Интернет-журнал «Мир науки» ISSN 2309-4265 http://mir-nauki.com/

2016, Том 4, номер 6 (ноябрь - декабрь) http://mir-nauki.com/vol4-6.html

URL статьи: http://mir-nauki.com/PDF/21PDMN616.pdf

Статья опубликована 22.11.2016

Ссылка для цитирования этой статьи:
Пирогов В.Ю. Олимпиадные задачи по программированию с рекурсией на графах // Интернет-журнал «Мир науки» 2016, Том 4, номер 6 http://mir-nauki.com/PDF/21PDMN616.pdf (доступ свободный). Загл. с экрана. Яз. рус., англ.

УДК 37

Пирогов Владислав Юрьевич[1]
ФГБОУ ВО «Шадринский государственный педагогический университет», Россия, Шадринск
Заведующий кафедрой «Программирования и автоматизации бизнес процессов»
Кандидат физико-математических наук, доцент
E-mail: Vladislav-133@yandex.ru

Олимпиадные задачи по программированию с рекурсией на графах

Аннотация. Статья посвящена олимпиадным задачам по программированию, для решения которых используются рекурсивные алгоритмы. Такие алгоритмы наиболее сложны для понимания студентов. Автор показывает, что для решения таких задач может быть использовано представление предметной области в виде графа. Это повышает наглядность и помогает найти нужный алгоритм. В статье разбираются две задачи. Обе задачи были предложены на одной из олимпиад по программированию, проводимых в Шадринском государственном педагогическом университете. Показано, что в первой задаче алгоритм может быть описан в виде циклов на неориентированном гамильтоновом графе. Для программной реализации был использован рекурсивный алгоритм. При анализе алгоритма автор обращает внимание на обработку ошибочных рекурсивных вызовов. Также указывается, что полученное решение дает возможность обнаружить критерии того, является данный граф гамильтоновым или нет. В условии второй задачи предметная область не имеет графического представления. В статье показывается, что с использованием ориентированных графов предметная область становится более понятной студентам для написания рекурсивного алгоритма. Автор подробно разбирает ту часть алгоритма, которая содержит условия поиска ситуаций, когда путь на графе возвращается в исходную точку. Автор также обсуждает критерии поиска оптимального пути.

Ключевые слова: программирование; язык программирования; граф; олимпиадная задача; рекурсия

Введение

Теория графов – один из разделов дискретной математики, рассматривающий свойства множества вершин и соединяющих их дуг [1]. Теория графов тесно связана с практическими задачами, в частности, с поиском кратчайшего пути между двумя точками на плоскости. В действительности многие задачи, не связанные, на первый взгляд, с какими-либо географическими построениями, можно также наглядно представить в виде задачи на графе.

[1] 641876, Россия, Курганская область, г. Шадринск, ул. Ефремова 24-21

Особенно это касается задач, связанных с поиском каких-либо оптимальных решений. Поскольку большое количество задач, которые приходится решать современным программистам, связана с нахождением именно оптимального решения, то теория графов стала тем инструментом, с которым должен быть знаком каждый студент, готовящийся стать программистом [2, 3].

Олимпиадное программирование в Вузе является одним из элементов подготовки будущих разработчиков программного обеспечения [5, 6, 7, 8]. В Шадринском государственном педагогическом университете на факультете Информатики, математики и физики с 2005 года проходят традиционные очные и заочные олимпиады по программированию. Накопился уже довольно большой архив оригинальных задач. В данной статье мы разберем две задачи, решение которых можно описать с помощью ориентированных и неориентированных графов. Обе представленные ниже задачи являются авторскими и были предложены участникам на одной из проводимых нами олимпиад. Для решения задач автором был использован язык программирования С [9].

Все задачи, связанные с использованием графов можно разделить на два класс: 1. Задачи, с некоторыми геометрическими построениями, явно указывающими на возможность использования графов. 2. Задачи, использование в которых графов, позволяет по-другому представить и постановку задачи, и ее решение.

