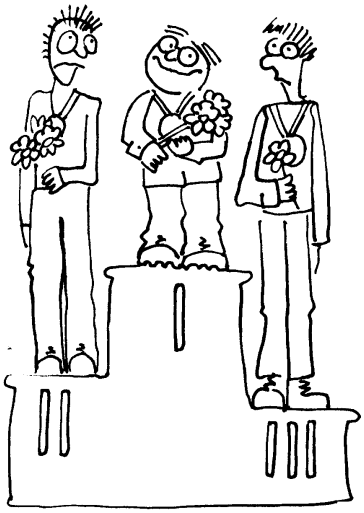
Черкасова Полина Геннадиевна

КРУЖКИ И ОЛИМПИАДЫ ПО ПРОГРАММИРОВАНИЮ

С незапамятных времен (с 1986 года) в городе проводятся олимпиады по программированию. Они имеют одну особенность - в них не существует возрастного деления. То есть в общем зачете соревнуются школьники всех возрастов. На первый взгляд, конечно, несправедливо, но это только на первый взгляд.

Дело в том, что в разных школах программирование преподают по-разному. Где-то начинают учить общению с компьютером с пятого класса, где-то с первого, а где-то с девятого. Информатике выделено различное число часов на усмотрение администрации школы (это зависит в первую очередь от возможности обеспечить учеников компьютерами). При этом кого-то учат программировать, а кого-то просто знакомят с операционной системой. Используются различные языки программирования - Logo, Basic, C++, Pascal.



...в общем зачете соревнуются школьники всех возрастов.

Поэтому сложно говорить о каком-то возрастном делении. Каждому школьнику предоставляется шанс проявить себя в период своего обучения. Это не значит, что пятикласснику на олимпиаде делать нечего. Если ученик пришел на олимпиаду, решил или попытался решить задачи и услышал их разбор, то на следующий год у него будет на порядок больше опыта и, следовательно, шансов пройти в следующий тур.

Существует масса кружков по математике, где учат решать математические олимпиадные задачи. И существует гораздо меньше подобных кружков по программированию.

Я несколько лет вела такой кружок в школе №470 и могу поделиться некоторым опытом.

Первый вопрос, который возникает - это с какими классами можно заниматься. Если дети еще недостаточно владеют языком программирования, это не значит, что организовывать такой кружок еще рано. В этом случае можно давать некоторые разделы математики, которые в программу не входят, или входят очень мало, но которые необходимы при решении олимпиадных задач по программированию. Например, очень желательно, чтобы ученики умели свободно обращаться с различными системами счисления и владели комбинаторикой. Их необходимо познакомить с графами, а также вспомнить, что такое делимость и сравнимость по модулю (при этом можно на пальцах объяснить, что такое алгебра вычетов) и т. д.

Разумеется, для участия в олимпиадах все эти вещи знать не обязательно, все это можно понимать интуитивно и каждый раз изобретать заново. Но чело-

век будет чувствовать себя гораздо увереннее, если он не будет плавать в этих вопросах, и сможет в любой момент использовать перечисленные инструменты.

Если ученики в достаточной степени владеют языком Pascal, C++ или хотя бы Basic, то с ними можно заниматься непосредственно олимпиадными задачами, по ходу дела поднимая вышеуказанные вопросы.

...ученики очень любят "жевать и комкать" алгоритмы.

Для проведения занятий кружка компьютерный класс вовсе не нужен. Задача считается решенной, если предложен верный алгоритм.

Если кружок дисциплинированный, то задачки даются на дом и дома дети над ними размышляют. Но у меня никогда не было дисциплинированного кружка, поэтому я давала время для размышления на занятии. После этого выслушиваются различные варианты решений, подвергаются умеренной критике и в процессе дискуссии появляется одно или несколько верных решений. Важно, чтобы ученики поняли, почему то или иное предложенное решение не является верным. Желательно дать возможность откорректировать решение экспромтом.

