1000.

Формат выхода: Выведите единственное целое число — финальные петины накопления.

Пример 1		Пример 2	
<u>Вход:</u>	<u>Выход:</u>	<u>Вход:</u>	<u>Выход:</u>
16 5	7	17 5	23
1 -14 45		1 -14 45	

Примечание: В примерах деньги даются Пете с 1-го по 4-й дни и затем с 10-го и далее. В первом примере он осуществляет закупку точно на 11-й день. Во втором примере ему нужно купить ещё одну заготовку на 12-й день, что он и делает, помещая оставшиеся деньги ($21 - 5 = 16$ рублей) в копилку в дополнение к тем 7-ми, что там есть на конец 11-го дня.

11.2. «Неубывающая последовательность». Имеется упорядоченный набор из n целых чисел, не превосходящих по модулю 1000. Петя Торопыжкин может прибавлять к любому числу любую неотрицательную величину. Он хочет достичь того, чтобы этот набор представлял собой неубывающую последовательность (каждый последующий член должен быть не меньше предыдущего). Каково наименьшее значение суммы величин, которые необходимо прибавить к элементам набор для достижения этой цели?

Формат входа: В первой строке задано целое число n ($1 \leq n \leq 1000$). В следующей строке через пробел заданы n целых чисел, по модулю не превосходящих 1000.

Формат выхода: Выдайте единственное целое неотрицательное число — сумму величин, которые надо добавить к членам набора, чтобы получить неубывающий набор.

Пример

<u>Вход:</u>	<u>Выход:</u>
5	6
2 5 1 3 8	

11.3. «Странный максимум». Пете Торопыжкину разонравились нечётные цифры (то есть цифры 1, 3, 5, 7 и 9). И теперь он сравнивает неотрицательные целые числа следующим образом: сначала из чисел вычёркиваются все нечётные цифры, затем вычёркиваются ведущие нули (если они появились), наконец, если от числа ничего не осталось, то оно полагается равным нулю. После такого «прореживания» числа сравниваются по обычным правилам. Если два числа получились равными, то Петя выбирает в качестве большего первое из исходных чисел. Напишите программу, которая находила бы максимум из набора чисел по петиним правилам сравнения. При поиске максимума в наборе каждое последующее число сравнивается как второе с максимумом из предыдущих чисел.

Формат входа: В первой строке задано целое число n — количество чисел ($1 \leq n \leq 1000$). В следующих n строках по одному на строке записаны числа, каждое из которых состоит не менее чем из одной и не более чем из 255 цифр.

Формат выхода: Выведите единственную строку, содержащую максимальное (в петиним смысле) число из данного набора.

Пример

<u>Вход:</u>	<u>Выход:</u>
4	597631
31595	
1997053	
597631	
96135	

11.4. «Прохождение игрушки». Во время, свободное от учёбы и решения задач по информатике, Петя Торопыжкин иногда играет. В том числе и в стрельки. Однажды он решил аккуратно проанализировать свой любимый шутер. В нём имеется n уровней. Для прохождения k -го уровня требуется b_k патронов. В начале игры игрок обладает s патронами. Для восстановления запаса в конце k -го уровня можно открыть секретную комнату и дополнительно получить a_k патронов, которые можно использовать только на следующих уровнях. Однако это

требует времени, и Петя хочет сократить посещение секретных комнат, насколько это возможно. Соответственно, игрушка считается успешно пройденной, если перед началом каждого из уровней запас патронов у игрока не менее чем требуемое количество. Помогите Пете, напишите программу, которая по имеющимся данным об игре укажет какое минимальное количество секретных комнат надо будет открыть и на каких именно уровнях.

Формат входа: В первой строке через пробел указаны два целых числа s , начальный запас патронов, и n , количество уровней в игре ($0 \leq s \leq 10^4$, $1 \leq n \leq 10^5$). В следующих n строках приведена информация о каждом из уровней игры — пара чисел b_k и a_k , расход патронов на k -м уровне и дополнительный запас патронов на k -м уровне ($0 \leq b_k, a_k \leq 10^4$).

Формат выхода: В первой строке выведите количество уровней, на которых необходимо открывать секретную комнату для пополнения запаса патронов. Если количество таких уровней больше нуля, во второй строке через пробел выведите номера этих уровней в любом порядке. Если решений несколько, выведите любое из них. Если прохождение игры невозможно, выведите единственное число -1.