Задача 1

Все задачи на олимпиадах по программированию принято сопровождать некоторой фабулами. Не составляет исключения и данная задача, которая был названа «Умная мышь». Рассмотрим условие.

В огромном замке квадратной формы, состоящей из n одинаковых комнат, живет мышь. Ночью, в одно и тоже время, открываются двери всех комнат и в каждой из комнат лежит маленький кусочек сыра. Мышь пробегает по всем комнатам, съедает сыр и возвращается в свой домик, который расположен в одной из комнат. Если она не успеет это сделать, то двери захлопнутся и мышь погибнет. Написать программу, которая находит все маршруты продвижения мыши по замку.

В исходном файле (input.txt) в первой строке указано количество комнат вдоль одного ребра квадрата (длина ребра). Во второй строке через пробел указаны координаты комнаты, где проживает мышь.

В выходном файле должны содержаться все пути, по которым мышь может обойти все комнаты и количество таких путей.

Например,

Input.txt

2

1 0

Output.txt

1 1

0 1

0 0

1 0

---

0 0

0 1

1 1

1 0

2

Таким образом, на выходе указываются координаты всех шагов мыши, в том числе и последние координаты – координаты комнаты, где мышь живет.

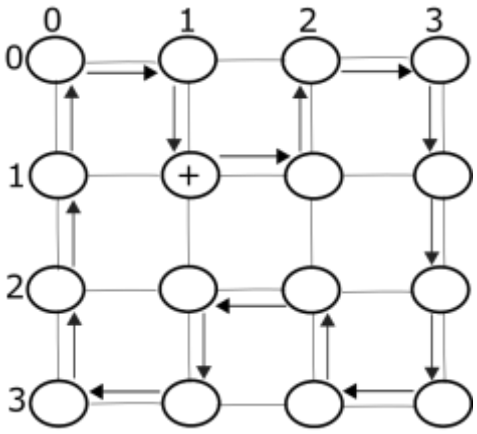

Рисунок 1. *К задаче 1. Граф с указанием одного из возможных обходов мыши с возвратом в исходную комнату. Исходная комната обозначена знаком «+» (разработано автором)*

Решение

На рисунке 1 представлен граф, показывающий возможный путь обхода умной мышью всех комнат замка. Такой путь в теории графов называется ***циклом*** [1, 2]. Домик мыши расположен в клетке, отмеченной знаком «+». Легко видеть, что таких путей несколько. Расчет показывает, что для квадрата 4 на 4 их 12, в каком бы месте ни был бы домик мыши. Обратим ваше внимание, что мышь проходит все комнаты (все узлы графа), но не все дуги (ребра графа). Если в графе для каждого из узлов существуют такие циклы, то граф называется ***гамильтоновым*** [1][2]. В данном примере (рисунок 1) граф, т.о. гамильтонов. Не существует точных критериев того, является данный граф гамильтоновым или нет, однако, написав программу, мы сможем чисто эмпирически находить гамильтоновы графы. Поиск нужных путей на графе удобно осуществлять, используя пошаговую проверку с возможностью отхода. Такая проверка достаточно просто реализуется с помощью рекурсивных алгоритмов.

Уже само условие задачи подсказывает нам, что движение мыши удобно показывать в двумерном числовом массиве (далее в программе он обозначен именем mz). Например, положение домика мыши обозначим -1, те, комнаты в которых мышь еще не была - 0, а те комнаты, которых она побывала - 1. Это конечно еще не решение, но уже не плохая идея, на основе которой можно писать программу. Для того, чтобы перебрать все возможные пути, необходимо реализовать алгоритм, в котором на каждом шаге можно выбирать путь, а также

[2] Если для каждого из узлов существует цикл с проходом всех ребер графа, то граф называется эйлеровым [1].

отступать, чтобы выбрать пути из предыдущего положения, которые еще не были реализованы. Напрашивается рекурсивный алгоритм, в котором рекурсивная функция представляет собой очередной шаг мыши, точнее попытка его сделать, с возможностью отойти назад. Решение задачи представлено в листинге 1.

