**ФЕДЕРАЛЬНОЕ**

**ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО**

**ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«Казанский (Приволжский) Федеральный Университет»**

**Институт информационных технологий и интеллектуальных систем**

**Задача коммивояжера**

**Выполнил:**

**Соловяненко Олег Юрьевич**

**Преподователь:**

**Гафурова Полина Олеговна**

**Казань - 2021**

**1 ) Задача коммивояжера**Задача коммивояжераодна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. Первые упоминания в качестве математической задачи на оптимизацию принадлежат [Карлу Менгеру](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%BD%D0%B3%D0%B5%D1%80,_%D0%9A%D0%B0%D1%80%D0%BB_(%D0%BC%D0%BB%D0%B0%D0%B4%D1%88%D0%B8%D0%B9)), который сформулировал её на математическом коллоквиуме в [1930 году](https://ru.wikipedia.org/wiki/1930_%D0%B3%D0%BE%D0%B4).  
В [1950-е](https://ru.wikipedia.org/wiki/1950-%D0%B5) и [1960-е](https://ru.wikipedia.org/wiki/1960-%D0%B5) годы задача коммивояжёра привлекла внимание ученых в США и Европе. Важный вклад в исследование задачи принадлежит [Джорджу Данцигу](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D1%86%D0%B8%D0%B3,_%D0%94%D0%B6%D0%BE%D1%80%D0%B4%D0%B6), Делберту Рею Фалкерсону, которые в [1954 году](https://ru.wikipedia.org/wiki/1954_%D0%B3%D0%BE%D0%B4) в институте [RAND Corporation](https://ru.wikipedia.org/wiki/RAND_(%D0%BA%D0%BE%D1%80%D0%BF%D0%BE%D1%80%D0%B0%D1%86%D0%B8%D1%8F)) сформулировали задачу в виде задачи дискретной оптимизации и применили для её решения [метод отсечений](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%93%D0%BE%D0%BC%D0%BE%D1%80%D0%B8).  
В марте [2005 года](https://ru.wikipedia.org/wiki/2005_%D0%B3%D0%BE%D0%B4) задача с 33 810 узлами была решена программой *Конкорд*: был вычислен путь длиной в 66 048 945 и доказано отсутствие более коротких путей. В апреле [2006](https://ru.wikipedia.org/wiki/2006) было найдено решение для экземпляра с 85 900 узлами.

Методы : полный перебор

* случайный перебор
* жадные алгоритмы
  + метод ближайшего соседа
  + [метод включения ближайшего города](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%BB%D0%B8%D0%B6%D0%B0%D0%B9%D1%88%D0%B5%D0%B3%D0%BE_%D0%B3%D0%BE%D1%80%D0%BE%D0%B4%D0%B0&action=edit&redlink=1)
  + [метод самого дешёвого включения](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D1%81%D0%B0%D0%BC%D0%BE%D0%B3%D0%BE_%D0%B4%D0%B5%D1%88%D1%91%D0%B2%D0%BE%D0%B3%D0%BE_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D1%8F&action=edit&redlink=1)
* [метод минимального остовного дерева](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE)
* [метод имитации отжига](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B8%D0%BC%D0%B8%D1%82%D0%B0%D1%86%D0%B8%D0%B8_%D0%BE%D1%82%D0%B6%D0%B8%D0%B3%D0%B0)

Все эффективные (сокращающие полный перебор) методы решения задачи коммивояжёра — методы [эвристические](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). В большинстве эвристических методов находится не самый эффективный маршрут, а [приближенное решение](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%B8%D0%B1%D0%BB%D0%B8%D0%B6%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5&action=edit&redlink=1). Зачастую востребованы так называемые [any-time-алгоритмы](https://ru.wikipedia.org/w/index.php?title=Any-time-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B&action=edit&redlink=1), то есть, постепенно улучшающие некоторое текущее приближенное решение.

Существуют также постановки, в которых задача коммивояжёра становится [NP-полной](https://ru.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0). Обычно такие постановки выглядят следующим образом: существует ли на заданном графе *G* такой обход, что его стоимость не превышает *x*. Часто на ней проводят обкатку новых подходов к эвристическому сокращению [полного перебора](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80).

На практике применяются различные модификации более эффективных методов: [метод ветвей и границ](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B2%D0%B5%D1%82%D0%B2%D0%B5%D0%B9_%D0%B8_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86) и [метод генетических алгоритмов](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), а также [алгоритм муравьиной колонии](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%80%D0%B0%D0%B2%D1%8C%D0%B8%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC).

**2)Основной принцип работы. Основные особенности**

Общая идея метода может быть описана на примере поиска минимума функцииf(x) на множестве допустимых значений переменной x. Функция f и переменная x могут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Процедура ветвления состоит в разбиении множества допустимых значений переменной x на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества множества значений переменной x).

Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений переменной x.

В основе метода ветвей и границ лежит следующая идея: если нижняя граница значений функции на подобласти A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальную из полученных верхних оценок записывают в глобальную переменную m; любой узел дерева поиска, нижняя граница которого больше значения m, может быть исключен из дальнейшего рассмотрения.

