



В. И. Галочкин,

Поволжский государственный технологический университет, г. Йошкар-Ола, Республика Марий Эл

ЗАДАЧИ ИНТЕЛЛЕКТУАЛЬНОЙ ОЛИМПИАДЫ ПРИВОЛЖСКОГО ФЕДЕРАЛЬНОГО ОКРУГА СРЕДИ ШКОЛЬНИКОВ

Аннотация

В статье рассмотрены авторские решения задач интеллектуальной олимпиады. Набор задач сбалансирован по тематике и трудности. Темы задач связаны с построением рациональных структур данных, рекуррентными соотношениями, оптимизацией на графах, способами отсека неперспективных вариантов решения. Выявлена наиболее распространенная причина неполного решения задач — недооценка вычислительной сложности алгоритмов. Статья предназначена для школьников, студентов, тренеров, ведущих занятия по олимпиадному программированию.

Ключевые слова: программирование, информатика, олимпиада, задача, алгоритм, программа.

DOI: 10.32517/2221-1993-2019-18-8-56-62

Введение

Уровень трудности школьных олимпиадных задач по информатике быстро повышается. В разряд тренировочных переходят задачи, ранее считавшиеся трудными, а программы олимпиадных занятий со школьниками превосходят по объему и глубине многие вузовские учебные программы [3]. Для совершенствования олимпиадникам необходимо постоянно участвовать в разнообразных соревнованиях помимо ежегодных регионального и заключительного этапов всероссийской олимпиады школьников.

Окружной этап Интеллектуальной олимпиады Приволжского федерального округа среди школьников проходил 5 марта 2019 года в городе Йошкар-Ола на базе Поволжского государственного технологического университета. В олимпиаде приняли участие команды четырнадцати регионов Приволжского федерального округа. Каждая команда состояла из двух школьников, прошедших отбор в региональных отборочных турнирах, поэтому уровень участников оказался достаточно высоким.

Команде выделялся один компьютер. Для проверки решений использовалась широко распространенная система ejudge [7]. На вход системы поступали тексты

программ, созданные в популярных средах программирования. Полное решение каждой задачи оценивалось 100 баллами. Частичные решения получали оценки, пропорциональные количеству пройденных тестов, что в целом соответствует правилам Всероссийской олимпиады школьников [3].

Было предложено восемь задач, одна из которых имела выраженный утешительный характер и была решена всеми командами. Автором статьи составлены остальные семь задач, разработаны алгоритмы и программы, подготовлены тесты. Все эти материалы были приняты жюри без изменений.

Для всех задач, кроме последней, время работы программы на одном тесте не должно превышать одной секунды, а затраты оперативной памяти — 256 мегабайт. Последняя задача имела ограничение по времени — две секунды.

Задача 1. Робот

Робот находится в точке плоскости с целыми координатами (X, Y) , где $0 \leq X, Y \leq 20$; $X + Y > 0$. Он должен достичь цели, которая находится в начале координат. Каждый ход робота — продвижение на единицу по горизонтали (налево или направо) или вертикали (вверх или

Контактная информация

Галочкин Владимир Иванович, канд. тех. наук, доцент, доцент кафедры информатики и системного программирования, Поволжский государственный технологический университет, г. Йошкар-Ола, Республика Марий Эл; адрес: 424000, Республика Марий Эл, г. Йошкар-Ола, пл. Ленина, д. 3; e-mail: info@volgatech.net

V. I. Galochkin,

Volga State University of Technology, Yoshkar-Ola, The Mari El Republic

TASKS OF THE INTELLECTUAL OLYMPIAD OF THE VOLGA FEDERAL DISTRICT AMONG SCHOOLCHILDREN

Abstract

The article discusses the author's solutions of the tasks of the intellectual Olympiad. The set of tasks is balanced by subject and difficulty. The topics of the tasks are related to the construction of rational data structures, recurrence relations, optimization on graphs, ways of cutting off unpromising solutions. The most common reason for the incomplete solution of problems is revealed that is underestimation of the computational complexity of the algorithms. The article is intended for schoolchildren, students, trainers, leading classes in Olympiad programming.

Keywords: programming, informatics, Olympiad, task, algorithm, program.

вниз). Если робот достигает цели, то немедленно останавливается. Необходимо найти количество способов достижения цели ровно за Z шагов, $1 \leq Z \leq 20$.

Входные данные. В единственной строке находятся три числа: X , Y и Z .

Выходные данные. Вывести количество способов достижения цели ровно за Z шагов.

Примеры.

