УДК 004.021

П. Г. Рагулина

P. G. Ragulina

Рагулина Полина Григорьевна, студентка 4 курса, группа ИНаз-16-1, НФИ КемГУ, г. Новокузнецк, Россия.
Научный руководитель: Можаров Максим Сергеевич, канд. пед. наук, профессор, зав. кафедры ИОТД ФГБОУ ВО НФИ КемГУ, г. Новокузнецк, Россия.

Ragulina Polina Grigoryevna, 4th year student, NFI KemSU, Novokuznetsk, Russia.
Scientific adviser: Mozharov Maksim Sergeevich, Candidate of Pedagogical Sciences, Professor, Head of the Department of Engineering and Technology, FSBEI HE NFI KemSU, Novokuznetsk, Russia.

РЕШЕНИЕ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ ПО ТЕМЕ «ЗАДАЧИ НА ГРАФАХ»

SOLUTION OF PROGRAMMING PROBLEMS ON THE TOPIC «TASKS ON GRAPHS»

Аннотация. *Данная статья знакомит с олимпиадными задачами повышенной сложности для школьников по программированию. В ней представлено решение задач на графах. Кроме того, в статье предложены задачи для самостоятельной работы.*

Abstract. *This article introduces the Olympiad problems of increased complexity for students in programming. It presents the solution of problems on graphs. In addition, the article suggests tasks for independent work.*

Ключевые слова: *обучение, программирование, графы, теория графов, задачи повешенной сложности, задача по программированию.*

Keywords: *training, programming, graphs, graph theory, hanging complexity problems, programming task.*

Теория графов – один из обширнейших разделов дискретной математики, широко применяется в решении экономических и управленческих задач, в программировании, химии, конструировании и изучении электрических цепей, коммуникации, психологии, психологии, социологии, лингвистике, других областях знаний. Теория графов систематически и последовательно изучает свойства графов, о которых можно сказать, что они состоят из множеств точек и множеств линий, отображающих связи между этими точками. Основателем теории графов считается Леонард Эйлер (1707-1882), решивший в 1736 году известную в то время задачу о кёнигсбергских мостах [5].

Граф – это абстрактный математический объект. Он состоит из вершин и ребер. Каждое ребро соединяет пару вершин. Если одну и ту же пару вершин соединяют несколько ребер, то эти ребра называются кратными. Ребро, соединяющее вершину с ней самой, называется петлей. По ребрам графа можно ходить, перемещаясь из одной вершины в другую. В зависимости от того, можно ли по ребру ходить в обе стороны, или только в одну, различают неориентированные и ориентированные графы соответственно. Ориентированные ребра называются дугами. Если у всех ребер графа есть вес (т.е. некоторое число, однозначно соответствующее данному ребру), то граф называется взвешенным. Вершины, соединенные ребром, называются соседними. Для неориентированного графа степень вершины – число входящих в нее ребер. Для ориентированного графа различают степень по входящим и степень по исходящим ребрам. Граф называется полным, если между любой парой различных вершин есть ребро [6].

Так как граф является абстрактным объектом, интерпретировать его мы можем по-разному, в зависимости от конкретной задачи. Рассмотрим пример. Пусть вершины графа – города, а ребра – дороги, их соединяющие. Если дороги имеют одностороннее движение, то граф ориентированный, иначе неориентированный. Если проезд по дорогам платный, то граф взвешенный.

На бумаге граф удобно представлять, изображая вершины точками, а ребра – линиями, соединяющими пары точек. Если граф ориентированный, на линиях нужно рисовать стрелочку, задающую направление; если граф взвешенный, то на каждом ребре необходимо еще надписывать число – вес ребра [8].

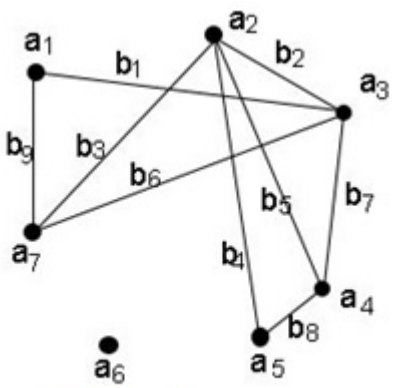

Рисунок 1. Чертёж к теории графов

На рисунке 1 представлен чертёж к теории графов: построение графа для отображения отношения знакомства между людьми [1].