Пример 1

<u>Вход:</u>	<u>Выход:</u>
100 2	0
10 10	
10 10	

Пример 2

<u>Вход:</u>	<u>Выход:</u>
10 3	1
10 20	1
10 10	
10 10	

Пример 3

<u>Вход:</u>	<u>Выход:</u>
10 3	-1
10 10	
20 10	
10 10	

11.5. «Организация связи». Кроме стрельлок Петя Торопыжкин иногда играет и в стратегические игры. В одной из них важным является организация связи между опорными пунктами. При очередном запуске этой игры Пете понадобилось обеспечить двустороннюю связь между пунктами A и B , расположенными на прямой. Для этого он может устанавливать ретрансляционные станции в некоторых из оговоренных точек C_i , расположенных на той же прямой. На каждой станции он должен предусмотреть два узконаправленных передатчика, транслирующих сигнал в направлении от A к B и от B к A . Кроме, конечно, станций, расположенных в самих пунктах A и B : там нужно установить только по одному передатчику, вещающему в соответствующую сторону. Всего имеется m видов передатчиков; передатчик j -го вида имеет дальность вещания d_j , а его установка стоит r_j денежных единиц. Естественно, Петя желает установить связь, потратив наименьшее количество денежных ресурсов. Сможет ли он организовать связь в указанных условиях и, если да, то в какую сумму ему это обойдётся?

Формат входа: В первой строке заданы два целых числа x_A и x_B — координаты пунктов A и B . Во второй строке задано число n — количество промежуточных точек C_i ($0 \leq n \leq 3000$). В третьей строке в каком-то порядке через пробел заданы координаты x_i точек C_i . В четвёртой строке задано число m — количество

типов передатчиков ($1 \leq m \leq 3000$). В следующих m строках описываются параметры каждого вида передатчиков — пары целых чисел d_j и r_j ($1 \leq d_j \leq 10^6$, $0 \leq r_j \leq 10^4$). Все координаты являются целыми числами, по модулю не превосходящими 10^5 . Гарантируется, что никакие две точки $C_{i'}$ и $C_{i''}$ не совпадают друг с другом и не совпадают с пунктами A и B .

Формат выхода: Выведите единственное целое число — наименьшую стоимость установленных передатчиков, обеспечивающих двустороннюю связь между пунктами A и B . Если связь установить невозможно, выведите -1 .

Пример 1		Пример 2	
<u>Вход:</u>	<u>Выход:</u>	<u>Вход:</u>	<u>Выход:</u>
0 10	80	0 10	-1
3		3	
2 6 9		2 6 9	
3		3	
2 10		2 10	
5 15		3 15	
1 8		1 8	

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2014–2015 учебном году 11 класс. Разбор решений и идеи тестов

Далее в разборах используются обозначения типов: short int — 2-байтовый целый тип (integer в TurboPascal/BorlandPascal и int в BorlandC++), int — 4-байтовый целый тип (longint в TurboPascal/BorlandPascal/FreePascal, long int в BorlandC++, integer в Delphi/PascalABC и int в VisualC++/C++ Builder), int64 — 8-байтный целый (отсутствует в BorlandPascal и BorlandC++, но присутствует в системах программирования под Windows).

Задачи оцениваются по 100 баллов за каждую; стало быть, полная стоимость пакета — 500 баллов. В рамках одной задачи все тесты считать равноценными.

11.1. «Модель Эйфелевой башни». *Петя Торопыжский решил построить модель Эйфелевой башни, для чего ему нужно купить p металлических прутков, стоимостью r рублей каждый. В n -й день от начала покупок ($n \geq 1$) родители выдают ему на карманные расходы $k(n) = an^2 + bn + c$ рублей, из которых он максимальное количество тратит на покупку заготовок, а оставшуюся неистраченную сумму помещает в копилку. Если коэффициенты таковы, что в какой-то день значение $k(n)$ отрицательно, то это означает, что в этот день ему карманных денег не дали, и он ничего не покупает и ничего в копилку не кладёт. Сколько денег окажется в копилке по окончанию покупки нужного количества заготовок?*

С точки зрения программного комитета, данная задача является лёгкой, поскольку подразумевает прямое вычисление

Идеи тестов:

1. вся закупка осуществляется в первый день, лишних денег нет;
2. вся закупка осуществляется в первый день, лишние деньги — только сдача;
3. вся закупка осуществляется в первый день, лишних денег много;
4. вся закупка осуществляется к концу второго дня, лишних денег нет ни в первый день, ни во второй;
5. вся закупка осуществляется к концу второго дня, лишних денег нет в первый день, но есть во второй (сдача);
6. вся закупка осуществляется к концу второго дня, лишних денег нет в первый день, но есть во второй (много);
7. вся закупка осуществляется к концу второго дня, лишние деньги есть в первый день, но их нет во второй;
8. вся закупка осуществляется к концу второго дня, лишние деньги есть и в первый день (сдача), и во второй (сдача);

9. вся закупка осуществляется к концу второго дня, лишние деньги есть и в первый день (сдача), и во второй (много);
- 10–18. то же, что в тестах 1–9, только в начале идёт период, когда Пете денег не дают;
- 19–25. тесты, когда в середине есть промежутки, когда Пете денег не дают.

11.2. «Неубывающая последовательность». Имеется упорядоченный набор из n целых чисел, не превосходящих по модулю 1000. Петя Торопыжкин может прибавлять к любому числу любую неотрицательную величину. Он хочет достичь того, чтобы этот набор представлял собой неубывающую последовательность (каждый последующий член должен быть не меньше предыдущего). Каково наименьшее значение суммы величин, которые необходимо прибавить к элементам набора для достижения этой цели?

Сложность данной задачи, по мнению программного комитета, ниже средней: над идеей решения надо поразмышлять, реализация является весьма простой.

Решение строится на следующем наблюдении: если максимум среди элементов массива с 1-го по i -й равен m_i , а $(i + 1)$ -й элемент a_{i+1} меньше m_i , то к нему надо добавить величину $m_i - a_{i+1}$ для сохранения неубывания. Соответственно, реализация состоит в том, чтобы, перебирая элементы массива один за одним, вычислять максимум перебранной части классическим алгоритмом поиска максимума и суммировать величины, прибавляемые для сохранения неубывания:

```
curMax := a[1]; /* начальное значение максимума равно a[1];
                  к первому элементу ничего добавлять не надо */
toAdd := 0; /* начальное значение добавляемых величин */
for i := 2 to n do begin
  if a[i] < curMax
  then toAdd := toAdd + curMax - a[i]
  else curMax := a[i];
end;
```

Идеи тестов:

1. минимальный тест, $n = 1$;
2. $n = 10$, последовательность постоянная;
3. $n = 10$, последовательность возрастающая;
4. $n = 10$, последовательность неубывающая;
5. $n = 10$, последовательность невозрастающая;
6. $n = 10$, последовательность убывающая;
- 7–9. $n = 10$, случайные тесты;
- 10–17. то же, что и в тестах 2–9, $n = 100$;
- 18–25. то же, что и в тестах 2–9, $n = 1000$ — максимальные тесты.

11.3. «Странный максимум». Пете Торопыжкину разонравились нечётные цифры (то есть цифры 1, 3, 5, 7 и 9). И теперь он сравнивает неотрицательные целые числа следующим образом: сначала из чисел вычёркиваются все нечётные цифры, затем вычёркиваются ведущие нули (если они появились), наконец, если от числа ничего не осталось, то оно полагается равным нулю. После такого «прореживания» числа сравниваются по обычным правилам. Если два числа получились равными, то Петя выбирает в качестве большего первое из исходных чисел. Напишите программу, которая находила бы максимум из набора чисел по петиным правилам сравнения. При поиске максимума в наборе каждое последующее число сравнивается как второе с максимумом из предыдущих чисел.

Данная задача представляется имеющей среднюю сложность, поскольку подразумевает активную работу со строками.

Алгоритм решения может быть следующим. Вначале читаем числа со входа числа в строчном виде, обрабатывая их: запоминаем исходную строку числа, строку–результат его прореживания по петиному алгоритму и длину строки–результата. Затем ищем максимум (по строке–результату) в наборе по классическому алгоритму, единственно, для этого понадобится процедура сравнения чисел, заданных строками. Впрочем, она не слишком сложна: сначала сравниваем длины (которые запомнены после первичной обработки входных данных). Если длины не равны, то большее число найдено: это то, которое имеет большую длину. Если же длины равны, то строчные записи чисел можно сравнивать как строки. Если и строчные представления чисел совпадают, то по условию задачи в качестве максимума выдаём первый объект.