№ п/п	Ввод	Вывод	Комментарий
1	1 0 1	1	
2	0 1 3	5	Возможны следующие варианты движения робота: (0,1) — (0,2) — (0,1) — (0,0) (0,1) — (1,1) — (0,1) — (0,0) (0,1) — (1,1) — (1,0) — (0,0) (0,1) — (-1,1) — (0,1) — (0,0) (0,1) — (-1,1) — (-1,0) — (0,0)

Описание алгоритма.

Обозначим через $F(i, j, k)$ функцию, определяющую количество способов достижения цели, где (i, j) — координаты начальной точки, а k — количество шагов. Ответ дает значение $F(X, Y, Z)$. Выведем рекуррентные соотношения для расчета значений функции F .

Если значения $X + Y$ и Z разной четности либо $X + Y > Z$, то цели достигнуть невозможно, т. е. $F(X, Y, Z) = 0$. При $X = 0$ и $Y = Z$, а также при $Y = 0$ и $X = Z$ имеется единственный способ, т. е. $F(X, Y, Z) = 1$.

Пусть $X + Y \leq Z$. Из позиции с координатами (i, j) можно сделать четыре разных первых хода, после чего останется $k - 1$ ход. Получаем основное рекуррентное соотношение:

$$F(i, j, k) = F(i - 1, j, k - 1) + F(i + 1, j, k - 1) + F(i, j - 1, k - 1) + F(i, j + 1, k - 1).$$

На этом соотношении легко составить рекурсивную программу. Ее недостаток в том, что значения функции многократно пересчитываются, что резко повышает время счета при увеличении размерности. Более рационально использовать подход динамического программирования [4, 6, 8].

Установим единичные значения $F(0, i, i)$ и $F(j, 0, j)$ а остальные значения будем считать нулевыми. Можно для удобства положить, что $F(0, 0, k) = 0$.

Рассчитаем значения $F(i, j, k)$ при $i \geq 0$ и $j \geq 0$. По симметрии:

$$F(i, j, k) = F(-i, j, k) = F(i, -j, k) = F(-i, -j, k) = F(j, i, k) = F(-j, i, k) = F(j, -i, k) = F(-j, -i, k).$$

Поэтому рассчитанные значения будем распространять и на отрицательные координаты.

Во внешнем цикле будем увеличивать разность $m = k - (i + j)$ между заданным числом ходов и наименьшим расстоянием от точки с координатами (i, j) до центра координат при перемещении параллельно координатным осям. Ввиду одинаковой четности величин $i + j$ и k значения m будут изменяться с шагом 2. Во внутреннем цикле будем последовательно увеличивать величину $i + j$, перебирая всевозможные пары значений i и j . Рассмотрим процесс подробнее.

Пусть сначала $m = 0$. При $i + j = 1$, как уже говорилось, $F(0, 1, 1) = F(1, 0, 1) = 1$.

При $i + j = 2$ получим:

$$F(1, 1, 2) = F(0, 1, 1) + F(2, 1, 1) + F(1, 0, 1) + F(1, 2, 1) = 1 + 0 + 1 + 0 = 2.$$

При $i + j = 3$:

$$F(1, 2, 3) = F(0, 2, 2) + F(2, 2, 2) + F(1, 1, 2) + F(1, 3, 3) = 1 + 0 + 2 + 0 = 3.$$

При $i + j = 4$:

$$F(1, 3, 4) = F(0, 3, 3) + F(2, 2, 3) + F(1, 2, 3) + F(1, 4, 3) = 1 + 0 + 3 + 0 = 4;$$

$$F(2, 2, 4) = F(1, 2, 3) + F(3, 2, 3) + F(2, 1, 3) + F(2, 3, 3) = 1 + 0 + 2 + 0 = 3.$$

При $i + j = 5$:

$$F(1, 4, 5) = F(0, 4, 4) + F(2, 4, 4) + F(1, 3, 4) + F(1, 5, 4) = 1 + 0 + 4 + 0 = 5;$$

$$F(2, 3, 5) = F(1, 3, 4) + F(3, 3, 4) + F(2, 2, 4) + F(2, 4, 4) = 4 + 0 + 4 + 0 = 10.$$

Когда окажется $i + j > Z$, увеличим на 2 значение m и снова проведем цикл расчетов по возрастанию суммы $i + j$ от 1 до k . Например:

$$F(0, 1, 3) = F(-1, 1, 2) + F(1, 1, 2) + F(0, 0, 2) + F(0, 2, 2) = 2 + 2 + 0 + 1 = 5;$$

$$F(1, 0, 3) = F(0, 1, 3) = 5;$$

$$F(0, 2, 4) = F(-1, 1, 2) + F(1, 1, 2) + F(0, 0, 2) + F(0, 2, 2) = 3 + 3 + 5 + 1 = 12;$$

$$F(1, 1, 4) = F(0, 1, 3) + F(2, 1, 3) + F(1, 0, 3) + F(1, 2, 3) = 5 + 3 + 5 + 3 = 16;$$

$$F(0, 3, 5) = F(-1, 3, 4) + F(1, 3, 4) + F(0, 2, 4) + F(0, 4, 4) = 4 + 4 + 12 + 1 = 21.$$

Вычисления происходят быстро, но при увеличении значения Z свыше 20 для результата не хватает 64 разрядов.

Задача 2. Цифры

Как-то раз Вася не мог заснуть. Сначала он представлял себе караван верблюдов, идущих через пустыню. Когда это не помогло, он стал считать суммарное количество цифр S во всех целых числах от M до N включительно при различных значениях M и N .

Вася заинтересовался обратной задачей: как при заданном количестве цифр S определить соответствующий диапазон (M, N) ? Он быстро понял, что достаточно взять единственное число P из S цифр и положить $M = N = P$. Но это было совсем неинтересно.

Далее Вася задумался о том, как минимизировать M , не превышая некоторого предела L . Сон окончательно пропал. Чтобы Вася спал спокойно, необходимо написать программу для решения этой задачи.

Входные данные. В единственной строке находятся целые числа S и L ($1 \leq S \leq 10^{18}$; $1 \leq L \leq 10^4$).

Выходные данные. Вывести значения M и N ($1 \leq M \leq N$; $M \leq L$). Если ответов несколько, вывести тот, для которого величина M минимальна. Если решения нет, вывести «No».

Примеры.

№ п/п	Ввод	Вывод
1	2 5	1 2
2	12 20	2 11
3	10 1	No

Описание алгоритма.

Обозначим через $D(i, j)$ суммарное количество цифр всех чисел от i до j включительно. При увеличении i величина $D(i, j)$ уменьшается, а при увеличении j увеличивается. Требуется найти минимальное значение $M \leq L$ и $N \geq M$, чтобы выполнялось равенство $D(M, N) = S$.

Обозначим через $T(i)$ суммарное количество цифр всех чисел, разрядность которых не превышает i . Легко видеть, что:

$$T(i) = 9 \times 10^0 + 9 \times 2 \times 10^1 + 9 \times 3 \times 10^2 + 9 \times 4 \times 10^3 + \dots + 9 \times i \times 10^{i-1}.$$

Обозначим через k минимальное значение i , при котором $T(i) \geq S$. Поскольку $S \leq 10^{18}$, то $k \leq 18$. Установим сначала $M = 1$ и будем определять N .

Если $T(k) = S$, то $N = 10^k - 1$. Пусть $T(k) > S$. Обозначим $U = T(k) - S$. Тогда величина $V = 10^k - 1 - [U / k]$, где квадратными скобками обозначена целая часть числа, дает минимальное значение, для которого $R = D(M, V) = T - k(10^k - 1 - V) \geq S$. Положим $N = V$.

Если $R = S$, то установленные значения $M = 1$ и $N = V$ удовлетворяют условиям задачи. Иначе величина $W = R - S$, где $W \leq 18$, определяет количество лишних цифр. Придется увеличивать значение M , проверяя выполнение условия $M \leq L$.

Если $W \leq 9$, то при $M = W + 1$ получим $D(M, N) = S$. Если $W > 9$ и W нечетное, то достаточно положить $M = 10 + (W - 9) / 2$, поскольку имеется девять чисел из одной цифры, а каждое двузначное число увеличивает на 2 количество цифр. Наконец, если $W > 9$ и W — четное, то решения при $N = V$ не существует. Положим $M = 10 + (W - 8) / 2$. Сейчас уже $D(M, N) = S - 1$. Увеличим на 1 значение N . Значение $D(M, N)$ увеличится на k или $k + 1$, т. е. окажется, что $D(M, N) > S$.

Будем аналогично прежнему увеличивать величину M , пока не достигнем условия $D(M, N) \leq S$. В случае превращения последнего неравенства в равенство задача решена. В противном случае снова увеличим на 1 значение N и продолжим вычисления.

Таким образом, значения M и N возрастают связно. Поскольку должно выполняться условие $M \leq L$, то процесс заканчивается нахождением значений M и N либо отрицательным ответом.