Листинг 1. Решение задачи Умная мышь

```
#include <stdio.h>
//прототипы функций
void rec(int, int, int);
//глобальные переменные
int mz[100][100];
int x2[10000],y2[10000];
int n,x,y,i,j,m,k;
//основная функция
int main(){
//получить длину стороны помещения в комнатах
 scanf("%d",&n);
 if(n>100||n<=0)return 1;
 m=n*n;
//получить начальные координаты мыши
 scanf("%d%d",&x,&y);
//инициализации
 for(i=0; i<n; i++){
          x2[i]=0; y2[i]=0;
          for(j=0; j<n; j++){
           mz[i][j]=0;
          }
 }
//вызов рекурсивной функции
 mz[x][y]=-1; k=0;
 rec(x+1,y,1);
 rec(x,y+1,1);
 rec(x-1,y,1);
 rec(x,y-1,1);
 printf("%d\n",k);
 return;
}
//рекурсивная функция обхода замка
void rec(int x1, int y1, int i){
```

```
//нет ли выхода за пределы замка
 if(x1<0||y1<0||y1==n||x1==n)return;
//ложные шаги?
 if(mz[x1][y1]==-1&&i<m){
        return;
 };
 if(mz[x1][y1]==1){
        return;
 };
 if(mz[x1][y1]==0&&i==m){
        return;
 };
//запоминаем положение
 x2[i-1]=x1; y2[i-1]=y1;
//мышь вернулась?
 if(mz[x1][y1]==-1){
//вывод пути
        for(j=0; j<i; j++){
         printf("%d %d\n",x2[j],y2[j]);
        }
        printf("---------\n");
        k++;
//шаг назад
        return;
 }
 mz[x1][y1]=1;
//делаем ход
 rec(x1+1,y1,i+1);
 rec(x1,y1+1,i+1);
 rec(x1-1,y1,i+1);
 rec(x1,y1-1,i+1);
 mz[x1][y1]=0;
 return;
}
```

Самое важное в программе из листинга 1 это рекурсивная функция rec, все остальное не вызывает сложности для понимания. Данная функция реализует попытку шага мыши. Обратим также внимание на массив mz, выше мы говорили об идее использования такого массива.

Остановимся на начальном фрагменте функции, в котором реализована обработка неправильных шагов мыши. Для удобства приведем его еще раз здесь.

```
//нет ли выхода за пределы замка
 if(x1<0||y1<0||y1==n||x1==n)return;
//ложные шаги?
 if(mz[x1][y1]==-1&&i<m){
        return;
 };
 if(mz[x1][y1]==1){
        return;
 };
 if(mz[x1][y1]==0&&i==m){
        return;
 };
```

Перечислим, в порядке их следования в программе, неправильные шаги мыши: 1. Выход за пределы замка. Проверяется положение мыши, поскольку выполнение функции это уже шаг (см. вызов рекурсивной функции в тексте программы). 2. Возврат в исходную точку (mz[x1][y1]=-1), когда еще не все комнаты пройдены. Здесь m это количество шагов мыши, необходимых, чтобы обойти весь замок, m=n×n. 3. Попытка перейти в комнату, где она уже была. 4. Ситуация, когда мышь прошла все комнаты, но не попала в исходную. При наступлении любого из этих событий происходит возврат в предыдущую позицию. В этом красота рекурсии.

Если шаг сделан правильно, положение мыши запоминается в массивах x2 и y2, при этом i – количество сделанных шагов (не забываем, что отчет координат и массивов ведется с нулевого значения).

Далее осуществляется проверка того, что полный путь пройден и мышь в исходной точке. Если это выполняется, то выводятся все шаги. При этом k – глобальная переменная будет содержать количество найденных путей. Если путь пройден, то осуществляется возврат, т.е. шаг назад.

Если мышь не достигла исходной точки, но шаг сделан правильно, то это место в массиве отмечается 1. После этого осуществятся набор рекурсивных вызовов. Обратим внимание, что после них, осуществляется шаг назад с восстановлением значения места в массиве (освобождением). Другими словами, мышь может уже пройти по этой комнате, прокладывая другой маршрут.