Идеи тестов:

1. минимальный тест, $n = 1$, число состоит только из нечётных цифр;
2. минимальный тест, $n = 1$, после выкидывания нечётных цифр, образуются ведущие нули, но имеются и значащие чётные цифры;
3. минимальный тест, $n = 1$, после выкидывания нечётных цифр, остаются только нули;
- 4–6. то же, что и в тестах 1–3, только $n = 2$, в рамках одного теста все числа одинакового типа;
- 7–9. то же, что и в тестах 1–3, только $n = 10$, в рамках одного теста все числа одинакового типа;
- 10–12. то же, что и в тестах 1–3, только $n = 1000$, в рамках одного теста все числа одинакового типа;
- 13–20. случайные тесты, длины чисел меньше 255;
- 21–25. случайные тесты, есть числа с длиной 255.

11.4. «Прохождение игрушки». Во время, свободное от учёбы и решения задач по информатике, Петя Торопыжкин иногда играет. В том числе и в стрелялки. Однажды он решил аккуратно проанализировать свой любимый шутер.

В нём имеется n уровней. Для прохождения k -го уровня требуется b_k патронов. В начале игры игрок обладает s патронами. Для восстановления запаса в конце k -го уровня можно открыть секретную комнату и дополнительно получить a_k патронов, которые можно использовать только на следующих уровнях. Однако это требует времени, и Петя хочет сократить посещение секретных комнат, насколько это возможно. Соответственно, игрушка считается успешно пройденной, если перед началом каждого из уровней запас патронов у игрока не менее чем требуемое количество. Помогите Пете, напишите программу, которая по имеющимся данным об игре укажет какое минимальное количество секретных комнат надо будет открыть и на каких именно уровнях.

Программный комитет считает, что эта задача имеет сложность выше средней. Идея решения достаточно прямолинейна, однако аккуратная реализация, претендующая на полный балл, требует дополнительных знаний в области структур данных.

Алгоритм решения состоит в последовательном переборе уровней игры с учётом текущего количества оставшихся патронов. Если перед очередным уровнем патронов на его прохождение недостаточно, это значит, что нужно (было) открыть секретную комнату на одном из предыдущих уровней. Если уж мы вскрываем секретную комнату, то следует вскрыть ту, в которой больше всего патронов; находим среди предыдущих уровней такой, на котором секретная комната ещё не открыта и содержит наибольшее количество патронов. Открываем её, делая соответствующую пометку, добавляем запас патронов из неё к нашему запасу и запоминаем номер этого уровня. Если патронов по-прежнему недостаточно для прохождения очередного уровня, повторяем поиск секретной комнаты. Если нам нужны дополнительные патроны, а неоткрытых комнат на предыдущих уровнях не осталось, то пройти игру невозможно, останавливаем цикл по уровням и сообщаем о неудаче. Если цикл по уровням успешно завершился, то прохождение возможно, о чём следует сообщить и выдать накопленный набор уровней, на которых вскрывались секретные комнаты.

Видно, что центральным местом алгоритма является процедура поиска уровня из предыдущих, на котором секретная комната ещё не открыта и содержит наибольшее количество патронов. «Наивный» поиск, связанный с честным перебором всех предыдущих уровней для поиска максимума, при большом общем количестве уровней может не укладываться в ограничения по времени. То есть нам нужна структура, которая позволяет организовать быстрый поиск максимума по своим элементам. Такая структура известна и называется *упорядоченное множество* (в англоязычной литературе — *sorted set* или просто *set*). (Не следует путать этот тип данных с типом **set** из языка Паскаль!) Упорядоченное множество хранит (как следует из названия) упорядоченный набор попарно различных элементов. В нашем случае это может быть пара (*количество патронов в секретной комна-*

те, номер уровня). Порядок элементов — лексикографический, то есть сначала происходит сравнение первых элементов по количеству патронов; при равенстве первых элементов пар сравниваются вторые (которые заведомо различны в нашем случае). Такая структура данных позволяет быстрые (за время $O(\log k)$, где k — количество элементов в хранилище) операции вставки, удаления, поиска элемента, поиска минимального и максимального элементов (в смысле используемого порядка). Стало быть, сложность алгоритма, опирающегося на упорядоченное множество, составляет $O(n \log n)$: суммарное количество вставок, так же, как и суммарное количество удалений, не превосходит n , а каждая из операций имеет сложность $O(\log n)$.