Задача 3. Автогонка

Круг кольцевой автогонки имеет длину S километров. Движение происходит только в одном направлении. Автогонщик Веттель отрабатывает на тренировке остановку и дозаправку бензином.

На расстояниях $X_1, X_2, \dots, X_N$ от старта расставлены N канистр с бензином. Бензобак в состоянии вместить содержимое всех канистр. Болид Веттеля с пустым бензобаком

доставляется к одной из канистр. После заправки бензином из этой канистры болид начинает движение по кольцу, останавливаясь и заправляясь дополнительно у каждой встреченной непустой канистры. Движение по кольцу продолжается до тех пор, пока бензобак не опустеет.

Известно, что бензина из i -й канистры, находящейся в точке X_i , хватает на C_i километров. Требуется выбрать стартовую канистру так, чтобы Веттель смог проехать максимальное расстояние.

Входные данные.

В первой строке заданы значения N и S ($1 \leq N \leq 2 \times 10^5$; $1 \leq S \leq 10^9$).

Во второй строке находятся $X_1 < X_2 < \dots < X_N$ ($0 \leq X_i < S$).

В третьей строке указаны значения $C_1, C_2, \dots, C_N$ ($1 \leq C_i \leq 10^5$).

Все числа целые.

Выходные данные. Вывести максимальное расстояние, которое пройдет болид при движении от найденной стартовой канистры.

Примеры.

№ п/п	Ввод	Вывод	Комментарий
1	3 10 1 3 6 3 4 1	8	
2	3 10 1 5 9 3 1 3	7	При движении от канистры 3 болид по кругу добирается до канистры 1. В бензобаке остается бензин на 1 км, а после дозаправки его хватит на 4 км. Этого бензина достаточно для того, чтобы доехать до канистры 2. Бак оказывается пустым, но после дозаправки из канистры 2 болид может проехать еще 1 км. Общая протяженность пути составляет 7 км

Описание алгоритма.

Нельзя делать полный перебор по начальной канистре. Сложность такого подхода составляет $O(N^2)$.

Если в какой-либо канистре бензина достаточно для полного круга, то именно эту канистру выберем в качестве начальной и найдем искомое расстояние.

Пусть это не так.

Начнем с первой канистры и будем двигаться, заправляясь из встреченных канистр. Если болид не сможет дойти до очередной канистры, запомним пройденное расстояние и начнем движение от следующей канистры. Из таких расстояний будем дополнительно сохранять максимальное. Конечно, если будет преодолен полный круг, то пройденное расстояние помимо этого круга будет определяться остатками топлива.

Если при начале движения от одной из канистр болид переходит по кольцу на следующий круг и доходит до первой канистры, то нужно определить, на какое дополнительное расстояние хватит остатка его топлива. Это расстояние нужно добавить к тому, которое найдено ранее при начале движения от первой канистры.

Может оказаться, что сейчас мы в состоянии доехать до более дальней канистры, чем это было первоначально. Снова найдем дополнительное расстояние и продолжим

расчеты. Как только окажется невозможным добраться до очередной канистры, моделирование движения прекращается и выдается полученный максимум. Как и ранее, может оказаться, что мы доберемся в конце концов до стартовой канистры.

Итак, достаточно промоделировать движение на двух первых кругах. Сложность алгоритма оценивается величиной $O(N)$.

Можно доказать, что если суммарного объема топлива достаточно для прохождения полного круга, то всегда можно выбрать начальную канистру, позволяющую преодолеть круг. Использование этого утверждения не меняет общую сложность алгоритма.

Задача 4. Ремонт дорог

В тридесятиом государстве имеется N городов, $3 \leq N \leq 300$. Некоторые из них связаны асфальтовыми дорогами. Между двумя городами имеется не более одной дороги. Города пронумерованы от 1 до N . По каждой дороге возможно движение в обоих направлениях. Из каждого города имеется путь в любой другой город напрямую либо через промежуточные города.

Решено отремонтировать часть дорог, покрыв их супербетоном марки «Патриот». Для каждой пары городов должен быть обеспечен хотя бы один кратчайший путь по дорогам с бетонным покрытием, связывающий эти города. При этом общая длина ремонтируемых дорог должна быть минимальной. Требуется по заданным длинам кратчайших путей между всеми городами определить дороги для предстоящего ремонта.

Входные данные.

Первая строка содержит значение N .

Далее следует $N - 1$ строка описания кратчайших расстояний между городами. Каждая i -я строка содержит $N - i$ значений C_{ij} ($1 \leq C_{ij} \leq 10^5$, $j = i + 1, i + 2, \dots, N$), где C_{ij} — кратчайшее расстояние от i -го города до j -го.