Запуская данную программу для исходных данных с различным начальным значением n, мы неожиданно обнаруживаем, что для нечетных значений n невозможно найти нужный путь для любого начального положения мыши. Другими словами, графы с нечетными n не являются гамильтоновыми. Конечно, это не может считаться строгим доказательством, но может послужить отправной точкой для дальнейших исследований. Данный факт можно использовать на занятиях по дисциплине, где рассматривается теория графов.

Задача 2

Данная задача тоже имеет занимательную фабулу и относится ко второму типу задач, когда условие непосредственно не указывает на использования для ее решения графов.

Рассмотрим следующую гипотетическую ситуацию. В некотором государстве работают N российских нелегальных разведчика (нелегала). В задачу этих разведчиков входит сбор ценной для нашего государства информации. Разведчики могут обмениваться информацией друг с другом. Ниже изображена схема связи четырех разведчиков A, B, C, D друг с другом.

	A	B	C	D
A	0	1	1	0
B	0	0	1	0
C	0	0	0	1
D	0	0	0	0

Цифра 1 на пересечении строки X и столбца Y означает, что разведчик X получает информацию от разведчика Y (имеет односторонний канал). Цифра 0, означает, что канал получения информации отсутствует. В нашем случае разведчик A получает информацию от разведчиков B и C, разведчик B получает информацию от разведчика C, разведчик C получает информацию от разведчика D, разведчик D не имеет каналов получения информации от других разведчиков. Разумеется, наша руководство заинтересовано, чтобы информация, добытая нелегалами, в конце концов, дошла до них. С другой стороны, чем меньше контактов будет между разведчиком и руководством, тем лучше. Таким образом, необходимо в общем случае найти минимальный набор нелегалов, контакт с которыми позволит получить всю информацию, добытую разведчиками. Например, в ситуации представленной выше достаточно контактировать только с разведчиком A, а в ситуации представленной схематично ниже с разведчиками A и D.

	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

Требования задачи.

Входной файл содержит строки одинаковой длины, состоящие из нулей и единиц. Агенты нашей разведки, таким образом, нумеруются цифрами от 0 до N-1 (для варианта, представленного ниже - от 0 до 4). Но первой строкой всегда идет число, равное количество нелегалов:

```
5
0 0 0 0 0
1 0 1 0 0
0 0 0 0 0
0 0 0 0 1
0 0 0 0 0
```

Необходимо найти группу с минимальным количеством нелегалов, с помощью которых можно получить всю добываемую информацию (всеми разведчиками). В выходном файле должны быть перечислены номера этих агентов. Например, для последней ситуации выходной файл будет содержать строку из двух цифр "1 3" (не забываем, что нумерация разведчиков ведется от 0).

Имя входного файла input.txt (или стандартный поток ввода), имя выходного файла output.txt (или стандартный поток вывода).

Важные замечания

1. В общем случае, может быть несколько групп, удовлетворяющих заданному условию (количество разведчиков в них минимально), но требуется найти только одну, все равно какую группу.

2. В поставленной выше задаче имеется одна серьезная проблема. Дело в том, что в общем случае информационные потоки между агентами могут замыкаться. Например, агент A получает информацию от агента B, а агент B получает информацию от агента A. Или более сложный вариант: агент A получает информацию от агента B, агент B получает информацию от агента C, агент C получает информацию от агента D, а агент D получает информацию от агента A (круг замкнулся). В случае наличия замыкания (петли) должно быть выведено сообщение об ошибке: Error.

3. Примем, что максимальное количество разведчиков-нелегалов не превышает 10.

