Подготовка школьников к олимпиадам по программированию: решение задач на полный перебор

Морозов В. В.

На олимпиадах по программированию частая гостья – задача, в которой приходится из данного множества выбирать некоторое подмножество, удовлетворяющее определенным условиям. Например, из некоторой группы людей выделить подгруппу, которая наилучшим образом взаимодействует друг с другом и делает подгруппу эффективной. Или, из множества ящиков выбрать подмножество, которое наилучшим образом помещается в грузовик и имеет наибольшую ценность.

Из курса комбинаторики известно, что если множество состоит из N элементов, то в нем можно выбрать 2^N всевозможных подмножеств. И не всегда очевидно, какое подмножество дает ответ на поставленный вопрос. И тогда приходится перебирать все 2^N подмножеств, пока не найдем нужное подмножество.

Для того, чтобы осуществить перебор всех возможных подмножеств можно использовать рекурсивные процедуры. Конечно, применение рекурсивных процедур говорит о мастерстве программиста, но у рекурсии есть существенный недостаток: если рекурсия вызывает себя многократно, то возникает переполнение стека, и вместо успешного решения задачи мы видим окно, которое сообщает об этой неприятности.

В настоящей статье предлагается универсальная идея, домашняя заготовка, которая поможет организовать перебор подмножеств без рекурсии. И эту заготовку учащийся может использовать на олимпиаде по программированию в другой задаче, где нужно осуществить полный перебор.

Обратим внимание читателя также на то, что если требуется перебрать все возможные способы разбиения данного множества на 3 подмножества, то способов разбиения уже 3^N. Но и в этом случае можно применить идею полного перебора, правда, при этом придется использовать троичную систему счисления вместо двоичной, но об этом чуть позже.

В качестве примера применения этой идей перебора рассмотрим решение задачи, которая предложена на сайте «Школа программиста»[1].

Постановка задачи.

Даны N целых чисел X_1, X_2, …, X_N. Требуется расставить между ними знаки «+» и «–» так, чтобы значение получившегося выражения было равно заданному целому S.

Входные данные

Входной файл INPUT.TXT в первой строке содержит числа N и S. В следующей строке располагается N чисел, разделенных пробелом. Ограничения: $2 <= N <= 24$, $0 <= X_i <= 5*10^7$, $-10^9 <= S <= 10^9$.

Выходные данные

В выходной файл OUTPUT.TXT выведите «No solution», если такой результат получить невозможно, иначе выведите получившееся равенство. Если решение не единственное, выведите любое.

[1] (Выражения, 2007).

Примеры входных и выходных данных.

Таблица 1

№	INPUT.TXT	OUTPUT.TXT
1	3 10 15 25 30	15+25–30=10
2	2 100 10 10	No solution

Пояснения к примеру

В первом примере в первой строке мы передаем программе количество чисел – 3 и целевое значение – 10. Во второй строке даем эти три числа. Программа определяет, что существует способ расставить между данными числами знаки «+» и «–» так, что получается целевое значение 10.

Во втором примере программа получает два числа. Целевое значение 100 получить невозможно, ведь максимально возможное число, которое может получиться – это 10+10=20. Программа должна выдать сообщение «No solution».

Обсуждаем с учащимися идею решения[2].

Перебирать всевозможные подмножества лучше всего с помощью двоичного представления чисел. Перебираем все числа от 0 до 2^n, каждое число преобразуем в двоичное представление, и в данное множество индексов включим те индексы, в позиции которых записана 1 в двоичном представлении данного числа. Например, число 17 соответствует строке '10001', будем рассматривать последовательность + – – – +, проверять, выполняется ли условие a[1] – a[2] – a[3] – a[4] + a[5] = целевому значению, и если выполняется, то печатаем этот результат.

Но можно экономнее расходовать машинное время, не тратя его на преобразование числа из десятичной в двоичную систему счисления. Например, пусть n = 10. Начнём со строки '0000000000', которая соответствует последовательности десяти минусов. Напишем процедуру, которая выполняет двоичное сложение текущей двоичной строки с единицей. Если эта процедура получает на вход строку, скажем, '1111011011', то процедура будет просматривать эту строку справа налево, отыскивая первый ноль ('11**0**1111111') и пропуская все единицы, затем процедура заменит найденный ноль единицей, а все предыдущие единицы на нули, получим: '11**1**0000000', а следовательно получили новую уникальную последовательность из знаков «+» и. Будем осуществлять такой перебор до тех пор, пока не получим строку из единиц. Таким образом, мы переберём все 2^n подмножеств множества индексов.

Приступаем теперь к реализации описанной идеи.

[2] (Морозов, 2001)

Таблица 2

`uses crt;` `const nn=24;`	Объявляем константу nn – максимально возможное количество чисел.

