ОЛИМПИАДА ШКОЛЬНИКОВ «ПЕРЫЕ ШАГИ В ЭНЕРГЕТИКЕ» ПО ПРЕДМЕТУ «ИНФОРМАТИКА»

Методическое пособие для школьников по подготовке к Олимпиаде «Первые шаги в энергетике»

О.П. Попкова

Пособие содержит материал, который может дать ориентиры для потенциальных участников Олимпиады школьников «Первые шаги в энергетике» по предмету информатика. На примерах продемонстрированы некоторые принципы, которыми полезно руководствоваться при решении олимпиадных задач.

Подготовлено в методической комиссии Олимпиады школьников «Первые шаги в энергетике» по предмету информатика.

Одной из важнейших задач в сфере образования является работа со школьниками, имеющими выдающиеся способности в области информатики. В связи с быстрым развитием информационных технологий, усиленным интересом школьников, олимпиады по информатике и ИКТ становятся одни из лидирующих.Терабайты информации и миллионы страниц олимпиадных задач и методов их решения публикуются не первый год в книгах и Интернете. Задания на олимпиадах чаще всего имеют простую формулировку, но не простое решение, требующее творческого мышления.

В последнее время большинство олимпиадных задач строят по такой классификации, как показано в таблице 1.

Направление задач	Обозначение
Задачи без программирования	Задачи на логику и расчеты, где решение не требует составления программного кода
Арифметические задачи	Задачи требуют знания формул и умения применять их, программный код обычно небольшой
Геометрические задачи	Задачи на проверку условий выполнения определенных свойств для заданных фигуры или тела, взаимодействия тел на плоскости и в пространстве
Задачи на динамическое программирование	Задачи, направленные на выявление рекуррентных соотношений
Задачи на сортировку и последовательность	Задачи на обработку данных, представленных в виде массивов.
Задачи на графы	Задачи со структурами данных, основанными на вершинах и ребрах
Задачи на рекурсию	Задачи на поиск с рекурсивным перебором вариантов

Некоторые задачи сложно отнести к какому-то определенному направлению, так как могут сочетать в себе сразу несколько. Рассмотрим примеры задач по каждому направлению.

Задачи без программирования

В доме у Пети установили новый лифт экспериментальной модели. В этом лифте все кнопки с номерами этажей заменены двумя кнопками. При нажатии на одну из них лифт поднимается на один этаж вверх, а при нажатии на вторую – опускается на один этаж вниз. Пете очень понравился новый лифт, и он катался на нем, пока не побывал на каждом из этажей хотя бы по одному разу.

Известна последовательность кнопок, которые нажимал Петя: 1221221221. Каково количество этажей в доме у Пети? С какого этажа Петя начал кататься на лифте? На каком этаже Петя закончил свой эксперимент? Решение: предположим, что 1 – это подняться наверх, а 2 – это спуститься вниз. Тогда, предположим, что Петя начал движение с 4 этажа, построим таблицу:

	1	2	2	1	2	2	1	2	2	1
1 эт	*									
2 эт		*		*						
3 эт			*		*		*			
4 эт						*		*		*
5 эт									*	

Ответ: 5 этажей в доме, Петя начал движение с 4 этажа и закончил движение на 2 этаже.

Арифметические задачи

Проверить, поместится ли на диске компьютера музыкальная композиция, которая длится mминут и n секунд, если свободное дисковое пространство 6 мегабайт, а для записи одной секунды звука необходимо 16 килобайт.

Решение: переведем мегабайты в килобайты, так как 1 мегабайт = 1024 килобайта, то 6 мегабайт = 6144 килобайт. Обозначим t – время, в секундах, v – объем файла, в килобайтах, тогда:

t=60*m+n;

v=16*t.