Часто в задачах встречается следующая конструкция – есть дома и дороги, их соединяющие; у каждой дороги есть длина. По другой терминологии такая конструкция называется графом, дома называются «вершинами», дороги – «ребрами» или «дугами», а длина дороги «весом ребра» или «весом дуги». Таким образом, фраза «Найти минимальный вес пути между вершинами s и k в графе» может быть переведена так: «Есть дома и дороги их соединяющие. Также заданы длины дорог. Найти кратчайшую длину пути от города s до города k, если двигаться можно только по дорогам». Пропускная способность дуги (i, j) означает, например, сколько груза может быть перевезено по дороге (по дуге) (i, j) за единицу времени); а поток по дуге (i, j) – это сколько перевозится сейчас на самом деле) [7].

Часто используют следующие обозначения: Г(x(i)) – множество вершин, в которые есть дуга из вершины i; Д(x(i)) – множество вершин, из которых есть дуга в вершину I [4].

Пусть в графе N вершин.

Длины дуг обычно заносятся в матрицу (назовем ее C) размера N на N, называемой матрицей смежности:

var C: array [1..N,1..N] of integer;

Элемент C[i, j] этой матрицы равен длине дуги (дороги), соединяющей вершины i и j, и равен (например) 0 или -1, если такой дуги нет. Если дорога двунаправленная (дуга неориентированная), то очевидно, что C[i, j] = C[j, i] [3].

Алгоритм расстановки пометок для задачи о максимальном (от s к t) потоке.

А. Расстановка пометок. Вершина может находиться в одном из трех состояний: вершине приписана пометка и вершина просмотрена (т.е. она имеет пометку и все смежные с ней вершины «обработаны»), пометка приписана, но вершина не просмотрена (т.е. она имеет пометку, но не все смежные с ней вершины обработаны), вершина не имеет пометки. Пометка произвольной вершины x(i) состоит из двух частей и имеет один из двух видов: (+x(j), m) или (-x(j), m). Часть +x(j) пометки первого типа означает, что поток допускает увеличение вдоль дуги (x(j), x(i)). Часть -x(j) пометки другого типа означает, что поток может быть уменьшен вдоль дуги (x(i), x(j)). В обоих случаях m задает максимальную величину дополнительного потока, который может протекать от s к x(i) вдоль построенной увеличивающей цепи потока. Присвоение пометки вершине x(i) соответствует нахождению увеличивающей цепи потока от s к x(i). Сначала все вершины не имеют пометок.

- **Шаг 1.** Присвоить вершине s пометку (+s, m(s) = бесконечность). Вершине s присвоена пометка и она просмотрена, все остальные вершины без пометок.

- **Шаг 2.** Взять некоторую непросмотренную вершину с пометкой; пусть ее пометка будет (+-x(k), m(x(i))) (+- обозначает, что перед x(k) может стоять как плюс, так и минус).

(I) Каждой помеченной вершине x(j), принадлежащей Г(x(i)), для которой c(i, j)<q(i, j=""> </q(i,>

(II) Каждой непомеченной вершине x(j), принадлежащей Д(x), для которой c(i, j)>0, присвоить пометку (-x(i), m(x(j))), где m(x(j)) = min[m(x(i)), c(j, i)].

(Теперь вершина x(i) и помечена, и просмотрена, а вершины x(j), пометки которым присвоены в (I) и (II), являются не просмотренными). Обозначить каким-либо способом, что вершина x(i) просмотрена.

- **Шаг 3.** Повторять шаг 2 до тех пор, пока либо вершина t будет помечена, и тогда перейти к шагу 4, либо t будет не помечена и никаких других пометок нельзя будет расставить; в этом случае алгоритм заканчивает работу с максимальным вектором потока c. Здесь следует отметить, что если X(0) – множество помеченных вершин, а X'(0) – множество не помеченных, то (X(0) --> X'(0)) является минимальным разрезом.

Б. Увеличение потока.

