Морзянка

Решение.

Первое задание совсем простое. Ключевое наблюдение: имеются буквы T с кодом '-' и E с кодом '.', и, пользуясь даже только этими буквами, можно раскодировать любое сообщение. Иначе говоря, любая последовательность точек и тире может быть раскодирована. Следовательно, для получения лексикографически наименьшей строки, соответствующей данному коду Морзе, мы можем действовать жадно: просто двигаться слева направо, отрезая от начала сообщения отрезок, соответствующий наименьшей возможной букве, и не заботиться о том, что в дальнейшем остающаяся часть сообщения не сможет быть раскодирована. Так что найденную букву можно даже не хранить, а сразу выводить.

Здесь, правда возникает одна проблема: какой способ хранения букв и их кодов выбрать. Причём от ответа на этот вопрос сильно зависит и решение остальных заданий нашей задачи. Я, конечно, немного забегаю в изложении вперёд, но в данной задаче этот вопрос – один из важнейших. Так что лучше бы определиться с ним поскорее.

Я пытался придумать какие-нибудь альтернативные решения задачи, включая весьма экзотичные и весьма неэффективные. И всюду оказывалось, что где-то внутри решения приходилось искать буквы, соответствующие некоторому отрезку кода Морзе (или определять, что этому отрезку не соответствует никакая буква). Но это и понятно – мы ведь морзянку переводим в обычный буквенный текст. А вот чуть дальше становится понятно, что мы весьма часто ищем буквы, соответствующие коду из одной, двух, трёх и четырёх (длиннее – повезло! – кодов Морзе у букв нет) точек-тире. Это наводит на мысль, что неплохо бы разместить коды так, чтобы было легко различать коды разной длины. А тогда с точками-тире всё становится ясно – считаем коды букв двоичными числами. Я лично считал точку нулём, а тире – единицей, можно и наоборот ☺. В итоге получилась такая конструкция:

```
Var code: Array [1..4,0..15] of Char;
...
code[1,0]:='E';  code[1,1]:='T';

code[2,0]:='I';  code[2,1]:='A';  code[2,2]:='N';  code[2,3]:='M';

code[3,0]:='S';  code[3,1]:='U';  code[3,2]:='R';  code[3,3]:='W';
code[3,4]:='D';  code[3,5]:='K';  code[3,6]:='G';  code[3,7]:='O';

code[4,0]:='H';  code[4,1]:='V';  code[4,2]:='F';  code[4,4]:='L';
code[4,6]:='P';  code[4,7]:='J';  code[4,8]:='B';  code[4,9]:='X';
code[4,10]:='C'; code[4,11]:='Y'; code[4,12]:='Z'; code[4,13]:='Q';
...
```

Первый индекс – длина кода, второй – сам код буквы, переведённый в десятичную систему из двоичной. Например, в `code[3,2]` находится буква 'R' потому, что код буквы 'R' состоит из трёх точек-тире и равен 2, если рассматривать его как двоичную запись числа, т.е. код 'R' равен 010, или `'.-.'` . Все незанятые места в массиве `code` заполним каким-нибудь особым символом. В процессе реализации удобнее оказалось сделать так, чтобы этот особый символ был больше всех букв. Немного скучно заполнять такой массив. Но что поделаешь – особых закономерностей, вроде, не видно. Остаётся только порадоваться, что букв немного, и длина кода не превосходит 4.

Ну, а оставшиеся три задания данной задачи – это стандартные задания на применение динамического программирования. О динамическом программировании в прошлогоднем Runsite говорилось много – задачи с 7 по 12 были посвящены именно динамическому программированию. Так что, если чувствуете необходимость, то посмотрите в прошлогодние материалы:

http://www.progmeistars.lv/CM/probarchive.html#RUNSITE2006_7

Особенно на решения и учебные материалы 11-й и 10-й задач. И на учебные материалы, выложенные к 7-й задаче – там их очень много.

В связи с этим я не буду подробно излагать решение и обсуждать реализацию сегодняшней задачи, только выпишу основные соображения.