Выходные данные. Если решения нет, вывести в первой строке «No». В противном случае в первой строке выводятся количество новых дорог M и их общая длина L . Следующие M строк содержат по три числа: номера двух городов, определяющих дорогу, и длина этой дороги.

Дороги выводятся в порядке возрастания первого номера в паре, а при равных первых номерах — в порядке возрастания второго номера. Первый номер в паре должен быть меньше второго. Если имеется несколько решений, вывести любое из них.

Примеры.

№ п/п	Ввод	Вывод
1	4 2 3 4 1 3 2	4 9 1 2 2 1 4 4 2 3 1 3 4 2
2	4 1 2 4 1 3 2	3 4 1 2 1 2 3 1 3 4 2
3	3 1 3 2	2 3 1 2 1 2 3 2

Описание алгоритма.

Будем формировать новый граф с матрицей длин A путем усечения полного графа, содержащего для каждого двух вершин соединяющее их ребро. Переберем всевозможные тройки (i, j, k) отличающихся друг от друга номеров вершин с учетом порядка элементов в тройке.

Если для некоторой тройки имеется соотношение $C_{ij} + C_{jk} < C_{ik}$, то решения нет. Действительно, в этом случае величина C_{ik} не может быть длиной кратчайшего пути из i -й вершины в k -ю.

Будем далее предполагать, что $C_{ij} + C_{jk} \geq C_{ik}$.

Для каждой тройки проверим соотношение $C_{ij} + C_{jk} = C_{ik}$. Если оно выполняется, то нужно запретить ребро нового графа (i, k) , поскольку имеется кратчайший путь из i -й вершины в k -ю через j -ю вершину. Этот путь предпочтительнее прямого, поскольку включает кратчайшие пути из i -й вершины в k -ю и из k -й вершины в j -ю. Заметим, что ребро (i, k) может и не входить в начальный граф.

Оставшимися ребрами графа бетонных дорог будут все незапрещенные ребра (i, j) . Для каждого такого ребра $A_{ij} = C_{ij}$.

В результате граф бетонных дорог будет состоять из ребер (i, j) , каждое из которых определяет единственный кратчайший путь из i -й вершины в j -ю. Действительно, если кратчайший путь из i -й вершины в j -ю проходит через некоторую промежуточную вершину k , то $C_{ij} = C_{ik} + C_{kj}$, а ребро (i, j) оказалось бы среди запрещенных.

Полученное решение единственное. Очевидно, что каждое ребро (i, j) графа бетонных дорог должно входить в некоторый кратчайший путь, поэтому само определяет кратчайший путь из i -й вершины в j -ю.

При выдаче ребер по матрице A дополнительная сортировка не требуется. Сложность алгоритма составляет $O(N^3)$. Он имеет определенное сходство с широко распространенным алгоритмом поиска кратчайших путей Флойда—Уоршелла [4, 8].

Задача 5. Найти кольцо

Прямоугольная комната размером $M \times N$ метров с мебелью покрыта квадратными паркетными плитками стороной один метр. В углу комнаты стоит тяжелый сейф, занимая всю площадь плитки. Под сейф закатилось кольцо с бриллиантом, поэтому решено полностью освободить плитку под сейфом. Для этого нужно переместить сейф на одну из соседних плит путем кантования — поворота в плоскости пола на 90° вокруг одного из двух своих углов. Соседними считаются плитки, имеющие общую сторону.

Некоторые плитки комнаты заняты стульями. Каждый стул может быть переставлен на свободную соседнюю с ним плитку. В комнате может быть и другая мебель. Она также занимает определенные плитки, но в отличие от стульев не может перемещаться.

При повороте сейфа никакая его точка не должна пересекать внутреннюю часть такой плитки, на которой стоит стул или другая мебель.

Матрица $M \times N$ определяет начальное положение мебели. Каждый элемент матрицы описывает одну паркетную плитку. Сейф находится в левом верхнем углу. Плите с сейфом соответствует символ «#», свободной плите паркета — символ «.» (точка), плите со стулом —

символ «@», плите с другой мебелью — символ «*». Поворот сейфа выполняется за одну минуту, а перестановка стула — за одну секунду.

Найти минимальное время в секундах, за которое сейф может быть перемещен.

Входные данные.

В первой строке содержатся числа M и N ($2 \leq M, N \leq 500$), разделенные пробелами.

В следующих M строках находится матрица, описывающая комнату и содержащая по N подряд идущих символов в строке.

Выходные данные. Вывести наименьшее число секунд, необходимых для освобождения плиты под сейфом. Если плиту освободить невозможно, вывести «No».