Олимпиады бывают теоретические и практические.

На теоретической олимпиаде требуется изложить алгоритм решения и текст программы на каком-нибудь языке программирования (с подробными комментариями). Если алгоритм неверен или отсутствует, то задача автоматически не засчитывается даже при верном тексте программы. При верном же алгоритме допускаются мелкие ошибки и недочеты в программном коде. Поэтому важно научить детей правильно излагать алгоритм решения. Практика показала, что этому следует уделять много внимания, так как обычно ученики очень любят “жевать и комкать” алгоритмы.

На практической олимпиаде от вас требуется только написать программу. При проверке текст программы никто не читает. Ее проверяют при помощи заранее подготовленных тестов. Вашей программе предлагается несколько вариантов исходных данных. Если она выдала верные ответы, то задача засчитывается. При этом вам может просто повезти, и при неверном алгоритме ваша программа выдаст правильный результат.

Если у школы есть возможность предоставить машинное время, то следует предлагать ученикам решить некоторые задачи на компьютере. Для этого можно использовать уже разобранные на теоретическом занятии задачи, так как все равно без готового алгоритма что-либо программировать не имеет смысла.

Если человек знает язык программирования, то сложно научить его чему-либо еще. Мастерство приходит с опытом, поэтому, чем больше дети программируют, тем лучше.

Когда ученик показывает вам программу, следует посмотреть ее текст независимо от того, работает она или нет.

...без готового алгоритма что-либо программировать не имеет смысла

Во-первых, программа должна быть “читабельной”. Имеется в виду не наличие комментариев, которые, безусловно, должны присутствовать (маслом кашу не испортишь). Имеется в виду удобная структурированность и правильное разбиение алгоритма на процедуры и функции. Если вы знаете условие задачи и алгоритм решения, вы должны без проблем прочитать предъявленную программу, даже если в ней

нет комментариев.

Школьники в большинстве своем себя не любят, поэтому при программировании стараются делать как можно больше ненужных действий и возможно сильнее запутать решение. Если после такого программирования школьнику удается отладить свою программу, то такому человеку следует поставить памятник.

Каждый программист знает, что не бывает программ, которые работают сразу после написания. Всегда что-нибудь да не так. Искусство программирования состоит в том, чтобы найти, что же не так, и исправить. И чем проще и понятнее написана программа, чем удобнее она разбита на процедуры и функции, тем легче это сделать.

Чтобы решать олимпиадные задачи по программированию, важно иметь представление о том, что такое время выполнения программы. Ведь в конечном итоге любую задачу можно решить перебором. Но на олимпиадах такое решение, как правило, не засчитывается. Переборные алгоритмы работают слишком долго.

Пример: требуется перетасовать колоду, содержащую N карт.

Решение № 1: заводим массив целых чисел A длиной N. Будем складывать в него карты на случайные места. Сначала этот массив заполнен нулями. Условимся, что карты пронумерованы от 1 до N.

Возьмем карту №1 и выберем для нее место случайным образом в диапазоне от 1 до N (в языке Pascal это делается с помощью вызова функции random(k), которая возвращает случайное число от 0 до k). Предположим, нам выпало место №6. На выбранное место положим карту, то есть присвоим шестой ячейке массива значение 1 (номер карты).

...программа должна быть "читабельной".

Теперь нужно выбрать место для карты с №2. После размещения первой карты у нас осталось N-1 пустое место. Перенумеруем их от 0 до N-1 и выберем одно из них случайным образом (пусть нам выпала восьмерка). Теперь на это место нужно поместить карту №2. Но для этого нужно найти пустое место с таким номером, ведь после размещения первой карты номера пустых мест не совпадают с номерами ячеек массива:

Номера пустых мест										
1	2	3	4	5		6	7	8	9	
0	0	0	0	0	1	0	0	0	0	...
1	2	3	4	5	6	7	8	9	10	
Номера ячеек										

Поэтому придется пробежаться вдоль массива и отсчитать нужное место:

Номера пустых мест										
1	2	3	4	5		6	7		8	9
0	0	0	0	0	1	0	0	2	0	...
1	2	3	4	5	6	7	8	9	10	
Номера ячеек										

В итоге для каждой из N карт нам нужно пробежаться вдоль массива длиной N. Значит, время работы программы будет порядка N^2 шагов.

