Нахождение кратчайшего гамильтонова цикла. Программа Рубана Е.А и Быстрых А.В

1. Ввод:

Сначала идёт кол-во вершин (городов), затем верхняя часть матрицы через пробел (пример ввода есть в файле graph.txt)

1. Вывод:

Сначала выводятся входные данные (длины ребер, в виде верхней части матрицы),

Затем длина полученного решения и порядок вершин через точку

(в папке находится пять примеров в формате .docx, программа выводит в out.txt)

1. Список основных переменных:

n – кол-во вершин

size – кол-во рёбер

lft – кол-во отсутствующих рёбер

RE – номер элемента в таблице, который мы проверяем на совместимость с уже выбранными рёбрами

MZ – максимальное значение, дальше которого нельзя брать рёбра

i – какое по счёту ребро мы добавляем к нашему циклу, от этого числа зависит MZ

F[] – первая координата в таблице рёбер

T[] – вторая координата в таблице рёбер

L[] – длина ребра в таблице

a[] – хранит рёбра из текущего решения

tares[] – хранит цикл самого короткого решения

C1[] – хранит начала цепочек из выбранных рёбер

C2[] – хранит концы цепочек из выбранных рёбер

Reb[] – хранит кратности всех вершин с учетом текущего решения

Sres – хранит длину самого короткого найденного решения

1. Алгоритм:

Программа берёт входные данные из файла graph.txt.

Создаётся представление всех рёбер в виде таблицы

Отсутствующие рёбра остаются в памяти но мы их не рассматриваем

Дальше цикл, пока не будут перебраны все варианты или брать новое решение для самого первого ребра нецелесообразно

А) проверяем, не достигли ли мы максимального значения для текущего ребра, если достигли – возвращаемся к поиску предыдущего ребра

Б) Проверка на допустимость.

Состоит из двух этапов:

1. проверка вершин ребра на кратность, если его вершины имеют кратность 2, то это ребро не допустимо
2. не получится ли с добавлением этого ребра цикл. Если этим ребром в текущем решении создаётся цикл, то это ребро не допустимо, поскольку цикл должен появится только в конце

В) Если ребро допустимо, проверяем по сумме следующих элементов. Если сумма длин выбранных ребер и j ближайших к ним (j – кол-во рёбер, необходимое для полного решения) больше или равна по уже найденному решению, то не рассматриваем ребро. Если меньше, то добавляем его к текущему решению. (добавляем его к список цепочек C1/2[], изменяем кратности вершин в reb[], сохраняем ребро в a[])

Г) Если осталось найди одно ребро, находим его (если возможно), считаем сумму и при необходимости сохраняем полученное решение в tares[]

4.1) Добавление ребра в цепочки:

Проверяем, можно ли присоединить к уже имеющейся цепочке выбранное ребро. Если можно – присоединяем, путём замены начала или конца цепочки на новое. Если нельзя – создаём новую цепочку.

Затем проверяем, можно ли соединить две имеющихся цепочки в одну (у них имеются одинаковые вершины на концах), если можно – соединяем