Примеры.

№ п/п	Ввод	Вывод
1	4 4 #... @.@. ...@	60
2	4 3 #.* .@. @** @.@	61
3	4 2 #. .@ @* @@	No

Описание алгоритма.

Итак, имеется клетчатое поле, в котором плите соответствует одна клетка. Для того чтобы переместить сейф, нужно освободить три клетки: справа, снизу и ниже по диагонали от клетки сейфа. Назовем эти клетки граничными. Путем из клеток будем считать цепочку клеток, у которой каждая последующая клетка является соседней с предыдущей.

Обозначим какую-либо граничную клетку, занятую стулом, через A_1 . Пусть имеется путь $A_1, A_2, \dots, A_L$, где клетки $A_2, \dots, A_L$ не являются граничными, A_L — свободная клетка, а клетки $A_2, \dots, A_{L-1}$ заняты стульями. Тогда можно переставить на свободную клетку A_L стул с клетки A_{L-1} , на освободившуюся клетку A_{L-1} — стул с предпоследней клетки A_{L-2} и т. д. В результате начальная граничная клетка тоже будет освобождена. Для этого потребуется $L - 1$ перестановка стульев. Все это напоминает дырочную проводимость, известную в физике.

Что изменится, если в путь от занятой граничной клетки к пустой входят другие граничные клетки, как занятые стульями, так и свободные? Оказывается, что количество перестановок стульев не изменится, а освободится только начальная граничная клетка. Рассмотрим такой пример исходных данных:

```
4 4
#.@.
@a**
@a*a
@a@a
```

Пусть строки нумеруются сверху вниз, а столбцы — слева направо. Кратчайший путь от граничной клетки (2, 1) до свободной клетки (1, 4) состоит из пяти клеток и проходит через граничные клетки (2, 2) и (1, 2). За те же четыре перестановки стульев клетка (2, 1) окажется свободной, а клетка (1, 4) — занятой.

Длина кратчайшего пути от каждой занятой граничной клетки до ближайшей свободной клетки определяется поиском в ширину [4, 6, 8]. Общий результат зависит от порядка перебора граничных клеток. Так, для примера:

```
4 7
#aaaaa.
aa.aaaa
aaaaaaa
.aaaaaa
```

сумма длин путей при поиске от граничных клеток в порядке (1, 2) — (2, 2) — (2, 1) составляет 12, а при порядке (2, 2) — (2, 1) — (1, 2) — только 8. Значит, нужно найти наименьшую сумму при всех шести вариантах перестановок граничных клеток.

Задача 6. Полеты со скидкой

Самолеты авиакомпании «Себе в убыток» летают в N городов ($2 \leq N \leq 100$) и обслуживают M рейсов. Города пронумерованы от 1 до N . Каждый рейс связывает два разных города. Между парой городов может быть не более одного рейса в каждом направлении.

Билет на каждый рейс имеет определенную стоимость. Компания продает единый билет на некоторый маршрут, предусматривающий пересадки с одного рейса на другой. В этом случае, если маршрут имеет не менее K пересадок ($1 \leq K < N$), то после K -й пересадки стоимость на каждый рейс маршрута снижается вдвое. Маршрут может иметь повторяющиеся города и рейсы. Требуется рассчитать минимальную стоимость перелета из столицы, имеющей номер 1, в другие города.

Входные данные.

Первая строка содержит числа N , M и K .

Далее следуют M строк описания рейсов по три числа A , B и C в строке ($1 \leq A, B \leq N$; $2 \leq C \leq 10^4$). Первые два числа задают номера городов, а третье — стоимость перелета.

Все числа целые.

Известно, что стоимость каждого рейса выражается четным числом.

Выходные данные. В i -й строке выводятся минимальная стоимость перелета от столицы до $(i + 1)$ -го города. Если нет ни одного пути, в соответствующей строке выводится 0. Таким образом, выходной файл состоит из $N - 1$ строки.

Пример.

Ввод	Вывод	Комментарий
3 3 2 1 2 4 2 1 4 2 3 20	4 20	Для полета в город 3 выгоднее всего слетать в город 2 и вернуться в столицу. Следующий повторный перелет в город 2 происходит после двух пересадок и обходится вдвое дешевле, как и последний перелет в город 3

Описание алгоритма.

Проведем небольшую модификацию известного алгоритма поиска кратчайших путей Беллмана—Форда [4].

