**Тур – базовый уровень**

**Задача 1 «VICTORY-PRO»**

Основная идея решения данной задачи основана на следующем факте: для того, чтобы нарисовать единичный квадрат, требуется либо по одному треугольнику типа 1 и 2, либо по одному треугольнику типа 3 и 4. Таким образом, из *a*1 треугольников типа 1 и *a*2 треугольников типа 2 можно составить не более min{*a*1, *a*2} квадратов, а из *a*3 треугольников типа 3 и *a*4 треугольников типа 4 – min{*a*3, *a*4} квадратов.

Теперь, чтобы решить поставленную задачу, требуется определить, какой максимальный квадрат можно составить из *K* = min{*a*1, *a*2} + min{*a*3, *a*4} единичных квадратов. Для этого можно либо перебирать все квадраты натуральных чисел   
(1, 4, 9, 16, …) до тех пор, пока не встретится число, большее *K*, либо извлечь из *K* квадратный корень и округлить результат до ближайшего целого вниз. Стоит учитывать малую точность операции взятия квадратного корня из большого числа в некоторых языках программирования, поэтому необходимо результат проверить обратным возведением в квадрат или использовать тип с расширенной точностью.

В первом случае сложность алгоритма будет равна О(*К*1/2), а во втором – О(1).

**Задача 2 «Дистанционная регистрация»**

При решении данной задачи необходимо сначала выделить номера школ из каждой записи с названием школы и сформировать массив, содержащий эти номера. Затем, используя полученный массив, определить, количество школ и номера школ, которые встречаются в нем не более пяти раз.

При выделении номера школы из ее названия *s* следует последовательно перемещаться по строке *s* слева направо, пока не встретится цифра. Это можно сделать с помощью следующего фрагмента программы:

i = 0

**while** s[i] **not in** '0123456789':

i += 1

start = i

**while** i < len(s) **and** s[i] **in** '0123456789':

i +=1

finish = i

После выполнения этого фрагмента программы подстрока строки *s* от символа с номером start до символа с номером finish (не включительно) и есть искомый номер школы. Следует выделить этот номер и сохранить его в массиве *sch*.

Поскольку номер школы может быть очень большим, то не следует переводить его в числовой тип данных, а сохранить как строку. Полученный массив номеров школ нужно отсортировать в лексикографическом (алфавитном) порядке. Теперь, чтобы получить искомый ответ, осталось подсчитать, сколько раз встречается в массиве каждая школа, и сформировать список школ, встречающихся не более 5 раз. Это можно сделать, например, с использованием следующего фрагмента программы:

count = 1

result = 0

answer = []

**for** i **in** **range**(1, **len**(sch)):

**if** sch[i] == sch[i – 1]:

count += 1

**else**:

**if** count <= 5:

result += 1

answer.append(sch[i-1])

count = 0

**if** count <= 5: # не забываем последнюю школу

result += 1

answer.append(sch[-1])

**Тур – продвинутый уровень**

**Задача 1 «Олимпиада по программированию»**

Разберем вначале частичное решение, основанное на предположении, что все задачи оцениваются одинаково. Представим каждую задачу отрезком на оси времени. Тогда задача сводится к классической: из заданного множества отрезков на прямой выбрать наибольшее количество отрезков, не имеющих общих точек.

Для решения этой задачи можно использовать следующий алгоритм. Вначале выберем из всех отрезков тот, который заканчивается левее всех. Из отрезков, которые начинаются правее него, опять выберем отрезок, заканчивающийся левее всех и т.д.

Не сложно определить, что реализация этого алгоритма имеет асимптотическую сложность O(*N*2). Более эффективное решение можно получить, отсортировав все начала и концы отрезков и пройдя слева направо по отсортированному массиву. Такое решение с учетом использования быстрой сортировки имеет уже асимптотическую сложность   
O(*N* log *N*).

