П Р И Л О Ж Е Н И Я Н А Р Е К У Р Е Н Т Н И Т Е У Р А В Н Е Н И Я

(ДОМАШНА РАБОТА ЗА СТУДЕНТИ ОТ СУ, ФМИ)

Задача 1. Разглеждаме редици от следния вид: $x_0 = 0$, $x_{n+1} = x_n / 2$ или $x_{n+1} = 1 - x_n$. Сложност на едно число A наричаме най-малкото n, за което $x_n = A$, ако изобщо съществува редица от описания вид, съдържаща числото A. Измежду числата от вида $m / 2^{50}$, където $m \in \{1;\ 3;\ 5;\ \ldots;\ 2^{50} - 1\}$, да се намери числото с най-голяма сложност. На колко е равна тази най-голяма сложност? **(50 точки)**

Задача 2. Даден е низ $S[1...n]$, съставен от n букви. Да се намери броят на всички непразни подредици на S, в които гласните и съгласните се редуват. Всяка такава подредица може да има произволна положителна дължина и може да започва както с гласна, така и със съгласна. Подредица е всяка редица, съставена от букви от S, взети в същия ред, в който се срещат в S, като не е задължително да са последователни в S.

а) Съставете възможно най-прости рекурентни уравнения за пресмятане на търсения брой на подредиците с желаното свойство. **(20 точки)**

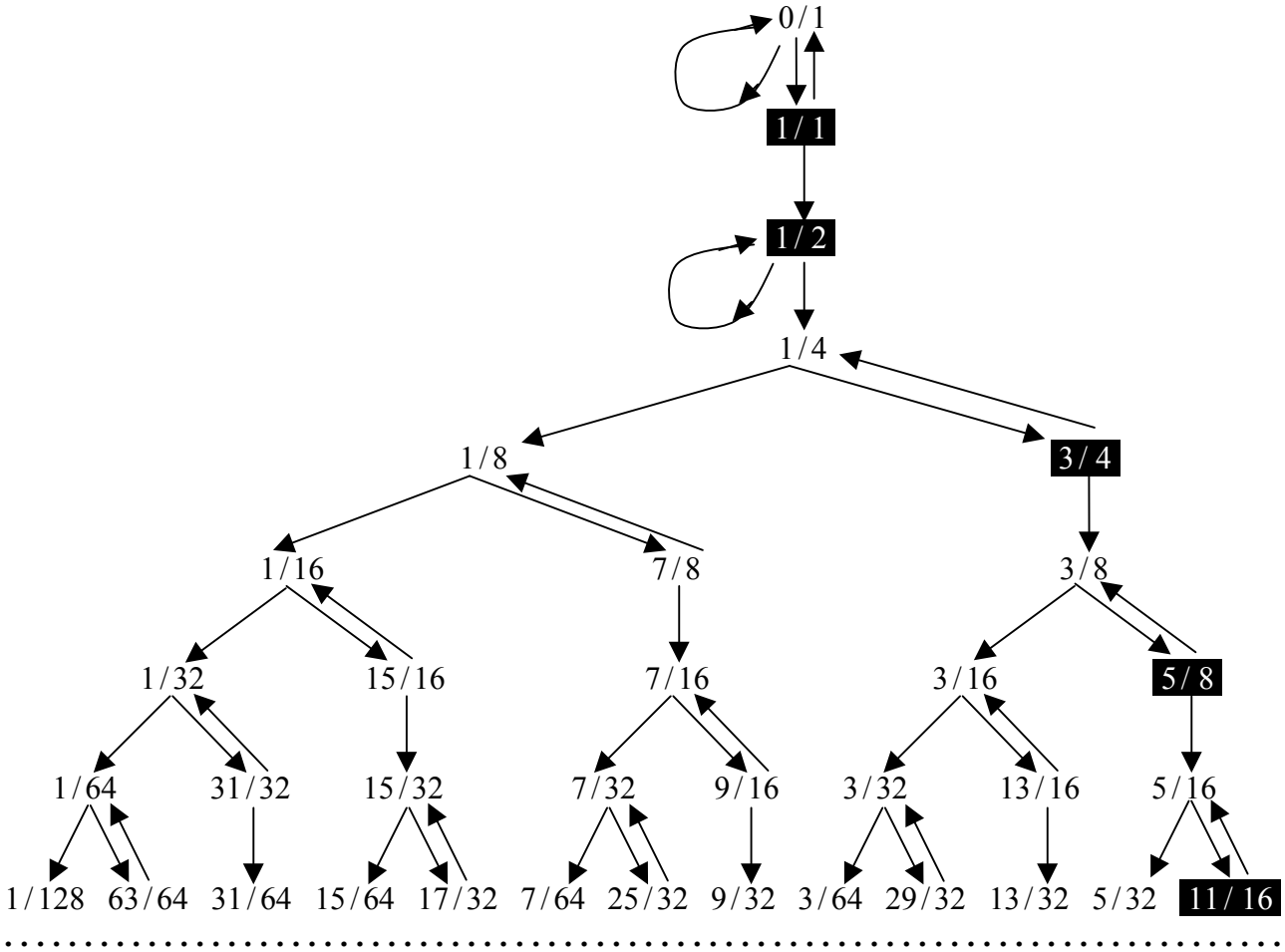
б) Превърнете уравненията в алгоритъм със сложност по време и памет O(n). Опишете алгоритъма като програма на Си. Програмата може да използва само типовете char, int, масив и указател и трябва да бъде конзолна, тоест да приема и предава данни само чрез стандартния вход-изход. Тя не може да използва никакви готови функции (библиотеки) с изключение на тези за четене и писане на стандартния вход-изход. **(10 точки)**