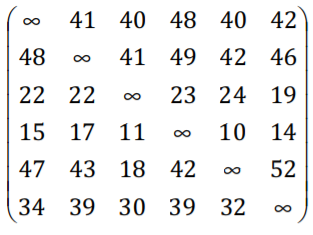
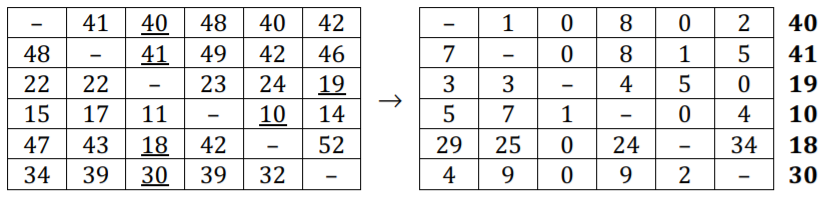
Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

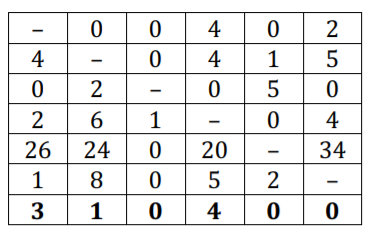
**Более короче :**

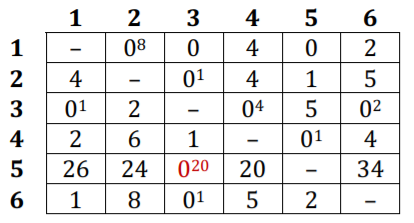
Имеется множетво S всех гамильтоновых циклов графа. На каждом шаге в S ищется ребро (i, j), исключение которого из маршрута максимально увеличит оценку снизу. Далее происходит разбиение множества на два непересекающихся S1 и S2. S1 — все циклы, содержащие ребро (i, j) и не содержащие (j, i). S2 — все циклы, не содержащие (i, j). Далее вычисляется оценка снизу для длины пути каждого множества и, если она превышает длину уже найденного решения, множество отбрасывается. Если нет — множества S1 и S2 обрабатываются так же, как и S.

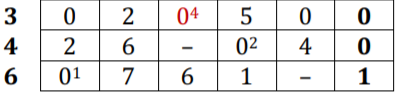
**Пример решения :**

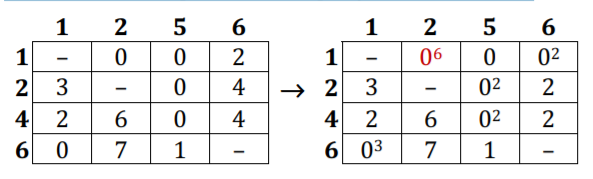
Решите методом ветвей и границ следующую задачу коммивояжера:

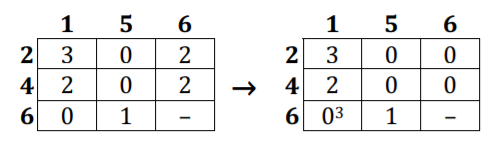
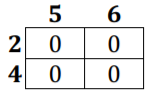
  
Приведем исходную матрицу по строкам, вычитая из элементов каждой строки ее минимальный элемент.   
  
Выделенные жирным шрифтом числа – это идеальный тур, полученный лексикографическим перебором. Сумма констант приведения: 40 + 41 + 19 + 10 + 18 + 30 = 158. Приведем исходную матрицу по столбцам, вычитая из элементов каждого столбца его минимальный элемент.



Сумма констант приведения: 3 + 1 + 0 + 4 + 0 + 0 = 8, сумма всех констант: 158 + 8 = 166.   
Тур, проходящий только через ребра нулевой стоимости, будет, очевидно, минимальным. Для того, чтобы определить его стоимость, прибавим к нулю только что вычисленную константу 166: 0 + 166 = 166 Таким образом, мы получили нижнюю оценку стоимости класса всех возможных туров, т.е. минимальный тур в данной задаче не может стоить меньше, чем 166. Назовем оценкой нуля в позиции (i, j) в матрице сумму минимальных элементов в i-й строке и j-м столбце (не считая сам этот ноль). Оценим теперь каждый ноль в приведенной матрице:   


Оценки, равные нулю, не указаны. Оценка k нуля, в позиции (i, j) означает следующее: если в тур не будет включен путь из i в j (стоимостью 0), то придется доплатить как минимум k. Поэтому, можно разделить класс всех возможных туров на два: туры, содержащие ребро (i, j) и туры, не содержащие его. Для последних минимальная оценка увеличится на k. Рассмотрим ребро, соответствующее нулю с максимальной оценкой. В данном случае это ребро (5, 3). Таким образом, класс всех туров разбивается на два: содержащих ребро (5, 3) и не содержащих его. Нижняя оценка стоимости второго класса туров увеличивается до 166 + 20 = 186. Чтобы определить оценку для первого класса туров удалим из матрицы строку 5 и столбец 3. Обозначим ее как C[(5,3)]:   
  


Оценка класса туров с ребром (5, 3) составит 166 + 1 + 1 = 168. Выберем теперь класс с наименьшей оценкой и повторим этот процесс для него. Затем из двух полученных классов выберем тот, у которого оценка минимальна и разобьем его. Так будем повторять до тех пор, пока не достигнем листа дерева, т. е. пока не получим матрицу 0×0: Удаляем из матрицы 3-ю строку и 4-й столбец:

Т. к. последнюю матрицу удалось привести на 2 (по 6-му столбцу), то оценка класса туров с ребром (3, 4) увеличивается на 2 и становится равной 168 + 2 = 170. Удаляем из матрицы 1-ю строку и 2-й столбец:  
  
Т. к. последнюю матрицу удалось привести на 2 (по 6-му столбцу), то оценка класса туров с ребром (1, 2) увеличивается на 2 и становится равной 170 + 2 = 172. Удаляем из матрицы 6-ю строку и 1-й столбец:   
Вершина (2,5) дерева будет соответствовать классу, содержащему ребра: (5, 3); (3, 4); (1, 2); (6, 1); (2, 5); (4, 6). Этот класс состоит из одного полного тура (1, 2, 5, 3, 4, 6, 1) со стоимостью 172

**3) Оценка временной сложности**

.Поскольку коммивояжёр в каждом из городов встает перед выбором следующего города из тех, что он ещё не посетил, существует (n-1)! маршрутов для асимметричной и (n-1)!/2

маршрутов для симметричной задачи коммивояжёра. Таким образом, размер пространства поиска [факториально](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0) зависит от количества городов.

