Всероссийская олимпиада школьников по информатике 2019-2020 учебного года.
Муниципальный этап, Амурская область,

Автор разбора – В.М.Гуровиц

Задача A. Симметричная последовательность

Начнем с функции, которая будет проверять, является ли последовательность a[1], a[2], …, a[k] палиндромной. Для этого нужно сравнить числа a[1] и a[k], a[2] и a[k-1], и т. д. до a[k div 2] и a[k+1-(k div 2)].

a[1]	a[2]	…	a[k div 2]	a[k+1-(k div 2)]	…	a[k-1]	a[k]

Если в какой-то из пар числа окажутся неравными, то последовательность палиндромной не является. Напишем соответствующий код:

```
function palindrom (k : integer) : boolean;
var
  i : integer;
begin
palindrom:=TRUE;
  for i:=1 to k div 2 do
    if a[i]<>a[k+1-i] then begin
      palindrom := false;
      break;
    end;
end;
```

Теперь вернемся к исходной задаче. Заметим, что в худшем случае нам придется дописать N-1 чисел. Например, если дана последовательность 1 2 3 … N-1 N, то кратчайшая палиндромная последовательность будет выглядеть так: 1 2 3 … N-1 N N-1 … 3 2 1. В самом лучшем случае наша последовательность уже является палиндромной, и дописывать ничего не нужно. Таким образом, ответом в задаче является число от 0 до N-1.

Опишем один из алгоритмов, решающих нашу задачу. Для начала проверим, является ли данная нам последовательность палиндромной, используя вызов функции `palindrom(N)`. Если нет, то проверим, достаточно ли добавить одно число. Ясно, что для того, чтобы получилась палиндромная последовательность, необходимо в конец дописать число a[1] :

```
a[N+1]:=a[1];
```

После этого опять проверим, получился ли палиндром, вызывая функцию `palindrom(N+1)`. Если нас вновь не постиг успех, допишем в конец исходной последовательности два числа:

```
a[N+2]:=a[1]; a[N+1]:=a[2]
```

И т. д. до тех пор, пока функция palindrom на очередном шаге не вернет результат `TRUE`.

Покажем, как организовать этот процесс, используя цикл `for`:

```
 [1] for i:=0 to N-1 do begin
 [2]   for j:=1 to i do
 [3]     a[N+j]:=a[i+1-j];  {Добавляем в конец i чисел}
 [4]   if palindrom(N+i) then begin
 [5]     writeln(i); {Печатаем количество добавленных чисел}
 [6]     for j:=1 to i do
 [7]       write(a[N+j],' ');  {Печатаем добавленные числа}
 [8]   exit;
 [9]   end;
[10] end;
```

Обратите внимание, что мы принудительно завершаем работу программы оператором `exit`, если при очередной итерации цикла получается палиндромная последовательность. Можно было обойтись без этого, используя цикл `while`.

Теперь попробуем несколько модифицировать приведенный алгоритм. Дело в том, что мы выполняем лишние операции, дописывая в конец те же числа, что стоят в начале последовательности и затем сравнивая эти числа в функции `palindrom`. Ясно, что достаточно проверять лишь числа, которые входят в исходную последовательность.

Для этого в цикле внутри функции `palindrom` нужно начинать проверять числа не с `a[1]`, а с числа, которое сравнивается с числом `a[N]`, то есть с числа `a[k+1-N]`:

```
  for i:=k+1-N to k div 2 do ...
```

Тогда и в основной программе нет нужды дополнять массив новыми числами: строки [2] и [3] можно удалить, а строку [7] заменить такой: `write(a[i+1-j],' ');`

Отметим, что максимальная возможная длина палиндрома - 199. Это нужно учитывать при объявлении массива `a`.

