Рекурсивное и нерекурсивное решение в задаче о сочетаниях

Условие задачи. Дано множество символов, построить все сочетания из данного множества без повторений.

Мы можем немного помочь интуиции анализом условия. Там сказано, что сочетания должны быть без повторений. Это означает, что элементы, уже включенные в очередное сочетание, необходимо как-то пометить, чтобы ненароком не включить их в это же сочетание еще раз. А это уже конкретное указание на то, что делать. Что же касается пометки, то заметим, что элемент может быть только в двух состояниях: он входит в сочетание или не входит. Логично элемент включенный в сочетание отметить единицей, а не включенный нулем. Для дальнейшего движения зададим себе естественный вопрос: "Что означает новый термин "отметить", куда будет ставиться единица или ноль".

Мы провели хорошую подготовительную работу, теперь выполнить заключительные рассуждения не так сложно. Идея такова: составим двоичное число с длиной, равной длине исходного множества, и тогда если, например, второй разряд двоичного числа равен 1, то второй элемент множества участвует в сочетании, а если в некотором разряде находится ноль, то соответствующий элемент множества в сочетании не участвует.

Если мы согласны с такой моделью сочетания, то сразу появляется и метод получения очередного сочетания из уже имеющегося. А именно, задача получения очередного сочетания сводится к задаче получения очередного двоичного числа из уже имеющегося. Получить же новое двоичное число можно обычной операцией прибавления двоичной единицы.

Приведем пример. Дано:

- исходное множество — Q, W, S , D;
- начальное число — 0001.

Запишем таблицу соответствия чисел и сочетаний.

Таблица. Двоичные числа и соответствующие им сочетания

№	Число	Сочетание
1	0001	
2	0010	S
3	0011	S D
4	0100	W
5	0101	W D
6	0110	W S
7	0111	W S D
8	1000	Q
9	1001	Q D
10	1010	Q S
11	1011	Q S D
12	1100	Q W

13	1101	Q W D
14	1110	Q W S
15	1111	Q W S D

Алгоритм можно построить как цикл, в начале которого строится число, состоящее из одних нулей, затем на каждом шаге выполнения цикла:

- к уже построенному двоичному числу прибавляется двоичная единица;
- распечатывается очередное сочетание, соответствующая полученному двоичному числу.

Построение структур данных не проблема. Исходное множество — это массив, двоичное число — это также массив. Оба массива имеют одинаковую длину.

Если вам уже когда-либо приходилось решать задачу сложения двух двоичных чисел, то сейчас вы можете уже готовую программу сложения оформить в виде процедуры и часть задачи решена. Это кстати очень важное умение использовать готовые результаты. Но мы пойдем другим путем. Заметим, что у нас нет задачи сложения двух любых двоичных чисел. Перед нами стоит задача прибавить к произвольному двоичному числу двоичную единицу. Это частная задача, а решения частных задач зачастую проще и эффективнее общих.

Действительно, для того чтобы прибавить единицу, достаточно найти первый ноль, начиная с младшего разряда, заменить его на единицу, а затем все единицы, расположенные правее (или левее в зависимости от того, где находится младший разряд) от только что поставленной, заменить на нули. Если вы желаете наглядно увидеть, как работает это правило, посмотрите внимательно на таблице, в ней много двоичных чисел, каждое получено из предыдущего прибавлением двоичной единицы и действие нового правила видно очень хорошо.

Итак, общие идеи мы проговорили достаточно хорошо, теперь запишем алгоритм:

```
Вводим исходное множество, в виде массива.
Формируем массив двоичного числа с длиной, равной количеству
элементов в исходном множестве и заполняем его нулями.
Пока двоичное число содержит хотя бы один ноль, делаем следующее:
  Прибавляем к двоичному числу двоичную единицу
  Для всех разрядов двоичного числа делаем следующее:
    Если текущий разряд равен единице, то распечатываем элемент
    исходного множества с тем же номером, что и единичный разряд.
```

Несколько замечаний по структуре программы

- Заголовок главного цикла выглядит достаточно сложно. Будет разумно разработать специальную функцию проверки на наличие нулей и вызывать ее в заголовке цикла.
- Задачи прибавления единицы к двоичному числу и распечатки очередной выборки выглядят как достаточно сложные самостоятельные задачи, поэтому разумно их

также оформить в виде отдельных процедур, которые будут вызываться поочередно в главном цикле.

Если выполнить эти советы, то главный цикл получится очень компактным и легким для понимания.