Различные варианты задачи коммивояжёра (метрическая, симметричная и асимметричная) NP-эквивалентны. Согласно распространенной, но недоказанной гипотезе о неравенстве классов сложности P и NP, не существует детерминированной [машины Тьюринга](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0), способной находить решения экземпляров задачи за полиномиальное время в зависимости от количества городов.

Также известно, что при условии

 не существует алгоритма, который для некоторого полинома

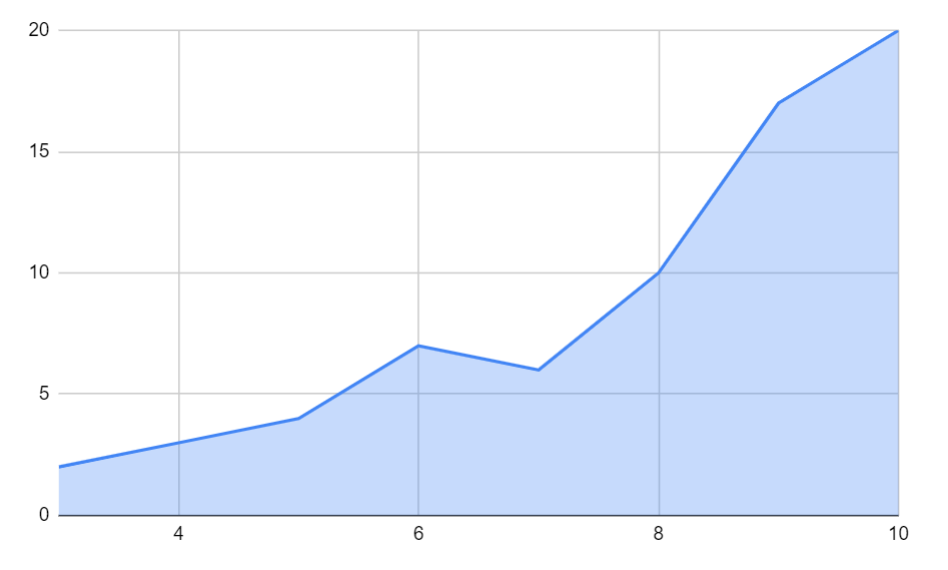
P вычислял бы такие решения задачи коммивояжёра, которые отличались бы от оптимального максимум на коэффициент 2^p(n) .

На практике поиск строго оптимального маршрута требуется не всегда. Существуют алгоритмы поиска приближенных решений для метрической задачи за полиномиальное время и нахождения маршрута максимум вдвое длиннее оптимального. До сих пор не известен ни один алгоритм с полиномиальным временем, который бы гарантировал точность лучшую, чем 1,5 от оптимальной. По предположению

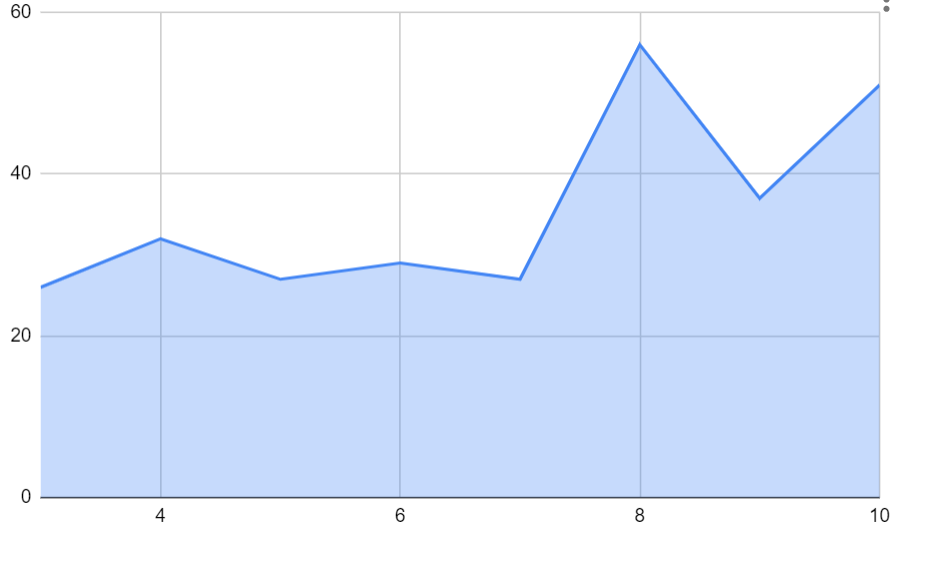
, существует (неизвестная) константа c>0 , такая, что ни один алгоритм с полиномиальным временем не может гарантировать точность 1+c.Как было показано Арора, для евклидовой задачи коммивояжёра существует схема полиномиального времени [PTAS](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%B1%D0%BB%D0%B8%D0%B6%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%85%D0%B5%D0%BC%D0%B0_%D0%BF%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8) для поиска приближённого решения.

Кроме того данные могут иметь особенности, которые могут помочь решить задачу. Например, города расположены не случайно, а по рельефу местности, либо даже вдоль уже давно найденного оптимального торгового маршрута. Дополнительная информация и эвристики позволяют за приемлемое время находить приемлемые решения.

**4) Таблицы полученных значений количества итераций и времени работы в зависимости от размера и характера данных  
**

**5) Графики  
**

**Время в мс(oY) от кол-ва вершин(oX)**

****

**Кбайты(oY) от кол-ва точек(oX)**

****

**6) Выводы. Плюсы и минусы алгоритма,его применимость.**

**Плюсы**

-Имея достаточно времени, они вычисляют кратчайший маршрут. Имея нижнюю границу для оптимальных решений, можно оценить то, насколько отличается найденный маршрут от оптимального. Например, имея нижнюю границу на уровне 100, и после нахождения маршрута длиной 102, оптимальный маршрут может находиться в пределах от 100 до 102.