Муниципальный этап всероссийской олимпиады школьников по информатике
в 2015/2016 учебном году (7-8 класс)

Задача B. Количество слов

В каждом слове есть первая буква. Поэтому вместо того, чтобы считать слова, можно считать первые буквы слов. Какая буква является первой буквой слова? Любая, слева от которой стоит не буква. Исключение составляет буква, слева от которой стоит символ “-“, а перед ним - буква. Осталось написать программу, которая перебирает все символы строки, и считает, сколько среди них первых букв.

```
read(s);
for i:=1 to length(s) do
  if (s[i] in ['a'..'z','A'..'Z'])  {s[i] – буква     }
   and not(s[i-1] in ['a'..'z','A'..'Z'])
                                     {s[i-1] – не буква}
   and not ((s[i-1]='-') and (s[i-2] in
                  ['a'..'z','A'..'Z']))  {s[i-1] – не дефис}
   then n := n + 1;
write(n);
```

У этой программы есть небольшой дефект. Когда мы проверяем, например, символ s[1], нам приходится обращаться к несуществующим символам s[0] и s[–1]. Чтобы избежать этого, можно изначально добавить в начало строки два символа, которые не повлияют на количество слов в строке, например, `s:='!!'+s`, и начинать искать первые буквы слов, начиная с третьего символа строки: **for** `i:=3 to length(s) do ...`

Задача C Метро

Идею решения этой задачи можно кратко описать так. Рассмотрим все линии, на которых расположена станция A. До любой станции на любой из этих линий можно добраться со станции A *без пересадок*. Пометим все эти линии числом 0. Теперь рассмотрим все непомеченные линии, у которых есть общие станции с линиям, помеченными числом 0. До любой станции на любой из этих линий можно добраться со станции A, используя *одну пересадку*, но нельзя добраться без пересадок. Пометим эти линии числом 1. Далее, аналогично, рассмотрим все непомеченные линии, у которых есть общие станции с линиями, помеченными числом 1. На любую

станцию этих линий можно попасть, используя *две пересадки*, поэтому пометим их числом 2, и т. д. до тех пор, пока мы впервые не пометим одну из линий, на которых есть станция B. Число, которым помечена эта линия, и будет являться ответом в нашей задаче. (Такой алгоритм называют *волновым*).

Теперь займемся реализацией этого алгоритма. Проще всего написать программу, используя структуру данных *множество* (`set`). Пусть `l[i]` - множество станций метро, лежащих на `i`-й линии (то есть `l` - массив множеств), `current` - множество станций, на которых можно оказаться, используя не более `step` пересадок. Построим множество `next` станций, на которых можно оказаться, используя не более (`step+1`) пересадок. Во-первых, все станции из множества `current` входят во множество `next`. Во-вторых, если на какую-то станцию можно попасть за не более чем step пересадок, и эта станция принадлежит в том числе и линии `i`, то тогда на любую станцию линии `i` можно попасть за не более чем `(step+1)` пересадок. То есть, если множество `current* l[i]` не пусто (звездочкой обозначено пересечение множеств, то есть их общая часть), все элементы множества `l[i]` нужно включить во множество next:

```
next:=next+l[i]; {объединение множеств}
```

Осталось заметить, что если метро состоит из M линий, то либо со станции A на станцию B можно попасть не более чем с (M-1)-й пересадкой, либо нельзя попасть вовсе (A и B лежат в разных *компонентах связности*).

Теперь напишем программу.

```
{Чтение данных из входного файла}
  read(N,M);
  for i:=1 to M do begin
    read(k);
    for j:=1 to k do begin
      read(t);
      Include(l[i],t);    {Добавляем станцию t }
                          {в множество l[i]    }
    end;
  end;
  read(A,B);
```

```
{Инициализация}
  emptyset:=[];  {Пустое множество}
  step:=0;
  current:=[A];

{Основной цикл}
  for step:=0 to M-1 do begin
    next:=current;
    for i:=1 to M do
      if current*l[i]<>emptyset then
        next:=next+l[i];
    if B in next then begin
      write(step); exit;
    end;
    current:=next;
  end;

{Пути от станции A до станции B нет}
  write(-1);
```

Задача D Сумма чисел

Вся задача в типе чисел
Например решение ниже на Паскале работает только в целых числах в определенном диапазоне С большими числами или дробными возникают проблемы
Пример возможного частичного или полного решения задачи

```
var
  a, b,c : Integer;
f1, f2 : text;
begin
  Assign (f1, 'input.txt');
  Assign (f2, 'output.txt');
Reset(f1);
ReadLn(f1, a, b);
c:=a+b;
Rewrite(f2);
 WriteLn(f2,c);
close(f1);close(f2);end.
```