- **Шаг 4**. Положить x=t и перейти к шагу 5.
- **Шаг 5**. (I) Если пометка в вершине x имеет вид (+z, m(x)), то изменить поток вдоль дуги (z, x) с c(z, x) на c(z, x) + m(t). (II) Если пометка в вершине x имеет вид (-x, z) с c(x, z) на c(x, z) – m(t).
- **Шаг 6.** Если z = s, то стереть все пометки и вернуться к шагу 1, чтобы начать расставлять пометки, но используя уже улучшенный поток, найденный на шаге 5. Если z <> s, то взять x = z и вернуть к шагу 5.

Обозначения: Г(x(i)) – множество вершин, в которые есть дуга из вершины i; Д(x(i)) – множество вершин, из которых есть дуга в вершину i; c(i, j) – это пропускная способность дуги (т.е., например, сколько груза может быть перевезено по дороге (по дуге) (i, j) за единицу времени); q(i, j) – поток по дуге (i, j) (т.е. сколько перевозится сейчас на самом деле).

Кратчайшее расстояние от вершины *нач* до остальных вершин.

Алгоритм Дейкстры.

Обозначения

C[i, j] – длина ребра (i, j), C[i, j] >= 0 (если ребра нет, то его длина полагается равной бесконечности).

D[i] – кратчайшее текущее расстояние от вершины нач до вершины i.

Флаг [i] – информация о просмотре вершины i: 0 – если вершина не просмотрена, 1 – если просмотрена. Если вершина просмотрена, то для нее D[i] есть наикратчайшее расстояние от вершины нач до вершины i.

Предок [i] – информация о номере вершины, предшествующей вершине i в кратчайшем пути от вершины нач.

Минрас – это минимальное расстояние [2].

Алгоритм

для i от 1 до N выполнять

нц

предок[i]:=нач;

флаг[i]:=0;

D[i]:=C[нач,i]

кц

флаг[нач]:=1; {пока мы знаем только расстояние}

предок[нач]:=0 {от вершины нач до нее же, равное 0}

для i от 1 до N-1 выполнять

нц

минрас:=бесконечность;

для j от 1 до N выполнять

если (флаг[j]=0 и (минрас > D[j]) {находим минимальное}

то минрас:=D[j]; {расстояние}

k:=j; {до непомеченных вершин}

все

флаг[k]:=1; {вершина k помечается просмотренной}

для j от 1 до N выполнять {выполняем просмотр}

если флаг[j]=0 и D[j]>D[k]+C[k,j]

{Т.е. если для вершины j еще не найдено кратчайшее расстояние

от нач, и из вершины k по дуге C[k,j] путь в j короче,

чем найденный ранее}

то D[j]:=D[k]+C[k,j] {то запоминаем его}

предок[j]:=k;

все

кц

Рассмотрим несколько таких задач с решением.

Задача 1. Имеется N городов. Для каждой пары городов (I, J) можно построить дорогу, соединяющую эти два города и не заходящие в другие города. Стоимость такой дороги A(I, J). Вне городов дороги не пересекаются.

Написать алгоритм для нахождения самой дешевой системы дорог, позволяющей попасть из любого города в любой другой. Результаты задавать таблицей B[1:N, 1:N], где B[I, J] = 1 тогда и только тогда, когда дорогу, соединяющую города I и J, следует строить.

Решение задачи 1.

Легко понять, что сеть дорог будет реализовывать некоторый связный (так как можно проехать из любого города в любой) граф без циклов (так как одно ребро из цикла можно выбросить, а связный граф останется связным). Поэтому алгоритм построения сети дорог минимальной суммарной стоимости очень прост. На каждой итерации необходимо находить дорогу минимальной стоимости, которая не образует цикла с уже выбранными дорогами на предыдущих итерациях. Основную трудность такого решения составляет проверка условия, образуют ли ребра цикл.

Однако решение существенно упрощается, если рассматривать только минимальные ребра только между двумя множествами: множеством помеченных вершин и множеством непомеченных вершин. Понятно, что эти множества должно соединять хотя бы одно ребро, чтобы граф был связным. Ясно, что оно должно быть минимальным по длине. В описываемом ниже алгоритме это делается следующим образом.