Программа на Паскале будет иметь вид:

programmuzi;

```
varm, n, t, v: integer;
begin writeln (‘Введитемин’);
readln (m, n);
t:=60*m+n;
v:=16*t;
 if v<=6144 then writeln (‘Композицияпоместится’)
elsewriteln (‘Нехватает ’, v-6144, ‘ килобайт’);
end.
```

Системы счисления

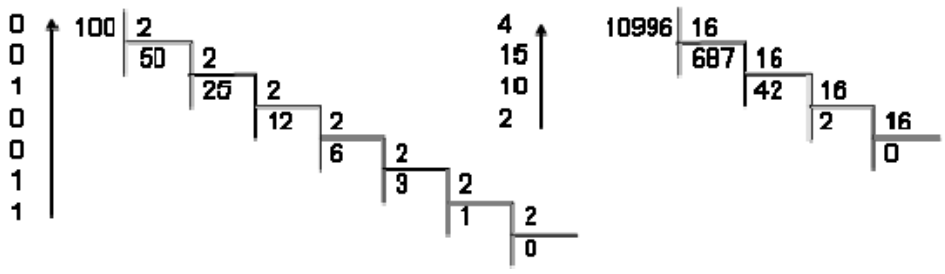
Обычно мы имеем дело с десятичной системой счисления, в которой используется ровно 10 цифр: 0..9, и мы все к этому сильно привыкли. Но для обозначения чисел возможно использование других систем, отличных от десятичной. Особенностью таких систем является, отличное от 10 количество используемых цифр. Например, в двоичной системе используется только две цифры - 0 и 1; и любое число может быть представлено в такой системе. Если система имеет порядок, больший чем 10, то в качестве следующих цифр используются латинские буквы; например, в шестнадцатеричной системе присутствуют следующие цифры: 0..9, A, B, C, D, E и F.

Для того чтобы представить какое либо число в десятичной системе счисления, следует вычислить сумму всех цифр данного числа, умноженных на порядок системы счисления, возведенной в степень, равную номеру разряда данной цифры. Следующие примеры демонстрируют суть данного метода:

$$(1100100)_2 = 1\cdot 2^6+1\cdot 2^5+0\cdot 2^4+0\cdot 2^3+1\cdot 2^2+0\cdot 2^1+0\cdot 2^0 = (100)_{10}$$
$$\underset{6\,5\,4\,3\,2\,1\,0}{}$$

$$(2AF4)_{16} = 2\cdot 16^3+10\cdot 16^2+15\cdot 16^1+4\cdot 16^0 = (10996)_{10}$$
$$\underset{3\,2\,1\,0}{}$$

Для перевода из десятичной системы счисления в систему с основанием k необходимо производить деление исходного числа на k, запоминая остатки от деления и продолжая данную операцию с целочисленным значением деления до тех пор, пока результатом деления не будет ноль. Искомым представлением числа будет запись, составленная из полученных остатков от деления, записанных в обратном порядке. Ниже представлены примеры перевода:



При решении олимпиадных задач обычно рассматривают системы счисления с основаниями от 2 до 36. Это связано с тем, что в латинском алфавите 26 букв (10 цифр + 26 букв = 36 символов). Для хранения цифр числа можно использовать как целочисленный массив, так и строку. Пусть S – число, записанное в K-ой системе счисления, а X – число в десятичной системе. Тогда алгоритм перевода из любой системы в десятичную можно записать следующим образом:

```
read(S);
X=0;
 for i = 1..len(S){ X = X*K+S[i]; }
write(X);
```

Алгоритм перевода из десятичной системы числа X в систему с основанием K и записи его в S может быть представлен так:

```
read(X);
 l=0;
do{ l=l+1;
 S[l] = X mod K;
X = X div K; }while(X>0);
write(reverse(S));
```

Для того чтобы перевести из k-ой системы счисления в систему с основанием m, можно воспользоваться «принципом чайника» и перевести сначала из k-ой системы в 10-ую, а затем из 10-ой в m-ую. Однако иногда этой двойной операции можно избежать. Речь идет о случае, когда мы имеем дело с кратными системами счисления. Например, несложно переводить из 2-ой в 4-ую, 8-ую, 16-ную и обратно; аналогично можно сказать о 3-ой и 27-ой системах или о 5-ой и 25-ой. В подобных случаях каждая цифра числа в системе с большим основанием представляется в виде нескольких цифр системы с меньшим основанием. Так, в 4-ой системе каждая цифра представляется 2-мя цифрами 2-ой системы, в 8-ой – 3мя цифрами 2-ой, а в 16-ой каждая цифра есть ничто иное как 4 цифры 2-ой системы. Например, для перевода двоичного числа 1011001010100011011 в 16-ую систему достаточно разбить число на 4-ки: 16 0010 1100 1010 1001 1011 и записать каждую из них в 16-ом формате: 2CA9B, т.к. 0010 = 2, 1100 = B, 1010 = A, 1001 = 9 и 1011 = B. Несложно догадаться, что обратное преобразование из 16-ой в 2-ую систему происходит аналогичным образом.