Заведем двумерный массив D . В элементе $D[v, j]$ будет храниться значение кратчайшего пути из начальной вершины 1 в вершину v , состоящего из $j + 1$ дуги для $j = 0, 1, \dots, 2N - 1$. Изначально элемент $D[1, 0]$ приравняем к нулю, а остальным элементам массива D присвоим значения, равные условной бесконечности (например, число, заведомо большее суммы всех весов).

Итак, есть граф $G = (V, E)$, и $N = |V|$, а $M = |E|$. Обозначим вес дуги, связывающей смежные вершины графа u и v , через $w(u, v)$. Во внешнем цикле будем увеличивать количество дуг пути. Во внутреннем цикле для каждой дуги (u, v) с конечной меткой вершины u проведем перерасчет по формулам:

$$D[v, j + 1] = \min(D[v, j + 1], D[u, j] + w(u, v)) \\ \text{при } j \leq K - 1;$$

$$D[v, j + 1] = \min(D[v, j + 1], D[u, j] + 0.5 \times w(u, v)) \\ \text{при } j \geq K.$$

Количество шагов не должно превышать $2N - 1$.

Минимальное значение $D[v, j]$ по всем значениям j определяет длину кратчайшего пути из начальной вершины 1 в вершину v . Сложность алгоритма составляет $O(N \times M)$.

Задача 7. Бильярд

Бильярдный стол расчерчен на квадратные клетки и имеет размеры $M \times N$ клеток. В углах стола находятся четыре лузы для шаров. В центрах некоторых клеток находятся шары. По определенному шару, называемому битком, наносится удар, в результате которого шар может катиться в одном из четырех направлений вдоль диагоналей клетки. Если этот шар достигает края стола, то отражается и продолжает движение, а если попадает в угол стола, то сваливается в лузу и игра заканчивается. Если биток не встречает на своем пути других шаров и не сваливается в лузу, то он проходит ровно K клеток, не считая начальной, но включая конечную клетку либо лузу. Если же биток достигает клетки с другим шаром, то встреченный шар удаляется, а биток занимает его место и используется для последующих ударов. По заданному расположению шаров найти наименьшее количество ударов, необходимое для «снятия» всех шаров, включая биток.

Входные данные.

Первая строка содержит целые числа M , N и K ($1 \leq M, N \leq 15$; $1 \leq K \leq 10$) через пробел.

Каждая i -я строка из следующих M строк содержит N символов C_{ij} ($1 \leq j \leq N$).

Свободные клетки стола помечаются символами «.», клетки с неподвижными шарами — символами «#», а клетка с битком — символом «&».

Общее количество шаров, включая биток, может быть от 1 до 15.

Выходные данные. В единственной строке выводится минимальное количество ударов, необходимое для снятия всех шаров. Если это невозможно, выводится «No».

Примеры.

№ п/п	Ввод	Вывод	Комментарий
1	2 3 2 #.. &.#	3	Биток из клетки (2, 1) после первого удара вверх и налево вытесняет шар с клетки (1, 1). Вторым ударом вниз и направо шар попадает в клетку (2, 3), вытесняя еще один шар. Наконец, третьим ходом вниз и направо он сваливается в лузу
2	3 3 5 #.. .&.. ...	No	Любой удар приводит к попаданию битка в лузу. При этом на столе остается шар в клетке (1, 2)

Описание алгоритма.

Нельзя выбивать ближайший шар, т. е. жадный подход [4] не годится. В этом убеждает следующий простой пример входных данных:

```
1 6 1
#.&###
```

При начальном движении вверх либо вниз и направо потребуется не менее девяти ходов. Если же сначала устремиться налево, то можно обойтись восемью ходами.

Из предыдущего примера видно, что клетка может посещаться не один раз. Более того, в зависимости от выбитых шаров предпочтительными оказываются разные варианты движения из клетки, т. е. траектория во многом определяется предшествующими ходами.

Определим состояние клетки как следующий набор параметров:

- координаты клетки;
- порядковый номер хода, после которого клетка достигнута;
- множество оставшихся шаров.

Организуем поиск в ширину [4, 6, 8] по возрастанию количества ходов, используя очередь из состояний. Сначала в очередь помещаются начальная клетка и полный набор шаров. В дальнейшем выбирается состояние из начала очереди, а в конец помещаются все состояния, в которые можно перейти из выбранного состояния. Полный перебор нереален, поэтому нужны отсечения.

Рассмотрим несколько вариантов сокращения перебора.

Можно ограничить количество посещений каждой клетки. Тестирование показывает, что трудно подобрать соответствующий параметр для различных тестов.