Решение

Данная задача может быть описана с помощью графа. Предположим, что на входе мы имеем следующие данные

4

0 0 0 1

1 0 0 0

0 0 0 0

0 0 0 0

На рисунке 2 представлен граф, описывающий данный набор входной информации. Граф ориентирован, так как информация между нелегалами передается только в одну сторону. Мы видим, что информацию от нелегала 2 можно получить только непосредственно от него. Нелегал 0 получает информацию от нелегала 1. Но нелегал 3 получая информацию от нелегала 0, получает тем самым информацию и от нелегала 1. Следовательно правильным ответом (решением) задачи будет последовательность 2 3. В данном случае решение единственное. Как уже было упомянуто, программа должна учитывать тот случай, когда связи между узлами (разведчиками) образуют замкнутую линию. Другими словами, разведчики не добывают новой информации. К тому же случаю относится ситуация, когда узел замыкается сам на себе – разведчик всю информацию выдумывает. Наша задача, таким образом, написать программу обхода всех узлов диаграммы по порядку и выяснения минимального их количества для охвата всего потока информации. Я бы сформулировал задачу и по-другому: после обхода всех узлов должна быть заполнена некоторая структура, из которой уже легко получить правильный ответ.

2016, Том 4, номер 6
ISSN 2309-4265
(499) 755 50 99
http://mir-nauki.com

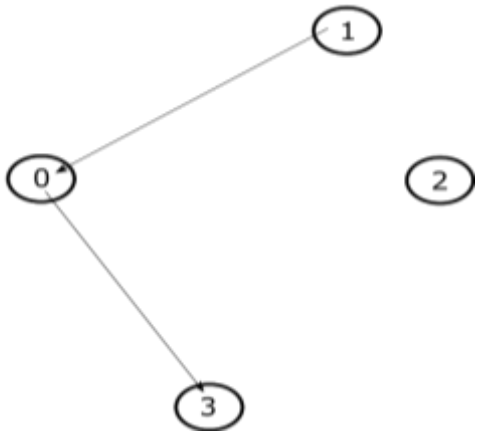

***Рисунок 2**. К задаче 2. Ориентированный граф информационных потоков между нелегалами для матрицы 4 на 4 (разработано автором)*

В условии задачи особо оговаривается ситуация, когда возникает замыкание или петля. Рассмотрим следующую матрицу, описывающую ситуацию.

0 0 0 0 1

1 0 0 0 0

1 0 0 0 0

0 1 0 0 0

0 0 0 1 0

Данный набор входной информации описывается следующим графом (см. Рисунок 3). На графе четко видно, что информация ходит по кругу и разведчики получают свою же информацию.

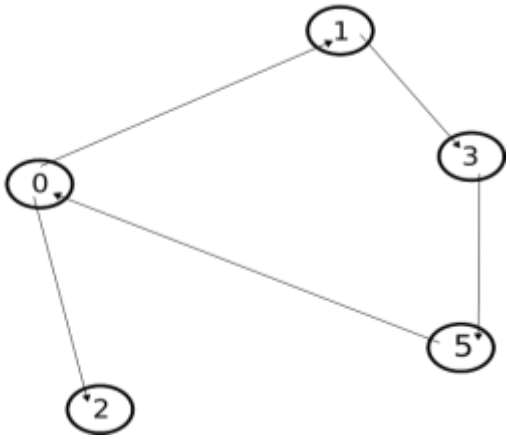

***Рисунок 3**. К задаче 2. Ориентированный граф информационных потоков между нелегалами для матрицы 5 на 5 (разработано автором)*

Из структуры исходных данных легко понять, что и в программе легче всего хранить информацию о разведчиках-нелегалах в виду числового массива. Единица на пересечении n-строки и m-ого столбца будет обозначать, что m-й разведчик снабжает информацией n-ого разведчика. Этот же массив можно использовать для хранения динамической картины, когда программа проходит маршруты, по которым проходит поток информации. При этом, можно изменять значение 1 на, скажем, 2, чтобы отмечать уже пройденный узел. При возврате из рекурсии нужно пошагово возвращать прежнее значение узла. Если при переходе в узел его

значение окажется равным 2, это будет означать, что поток замкнулся, т.е. программа должна закончить работу и сообщить об ошибке.