Таким образом, алгоритм имеет следующий вид:

1. подготовка основного цикла:
`set := ∅` — информация о секретных комнатах на предыдущих уровнях;
`levels := ∅` — информация об уровнях, на которых открывали комнаты;
`ammo := s` — текущий запас патронов равен начальному запасу;
2. цикл по уровням, k меняется от 1 до n
3. готовимся пойти на следующий уровень:
`while (ammo <  $b_k$ ) && (set ≠ ∅)`
(собираем патроны, пока надо и пока есть, откуда их брать)
`(maxAmmo, maxLevel) ←max set;`
(открываем самую богатую комнату и удаляем соответствующую информацию из `set`)
`levels ← maxLevel;` (запоминаем уровень, где открыли комнату)
`end while`
4. если `ammo <  $b_k$` , то не смогли набрать патронов на следующий уровень, выходим, сообщаем о невозможности пройти игру
5. `ammo := ammo -  $b_k$` ; (тратим патроны на прохождение уровня)
6. `set ← ( $a_k, k$ )`; (запоминаем информацию о комнате на текущем уровне)
7. конец цикла по уровням
8. игру прошли, сообщаем об уровнях, на которых открывали комнаты (используем `levels`)

Тесты подобраны таким образом, что участники, построившие своё решение на «наивном» алгоритме поиска секретных комнат, получают 60% от максимального балла.

Идеи тестов:

1. минимальный тест, $n = 1$, начального запаса патронов хватит, чтобы пройти единственный уровень;
2. минимальный тест, $n = 1$, начального запаса патронов не хватит, чтобы пройти единственный уровень;
3. $n = 2$, начального запаса патронов хватит, чтобы пройти оба уровня;

4. $n = 2$, начального запаса патронов не хватит, чтобы пройти оба уровня; секретная комната даст достаточное количество патронов, чтобы пройти второй уровень;
5. $n = 2$, начального запаса патронов не хватит, чтобы пройти оба уровня; секретная комната не даст достаточного количества патронов, чтобы пройти второй уровень;
6. $n = 10$, начального запаса хватит, чтобы пройти всю игру;
7. $n = 10$, начального запаса не хватит, чтобы пройти всю игру; секретные комнаты дадут достаточное количество;
8. $n = 10$, начального запаса не хватит, чтобы пройти всю игру; секретные комнаты не дадут достаточного количества;
- 9–14. случайные тесты, $n < 10000$;
- 15–25. случайные тесты, $n > 20000$.

11.5. «Организация связи». *Кроме стрелялок Петя Торопыжский иногда играет и в стратегические игры. В одной из них важным является организация связи между опорными пунктами. При очередном запуске этой игры Пете понадобилось обеспечить двустороннюю связь между пунктами A и B , расположенными на прямой. Для этого он может устанавливать ретрансляционные станции в некоторых из оговоренных точек C_i , расположенных на той же прямой. На каждой станции он должен предусмотреть два узконаправленных передатчика, транслирующих сигнал в направлении от A к B и от B к A . Кроме, конечно, станций, расположенных в самих пунктах A и B : там нужно установить только по одному передатчику, вещающему в соответствующую сторону. Всего имеется m видов передатчиков; передатчик j -го вида имеет дальность вещания d_j , а его установка стоит r_j денежных единиц. Естественно, Петя желает установить связь, потратив наименьшее количество денежных ресурсов. Сможет ли он организовать связь в указанных условиях и, если да, то в какую сумму ему это обойдётся?*

Данная задача, по мнению программного комитета, является сложной, поскольку не вполне тривиальна идейно, а кроме того, привлекает для своего решения принцип динамического программирования. Однако подготовленным 11-классникам данная задача по силам в полном объёме, а для прочих участников допускает частичные решения, создаваемые из общих соображений.

Отметим, что задача содержит некоторую хитрость: возможные точки установки станций могут находиться не только между точками A и B , но и вне отрезка между этими точками. Несложно понять, что бессмысленно ставить станции в точках, не принадлежащих отрезку AB , поэтому при чтении такие точки нужно отсеивать. В дальнейших рассуждениях считаем, что у нас остались только точки из отрезка AB .