Код на языке Pascal выглядит следующим образом:

```
program Pack1;

const MaxN
=10000;  {макси-
мально возможная
длина колоды}
var A: array
[1..MaxN] of
integer;
    N: 1..MaxN;
function
FindPlace(k:
integer):
integer;
{находит свобод-
ное место с но-
мером k}
var i, m:
```

```
integer;
begin
  m:=0; {l - число уже пройден-
ных свободных мест}
  for i:= 1 to N do
        begin
      if A[i] = 0 then inc(m);
{при обнаружении свободного мес-
та увеличиваем m}
      if m = k then break;
              {как только нашли
нужное,выходим из цикла}
    end;
  FindPlace:= i; {возвращаем но-
мер найденного места}
end;          {конец описания
функции FindPlace}

var i:integer;

begin         {тело программы}
  writeln('Введите количество
карт в колоде');
  readln(N);
  for i:= 1 to N do A[i]:= 0;
              {заполняем массив
нулями}
  for i:= 1 to N do A[
FindPlace(random(i-1)+1 ) ]:= i;
             {кладем каждую кар-
ту на случайно выбранное пустое
место}
  writrln('Перетасованная коло-
да:')          {вывод результата}
  for i:= 1 to N do write(A[i]);
end.
```

Решение № 2: заводим массив целых чисел A длиной N (пусть N=10). Расположим в нем карты по порядку. Теперь вытащим случайным образом карту (предположим, у нее будет № 3) и отложим в конец колоды, то есть в последнюю ячейку массива. А карту из последней ячейки (чтобы не пропала) положим на освободившееся место:

1	2	**10**	4	5	6	7	8	9	**3**
1	2	3	4	5	6	7	8	9	10
Номера ячеек									

Теперь перед нами стоит задача перетасовать колоду длиной N-1. Мы проделываем то же самое: выбираем любую карту (например, пятую) и меняем ее местами с последней (то есть с N-1-ой):

1	2	10	4	**9**	6	7	8	**5**	3
1	2	3	4	5	6	7	8	9	10
Номера ячеек									

Теперь нам осталось перетасовать N-2 карты.

И так далее. За каждый шаг мы уменьшаем колоду на 1 карту. Следовательно, программа выполнит N шагов.

Код программы выглядит следующим образом:

```
program Pack2;

const MaxN =10000;   {максимально возможная длина колоды}
var A: array [1..MaxN] of
integer;
   N: 1..MaxN;

procedure DrawCard(k:
integer);    {вытаскивает случайную карту из колоды
длиной N меняет ее местами с
последней}
var Help: integer;  {вспомогательная переменная}
    m: integer;{номер случайной ячейки}
begin
  m:= random(k-1)+1; {вычисля
ем номер случайной ячейки}
  Help:= A[m]; {меняем местми}
  A[m]:= A[k]; {выбранную карту}
  A[k]:= Help; {и последнюю}
end;         {конец описания
функции DrawCard}

var i:integer;

begin         {тело программы}
  writeln('Введите количество
карт в колоде');
```

```
  readln(N);
  for i:= 1 to N do A[i]:= i;
            {выстраиваем карты по порядку}
  for i:= N downto 1 do
DrawCard(i);
        {случайным образом вытаскиваем из колоды N карт}
  writrln('Перетасованная колода:')     {вывод результата}
  for i:= 1 to N do
write(A[i]);
end.
```

Понятно, что второе решение более оптимально, так как оно работает за меньшее число шагов.