Теперь осталось определить структуру, в которую мы будем заносить информацию о том, с какими разведчиками связан данный разведчик. В качестве такой структуры можно взять опять же массив. Номер строки – это конкретный разведчик, единица в положении n, означает, что данный разведчик прямо или через других получает информацию от этого разведчика. После того, как такой массив заполнен остается найти минимальное количество строк (нелегалов) которые содержат ссылки от других строк. Если в строке единиц нет (кроме диагональной – разведчик сам для себя является источником), и нет ссылки на эту строку, то тогда эта строка также включается в результирующий набор строк. Принцип анализа данной структуры прост: 1. Находим строку с минимальным количеством нулей; 2. Результатом будет номер строки и номера нулей в этой строке.

Изложенный словесно алгоритм представлен программой на языке С, в листинге 2.

Листинг 2. Программа – решение задачи «Резидентура»

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void rec(int);
int prov();

int res[11][11],res1[11][11];
int l=0,i=0,j,n;
//основная функция
int main(){
//количество элементов в массиве
 scanf("%d",&n);
 if(n>10)return 1;
//заполнение массивов
 for(i=0; i<n; i++){
          for(j=0; j<n; j++){
           if(i!=j)
                    res1[i][j]=0;
           else
                    res1[i][j]=1;
           scanf("%d",&res[i][j]);
          }
 }
//по всем строкам
```

2016, Том 4, номер 6
ISSN 2309-4265
(499) 755 50 99
http://mir-nauki.com

```
 for(i=0; i<n; i++){
        rec(i);
 }
//анализируем и выводим результаты
 j=prov();
//первый агент
 printf("%d ",j);
//остальные
 for(i=0; i<n; i++){
        if(!res1[j][i])printf("%d ",i);
 }
 printf("\n");
 return 0;
}
//вспомогательная функция проверки результата
int prov(){
 int s=0,s1,k,l;
//найдем макисмальное количество связей у одного агента
 for(k=0; k<n; k++){
        s1=0;
        for(l=0; l<n; l++){
         s1=s1+res1[k][l];
        }
        if(s1>s)s=s1;
 }
//найдем агента (любого) с максимальным количеством связей
 for(k=0; k<n; k++){
        s1=0;
        for(l=0; l<n; l++){
         s1=s1+res1[k][l];
        }
        if(s==s1)break;
 }
 return k;
}
//рекурсивные функции
void rec(int ii){
```

```
 int k,t;
//проверка на зацикленность
 for(k=0; k<n; k++){
        if(res[ii][k]==2){
         printf("Error\n");
         exit(1);
        }
 }
//цикл по строке массива с рекурсивным вызовом
 for(t=0; t<n; t++){
        if(res[ii][t]==1){
         res[ii][t]=2; //отметили точку перехода
         res1[i][t]=1;
         rec(t);
         res[ii][t]=1; //вернули в исходное
        }
 }
 return;
}
```

Прокомментируем теперь текст программы из листинга 2. Прежде все обратимся к структурам данных. Массив res содержит исходные дынные о нелегальных разведчиках и получаемой ими информации. Он заполняется на основе входного потока. Параллельно в этом же цикле инициируется массив res1, который будет содержать результирующую информацию анализа потоков данных. Диагональ массива мы сразу заполняем единицами.

После ввода данных в цикле вызывается рекурсивная функция rec, единственным параметром которой является номер нелегала. Один такой вызов проверяет все потоки данных, связанных данным нелегалом. Обратим внимание на следующий фрагмент