Для начала сделаем наблюдение, что для самой дешёвой организации двусто-

ронней связи достаточно найти самый дешёвый вариант организации односторонней связи из A в B , а потом в обратную сторону устанавливать те же самые ретрансляторы. То есть если при организации связи из A в B какая-то станция из точки C_i вещает при помощи передатчика типа j на станцию в точке $C_{i'}$, то для организации обратной связи надо установить на станцию в точке $C_{i'}$ передатчик того же типа j (очевидно, его дальности хватит, чтобы покрыть станцию в точке C_i). Действительно, пусть мы как-то организовали связь из A в B , а обратную связь каким-то другим образом удалось установить дешевле. Тогда, применив ту же идею с повторением передатчиков, мы можем удешевить организацию связи из A в B . Стало быть, достаточно решить задачу самого дешёвого установления связи из A в B и для получения ответа для двусторонней связи умножить полученный результат на 2.

Основным объектом будет одномерный массив `price` с индексами от 0 до $(n + 1)$. Элемент с индексом i содержит наименьшую стоимость организации односторонней связи из A до C_i . Точку C_0 отождествим с A , а точку C_{n+1} — с B . Элемент `price[0]` изначально положим нулём (из A в A связь устанавливается бесплатно), а остальные элементы зашлём $+\infty$ (значением, заведомо большим возможной стоимости связи, например, 10^9). Обработка точек C_i идёт для индексов i от 0 до n . Шаг принципа динамического программирования заключается в следующем.

Считаем, что передатчики перенумерованы так, что $d_j \leq d_{j+1}$ — передатчик с большим номером имеет не меньший радиус вещания, чем передатчик с меньшим номером. Считаем, что точки C_i перенумерованы так, что $x_i < x_{i+1}$ — в порядке расположения на прямой. Этих перенумераций можно достичь, проведя соответствующие сортировки передатчиков и точек. Кроме того, в этом месте можно сделать небольшую оптимизацию: после сортировки типов передатчиков из каждой группы с одинаковым расстоянием вещания оставить только один тип с наименьшей стоимостью установки. Хотя следует отметить, что в стандартной библиотеке языков BASIC, Паскаль, C/C++ нет алгоритма, выделяющего в отсортированном массиве группы равных элементов, а при ручной его реализации следует соблюдать определённую аккуратность.

Для точки C_i выбираем индекс j^* такой, что $d_{j^*} \geq x_{i+1} - x_i$, то есть передатчик с номером j^* и передатчики с большими номерами могут достать хотя бы следующую станцию C_{i+1} со станции C_i . В силу упорядоченности набора передатчиков можно применить алгоритм двоичного поиска, который даст результат за время $O(\log m)$. Если такого индекса не нашлось, то есть не нашлось передатчика, вещающего достаточно далеко, то связь установить невозможно, цикл останавливаем и сообщаем о неудаче. (Хотя этот этап можно и пропустить, устроив далее перебор по всему множеству типов передатчиков. Соответственно, в этом переборе нужно предусмотреть завершение алгоритма с неудачей.)

Здесь потребуется ещё одно наблюдение. А именно, если передатчик какого-то типа j может вещать со станции C_i на какую-то станцию $C_{i'}$, то нет нужды

рассматривать ситуации, когда этот передатчик вещает на другие станции, расположенные между C_i и $C_{i'}$. Действительно, если оптимальная цепочка вещания выглядит как $C_i \xrightarrow{j} C_{i_1} \rightarrow C_{i_2} \rightarrow \dots \rightarrow C_{i_\sigma} \rightarrow C_{i'}$ (то есть со станции C_i на какую-то промежуточную станцию C_{i_1} вещание идёт посредством передатчика типа j и далее до $C_{i'}$), то стоимость такой цепочки станций будет не меньше, чем если вещать непосредственно с C_i на $C_{i'}$ при помощи передатчика типа j . Если же оптимальная цепочка выглядит как $C_i \xrightarrow{j} C_{i_1} \rightarrow C_{i_2} \rightarrow \dots \rightarrow C_{i_\sigma} \xrightarrow{j'} C_{i''}$, причём $x_{i_\sigma} < x_{i'} < x_{i''}$, то такую цепочку можно преобразовать в цепочку $C_i \xrightarrow{j} C_{i'} \xrightarrow{j'} C_{i''}$, которая также имеет не бóльшую стоимость.

Таким образом, выбрав какой-то передатчик, нужно вещать с него на самую далёкую станцию. Поэтому затем для всех типов передатчика j с номерами от j^* до m находим номер i' станции, самой далёкой, на которую он может вещать со станции C_i . Соответствующая станция находится опять двоичным поиском по условиям $x_{i'} - x_i \leq d_j$, $x_{i'+1} - x_i > d_j$. При этом пересчёт массива **price** осуществляется как

$$\text{price}[i'] = \min\{\text{price}[i'], \text{price}[i] + r_j\},$$

то есть если установка передатчика d_j на станцию C_i удешевляет связь на станции $C_{i'}$, то запомним эту новую, более низкую цену.