При подсчете времени работы программы следует смотреть только порядок. Как правило, невелика разница - работает программа за N шагов или за 3N шагов. Но программа, выполняющая 1000N шагов, работает значительно оптимальнее, чем программа, выполняющая $N^2/1000$ шагов.

Время работы алгоритма играет большую роль в оценивании работ на теоретических турах олимпиад по программированию.

На практических же турах среди тестов, проверяющих ваше решение, существует специальный тест для проверки быстродействия алгоритма. Если ваша программа работает недостаточно быстро, она этот тест просто не пройдет.

На практических турах в условии задачи указывается максимальное время работы вашей программы. Как правило, оно берется “с потолка” и очень приблизительно (от 1 секунды до 1 минуты). На самом деле программа должна работать “мгновенно”. Если приходится ждать, значит, программа тест не прошла.

Искусство программирования состоит в том, чтобы найти, что же не так...

Есть еще один важный фактор в разработке алгоритмов - это ограниченность машинной памяти. Вы должны следить за тем, чтобы данные, которые вы храните, помещались в памяти компьютера. Так как все задачи предлагаются для решения в операционной системе DOS, то объем данных не должен превышать 64K.

Пример: вводится длинное целое число A справа налево (т.е. сначала младшие разряды, а затем старшие) до 1 000 000 000 знаков. В конце стоит точка. Выяснить, делится ли A на 7.

Решение: если бы число не было таким большим, мы бы просто взяли его и поделили (как говаривал товарищ Шариков). И посмотрели бы - делится или нет. Вся загвоздка в том, что мы не можем хранить его в памяти ни как число, ни как строку цифр, ни как массив цифр - не поместится.

Рассмотрим десятичное представление числа:

$$\overline{abcd} = a*1000 + b*100 + c*10 + d*1$$

Попробуем найти остаток от деления этого числа на 7. Как известно, при сложении, вычитании и умножении мы можем заменять числа их остатками от деления на 7 (впрочем, как и на любое другое число). После такой замены остаток от деления результата на 7 не изменится.

$$a*1000 + b*100 + c*10 + d*1 \equiv$$
$$\equiv a*6 + b*2 + c*3 + d*1 \pmod 7$$[1]

Следовательно, если мы подставим вместо степени десятки ее остаток от деления на 7, то, вычислив значение такого выражения для нашего числа, мы узнаем его остаток от деления на 7.

Остается вычислить остатки всех нужных степе-

ней десяток. Здесь мы опять не сможем просто взять и поделить, потому что миллионная степень десятки у нас никуда не поместится. Зато, зная предыдущую степень, мы можем вычислить следующую. Предположим, k-ая степень десятки имеет остаток 3 от деления на 7. Тогда k+1–ая степень будет иметь такой же остаток, как у числа 30, то есть 2. Соответственно, следующая будет иметь остаток, как у числа 20, то есть 6. И так далее... Скажу по секрету, что остатки степеней десяток от деления на семь зацикливаются:

Число									
10^0	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
1	3	2	6	4	5	1	3	2	6
Остаток от деления на 7									

Алгоритм: считаем остаток r от деления A на 7. Сначала он равен 0. Двигаемся по числу A справа налево. Для каждой цифры вычисляем остаток соответствующей степени десятки от деления на 7. Умножаем его на текущую цифру числа и прибавляем к r. Затем вместо r берем его остаток от деления на 7 (чтобы не таскать за собой больших чисел) и переходим к следующей цифре.

Быстродействие алгоритма будет равняться N, где N - это количество знаков в числе A. Иначе говоря, программа работает за линейное время.