Для каждой вершины к из множества непомеченных вершин (а на начальном этапе это все вершины, кроме первой) определяется ближайшая вершина из множества помеченных вершин БЛИЖ[к]. На каждой итерации определяется кратчайшее ребро (i, j) между множеством помеченных вершин и множеством непомеченных вершин, используя массив БЛИЖ. Найденное ребро выбирается для системы дорог, а соответствующая вершина j считается помеченной. После этого пересчитывается массив БЛИЖ. При этом учитывается, что к изменение некоторой величины БЛИЖ[k] может произойти только тогда, когда расстояние от k до j меньше, чем от k до БЛИЖ[k].

Алгоритм

для i от 1 до N выполнять

нц

флаг[i]:=0;

БЛИЖ[i]:=1

кц

флаг[1]:=1;

для k от 1 до N-1 выполнять

нц

минрас:=бесконечность;

для i от 2 до N выполнять

если флаг[i]=0 и минрас > C[БЛИЖ[i],i]

то минрас:=C[БЛИЖ[i],i];

j:=i;

все

Вывод ребра (БЛИЖ[j],j)

флаг[j]:=1;

для i от 2 до N выполнять

если флаг[i]=0 и C[БЛИЖ[i],i]>C[i,j]

то БЛИЖ[i]:=j;

все

кц

Задача 2. Задан набор неповторяющихся пар (Ai, Aj), Ai и Aj принадлежат множеству A={A1, A2, ..., An}. Необходимо составить цепочку максимальной длины по правилу (Ai, Aj) + (Aj, Ak) = (Ai, Aj, Ak).

При образовании этой цепочки любая пара может быть использована не более одного раза.

Решение задачи 2.

Для более удобного хранения информации заведем матрицу C[1...n,1..n] (так называемую матрицу смежности), в которой C[i, j] = 1, если в наборе есть пара (Ai, Aj) и C[i, j] = 0 иначе.

Будем строить все возможные цепочки (по правилу, данному в условии) и искать среди них ту, которая имеет максимальную длину.

В качестве начального символа цепочки можно взять любой символ из A. Пусть это символ Ai. Ищем, просматривая строку i матрицы C слева направо элемент C[i, j] = 1 (другими словами, ищем пару с первым элементом Ai). Если такого элемента не существует, то берем в качестве начала строки другой элемент множества A. Если элемент C[i, j] = 1 найден, то ему соответствует пара (Ai, Aj). Помечаем ее как уже использованную, полагая, например, C[i, j] = -1. Далее просматриваем слева направо строку j матрицы C в поисках еще не использованной пары (Aj, Ak) (C[j, k] = 1). Присоединяем элемент Ak к имеющейся цепочке, полагаем C[j, k] = -1, ищем единичный элемент в строке k и т.д. Предположим, на некотором шаге мы получили цепочку Ai Aj Ak ... As Al Ap и в строке p матрицы больше нет ни одного единичного элемента. Это означает, что при таком подборе предыдущих элементов мы нашли максимальную по длине строку. Если ее длина больше длин всех найденных ранее строк, запоминаем эту строку как рекорд. После этого «отщепляем» от строки последний элемент Ap и смотрим, есть ли еще в строке l единичный элемент с индексом, большим p. Если да, то приписываем уже этот элемент к строке и пытаемся затем снова увеличить длину полученной строки, если же нет, то «отщепляем» от строки элемент A1, в строке S ищем единичный элемент с индексом, большим l и т.д.

Останов осуществляется тогда, когда мы должны «отщепить» от строки Ai.