Если мы успешно прошли весь цикл, то можно установить одностороннюю связь из A в B (а значит, можно установить и двустороннюю связь), наименьшая стоимость которой будет равна $\text{price}[n+1]$ (а стоимость самой дешёвой двусторонней связи будет вдвое больше).

Сложность предложенного алгоритма есть

$$O(n \log n + m \log m + n \cdot (\log m + m \cdot \log n)) = O(m \log m + mn \log n).$$

В исходной формуле оценки сложности член $n \log n$ отвечает за сортировку станций на оси, член $m \log m$ отвечает за сортировку передатчиков по дальности вещания, член n отвечает за цикл по станциям, $\log m$ — за поиск диапазона достаточно дальнобойных передатчиков, $m \log n$ — за перебор передатчиков и поиск самой дальней станции для каждого передатчика.

Некоторое замечание: в блоке поиска самой дальней станции передатчика независимые двоичные поиски для каждого типа передатчика можно заменить общим линейным поиском, пользуясь тем, что упорядочены и станции, и передатчики — самая дальняя станция для типа передатчика с бóльшим номером (и, соответственно, с бóльшей дальностью вещания) лежит к C_i не ближе, чем самая дальняя станция для предыдущего типа. Соответственно, линейный поиск самой дальней станции для следующего типа передатчика можно начинать не с номера i текущей станции, а с номера, найденного для предыдущего передатчика. Такая оптимизация приведёт к сложности $O(m+n)$ блока перебора передатчиков и поиска для каждого самой дальней станции — мы один раз проходим по передатчикам от номера j^* до номера m (слагаемое m) и один раз по станциям от номера i до номера

n в худшем случае (слагаемое n). Общая сложность алгоритма при таком подходе будет $O(n(m+n)) = O(n^2 + nm)$.

Идеи тестов:

1. минимальный тест, $n = 0$, $m = 1$, этот передатчик достаёт от A до B ;
2. минимальный тест, $n = 0$, $m = 1$, этот передатчик не достаёт от A до B ;
3. минимальный тест, $n = 0$, $m > 1$, есть один передатчик, который достанет от A до B ;
4. минимальный тест, $n = 0$, $m > 1$, есть несколько передатчиков, которые достанут от A до B ;
5. минимальный тест, $n = 0$, $m > 1$, нет передатчиков, которые достанут от A до B ;
6. $n = 1$, единственная точка C_1 посередине между A и B , $m > 1$, есть один передатчик, который достанет от A до C_1 ;
7. $n = 1$, единственная точка C_1 посередине между A и B , $m > 1$, есть несколько передатчиков, которые достанут от A до C_1 ;
8. $n = 1$, единственная точка C_1 посередине между A и B , $m > 1$, нет передатчиков, которые достанут от A до C_1 ;
9. $n = 10$, точки C_i расположены равномерно между A и B , $m > 1$, есть один передатчик, который достанет от A до C_1 ;
10. $n = 10$, точки C_i расположены равномерно между A и B , $m > 1$, есть несколько передатчиков, которые достанут от A до C_1 ;
11. $n = 10$, точки C_i расположены равномерно между A и B , $m > 1$, нет передатчиков, которые достанут от A до C_1 ;
12. максимальный тест, $n = 3000$, $m = 3000$, точки C_i расположены равномерно между A и B , $m > 1$, есть один передатчик, который достанет от A до C_1 ;
13. максимальный тест, $n = 3000$, $m = 3000$, точки C_i расположены равномерно между A и B , $m > 1$, есть несколько передатчиков, которые достанут от A до C_1 ;
14. максимальный тест, $n = 3000$, $m = 3000$, точки C_i расположены равномерно между A и B , $m > 1$, нет передатчиков, которые достанут от A до C_1 ;
- 15–18. случайные тесты, $n < 100$, $m < 100$, все точки на отрезке AB ;
- 19–20. случайные тесты, $n < 100$, $m < 100$, есть точки вне отрезка AB ;
- 21–23. случайные тесты, $n > 100$, $m > 100$, все точки на отрезке AB ;
- 24–25. случайные тесты, $n > 100$, $m > 100$, есть точки вне отрезка AB .