Геометрические задачи.

Фермер Иван с юности следит за своим газоном. Газон можно считать плоскостью, на которой в каждой точке с целыми координатами растет один

пучок травы. В одно из воскресений Иван воспользовался газонокосилкой и постриг некоторый прямоугольный участок газона. Стороны этого участка параллельны осям координат, а две противоположные вершины расположены в точках (x1, y1) и (x2, y2). Следует отметить, что пучки травы, находящиеся на границе этого прямоугольника, также были пострижены. Довольный результатом Иван купил и установил на газоне дождевальную установку. Она была размещена в точке с координатами (x3, y3) и имела радиус действия струи r. Таким образом, установка начала поливать все пучки, расстояние от которых до точки (x3, y3) не превышало r. Все было хорошо, но Ивана заинтересовал следующий вопрос: сколько пучков травы оказалось и пострижены, и полито в это воскресенье? Требуется написать программу, которая позволит дать ответ на вопрос Ивана.

lawn.in Имя входного файла: lawn.out Формат входных данных

В первой строке входного файла содержатся четыре целых числа: x1, y1, x2, y2 (-100 000 < 100 000; -100 000 < 100 000; 1 <r< 100 000).

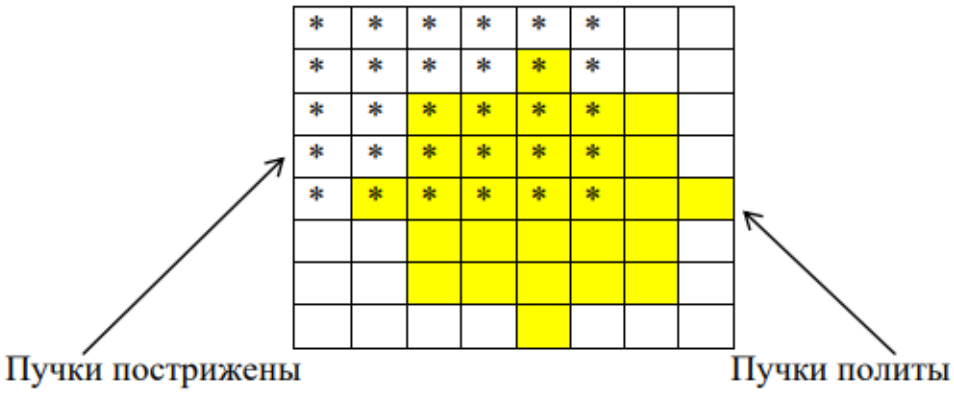
Формат выходных данных

В выходной файл необходимо вывести одно целое число – число пучков травы, которые были и пострижены, и политы.

Пример входных и выходных данных

Иллюстрация к примеру

Для каждой прямой, содержащей все пучки травы с равной координатой по оси абсцисс (аналогично можно рассматривать и ось ординат), найдем ее точки пересечения с окружностью, отображающей область действия поливальной установки, если, конечно, такие точки существуют. Получив данные точки, не сложно за время O(1) посчитать количество пучков травы на данной прямой одновременно и политых, и постриженных. Это делается с помощью операции вычисления целой части числа. Далее, просуммировав найденные числа для всех прямых, можно найти ответ на поставленную задачу.



```
programgazon;
type abc=Int64;
var x1, y1, x2, y2, x3, y3, x:logint; k, r, y:abc;
input, output:text;
functionkol (x, z1, z2:longint):longint;
var
min, max, k:longint;
begin
if (xx2) or (z1>y2) or (z2) k:=0
else
begin min:=y1;
if z1>y1 then
min:= z1;
max:=y2;
if z2 max:=z2;
k:=max-min+1
end;
kol:=k
end;
begin assign (input, ‘input.txt’);
reset (unput);
assign (output, ‘output.txt’);
rewrite (output);
read (input, x3, y3, r);
k:=kol (x3, y3-r, y3+r);
y:=r;
for x:=1 to r-1 do begin whilesqr (x) + sqr (y) >sqr (r) do y:=y-1;
 k:=k+kol (x3+x, y3-y, y3+y) + kol (x3-x, y3-y, y3+y)
end;
 k:=k+kol (x3+r, y3, y3) + kol (x3-r, y3, y3);
write (output, k);
close (input);
close (output);
end.
```