```
 for(k=0; k<n; k++){
        if(res[ii][k]==2){
         printf("Error\n");
         exit(1);
        }
 }
```

Именно в нем проверяется не оказались ли мы в том месте, где уже были (петля, см. рисунок 3). Если это так, то выдается сообщение об ошибке и осуществляется выход из программы. Мы не анализируем, есть ли еще зацикливания, нам достаточно одного, чтобы закончить работу программы. Второй цикл в рекурсивной процедуре проходит по текущей строке в поисках единицы. В случае, если единица найдена, то снова осуществляется вызов функции rec.

Отметим также роль функции prov. В ней осуществляется поиск строки с максимальным количеством единиц. Нам при этом нужна только одна из таких строк (если их несколько). Результирующие данные будут состоять из номера этой строки и номеров всех нулевых ее элементов.

Заключение

Мы рассмотрели примеры олимпиадных задач по программированию и их решения на основе рекурсивных алгоритмов. Особое внимание было уделено представлению предметной области задач с помощью графов. Такое представление не дает готового решения, но позволяет быстрее найти искомый алгоритм. Если при этом студент знаком с теорией графов, то он может использовать ее выводы в решении или же воспользоваться уже готовыми алгоритмами, разработанными в рамках этой теории. Кроме того, разрабатываемые в рамках решения задачи алгоритмы, сами могут послужить отправной точкой для дальнейшего исследования в области теории графов и использовать на соответствующих дисциплинах математического блока.

ЛИТЕРАТУРА

1. Липский, В. Комбинаторика для программистов [Текст] / В. Липский. – М.: Мир, 1988. – 200 с.
2. Кнут, Дональд Ж. Искусство программирования [Текст]. Т. 4. Ч. 1. Комбинаторные алгоритмы / Дональд Ж. Кнут. – М.: Вильямс, 2013. – 962 с.
3. Кнут, Дональд Ж. Искусство программирования [Текст]. Т. 4. Ч. 2. Генерация всех кортежей / Дональд Ж. Кнут. – М.: Вильямс, 2008. – 162 с.
4. Олимпиады по программированию в ШГПИ: подборка олимпиад. задач с эталонными решениями в системе Solver [Электронный ресурс] // Шадринский государственный педагогический университет: офиц. сайт. – Шадринск: ШГПУ, 2007-2016. – Режим доступа: http://shgpi.edu.ru/solver/. – 19.10.16.
5. Брудно, А.Л. Московские олимпиады по программированию [Текст] / А.Л. Брудно, Л.И. Каплан. – М.: Наука, 1990. – 208 с.
6. Долинский, М.С. Решение сложных и олимпиадных задач по программированию [Текст] / М.С. Долинский. – М.: Питер, 2006. – 368 с.
7. Меньшиков, Ф. Олимпиадные задачи по программированию [Текст] / Ф. Меньшиков. – М.: Питер, 2006. – 315 с.
8. Скиена, Стивен С. Олимпиадные задачи по программированию. Руководство по подготовке к соревнованиям [Текст] / Стивен С. Скиена, Мигель А. Ревилла. – М.: Кудиц-Образ, 2005. – 416 с.
9. Брайан У. Керниган, Деннис М. Ритчи. Язык программирования С [Текст] / Брайан У. Керниган, Деннис М. Ритчи. – М.: Вильямс, 2016. – 288 с.

Мир Науки
научный интернет-журнал
2016, Том 4, номер 6
ISSN 2309-4265
(499) 755 50 99
http://mir-nauki.com

Pirogov Vladislav Jurievich
Shadrinsk pedagogical university, Russia, Shadrinsk
Vladislav-133@yandex.ru

Olympiad programming problems with recursion on graphs

Abstract. The article deals with olympiad programming problems, which may be solved with use of recursive algorithms. These algorithms are the most difficult to understand for students. The author shows that for solution of such problems the view of the subject area in a graph can be used. This approach improves visibility and helps to find right algorithm. The article deals with two problems. Both problems were proposed on one of the programming olympiads that held in Shadrinsk State Pedagogical University. It is shown that in first problem the algorithm can be described in form of cycles in Hamiltonian undirected graphs. The recursive algorithm was used for the program realization. In the analysis of the algorithm, the author draws attention to the processing of erroneous recursive calls. It also indicates that the resulting solution makes it possible to discover the criteria for when this graph is Hamiltonian. The subject area of the second problem have not a graphical representation. The article shows that the use of directed graphs makes the subject area more understandable for students to write recursive algorithms. The author examines in detail the part of the algorithm that contains conditions of the search of the situations in which the path on the graph returns to the starting point. The author also discusses the search criteria of an optimal path.

Keywords: programming; programming language; graph; olympiad problems; recursion

21PDMN616