Можно в каждой клетке запоминать индексы элементов очереди, соответствующих посещениям данной клетки. Тогда перед заходом в клетку имеет смысл просмотреть предыдущие ее посещения. Если множество остающихся шаров включает соответствующее множество хотя бы при одном из прошлых посещений, то не стоит повторно рассматривать эту клетку. Этот подход уменьшает объем счета, но имеет свои сложности. Очередь состояний растет слишком быстро. К тому же трудно определить, насколько далеко стоит просматривать предыдущие состояния.

Можно ограничиться случаем, когда состояние рассматриваемой клетки имеет в точности такое же множество оставшихся шаров, какое встречалось у этой клетки ранее в очереди. Выявления этой ситуации можно добиться, не просматривая очередь назад. Каждое множество характеризуется битовой строкой, длина которой равна максимально возможному количеству элементов множества. Отдельный бит показывает присутствие или отсутствие соответствующего элемента. Пронумеруем все шары. Пусть, например, их три. Тогда битовая строка «101» определяет множество шаров {1, 3}. Рассматривая битовую строку как двоичное целое число, можно организовать трехмерный массив булевских значений. Первые два индекса соответствуют координатам клетки, а третий является целым числом от 0 до $2^{15} - 1$ и однозначно определяет множество из 14 элементов. По мере рассмотрения состояний очереди можно отмечать пройденные состояния в данном массиве, не повторяя их при переборе.

Последний вариант наиболее выгоден. Расчеты могут привести к двум случаям:

- попадание бита в лузу при пустом множестве оставшихся шаров определяет минимальное количество ударов;
- исчерпание очереди состояний при остающихся шарах означает отсутствие решения.

Результаты и выводы

Некоторым командам для успешного выступления не хватило сыгранности. Менее удачно выступили команды, в которых либо все задачи решались совместно, либо участники выбирали свои задачи и работали независимо. При подготовке к подобным олимпиадам стоит большее внимание уделять тактике командной работы, используя опыт студенческих команд [1, 5].

Решение каждой задачи требует организованного подхода. Поспешный переход к кодированию приводил к потере времени и баллов. В работе [5] дано системати-

зированное описание методов и приемов, помогающих лучше решать олимпиадные задачи.

Многие программы прошли только часть тестов. Иногда причина заключалась в недооценке вычислительной сложности алгоритмов. В других случаях не всегда достигался баланс между стремлением к рациональному решению и желанием получить хоть какую-то оценку. Во время разбора задач выяснилось, что некоторые частичные решения легко могли быть доведены до полных.

При составлении олимпиадных задач не всегда удается выдержать необходимый уровень сложности, соответствующий проводимой олимпиаде. Опыт проведения школьных и студенческих олимпиад [2] позволил решить эту проблему. Победителями стали школьники Кировской области, набравшие 625 баллов из 800 возможных.

Список использованных источников

1. Акишев И. Р. Об опыте участия в командных соревнованиях по программированию формата ACM ICPC // Информатика. 2008. № 19. С. 20–28. http://is.ifmo.ru/works/_akishev.pdf
2. Галочкин В. И. Задачи заключительного тура международной интернет-олимпиады по информатике и программированию 2012 года для студентов России и ближайшего зарубежья // Программные системы и вычислительные методы. 2012. № 1. С. 17–27.
3. Кирюхин В. М., Цветкова М. С. Информатика. Программы внеурочной деятельности учащихся по подготовке к Всероссийской олимпиаде школьников: 5–11 классы. М.: БИНОМ. Лаборатория знаний, 2014. 224 с.
4. Кормен Т., Лайзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 3-е изд. / пер. с англ. М.: Вильямс, 2013. 1328 с.
5. Оршанский С. А. О решении олимпиадных задач по программированию формата ACM ICPC // Информатика. 2006. № 1. <http://inf.lseptember.ru/article.php?ID=200600106>
6. Порублев И. Н., Ставровский А. Б. Алгоритмы и программы. Решение олимпиадных задач. М.: Вильямс, 2007. 480 с.
7. Сайт системы ejudge. https://ejudge.ru/wiki/index.php/Система_ejudge
8. Скиена С. С., Ревилла М. А. Олимпиадные задачи по программированию. Руководство по подготовке к соревнованиям. М.: КУДИЦ-ОБРАЗ, 2005. 416 с.

Уважаемые авторы и читатели!

Издательство «Образование и Информатика» объявляет о проведении конкурса цифровых изображений и фотографий, которые будут размещены на обложке журнала «Информатика в школе» в первом полугодии 2020 года.

Подробности — на сайте издательства «Образование и Информатика»: <http://www.infojournal.ru/>