Для решения исходной задачи в общем случае, когда каждая задача на межрегиональной олимпиаде оценивается своим количеством баллов, можно использовать различные подходы. Некоторые из них рассмотрены ниже в представленных вариантах решений.

Первый вариант решения**.** Он основан на использовании метода динамического программирования и заключается в следующем. Отсортируем все задачи, предложенные на межрегиональной олимпиаде, по *времени окончания* их решения. Пусть *a*[*i*] – максимальное количество баллов, которые можно набрать, решая только задачи из числа "первых" *i* задач. Рассмотрим *i*-ю задачу. Пусть *j* – это номер последней задачи, которая заканчивается не позже, чем выдается *i*-я задача (то есть *sj* + *tj* ≤ *si, a sj+1* + *tj+1* > *sj*). Тогда

*a*[*i*] = max(*a*[*i* – 1], *a*[*j*] + *c*[*i*]).

Из этого следует, что можно либо решить задачу с номером *i* и какой-то набор задач с номерами не больше *j*, либо не решать задачу с номером *i*. Ответом на поставленный вопрос будет число *a*[*n*].

Осталось пояснить, как находить число *j*. Это можно делать несколькими способами:

* перебором всех чисел от 1 до *i* (решение с асимптотической сложностью O(*N*2));
* бинарным поиском на отрезке от 1 до *i* (сложность O(*N*log *N*));
* используя метод "двух указателей", то есть, запоминая предыдущее *j* и осуществляя поиск каждого следующего *j* на отрезке от предыдущего *j*, поскольку новое значение будет больше или равно предыдущему (таким образом, для вычисления всех *a*[*i*] каждого кандидата каждый *j* мы проверим лишь единожды и сложность такого решения будет равна O(*N* log *N*) с учетом сортировки).

Восстановление ответа.Кроме поиска наибольшей суммы баллов, которую наберет Артур на олимпиаде, требуется также вывести количество и перечень задач, решение которых позволит ему набрать такую сумму. Это можно сделать одним из стандартных способов восстановления ответа в динамическом программировании. Например, будем кроме массива *a* хранить дополнительный массив *prev*, в котором укажем, какой выбор был сделан на каждом шаге. Если *a*[*i*] = *c*[*i*] + *a*[*j*], то в *prev*[*j*] запишем *j*. Это будет означать, что решается задача с номером *i*, а предыдущая задача имеет номер не больше *j*. В противном случае необходимо записать в *prev*[*i*] число –1. Теперь, пройдя с конца по массиву *prev*, можно восстановить список решаемых задач.

Второй вариант решения. При реализации этого варианта вначале отсортируем вместе все начала и концы отрезков, сохраняя для каждой точки ее тип (начало или конец) и номер отрезка, к которому она относится. Будем рассматривать отсортированные события слева направо. В переменной *best* будем хранить лучший результат, который можно получить, используя только задачи, время выполнения которых уже закончилось к текущему моменту («закончившиеся» задачи). Если очередная точка – это начало отрезка с номером *i*, то запишем в *b*[*i*] текущее значение переменной *best*. Таким образом, *b*[*i*] – это наилучший результат, который можно получить до начала решения *i*-й задачи. Если очередная точка – это конец *i*-го отрезка, то необходимо обновить значение переменной *best* следующим образом: если решается *i*-я задача, то *best* = *b*[*i*] + *c*[*i*], в противном случае значение переменной *best* не изменяется. Из двух вариантов нужно выбрать тот, в котором значение переменной *best* наибольшее, то есть *best* = max{*best*, *b*[*i*] + *c*[*i*]}.

**Задача 4 «Путь домой»**

Если рассматривать небольшие ограничения, то решение задачи может быть следующим. Построим граф, соответствующий городу без застройки, при этом вершинам графа будут соответствовать перекрестки, а ребрам – дороги. Затем удалим в графе все ребра, попавшие в застроенные кварталы (сложность такой процедуры равна   
O(*n* \* площадь города)). В полученном графе начнем поиск в ширину из точки (0, 0), чтобы определить расстояния до всех возможных расположений будущего дома Мэра, и выберем из них самое близкое. Восстанавливая путь при реализации поиска в ширину стандартным способом, можно найти точки поворота пути.