Забележка: Ако алгоритъмът изразходва памет O(1) и време O(n), дават се допълнителни 10 точки за висока ефективност.

в) Демонстрирайте алгоритъма, като проследите изпълнението му върху символния низ S = “книга”. **(10 точки)**

РЕШЕНИЯ

Задача 1. Числата от описания вид са рационални, знаменателите им са степени на двойката (изваждането от единица не променя знаменателя, а делението на 2 увеличава показателя на степента), всички числа са между 0 и 1 включително (доказва се с обикновена индукция по индекса на редицата). За да се ориентираме в структурата на множеството на тези числа, представяме го чрез безкраен граф: върхове са числата, а ребрата сочат от x_n към x_{n+1}. Числото $x_0 = 0$ смятаме за начален връх, от който обхождаме графа в ширина. Така графът се разделя на слоеве, номерирани с числата 0, 1, 2, 3 и т.н. Номерът на слоя, в който се среща число от разглежданото множество, е тъкмо сложността на числото. Иначе казано, сложността на едно число е дължината на най-къс път от нулата до числото. А най-къс път се намира именно чрез търсене в ширина.



Да означим с a_n числото с най-голяма сложност измежду числата от вида $m / 2^n$, където $m \in \{1;\ 3;\ 5;\ \ldots;\ 2^n - 1\}$. Числата с най-голяма сложност са изобразени на графа в негатив — бели цифри на черен фон. Например числото 5 / 8 е с най-голяма сложност измежду числата със знаменател 8, защото от всички тях е разположено най-долу, тоест в слой с най-голям номер.

С изключение на първия преход (от 1 / 1 към 1 / 2) всяко число с най-голяма сложност се получава от предишното чрез две аритметични операции — деление на 2, последвано от изваждане от 1. Тоест числата с най-голяма сложност се разполагат по десния клон на графа. Обратното не е вярно: не всяко число от десния клон на графа е с най-голяма сложност; такива са само числата от десния клон, взети през едно. Причината е в споменатите две аритметични операции: прескача се междинният резултат.

Обосновка на забелязаната зависимост: Нека a_{n+1} е числото с най-голяма сложност измежду числата от вида $m / 2^{n+1}$. Последното аритметично действие, с което е получено това число, не е деление на 2; в противен случай, като го извадим от единица, ще получим число от вида $m / 2^{n+1}$ в следващия слой, тоест с по-голяма сложност, а това противоречи на максималната сложност на a_{n+1}. И така, щом последното аритметично действие не е деление на 2, значи е изваждане от 1. Следва, че умалителят също има знаменател 2^{n+1} (и нечетен числител).