-Для поиска маршрутов приемлемой длины точные методы могут комбинироваться с эвристическими.

**Минусы**-такой метод мало пригоден для решения задач с огромным количеством точек,ведь он будет решать задачу за очень огромное время .

**Вывод**Такой алгоритм очень точен ,но и одновременно и неэффективен ,ведь при большом количестве точек будет разумно использовать эвристические методы . Люди используют алгоритмы поиска,они являются компромиссом между этими двумя методами решений,но требуют подбора параметров.

**7) Список использованной литературы**

<https://www.matburo.ru/>

https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0\_%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D1%91%D1%80%D0%B0#%D0%9C%D0%B5%D1%82%D0%BE%D0%B4\_%D0%B2%D0%B5%D1%82%D0%B2%D0%B5%D0%B9\_%D0%B8\_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86

**8) Код алгоритма**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <ctime>

using namespace std;

const int inf = 1E9, NMAX = 16;

int n, i, j, k, m, temp, ans, d[NMAX][NMAX], t[1 << NMAX][NMAX];

bool get(int nmb, int x)

{

return (x & (1 << nmb)) != 0;

}

int main()

{

unsigned int start\_time = clock(); // начальное время

string line;

ifstream in("D:\\test.txt");

string buffer = "";

vector<string> commands;

int const n = 3;

/\*if (in.is\_open())

{

while (getline(in, line))

{

for (int i = 0; i < line.size(); i++)

{

if (line[i] != ' ')

{

buffer += line[i];

}

else

{

commands.push\_back(buffer);

buffer = "";

}

}

}

}\*/

for (i = 0; i < n; ++i)

for (j = 0; j < n; ++j)

{

in >> d[i][j];

}

for (i = 0; i < n; ++i)

for (j = 0; j < n; ++j)cout << d[i][j]<<" ";

cout << endl;

t[1][0] = 0; m = 1 << n;

for (i = 1; i < m; i += 2)

for (j = (i == 1) ? 1 : 0; j < n; ++j)

{

t[i][j] = inf;

if (j > 0 && get(j, i))

{

temp = i ^ (1 << j);

for (k = 0; k < n; ++k)

if (get(k, i) && d[k][j] > 0) t[i][j] = min(t[i][j], t[temp][k] + d[k][j]);

}

}

for (j = 1, ans = inf; j < n; ++j)

if (d[j][0] > 0) ans = min(ans, t[m - 1][j] + d[j][0]);

if (ans == inf) cout << -1; else cout << ans;

unsigned int end\_time = clock(); // конечное время

unsigned int search\_time = end\_time - start\_time; // искомое время

cout << endl;

cout << "0";

}

**9) Код генерации входных файлов**

int main()

{

int adj\_matrix[10][10];// массив для генерации чисел

int n = 50;

for (int i = 1; i <= n; i++)

{

std::ofstream fout; // объект класса ofstream

fout.open(std::to\_string(i) + "test.txt"); // открываем файл. если такого нет, создается. расположение файла

srand(time(0) + i);

for (int k = 0; k < 50; k++)

{

for (int u = 0; u < 10; u++)

{

for (int v = 0; v < 10; v++)

{

adj\_matrix[u][v] = -1;

}

}

for (int u = 0; u < 10; u++)

{

for (int v = 0; v < 10; v++)

{

if (adj\_matrix[u][v] == -1)

{

adj\_matrix[u][v] = rand() % 10;

adj\_matrix[v][u] = adj\_matrix[u][v];

}

}

}

for (int i = 0; i < 10; i++)

{

adj\_matrix[i][i] = 0;

}

for (int u = 0; u < 10; u++)

{

for (int v = 0; v < 10; v++)

{

fout << adj\_matrix[u][v]<<" ";

}

fout << "\n";

}

fout << "#";

fout << "\n";

for (int u = 0; u < 10; u++)

{

for (int v = 0; v < 10; v++)

{

adj\_matrix[u][v] = -1;

}

}

}

fout.close();