Если площадь города велика, а кварталов застройки не много, то можно воспользоваться *сжатием координат.* Для этого при построении графа будем использовать только те горизонтали и вертикали, на которых расположены мэрия, будущие дома или границы кварталов застройки.

В случае максимальных ограничений решение исходной задачи может быть следующим. Будем решать задачу отдельно для каждого возможного расположения будущего дома Мэра. Пусть рассматриваемый дом имеет координаты (*xi*, *yi*). Будем считать, что *xi* ≥ 0, *yi* ≥ 0. Легко понять, что нужно изменить в решении, чтобы рассмотреть и случаи отрицательных координат.

Рассмотрим пути, выходящие из мэрии на Север. Возможны два варианта пути:

1. едем на Север, затем поворачиваем направо (на Восток), и затем налево (опять на Север); при этом любой отрезок пути может иметь длину, равную 0;
2. едем на Север, пересекая горизонтальную улицу, на которой будет расположен дом Мэра, затем поворачиваем направо (на Восток), и затем еще раз направо (обратно на Юг).

Сравнивая эти два пути можно сказать, что первый путь всегда короче второго.

Далее выясним, как далеко мы сможем проехать из точки (0, 0) на Север (см. рис. 1). Для этого переберем все кварталы застройки и пересечем их лучом, выходящим из начала координат на Север. Пусть мы можем беспрепятственно проехать до точки (0, *R*). Если рассматриваемый дом находится на этом отрезке, то задача решена.

Аналогично рассмотрим луч, выходящий из точки (*xi*, *yi*) на Юг, и найдем на нем точку (*xi*, *S*) с минимальной положительной ординатой *S*, такую что из этой точки мы сможем беспрепятственно доехать до дома Мэра. Если *S* > *R*, то пути первого типа нет. В противном случае, попытаемся найти минимальное значение *t* на отрезке [*S*, *R*], такое, что можно проехать из точки (0, *t*) в точку (*xi*, *t*). Для этого рассмотрим все кварталы, которые пересекаются с полосой 0 ≤ *x* ≤ *xi.* Рассмотрим их проекции на ось *Оу*, найдем объединение этих проекций (открытых интервалов), и найдем точку *Y* на оси *Оy* с минимальной координатой, не меньшей *S,* не покрытую объединением интервалов. Для этого можно отсортировать вместе начала и концы интервалов и пройти по этим точкам, считая *баланс*. Если *Y* окажется не больше *R*, то искомый путь имеет поворот направо в точке (0, *Y*) и поворот налево в точке (*xi*, *Y*). В противном случае, пути первого вида нет, и нужно перейти к поиску пути второго вида.

Если *R* ≤ yi, то пути второго вида также нет. Иначе рассмотрим луч, выходящий из точки (*xi*, *yi*) на Север (см. рис. 2), и найдем на нем самую далекую точку *S*, до которой можно беспрепятственно добраться из точки (*xi*, *yi*). Рассмотрим отрезок [*x*i, min{*R*, *S*}] и найдем на нем минимальную точку *Y*, не покрытую объединением проекций интервалов, рассмотренным выше. Если такая точка *Y* существует, то из всех путей, выходящих из точки (0, 0) на Север, кратчайший путь имеет повороты в точках (0, *Y*) и (*xi*, *Y*). В противном случае таких путей нет.

Заметим, что если *Y* = *yi*, то второй поворот не нужен.

Данный алгоритм имеет асимптотическую сложность O(*kn* log *n*).

0

0

*xi*

yi

*R*

Рис. 1.

0

0

*xi*

yi

*R*

*S*

Рис. 2.

*S*

*Y*

*Y*