Предпоследното действие не може да е изваждане от единица, защото в такъв случай числото a_{n+1} би се срещало в по-горен слой, тоест то би се повтаряло в графа. Ето защо предпоследното действие е деление на 2 и делимото има знаменател 2^n. Следователно сложността на a_{n+1} е с две по-голяма от сложността на делимото. Поради максималността на сложността на a_{n+1} трябва делимото също да има максимална сложност измежду числата от вида $m / 2^n$, тоест делимото трябва да е числото a_n.

Получаваме линейно-рекурентното уравнение

$$a_{n+1} = 1 - a_n / 2.$$

Разсъжденията, с които то беше изведено, не важат за прехода от $1/1$ към $1/2$. По-точно, нищо не пречи последното действие да е било деление на 2, защото като извадим $1/2$ от 1, получаваме пак $1/2$, а не друго число със знаменател 2 (което би отишло в следващия слой и би имало по-голяма сложност).

Въпреки че доказателството на уравнението не важи за прехода от $1/1$ към $1/2$, все пак самото уравнение остава в сила (това се проверява непосредствено — чрез заместване).

И така, числата с най-голяма сложност удовлетворяват рекурентното уравнение

$$a_{n+1} = 1 - a_n / 2 \text{ за всяко цяло } n \geq 0.$$

Първото такова число е $1 = 1 / 2^0$, откъдето получаваме началното условие $a_0 = 1$.

Това линейно-рекурентно уравнение може да се реши чрез характеристично уравнение. Решението е $a_n = \frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^n$ за всяко цяло $n \geq 0$. Това е отговорът на задачата — числото с най-голяма сложност измежду числата от вида $m / 2^n$, $m \in \{1;\ 3;\ 5;\ \ldots;\ 2^n - 1\}$. Въпросната най-голяма сложност е $2n$ за всяко цяло $n \geq 1$, защото всяко число се получава от предишното число с помощта на две аритметични операции, тоест сложността расте с 2 на всяка стъпка. Изключение е преходът от 1 към $1/2$; числото 1 има сложност 1 (а не 0, колкото би се получило от израза $2n$, ако заместим n с 0).

След заместване на $n = 50$ в получената формула стигаме до отговора на задачата: измежду числата от вида $m / 2^{50}$, където $m \in \{1;\ 3;\ 5;\ \ldots;\ 2^{50} - 1\}$, най-голяма сложност има числото $a_{50} = \frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^{50}$. Тази най-голяма сложност е $2 \cdot 50 = 100$.

Задачата е била дадена на XLVIII Московска математическа олимпиада през 1985 г.

Задача 2. Да означим с a_k броя на непразните подредици на низа $S[1...k]$, които редуват гласни и съгласни букви и завършват на гласна, а с b_k — броя на подредиците на $S[1...k]$, които редуват гласни и съгласни букви и завършват на съгласна. Получаваме две крайни редици: $a_1, a_2, \ldots, a_n$ и $b_1, b_2, \ldots, b_n$.

Начални условия: Ако $S[1]$ е гласна, то важат следните равенства: $a_1 = 1$ и $b_1 = 0$. Обратно, ако $S[1]$ е съгласна, тогава $a_1 = 0$ и $b_1 = 1$.

Рекурентни уравнения: Нека $k > 1$. Ако $S[k]$ е гласна, то в сила са следните равенства:

$$b_k = b_{k-1} \text{ и } a_k = a_{k-1} + b_{k-1} + 1.$$

Щом $S[k]$ е гласна, то броят на подредиците, завършващи на съгласна, не се мени ($b_k = b_{k-1}$) при прехода от $k-1$ към k, защото към досегашното им множество не влизат нови редици: те би трябвало да завършват на гласната $S[k]$, а това противоречи на тяхното определение. Към редиците, завършващи на гласна (чийто досегашен брой е a_{k-1}), се добавят редиците, завършващи на съгласна (чийто досегашен брой е b_{k-1}), понеже всяка от последните редици може да бъде удължена чрез долепване на една буква в края на редицата — гласната $S[k]$. Добавяме и събираемото 1: то съответства на редицата от една буква $S[k]$; тя завършва на гласна, обаче не е броена нито в събираемото a_{k-1}, нито в събираемото b_{k-1}, защото, ако изтрием буквата $S[k]$, ще остане празна редица, а пък празните редици не се броят.