}

}

**10) Входные данные**

0 8 1 7 4 1 6 1 8 2

8 0 5 9 8 8 6 3 5 9

1 5 0 2 6 7 9 6 4 1

7 9 2 0 0 3 4 9 5 9

4 8 6 0 0 8 0 6 6 7

1 8 7 3 8 0 4 1 1 5

6 6 9 4 0 4 0 6 5 5

1 3 6 9 6 1 6 0 8 7

8 5 4 5 6 1 5 8 0 6

2 9 1 9 7 5 5 7 6 0

#

0 2 8 9 5 5 7 9 3 8

2 0 0 8 4 6 1 2 1 9

8 0 0 5 7 6 4 6 4 5

9 8 5 0 2 8 5 9 3 9

5 4 7 2 0 1 2 7 4 6

5 6 6 8 1 0 5 2 5 1

7 1 4 5 2 5 0 3 1 9

9 2 6 9 7 2 3 0 3 4

3 1 4 3 4 5 1 3 0 3

8 9 5 9 6 1 9 4 3 0

#

0 5 2 7 9 9 3 3 8 9

5 0 2 8 4 2 7 8 9 4

2 2 0 9 0 5 0 1 1 1

7 8 9 0 9 7 7 3 4 3

9 4 0 9 0 7 2 0 2 0

9 2 5 7 7 0 5 3 7 9

3 7 0 7 2 5 0 6 6 7

3 8 1 3 0 3 6 0 0 5

8 9 1 4 2 7 6 0 0 0

9 4 1 3 0 9 7 5 0 0

#

0 7 0 6 7 7 8 9 5 6

7 0 2 5 5 8 1 8 7 1

0 2 0 8 6 9 9 3 4 6

6 5 8 0 8 0 5 3 2 2

7 5 6 8 0 5 4 1 6 1

7 8 9 0 5 0 0 3 7 0

8 1 9 5 4 0 0 4 0 0

9 8 3 3 1 3 4 0 1 1

5 7 4 2 6 7 0 1 0 6

6 1 6 2 1 0 0 1 6 0

#

0 9 5 2 0 2 9 5 5 7

9 0 9 9 8 6 7 4 9 8

5 9 0 7 9 0 6 0 6 9

2 9 7 0 1 5 5 9 1 0

0 8 9 1 0 4 8 4 5 2

2 6 0 5 4 0 0 6 2 0

9 7 6 5 8 0 0 9 6 8

5 4 0 9 4 6 9 0 3 0

5 9 6 1 5 2 6 3 0 4

7 8 9 0 2 0 8 0 4 0

#

0 5 2 1 7 6 1 3 0 5

5 0 5 9 7 8 0 9 6 0

2 5 0 0 3 3 7 6 2 0

1 9 0 0 9 8 7 3 5 5

7 7 3 9 0 5 5 3 5 3

6 8 3 8 5 0 2 9 5 9

1 0 7 7 5 2 0 0 7 6

3 9 6 3 3 9 0 0 6 7

0 6 2 5 5 5 7 6 0 2

5 0 0 5 3 9 6 7 2 0

#

0 1 0 7 4 1 5 4 8 8

1 0 0 8 7 8 5 0 2 9

0 0 0 0 3 8 6 5 3 0

7 8 0 0 8 4 4 2 9 8

4 7 3 8 0 7 1 8 7 0

1 8 8 4 7 0 8 3 9 9

5 5 6 4 1 8 0 7 9 7

4 0 5 2 8 3 7 0 2 7

8 2 3 9 7 9 9 2 0 5

8 9 0 8 0 9 7 7 5 0

#

0 5 3 6 5 9 6 7 2 3

5 0 5 3 5 9 8 3 6 0

3 5 0 2 9 0 4 8 9 1

6 3 2 0 5 4 0 8 6 6

5 5 9 5 0 2 0 9 4 2

9 9 0 4 2 0 1 2 2 6

6 8 4 0 0 1 0 3 9 7

7 3 8 8 9 2 3 0 0 7

2 6 9 6 4 2 9 0 0 9

3 0 1 6 2 6 7 7 9 0

#

0 4 8 4 3 1 2 7 0 5

4 0 5 0 9 7 1 0 5 6

8 5 0 5 8 0 4 4 1 7

4 0 5 0 0 9 9 0 0 7

3 9 8 0 0 1 1 3 7 6

1 7 0 9 1 0 4 7 9 7

2 1 4 9 1 4 0 4 0 5

7 0 4 0 3 7 4 0 9 6

0 5 1 0 7 9 0 9 0 3

5 6 7 7 6 7 5 6 3 0

#

0 9 9 3 1 6 3 4 3 3

9 0 5 4 1 1 0 8 0 7

9 5 0 8 7 3 9 9 8 5

3 4 8 0 3 8 1 5 4 4

1 1 7 3 0 6 2 3 0 3

6 1 3 8 6 0 6 3 3 8

3 0 9 1 2 6 0 7 6 0

4 8 9 5 3 3 7 0 7 5

3 0 8 4 0 3 6 7 0 5

3 7 5 4 3 8 0 5 5 0

#

0 8 1 5 9 8 1 5 3 4

8 0 4 7 0 4 1 9 6 6

1 4 0 4 5 4 6 0 3 0

5 7 4 0 9 5 5 2 6 3

9 0 5 9 0 0 4 5 8 3

8 4 4 5 0 0 0 6 7 4

1 1 6 5 4 0 0 9 5 3

5 9 0 2 5 6 9 0 7 6

3 6 3 6 8 7 5 7 0 9

4 6 0 3 3 4 3 6 9 0

#

0 7 9 0 2 5 9 6 3 9

7 0 8 4 2 3 7 1 2 8

9 8 0 4 3 1 8 8 2 3

0 4 4 0 8 7 0 5 2 4

2 2 3 8 0 4 6 6 3 6

5 3 1 7 4 0 3 7 3 4

9 7 8 0 6 3 0 9 0 0

6 1 8 5 6 7 9 0 5 0

3 2 2 2 3 3 0 5 0 7

9 8 3 4 6 4 0 0 7 0

#

0 6 5 1 2 4 1 3 0 1

6 0 5 0 7 5 4 1 5 3

5 5 0 7 0 3 9 3 7 9

1 0 7 0 3 5 2 6 2 5