Динамическое программирование

Начисловой прямой школьник Ваня может идти вправо на одну или две единицы. Первоначально Ваня находится в точке с координатой 0. Определите количество маршрутов Вани, приводящих его в точку с координатой n. Обозначим количество маршрутов Вани, ведущих в точку с координатой n, как

F(n). Теперь вычислим функцию F(n). Заметим, что F(0)=1 (это вырожденный случай, существует ровно один маршрут из точки 0 в точку 0 – он не содержит ни одного прыжка), F(1)=1, F(2)=2. Как вычислить F(n)? В точку n школьник может попасть двумя способами – из точки n–2 при помощи прыжка длиной 2 и из точки n-1 прыжком длины 1. То есть число способов попасть в точку n равно сумме числа способов попасть в точку n-1 и n-2, что позволяет выписать рекуррентное соотношение: F(n) = F (n-2) + F (n-1) , верное для всех n> 2.

Решение на языке Pascal в виде рекурсивной функции:

```
function f (n: longint): longint;
begin
if n < 2 then
f:=1
else f:= f (n-1) + f (n-2);
end;
```

Вычисление решения этой функции, например, для n=20, оказывается, что функция работает крайне медленно. То есть одна из причин неэффективности рекурсивного решения – одно и то же значение функции вычисляется несколько раз, так как оно используется для вычисления нескольких других значений функции.

В данном случае, значения рекурсивной функции совпадает с числами Фибоначчи, так как вычисляются по тем же рекуррентным соотношениям. Для вычисления чисел Фибоначчи можно использовать цикл.

Решение:

```
F: array [0..100] of longint;
i, n: longint;
begin readln (n);
for i:= 1 to n do F [i]:=0;
F[0]:=1;
F[1]:=1;
for i:= 2 to n do
F[i]:= F [i – 1] + F [i – 2];
writeln (F [n]);
end.
```

Динамическое программирование использует те же рекуррентные соотношения, что и рекурсивное решение, но в отличии от рекурсии в динамическом программировании значения вычисляются в цикле и сохраняются в списке.

Сортировка и последовательность.

Симметричной называют последовательность чисел, которая одинаково читается как слева направо и наоборот. Например, следующие последовательности являются симметричными: 11222211, 987656789 Вашей программе будет дана последовательность чисел. Требуется определить, какое минимальное количество и каких чисел надо приписать в конец этой последовательности, чтобы она стала симметричной.

Входные данные: в первой строке содержится число N – количество элементов исходной последовательности. Во второй строке записано N чисел, разделенных пробелом – элементы этой последовательности. 1 < N < 100, элементы последовательности – натуральные числа от 1 до 9.

Выходные данные: в первой строке число M – минимальное количество элементов, которое надо дописать к последовательности, во второй строке M чисел, которые надо дописать к последовательности.

Решение:

```
programsum;
vara, b: array [1..300] of integer;
i, j, n, k: integer;
functioncheck: boolean; //проверяет последовательность на симметричность
var i: integer;
begin
result:= false;
for i:= 1 to n+k do
 if b[i] <> b [n+k-i+1] then exit;
result:= true;
end;
 begin assign (input, ‘input.txt’);
reset (input);
assign (output, ‘output.txt’);
rewrite (output);
read (n); //считываеммассив
for i:= 1 to n do read (a[i]);
fork:= 0 tondo //перебираем всевозможные добавления
begin fullchar (b, sizeof (b),0); //очищаеммассивb
b:= a; // копируем массив ав массив b
for i:= n+1 to n+k do //добавляеммассивbкчислу
b[i]:= a{n+k-i+1];
ifcheckthen// проверяем массив на симметричность
begin writeln (k); // если симметричная последовательность получена, то
```

```
fori:= n+1 ton+kdo //выводим количество добавленных чисел и сами числа
write (b[i],’’);
close (input);
close (output);
halt (0); //прерываем работу программы
end;
end;
close (input);
close (output);
end.
```

Графы

Дано целое число A. Вставить между некоторыми цифрами 1, 2, 3, 4, 5, 6, 7, 8, 9, записанными именно в таком порядке, знаки «+» и «-» так, чтобы значением получившегося выражения было число n. Например, если A=122, то подойдет выражение: 12+34-5-6+78+9.