`Type TVector=array[1..nn] of integer;`
`var x:TVector;`
`n,c,k,i:integer;`
`h:string;`
`L,LL:boolean;`
	Создаем тип для хранения n данных чисел.
	Объявляем переменную x – массив из данных чисел.
	Целые переменные: n – для хранения количества чисел, c – целевое значение, h – последовательность нулей и единиц; логические переменные L (признак завершения перебора) и LL (признак, что найдено хотя бы одно решение).

`procedure bininc(var h:string;`
`var L:boolean);`
`var i,j:integer;`
	Пишем процедуру двоичного увеличения на единицу. В качестве формальных параметров процедура получает изменяемую в процедуре переменную h – двоичное число и логическую переменную L – признак завершения перебора.
	Объявляем локальные переменные – вспомогательные целые числа i и j.

`begin`
`if h[k]='0'`
`then`
`h[k]:='1'`
`else`
`begin`
	Если последняя цифра 0, то увеличить это число на 1 очень просто – вместо нуля пишем 1.

`i:=k;`
`while`
`(h[i]='1')and(i>1) do dec(i);`
	В противном случае ищем первый ноль справа (пример: h='01010101001100101111111')

`if h[i]='1'`
`then`
`L:=False`
	После завершения этого цикла в переменной i либо номер первого нуля, либо символ ‘1’. В последнем случае перебор завершен.

`else`
`begin`
`h[i]:='1';`
`for j:=i+1 to k do`
`h[j]:='0'`
`end`
`end`
`end;`
	В первом случае вместо найденного нуля пишем 1, а все единицы справа заменяем на нули.
	Процедура завершена.

function rez(h:string):integer; var i,s:integer;	Напишем функцию, которая вычисляет значение выражения для данной комбинации знаков. В качестве формальных параметров функция получает двоичное число (строку из нулей и единиц). Функция возвращает целое число – значение соответствующего выражения. Объявляем локальные переменные: i – индекс, S – сумма данных чисел с соответствующим знаком.

begin
  s:=x[1];
  for i:=1 to k do
  if h[i]='0'
  then
  s:=s-x[i+1]
  else
  s:=s+x[i+1];
  rez:=s
end;
	Вычисляем сумму данных числе с соответствующим знаком.
	В имя функции записываем результат.

Procedure Print(h:string);
var i:integer;
begin
  write(x[1]);
  for i:=1 to k do
  begin
  if h[i]='0'
  then write('-')
  else
  write('+');
  write(x[i+1]);
  end;
  write('=',c);
  writeln;
end;
	Напишем процедуру, которая получает в качестве формального параметра двоичное число h, а в результате печатает соответствующее арифметическое выражение.

begin
  clrscr; LL:=False;
  n:=10; c:=292;
  x[1]:=12; x[2]:=25;x[3]:=36;
  x[4]:=47; x[5]:=35;x[6]:=29;
  x[7]:=38;x[8]:=16;x[9]:=159;
  x[10]:=387;
	Тело программы.
	Вводим данные простым присваиванием. Чтение данных из файла и запись данных в файл – простая процедура, и это не предмет настоящей статьи.

k:=n-1;	k – длина двоичного числа – на 1 меньше количества данных чисел.

`h:='';`
`for i:=1 to k do`
`  h:=h+'0';`
	Первое двоичное число состоит из k нулей, что соответствует последовательности из k знаков минус.

`L:=true; LL:=False;`
`if Rez(h)=c`
`  then`
`  begin`
`  Print(h);`
`  LL:=true;`
`  end;`
	Проверяем выражение только с одними минусами.

`Repeat`
`  bininc(h,L);`
`  if (Rez(h)=c) and (L)`
`  then`
`  begin`
`  Print(h);`
`  LL:=true;`
`  end`
`until not(L);`
	Открываем цикл.
	Выполняем двоичное прибавление единицы. Для новой полученной последовательности вычисляем значение соответствующего выражения. Если значение выражения равно целевому значению, то печатаем удачное выражение и поднимаем флаг – признак того, что решение найдено.
	Крутимся в цикле, пока не будет достигнут полный перебор.

`if not(LL)`
`  then`
`  writeln('No solution');`
`end.`
	Если флаг успеха опущен, то выдаем сообщение об отсутствии решения.
	Задача решена.

Наиболее продвинутым учащимся можно предложить развитее этой задачи: кроме знаков «+» и «–» добавим знак умножения. Можно подсказать, что теперь для полного перебора будем использовать троичную систему счисления. Правда, при вычислении значения выражения придется учитывать более высокий приоритет умножения.

Предметный указатель

Список таблиц

Список литературы

1. *Выражения.* (8 1 2007 г.). Получено 15 11 2013 г., из Школа программиста: http://acmp.ru/index.asp?main=task&id_task=366

2. Морозов, В. В. (18 10 2001 г.). *Городские олимпиады школьников по информатике, 2004-2006 гг. Кемеровская область*. Получено 17 11 2014 г., из Сайт школы 17 г. Полысаево: http://polysaevoschool17.narod.ru/articles/morozov2.htm