2 7 0 3 0 8 3 0 0 5

4 5 3 5 8 0 9 1 5 5

1 4 9 2 3 9 0 3 4 1

3 1 3 6 0 1 3 0 2 7

0 5 7 2 0 5 4 2 0 8

1 3 9 5 5 5 1 7 8 0

#

0 0 2 4 5 5 8 3 5 3

0 0 4 6 7 5 2 0 3 4

2 4 0 8 1 4 7 7 7 4

4 6 8 0 6 4 7 0 7 6

5 7 1 6 0 2 2 8 2 7

5 5 4 4 2 0 2 7 1 6

8 2 7 7 2 2 0 9 4 4

3 0 7 0 8 7 9 0 4 2

5 3 7 7 2 1 4 4 0 7

3 4 4 6 7 6 4 2 7 0

#

0 3 7 6 9 9 2 5 8 2

3 0 1 8 4 0 0 9 6 8

7 1 0 7 0 8 8 3 3 1

6 8 7 0 3 3 0 6 2 7

9 4 0 3 0 0 2 4 2 0

9 0 8 3 0 0 0 6 4 4

2 0 8 0 2 0 0 7 0 2

5 9 3 6 4 6 7 0 5 2

8 6 3 2 2 4 0 5 0 7

2 8 1 7 0 4 2 2 7 0

#

0 6 8 3 8 7 6 0 9 6

6 0 1 6 2 3 5 3 1 9

8 1 0 6 7 1 5 5 7 8

3 6 6 0 1 4 6 8 0 6

8 2 7 1 0 2 6 9 9 1

7 3 1 4 2 0 5 9 6 6

6 5 5 6 6 5 0 9 3 2

0 3 5 8 9 9 9 0 8 9

9 1 7 0 9 6 3 8 0 0

6 9 8 6 1 6 2 9 0 0

#

0 3 3 9 8 6 0 7 4 1

3 0 2 5 5 7 5 2 2 4

3 2 0 9 1 2 9 1 8 0

9 5 9 0 4 4 8 5 8 6

8 5 1 4 0 5 8 2 3 9

6 7 2 4 5 0 9 1 8 2

0 5 9 8 8 9 0 6 8 1

7 2 1 5 2 1 6 0 8 9

4 2 8 8 3 8 8 8 0 2

1 4 0 6 9 2 1 9 2 0

#

0 3 9 3 9 4 4 0 6 0

3 0 9 5 3 2 2 9 6 8

9 9 0 9 5 5 3 1 1 4

3 5 9 0 0 9 1 6 6 3

9 3 5 0 0 7 4 6 7 3

4 2 5 9 7 0 0 6 5 6

4 2 3 1 4 0 0 0 7 8

0 9 1 6 6 6 0 0 4 1

6 6 1 6 7 5 7 4 0 9

0 8 4 3 3 6 8 1 9 0

#

0 1 2 6 9 8 9 0 8 3

1 0 3 9 5 6 1 2 0 8

2 3 0 1 4 0 0 5 8 8

6 9 1 0 5 4 6 7 3 6

9 5 4 5 0 8 1 1 5 3

8 6 0 4 8 0 4 2 9 2

9 1 0 6 1 4 0 9 4 3

0 2 5 7 1 2 9 0 0 8

8 0 8 3 5 9 4 0 0 4

3 8 8 6 3 2 3 8 4 0

#

0 2 7 7 7 7 7 5 8 2

2 0 4 7 1 6 3 2 2 4

7 4 0 3 2 3 7 1 9 9

7 7 3 0 2 9 2 2 6 8

7 1 2 2 0 9 1 2 0 2

7 6 3 9 9 0 7 8 9 3

7 3 7 2 1 7 0 4 8 7

5 2 1 2 2 8 4 0 5 8

8 2 9 6 0 9 8 5 0 6

2 4 9 8 2 3 7 8 6 0

#

0 4 3 8 4 8 9 7 8 0

4 0 0 9 6 7 4 0 3 5

3 0 0 0 2 9 7 2 8 0

8 9 0 0 5 5 2 2 9 9

4 6 2 5 0 9 0 4 6 2

8 7 9 5 9 0 0 3 1 8

9 4 7 2 0 0 0 9 2 7

7 0 2 2 4 3 9 0 9 0

8 3 8 9 6 1 2 9 0 3

0 5 0 9 2 8 7 0 3 0

#

0 3 1 8 4 3 8 5 5 9

3 0 1 7 6 8 7 5 8 2

1 1 0 8 4 1 5 0 9 9

8 7 8 0 5 7 5 3 2 5

4 6 4 5 0 9 5 2 9 3

3 8 1 7 9 0 5 3 5 8

8 7 5 5 5 5 0 8 8 9

5 5 0 3 2 3 8 0 4 4

5 8 9 2 9 5 8 4 0 7

9 2 9 5 3 8 9 4 7 0

#

0 2 6 4 6 2 8 6 4 5

2 0 9 3 6 6 2 7 4 3

6 9 0 9 5 1 1 4 1 7

4 3 9 0 7 1 9 2 2 3

6 6 5 7 0 0 3 3 8 6

2 6 1 1 0 0 2 2 9 6

8 2 1 9 3 2 0 5 9 0

6 7 4 2 3 2 5 0 2 9

4 4 1 2 8 9 9 2 0 7

5 3 7 3 6 6 0 9 7 0

#

0 9 1 0 4 4 8 4 1 1

9 0 4 2 1 4 9 6 9 2

1 4 0 1 4 5 2 1 4 4

0 2 1 0 6 9 2 2 7 9

4 1 4 6 0 1 8 2 8 1

4 4 5 9 1 0 0 8 8 2

8 9 2 2 8 0 0 3 3 4

4 6 1 2 2 8 3 0 1 2

1 9 4 7 8 8 3 1 0 8

1 2 4 9 1 2 4 2 8 0

#

0 7 2 9 4 4 0 9 2 8

7 0 4 3 9 2 3 0 3 9

2 4 0 2 4 0 6 4 0 5

9 3 2 0 7 1 1 6 9 7

4 9 4 7 0 0 0 5 6 9

4 2 0 1 0 0 3 9 4 0

0 3 6 1 0 3 0 9 5 7

9 0 4 6 5 9 9 0 9 1

2 3 0 9 6 4 5 9 0 1