Перебираем цепочки, начинающиеся со всех возможных элементов множества A. Находим строку максимальной длины:

```
const M=10; {максимально число элементов в A}

{будем считать, что A состоит из чисел от 1 до N}

var c:array[1..M,1..M] of integer;

curstr, maxstr: array[0..M] of integer;

{в этих переменных хранятся текущая цепочка и}

{цепочка максимальной длины.}

{В нулевом элементе хранится длина цепочки}

N,E : integer; {N - число элементов в A}

i,j,k : integer; {E - число пар в наборе}

procedure find;

var l,j : integer;

begin

l:=curstr[curstr[0]]; {l = последний элемент цепочки}

for j:=1 to N do {просмотр строки l}
```

```
if C[l,j]=1
then begin
curstr[0]:=curstr[0]+1;
curstr[curstr[0]]:=j; {j -> в цепочку}
c[l,j]:=-1; {пара использована}
find;
c[l,j]:=1; {пару снова разрешено использовать}
curstr[0]:=curstr[0]-1;
end;
if curstr[0]>maxstr[0] {если нашли более}
then maxstr:=curstr {длинную строку}
end;
begin
readln(N); readln(E);
for i:=1 to N do
for j:=1 to N do
C[i,j]:=0;
for k:=1 to E do begin
write('очередная пара: ',i,j);
c[i,j]:=1
end;
for i:=1 to N do begin
curr[0]:=1; {поиск цепочки}
curr[1]:=i; {начинающейся элементом i}
find;
end;
for i:=1 to maxstr[0] do
write(maxstr[i]); {печать максимальной строки} end.
```

Задачи для самостоятельной работы.

Задача 1. На плоскости расположено N точек. Имеется робот, который двигается следующим образом. Стартуя с некоторой начальной точки и имея некоторое начальное направление, робот движется до первой встреченной на его пути точки, изменяя в ней свое текущее направление на 90 градусов, т.е. поворачивая налево или направо. После этого он продолжает движение аналогично. Если робот достиг начальной точки, либо не может достичь новой точки (которую он еще не посещал), то он останавливается.

Определить, может ли робот посетить все N точек, если:

- определены начальные точка и направление робота;
- определена начальная точка, а направление робота можно выбирать;
- начальную точку и направление робота можно выбирать.

Координаты точек – целые числа, угол измеряется в радианах относительно оси OX.

Задача 2. «ПУТЬ». Найти кратчайшее расстояние между двумя вершинами в графе. Найти все возможные пути между этими двумя вершинами в графе не пересекающиеся по

а) ребрам
б) вершинам.

Задача 3. Имеется N прямоугольных конвертов и N прямоугольных открыток различных размеров. Можно ли разложить все открытки по конвертам, чтобы в каждом конверте было по одной открытке.

Замечание. Открытки нельзя складывать, сгибать и т.п., но можно помещать в конверт под углом. Например, открытка с размерами сторон 5:1 помещается в конверты с размерами 5:1, 6:3, 4.3:4.3, но не входит в конверты с размерами 4:1, 10:0.5, 4.2:4.2.

Задача 4. Имеется N человек и прямоугольная таблица A[1:N,1:N]; элемент A[i, j] равен 1, если человек i знаком с человеком j, A[i, j] = =A[j, i]. Можно ли разбить людей на 2 группы, чтобы в каждой группе были только незнакомые люди.

Задача 5. N различных станков, один за другим объединенных в конвейер. Имеется N рабочих. Задана матрица C[N , N], где C[i, j] производительность i-ого рабочего на j-ом станке. Определить:

а) на каком станке должен работать каждый из рабочих, чтобы производительность была максимальной;
б) то же, но станки расположены параллельно и выполняют однородные операции.

Список литературы

1. Альхова, З. Н. Внеклассная работа по математике [Текст]. / З. Н. Альхова, А. В. Макеева. – Саратов : «Лицей», 2002.

2. Башмаков, М. И. Математика в кармане «Кенгуру» [Текст]. / М. И. Башмаков. – М. : Дрофа, 2010.
3. Игнатьев, Е. И. Математическая смекалка [Текст]. / Е. И. Игнатьев. – М. : «Омега», 1994.
4. Григорьева, Г. И. Подготовка школьников к олимпиадам по математике. 5-6 классы [Текст]. / Сост. Г. И. Григорьева. – М. : «Глобус», 2009.
5. Квант [Текст]. // Физико-математический журнал «Квант», 1994. – № 6.
6. Савин, А. П. Энциклопедический словарь юного математика [Текст]. / Сост. А. П. Савин. – М. : Педагогика, 1989.
7. Фарков, А. В. Олимпиадные задачи по математике и методы их решения [Текст]. / А. В. Фарков. – М. : Народное образование, 2003.
8. Aisuluu Основные методы и приёмы решения олимпиадных математических задач [Электронный ресурс]. – г. Свислочь, 2011. – Режим доступа : http://pandia.ru/text/78/086/35870.php