Решение:

```
programzi;
var t: array [2..9] of string;
procedurezi_fra (s, r: longint; o:char; n:integer);
varnewr: longint;
begin if o = ‘+’ then newr:= r+s else newr:= r-s;
if n <= 9 then begin t[n]:= ‘+’;
zi_fra (n, newr, ‘+’, n+1);
 t[n]:= ‘-’; zi_fra (n, newr, ‘-’, n+1);
t[n]:= ‘ ’;
zi_fra (s*10+n, r, o, n+1);
end else// проверкаравенства
ifnewr = A then
begin
for i:= 2 to 9 do write (i-1, t[i]);
writeln (‘9’);
end;
end;
begin
readln (A);
zi_fra (1, 0, ‘+’, 2);
end
```

Процесс решения большинства задач можно разделить на следующие этапы:

1. Разобрать условие задачи;
2. Абстрагировать условие задачи;
3. Разработать алгоритм решения;
4. Реализовать программный алгоритм;
5. Провести апробацию программы.

Между ближайшими друг к другу этапами провести четкие границы довольно проблематично, а иногда и вовсе невозможно. Поэтому чаще всего их можно объединить на каком-то этапе, заменить или вовсе пропустить, поскольку процесс решения задач является творческим процессом. Но выделить этапы стоит даже только потому, как это помогает увидеть полную картину и начать последовательно действовать при решении задач.

Первый этап является самым важным, ведь при разборе условия происходит определение, как будет решаться задача. Для правильного понимания условия и решения задач необходимо внимательно их читать, иначе можно начать решать совершенно другую задачу. При прочтении необходимо обратить внимание на каждое предложение по отдельности, не редкостью бывает, что в них кроется нужная и важная информация. На этом этапе участник определяет, с какой задачи он начнет, но может произойти и так, что задача, которая кажется легкой, на первый взгляд, в решении, на самом деле будет самой сложной, и лучше будет потратить время на понимание и решение другой задачи, более простой, оставляя больше времени на остальные. Даже если задача, кажется, аналогична той, которую участник уже решал, но на самом деле решение ее лежит совершенно в другом ключе.

На этапе абстрагирования условия задачи необходимо перейти от словесного описания текста к выбору схемы решения. Может быть и такое, что при представлении условия задачи, будут выявлены не состыковки при чтении, и тогда необходимо будет возвратиться на чтение условия. Не один раз лучше представить решение задачи и по возможности упростить. Для начала лучше решить задачу ручным способом, без использования компьютера, используя по максимуму входных и выходных данных, рассматривая решение с самых простых случаев. Чаще всего существует не один подход к решению одной и той же задачи. И сложность заключается в том, чтобы увидеть эти варианты подходов и выбрать наиболее подходящий для этой задачи здесь и сейчас и реализовать его. Всвязи с множеством вариантов решений можно наткнуться на неверно выбранный путь, и чаще всего увидеть это можно на последующих этапах, что приводит на возврат к этому этапу.