8 9 5 7 9 0 7 1 1 0

#

0 2 8 5 9 6 2 9 1 0

2 0 3 9 1 5 4 4 2 5

8 3 0 1 2 5 4 3 5 6

5 9 1 0 0 3 7 0 2 1

9 1 2 0 0 5 1 3 8 1

6 5 5 3 5 0 7 3 5 6

2 4 4 7 1 7 0 1 5 5

9 4 3 0 3 3 1 0 6 9

1 2 5 2 8 5 5 6 0 3

0 5 6 1 1 6 5 9 3 0

#

0 8 2 4 3 0 0 4 0 5

8 0 4 4 9 0 1 7 9 2

2 4 0 5 0 0 0 6 1 1

4 4 5 0 2 5 9 1 9 1

3 9 0 2 0 9 9 9 6 0

0 0 0 5 9 0 8 1 5 7

0 1 0 9 9 8 0 8 1 3

4 7 6 1 9 1 8 0 1 5

0 9 1 9 6 5 1 1 0 6

5 2 1 1 0 7 3 5 6 0

#

0 5 2 0 2 4 2 8 0 6

5 0 9 0 8 9 0 4 7 9

2 9 0 4 3 0 8 1 3 3

0 0 4 0 6 1 1 2 5 3

2 8 3 6 0 7 4 3 0 5

4 9 0 1 7 0 9 4 0 1

2 0 8 1 4 9 0 8 0 7

8 4 1 2 3 4 8 0 7 5

0 7 3 5 0 0 0 7 0 0

6 9 3 3 5 1 7 5 0 0

#

0 6 6 7 0 3 9 0 2 5

6 0 0 0 9 5 9 4 8 6

6 0 0 1 2 0 4 1 2 4

7 0 1 0 5 3 1 8 7 6

0 9 2 5 0 6 9 2 4 7

3 5 0 3 6 0 9 7 5 7

9 9 4 1 9 9 0 6 7 0

0 4 1 8 2 7 6 0 0 0

2 8 2 7 4 5 7 0 0 5

5 6 4 6 7 7 0 0 5 0

#

0 1 2 0 4 9 0 9 3 7

1 0 6 0 6 0 9 6 0 0

2 6 0 8 0 6 7 6 6 6

0 0 8 0 5 8 9 8 3 2

4 6 0 5 0 9 6 5 2 7

9 0 6 8 9 0 8 7 0 2

0 9 7 9 6 8 0 7 6 6

9 6 6 8 5 7 7 0 4 2

3 0 6 3 2 0 6 4 0 9

7 0 6 2 7 2 6 2 9 0

#

0 0 6 9 2 5 5 9 6 0

0 0 1 0 1 5 4 0 5 6

6 1 0 5 1 9 7 5 6 2

9 0 5 0 6 9 0 6 4 7

2 1 1 6 0 7 6 7 3 7

5 5 9 9 7 0 1 7 0 1

5 4 7 0 6 1 0 0 7 9

9 0 5 6 7 7 0 0 9 2

6 5 6 4 3 0 7 9 0 7

0 6 2 7 7 1 9 2 7 0

#

0 7 8 9 3 3 7 8 2 1

7 0 4 7 5 4 0 3 4 8

8 4 0 2 9 4 6 9 1 1

9 7 2 0 6 8 9 0 6 3

3 5 9 6 0 2 8 8 7 9

3 4 4 8 2 0 4 6 8 7

7 0 6 9 8 4 0 4 8 5

8 3 9 0 8 6 4 0 4 8

2 4 1 6 7 8 8 4 0 9

1 8 1 3 9 7 5 8 9 0

#

0 2 9 8 0 9 4 2 8 4

2 0 5 8 9 7 4 1 3 4

9 5 0 3 1 5 8 4 2 1

8 8 3 0 4 1 8 4 5 9

0 9 1 4 0 1 2 7 5 4

9 7 5 1 1 0 5 5 3 1

4 4 8 8 2 5 0 5 0 6

2 1 4 4 7 5 5 0 6 9

8 3 2 5 5 3 0 6 0 2

4 4 1 9 4 1 6 9 2 0

#

0 9 0 4 0 4 3 9 8 3

9 0 4 3 0 1 0 0 3 2

0 4 0 2 4 0 0 8 4 1

4 3 2 0 9 7 2 1 1 3

0 0 4 9 0 7 9 4 1 5

4 1 0 7 7 0 4 5 4 5

3 0 0 2 9 4 0 5 7 6

9 0 8 1 4 5 5 0 3 9

8 3 4 1 1 4 7 3 0 9

3 2 1 3 5 5 6 9 9 0

#

0 3 9 0 8 3 5 5 7 7

3 0 2 1 2 8 7 9 4 2

9 2 0 7 1 2 7 3 9 3

0 1 7 0 2 0 7 4 4 5

8 2 1 2 0 0 0 6 2 4

3 8 2 0 0 0 8 1 4 4

5 7 7 7 0 8 0 8 4 3

5 9 3 4 6 1 8 0 7 6

7 4 9 4 2 4 4 7 0 2

7 2 3 5 4 4 3 6 2 0

#

0 8 2 7 5 0 1 2 4 1

8 0 1 7 9 1 4 5 4 4

2 1 0 5 3 1 2 0 8 9

7 7 5 0 4 0 7 0 4 2

5 9 3 4 0 9 5 5 0 5

0 1 1 0 9 0 1 6 0 8

1 4 2 7 5 1 0 8 3 8

2 5 0 0 5 6 8 0 9 5

4 4 8 4 0 0 3 9 0 4

1 4 9 2 5 8 8 5 4 0

#

0 4 0 4 7 2 8 0 8 0

4 0 7 3 9 9 5 8 3 4

0 7 0 6 9 6 1 0 2 2

4 3 6 0 9 9 6 7 4 6

7 9 9 9 0 1 4 8 1 0

2 9 6 9 1 0 4 7 5 3

8 5 1 6 4 4 0 4 6 2

0 8 0 7 8 7 4 0 3 9

8 3 2 4 1 5 6 3 0 6

0 4 2 6 0 3 2 9 6 0

#

0 9 9 0 2 7 7 3 2 3

9 0 8 1 7 8 5 2 9 6

9 8 0 5 4 7 7 0 9 4

0 1 5 0 8 8 7 7 7 3

2 7 4 8 0 6 7 6 5 6