Листинг

```
program example;
 uses crt;
 var
  a:array[1..100] of char;
  flag:array[1..100] of byte;
  i,n:integer;
function proverka:boolean;
var
 i:integer;
 f:boolean;
begin
 f:=false;
 for i:=1 to n do
  if flag[i]=0 then f:=true;
 proverka:=f;
end;
procedure summa;
var
 i,j:integer;
begin
 i:=1;
 while (flag[i]=1) and (i<=n) do i:=i+1;
 if i<=n then
  begin
   flag[i]:=1;
   for j:=1 to i-1 do flag[j]:=0;
  end;
end;
procedure printing;
var
 i:integer;
begin
 writeln('-------------------------------------------------------
----');
 for i:=1 to n do
  if flag[i]=1 then write(a[i]);
 writeln;
end;
begin
 clrscr;
 readln(n);
 for i:=1 to n do
  begin
   readln(a[i]);
   flag[i]:=0;
```

```
  end;
 while proverka do
  begin
   summa;
   printing;
  end;
end.
```

Некоторые важные выводы.

Исходная задача является задачей комбинаторной, то есть относящейся к разделу, который считается традиционно сложным для программирования. Не зря значительное количество задач, предлагаемых на олимпиадах различных уровней, — это задачи именно комбинаторные. Нам удалось перевести задачу совсем в иной раздел, мы переформулировали ее как задачу на сложение двоичных чисел с небольшим довеском в виде исходного множества элементов, обработка которого целиком определяется двоичным числом.

Это говорит о важности разносторонних знаний для человека, занимающегося программированием. В смежных, а иногда и даже отдаленных областях знания можно найти самое неожиданное решение для достаточно сложной задачи. Конечно, надо понимать, что четко и ясно никто не скажет, где может найтись такое простое и неожиданное решение, на то оно и неожиданное, поэтому самой верной стратегией подготовки будет широкая любознательность, изучение самых различных областей знания, но в первую очередь необходимо уделить внимание разделам математики.

Применив свои математические знания мы действительно нашли в соседней области хорошее и даже очень хорошее решение задачи, но чтобы действительно понять, насколько оно хорошо, его надо с чем-то сравнить. Поэтому сейчас решим нашу задачу еще раз, при этом постараемся, чтобы второе решение также было хорошим, а затем сравним их.

Второе решение:

Заметим, что каждый элемент исходного множества может либо входить в сочетание, либо не входить. Если для выборки завести отдельный массив, то для каждого элемента массива сочетания возможно только два состояния: либо этот элемент пуст, либо он содержит соответствующий элемент исходного множества.

Для первого элемента массива сочетания существуют два состояния, для каждого состояния первого элемента возможны два состояния второго элемента и т. д. Введем понятие последовательности. *Последовательностью* длины К будем называть часть массива выборки длины К, начиная с первого элемента. *Множество последовательностей* длины К мы можем определить через множество последовательностей длины К–1 следующим образом: каждая последовательность множества длины К — это последовательность из множества длины К–1, к которой в позицию К добавлен либо пробел, либо К-ый элемент исходного множества.

Тогда множество сочетаний — это множество последовательностей длины N. Полученное нами определение имеет рекуррентный вид, из этого следует, что мы можем

получить рекурсивное решение. Строиться рекурсивное решение на следующих почти очевидных соображений:

- цель искомой рекурсивной процедуры — за N вызовов построить очередную выборку, следовательно, на N-ом вызове можно завершать построение очередного сочетания и выполнять распечатку;

- каждый вызов удлиняет уже построенное сочетание на одну позицию, следовательно, в каждом вызове должен определяться очередной элемент сочетания;

- следующий элемент сочетания можно определить двумя способами (пустой или элемент исходного множества), следовательно, каждый вызов процедуры должен порождать еще два вызова.

Приведем код второго решения задачи в листинге.

Листинг

```
program example;
 uses crt;
 var
  a,b:array[1..10] of char;
  n,i:integer;
procedure vb(t,k:integer);
procedure print;
var
 i:integer;
begin
 if k+1=n then
  begin
   for i:=1 to n do
    if b[i]<>' ' then write(b[i],' ');
   writeln;
  end;
end;
begin
 if t=0 then b[k]:=' ' else b[k]:=a[k];
 if k<n then
  begin
   vb(0,k+1);
   print;
   vb(1,k+1);
   print;
  end;
end;
begin
 clrscr;
 readln(n);
 for i:=1 to n do readln(a[i]);
 vb(0,1);
 vb(1,1);
end.
```

В заключение

Нетрудно заметить, что второе решение даже изящнее и короче первого. Это решение, хотя оно и рекурсивное, значительной нагрузки на стековую память не накладывает, так как оба используемых массива являются глобальными, локальных переменных практически нет. Но логика второго алгоритма существенно сложнее для понимания, и с этой точки зрения первый алгоритм более предпочтителен, если есть потребность в быстром получении решения.