Второе задание. Пусть C_k – это количество способов декодировать последовательность первых k точек-тире из заданных. Мы уже видели, что все коды букв состоят не более, чем из 4 точек-тире, и что любая последовательность из одной, двух или трёх точек-тире является кодом некоторой буквы. Имеется C_{k-1} способов декодирования строки из k точек-тире, в которых код последней буквы состоит из одной точки-тире, C_{k-2} способов декодирования строки из k точек-тире, в которых код последней буквы состоит двух точек-тире, C_{k-3} способов декодирования строки из k точек-тире, в которых код последней буквы состоит из трёх точек-тире, и, если последние четыре точки-тире образуют код некоторой буквы, то имеется ещё C_{k-4} способов декодирования, заканчивающихся этой буквой. Итого, для всех k>4

$C_k = C_{k-1} + C_{k-2} + C_{k-3} + C_{k-4}$,

если последние четыре точки-тире образуют код некоторой буквы, и

$C_k = C_{k-1} + C_{k-2} + C_{k-3}$ в противном случае.

Легко видеть, что $C_1 = 1$, $C_2 = 2$, $C_3 = 4$, а $C_4=8$, если первые 4 точки-тире образуют код некоторой буквы, и $C_4=7$, если первые 4 точки-тире не являются кодом никакой буквы.

Понятно, что, во-первых, тащить за собой гигантские числа ни к чему, и все вычисления следует вести по модулю 123456789, и, во-вторых, для вычисления ответа не нужно заводить стотысячный массив – вполне достаточно пяти переменных (ну, или пятиэлементного массива).

Во втором задании мы нашли ответ, двигаясь вдоль массива данных слева направо - от начала к концу. Могли бы двигаться и справа налево – это практически ничего не меняет. Аналогично обстоят дела и в третьем задании. Но! Есть ещё и четвёртое задание. И, если в третьем задании надо найти только длину наикратчайшего декодирования, то в четвёртом – найти некоторое декодирование кратчайшей длины. Некоторое – это, в данном случае, лексикографически наименьшее, его удобнее строить, двигаясь слева направо и выбирая при этом наименьшую возможную букву.

Очень часто встречающаяся ситуация: необходимо найти объект, удовлетворяющий некоторому экстремальному свойству, а экстремальное (минимальное или максимальное) значение этого свойства ищем с помощью динамического программирования. Обычное дело в такой ситуации – сохранить результаты промежуточных вычислений и генерировать объект постепенно, двигаясь навстречу направлению динамического программирования – от последних вычислений к начальным.

В четвёртом задании удобно двигаться слева направо, значит будем в третьем задании двигаться в противоположном направлении – справа налево, от конца заданного кода Морзе к его началу.

Итак, обозначим Len длину заданной строки точек-тире. Пусть S_k – длина кратчайшего раскодирования строки из точек-тире, начиная с k-й и до последней (Len-ой) точки-тире. Легко видеть, что

$S_k = \min (S_{k+1} , S_{k+2} , S_{k+3} , S_{k+4}) +1$

если четыре точки-тире, стоящие на k-ом, (k+1)-ом, (k+2)-ом и (k+3)-ом местах образуют код некоторой буквы, и

$S_k = \min (S_{k+1}, S_{k+2}, S_{k+3}) +1$ в противном случае.

S_1 и содержит ответ третьего задания.

Для решения четвёртого задания будем хранить массив с вычисленными величинами S_k. Для определения лексикографически наименьшего раскодирования наименьшей возможной длины двигаемся слева направо и при этом поступаем так:

для i=1, 2, 3, 4 определяем букву, соответствующую коду из i первых точек-тире (возможно, что для i=4 такой буквы и нет);

среди этих четырёх (или трёх) букв выбрасываем те, которые мешают нам получить кратчайшее раскодирование; иначе говоря, выбрасываем те буквы, для которых $S_{k+i} > S_k-1$, где i – длина кода буквы, k – номер первой из оставшихся точек-тире; при этом хотя бы одна буква останется – ведь хотя бы одно раскодирование длины S_k существует;

из оставшихся букв выбираем наименьшую, выводим её и отрезаем от начала заданной строки точек-тире код этой буквы (точнее, отрезаем от того, что осталось от заданной строки).

Легко видеть, что все предложенные алгоритмы требуют O(Len) операций, где Len – количество заданных точек-тире.