7 8 7 8 6 0 0 3 0 5

7 5 7 7 7 0 0 1 5 2

3 2 0 7 6 3 1 0 9 0

2 9 9 7 5 0 5 9 0 0

3 6 4 3 6 5 2 0 0 0

#

0 0 0 7 4 8 2 1 8 3

0 0 5 7 4 6 9 2 8 1

0 5 0 7 9 9 3 2 4 0

7 7 7 0 1 9 0 3 4 8

4 4 9 1 0 3 3 0 4 5

8 6 9 9 3 0 6 9 4 0

2 9 3 0 3 6 0 6 6 2

1 2 2 3 0 9 6 0 9 5

8 8 4 4 4 4 6 9 0 0

3 1 0 8 5 0 2 5 0 0

#

0 3 5 6 1 0 2 5 0 3

3 0 3 8 1 5 0 7 7 5

5 3 0 2 6 6 3 5 7 9

6 8 2 0 3 7 7 7 7 7

1 1 6 3 0 5 7 2 6 3

0 5 6 7 5 0 8 8 9 4

2 0 3 7 7 8 0 7 5 5

5 7 5 7 2 8 7 0 2 1

0 7 7 7 6 9 5 2 0 8

3 5 9 7 3 4 5 1 8 0

#

0 2 8 4 2 1 9 8 0 7

2 0 2 5 9 2 9 6 0 8

8 2 0 4 7 6 4 1 9 9

4 5 4 0 7 2 4 4 6 7

2 9 7 7 0 6 7 4 0 5

1 2 6 2 6 0 2 8 8 4

9 9 4 4 7 2 0 5 9 2

8 6 1 4 4 8 5 0 2 3

0 0 9 6 0 8 9 2 0 0

7 8 9 7 5 4 2 3 0 0

#

0 5 8 1 4 1 7 7 8 4

5 0 1 5 2 4 4 0 7 0

8 1 0 4 0 6 4 9 5 1

1 5 4 0 7 6 9 3 5 0

4 2 0 7 0 4 4 3 8 5

1 4 6 6 4 0 5 4 0 6

7 4 4 9 4 5 0 1 3 3

7 0 9 3 3 4 1 0 0 1

8 7 5 5 8 0 3 0 0 1

4 0 1 0 5 6 3 1 1 0

#

0 5 0 5 4 7 6 8 6 0

5 0 8 1 1 2 8 1 4 6

0 8 0 6 7 1 9 7 0 4

5 1 6 0 4 1 4 2 8 3

4 1 7 4 0 0 2 0 7 4

7 2 1 1 0 0 5 3 9 7

6 8 9 4 2 5 0 8 1 1

8 1 7 2 0 3 8 0 9 4

6 4 0 8 7 9 1 9 0 9

0 6 4 3 4 7 1 4 9 0

#

0 8 7 5 0 1 8 5 4 6

8 0 9 4 5 4 4 1 1 5

7 9 0 6 0 5 1 9 5 0

5 4 6 0 4 1 5 7 2 1

0 5 0 4 0 3 0 9 7 1

1 4 5 1 3 0 3 4 4 5

8 4 1 5 0 3 0 2 5 4

5 1 9 7 9 4 2 0 4 6

4 1 5 2 7 4 5 4 0 1

6 5 0 1 1 5 4 6 1 0

#

0 5 5 4 2 2 8 3 1 6

5 0 7 7 5 1 5 3 2 1

5 7 0 7 7 3 5 1 3 4

4 7 7 0 1 5 8 8 6 7

2 5 7 1 0 9 1 0 2 8

2 1 3 5 9 0 3 6 1 5

8 5 5 8 1 3 0 8 5 4

3 3 1 8 0 6 8 0 4 3

1 2 3 6 2 1 5 4 0 6

6 1 4 7 8 5 4 3 6 0

#

0 3 2 1 5 9 1 2 6 8

3 0 8 5 8 5 9 7 1 1

2 8 0 5 8 3 1 7 1 0

1 5 5 0 4 9 6 0 7 9

5 8 8 4 0 1 2 6 5 1

9 5 3 9 1 0 5 2 9 1

1 9 1 6 2 5 0 2 4 0

2 7 7 0 6 2 2 0 1 0

6 1 1 7 5 9 4 1 0 0

8 1 0 9 1 1 0 0 0 0

#

0 4 2 0 7 2 0 5 8 5

4 0 9 1 0 7 3 8 7 3

2 9 0 3 2 3 7 5 0 5

0 1 3 0 3 6 4 1 2 9

7 0 2 3 0 9 0 0 2 6

2 7 3 6 9 0 0 6 2 3

0 3 7 4 0 0 0 2 4 2

5 8 5 1 0 6 2 0 8 4

8 7 0 2 2 2 4 8 0 5

5 3 5 9 6 3 2 4 5 0

#

0 4 0 6 6 9 8 6 3 3

4 0 4 9 8 5 1 6 4 3

0 4 0 7 2 5 7 2 5 0

6 9 7 0 0 2 8 3 4 6

6 8 2 0 0 2 1 9 0 6

9 5 5 2 2 0 8 3 8 6

8 1 7 8 1 8 0 0 0 3

6 6 2 3 9 3 0 0 7 4

3 4 5 4 0 8 0 7 0 1

3 3 0 6 6 6 3 4 1 0

#

0 7 2 6 0 6 5 5 1 9

7 0 0 3 3 5 1 0 2 3

2 0 0 1 0 4 3 6 3 3

6 3 1 0 5 0 1 8 6 8

0 3 0 5 0 8 7 3 7 0

6 5 4 0 8 0 9 2 5 9

5 1 3 1 7 9 0 2 8 9

5 0 6 8 3 2 2 0 9 3

1 2 3 6 7 5 8 9 0 4

9 3 3 8 0 9 9 3 4 0

#

0 3 3 5 4 2 5 5 5 5

3 0 5 4 4 5 8 9 4 1

3 5 0 0 6 7 2 9 5 3

5 4 0 0 4 0 4 8 7 9

4 4 6 4 0 0 1 8 5 0

2 5 7 0 0 0 8 9 6 4

5 8 2 4 1 8 0 0 0 6

5 9 9 8 8 9 0 0 3 5

5 4 5 7 5 6 0 3 0 6

5 1 3 9 0 4 6 5 6 0

#