Обратно, ако $S[k]$ е съгласна, то в сила са следните равенства:

$$a_k = a_{k-1} \text{ и } b_k = a_{k-1} + b_{k-1} + 1.$$

Накратко, на всяка стъпка едната бройка остава непроменена, а в другата се записва техният сбор плюс 1. Отговорът на задачата (търсеният брой на редиците) е сборът $a_n + b_n$.

Пример: Ако S = “книга”, то получаваме следните редици:

k	1	2	3	4	5
$S[k]$	к	н	и	г	а
a_k	0	0	3	3	10
b_k	1	2	2	6	6

Числата във всеки стълб се получават по горните формули. Например $a_3 = 3$ и $b_3 = 2$, защото в низа $S[1...3]$ = “кни” има пет непразни подредици, редуващи гласни и съгласни: *ки*, *ни*, *и*; *к*, *н*. Първите три подредици завършват на гласна, а другите две — на съгласна.

Добавянето на съгласната $S[4]$ = “г” не променя броя на редиците, които завършват на гласна: това са същите три редици, затова $a_4 = a_3 = 3$. Обаче броят на редиците, които завършват на съгласна, се увеличава: $b_4 = a_3 + b_3 + 1 = 3 + 2 + 1 = 6$. В този брой влизат:
— двете стари редици, завършващи на съгласна: *к*, *н*;
— трите стари редици, завършващи на гласна, но с долепена в края буква “г”: *киг*, *ниг*, *иг*;
— редица, съставена от една буква (новата буква): *г*.

Аналогично се получават числата в последния стълб. А щом низът S = “книга” съдържа 10 редици, завършващи на гласна, и 6 редици, завършващи на съгласна, то броят на всички редици (редуващи гласни и съгласни) е равен на 10 + 6 = 16.

Трябва да пресметнем всички членове на редиците, затова времето на алгоритъма е $\Theta(n)$. Не е нужно да пазим всички членове през цялото време: достатъчни са последните два члена. Алгоритъмът, описан на Си, изглежда така:

```
#include <stdio.h>
#pragma warn -pia
#define MAX 11

int isVowel(char c) {
  return c == 'а' || c == 'ъ' || c == 'о' || c == 'у' ||
         c == 'е' || c == 'и' || c == 'ю' || c == 'я' ;
}
```

```
unsigned int ALG(char * S) {
  char current;
  unsigned int a, b, k;
  a = b = k = 0;
  while ((k < MAX) && (current = S[k])) {
    if (isVowel(current))
      a += (b + 1);
    else
      b += (a + 1);
    ++k;
  };
  return (a + b);
}

int main() {
  char S[MAX];
  printf("\n");
  printf("Given a string, this program calculates the number\n");
  printf("of subsequences that alternate between vowels and consonants.\n");
  printf("\n");
  printf("Input a string of small cyrillic letters encoded in 8-bit ASCII.\n");
  printf("Its length must not exceed %u.\n", MAX-1);
  scanf("%s", S);
  unsigned int NumSeq = ALG(S);
  printf("\n");
  printf("The number of alternating subsequences is %u.\n", NumSeq);
  printf("\n");
  return 0;
}
```

За простота на програмния код са добавени нови начални членове: $a_0 = 0$ и $b_0 = 0$, а пък е прието, че рекурентните уравнения важат и при $k = 1$. Това не променя решението.

Този начин за решаване на алгоритмични задачи чрез рекурентни уравнения се нарича ***динамично програмиране.*** Названието произлиза от задачи, в които етапите на изчисление съответстват на етапите на някакъв природен, икономически или друг процес.

СХЕМА НА ТОЧКУВАНЕ

Задача 1 носи 50 точки, разпределени, както следва:
— за намиране на рекурентна връзка, изразена неформално: 11 точки;
— за съставяне на рекурентно уравнение: 10 точки;
— за подробна обосновка на рекурентното уравнение: 10 точки;
— за избор на начално условие: 2 точки;
— за решаване на рекурентното уравнение до обща формула: 5 точки;
— за намиране на неопределените коефициенти: 5 точки;
— за заместване на n с 50 във формулата за числото с най-голяма сложност: 2 точки;
— за намиране на самата най-голяма сложност при $n = 50$ (дори без обща формула): 5 точки (отнема се 1 точка, ако има обща формула, но n не е заместено с 50).

Задача 2 носи 50 точки, разпределени, както е указано в условието на задачата.