Разработка алгоритма решений – это поиск эффективного алгоритма решения и пути его реализации, на данном этапе получаем ответ на вопрос «как это делать?». Творческий подход к решению задач на этом этапе играет главную роль, как и знание теоретических основ информатики, и опыт решения олимпиадных задач. У каждого исполнителя свой уникальный алгоритм решения для определенной задачи, который он сам придумал или переработал и усовершенствовал. Но существуют общепринятые рекомендации. Как правило, этап начинается с определения ведущей идеи выбранного алгоритма. Для упрощения задачи можно переключиться с ограничивающих обстоятельств (время, компьютерная техника и ее свойства) и постараться найти один из возможно-верных путей решений. В таком случае каждая из возникших идей требует предварительной проработки ручным способом и доведении ее до конкретного результата. Обязательным перед отправкой задания на проверку жюри является использование из условия задачи примеров входных и выходных файлов или тестирование программы с помощью своих тестовых данных. При возникновении трудностей на этапе разработки приемлемой идеи решения задачи, необходимо попробовать разложить ее на более простые задачи. Данный порядок следует последовательно применить к каждой из задач по отдельности. Более того, вместе с понимание хода решения задач можно разработать структуру алгоритма, и наполнять ее соответствующим содержанием. Только при наличии предварительно проработанной идеи можно приступить к описанию структуры алгоритма, выделяя при этом основные компоненты, выстраивая взаимосвязь между ними. В этом случае могут пригодиться домашние заготовки алгоритмов, используя их проще получить окончательное решение задачи. Во время подготовки к олимпиадеучастникунеобходимо хорошо освоить определенный набор типовых алгоритмов и умело в дальнейшем их применить. Так же решения задач могут основываться на математических идеях, которые необходимо придумать, в этом случае элементами могут быть типичные процедуры и функции (сортировка, перебор, динамическое программирование и др.). Исключить нельзя и тот случай, когда было быстро придумано решение, но реализовывалось которое долго и упорно, тогда как существует более эффективный и простой путь для решения данной задачи, но чтобы увидеть его, необходимо чуть больше подумать.

На протяжении всего этапа желательно все фрагменты решения записывать на бумагу, при написании мыслей происходит фиксация важных фактов и завершается их упорядочивание, в дальнейшем это помогает избежать ошибок в процессе реализации решения задач. Начальные значения переменных, процедуры это те факты, которые необходимо записать. При прочтении таких записей будет проще обнаружить, все ли данные учтены,

правильно ли определены переменные. В записях быстрее получится найти нужное место при отладке программы и позволит избежать нелепых ошибок, допустим, таких как забывание присвоения начальному значению переменной.

Данный этап для самого участника должен заканчиваться корректным получением алгоритма и оценкой его эффективности. Эти компоненты решения олимпиадных задач являются очень важными, так как именно они в большей степени определяют правильность полученного решения. Довольно 47 часто, кажется, что алгоритм интуитивно работает правильно, но на самом деле при более внимательном рассмотрении могут возникнуть проблемы во внутренней структуре алгоритма, довольно сложного алгоритма, что не так просто сразу понять в голове. Для доказательства корректности алгоритма можно разработать набор тестов, учитывающих особенности алгоритма, и провести тестирование программы на них. Как показывает практика, лучше сразу подумать над доказательством корректности и либо уметь вычислять такие оценки для используемых ими стандартных алгоритмов или знать их. Если при тестировании алгоритма оценки не удовлетворяют нашим ограничениям, то необходимо оптимизировать наше решение полностью или отдельными подзадачами. В такой ситуации, не лишнем будет остановиться и подумать, нужно ли дальше тратит время на усовершенствование программы или стоит найти более простое решение, которое позволит сохранить больше времени. Ведь нередко бывает так, что участник долго разрабатывает точное решение для поставленной задачи, но безуспешно, только потому, что его в принципе не существует.

Литература

1. Алексеев, А.В. Задачи олимпиад по математике и информатике. Практикум для школьников и студентов. – Ханты-Мансийск: ЮГУ, 2005
2. Глинка, Н.В. Школьные олимпиады. Информатика. 8-11 классы. – М.: Айрис-пресс, 2007. – 240 с.
3. Долинский, М.С. Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач. – СПб.: Питер, 2005. – 237 с.
4. Долинский, М.С. Решение сложных и олимпиадных задач по программированию. – СПб.: Питер, 2006. – 336 с.
5. Кирюхин, В.М., Окулов, С.М. Методика решения задач по информатике. Международные олимпиады. – М.: БИНОМ. Лаборатория знаний, 2007. – 600 с
6. Окулов, С.М. Программирование в алгоритмах. – М.: БИНОМ. Лаборатория знаний, 2002. – 341 с.
7. Меньшиков, Ф.В. Олимпиадные задачи по программированию. – СПб.: Питер, 2006. – 315 с.
8. Андреева, Е.В., Егоров, Ю.Е. Вычислительная геометрия на плоскости. // Информатика, № 39, 40, 43, 44 за 2002 г.