Код на языке Pascal:

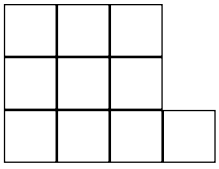
```
program Divisibility;

var r: integer; {остаток от
деления числа A на семь}
    p: integer; {остаток от
деления текущей степени десят-
ки на семь}
    ch: char;   {очередная
цифра в виде символа}
    n : 0..9;   {очередная
цифра в виде числа}

begin          {тело программы}
  r:=0;        {сначала остаток
равен нулю}
  p:=1;        {остаток от деле-
ния 10^0 на 7}
  writeln('Введите число');
  read(ch);  {читаем первую
справа цифру}
  repeat
    n:= ord(ch) - ord(0)
            {преобразуем сим-
вол в число}
    r:= (r + p*n) mod 7;
            {прибавляем к ос-
татку r цифру, умноженную на
остаток степени десятки}
    p:=p*10 mod 7;
            {вычисляем оста-
ток следующей степени десятки}
    read(ch);
            {считываем следу-
ющую цифру}
  until ch <> '.';

  writeln;
  if r = 0 then writeln('Число
делится на 7')
           else writeln('Число
не делится на 7');
end.
```

Откуда берутся задачи? Во-первых, просто берется некая жизненная ситуация и придумывается ее модель. Вот несколько задач такого плана:

1. Имеется некоторое количество (от 1 до 100) монет заданного достоинства. Требуется набрать определенную сумму или определить, что это невозможно.

2. Требуется сосчитать количество «счастливых» билетиков с номерами из 2n (n<10) цифр.

3. Выиграть в крестики-нолики на таком поле:



4. Игра в крестики-нолики на бесконечном поле остановлена после N (1<N<1000) ходов крестиков и ответных ходов ноликов. Определить, могут ли крестики выиграть

следующим ходом, поставив пятый крестик по диагонали, вертикали или горизонтали.

5. В картинной галерее работают сторожа. Для каждого сторожа известно время прихода на работу и время ухода. Определить, всегда ли галерея охраняется, и, если да, то можно ли уволить кого-нибудь из сторожей так, чтобы сторожей осталось как можно меньше, но при этом галерея бы все время охранялась.

6. Разбить абзац на строчки длины N, при этом добавив недостающие пробелы.

И так далее.

Еще один способ придумывать задачи: взять известный классический алгоритм и придумать задачу на его использование, при этом вставив некоторую изюминку. Например:

Дана карта островов: на клетчатой плоскости N клеток помечены единицами, остальные - нулями. Две клетки являются соседними, если они имеют общую границу - единичный отрезок координатной сетки. Остров - это набор соседних единичек. Клетки острова, имеющие менее четырех соседей, являются граничными. Найти остров с самой длинной границей и вывести ее длину.

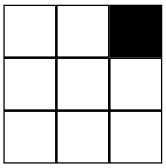
Это задача на применение классического алгоритма поиска компонент связности в графе. Изюминка состоит в том, чтобы найти длину границы острова.

Если ваша программа работает недостаточно быстро...

При придумывании задач можно использовать огромные запасы математических олимпиад. Обобщение математической задачи дает задачу по информатике.

Вот, например, математическая задача: в квадрате 3x3 можно перекрашивать строки и столбцы. Перекраска означает замену всех черных клеток белыми, и наоборот. Требуется перекрасить в белый цвет белый квадрат с черной клеткой в углу или доказать, что это невозможно.

Из нее получается замечательная задача по программированию: вводится квадрат NxN из белых и черных клеток. Требуется перекрасить его в белый цвет (то есть вывести последовательность строк и столбцов, которые следует перекрасить) либо сообщить, что это невозможно.



Еще один пример: всем известна задача о нахождении количества нулей в конце числа 100!. Если ее обобщить чуть-чуть, то получим задачу о нахождении количества нулей в конце числа N!. А если обобщить побольше, то получим задачу, требующую определить, делится ли число N! на K.

[1] Если два числа имеют одинаковые остатки от деления на третье число, то пишут A≡B (mod k) и говорят, что A сравнимо с B по модулю k.

Черкасова Полина Геннадиевна, в 1994 году окончила физико-математическую школу №470, в настоящее время - студентка IV курса СПбИТМО.

НАШИ АВТОРЫ