**КОМБИНАТОРНЫЕ АЛГОРИТМЫ**

*Дополнительная литература: Окулов С.М. Программирование в алгоритмах*

При поиске оптимального варианта из некоторого допустимого набора часто приходится перебирать все возможные альтернативы. Во многих задачах перебираемые варианты являются членами сочетаний, размещений и других комбинаторных конфигураций. Поэтому большое значение имеют алгоритмы генерации элементов соответствующих конечных множеств.

Главная сложность составления таких ”генераторов” состоит в необходимости сделать их структуру независимой от размерности порождаемых ими множеств, которая может очень сильно варьироваться в различных задачах. Например, алгоритм генерации r-размещений с повторениями очень легко реализовать с помощью r вложенных циклов:

for j1 := 1 to n do

for j2 := 1 to n do

…………………..

for jR := 1 to n do writeln (j1, j2, … jR);

Ясно, что всякий раз при изменении значения r такую программу пришлось бы переделывать, добавляя или удаляя циклы, и на самом деле алгоритм генерации должен быть другим.

Другой особенностью предлагаемых алгоритмов является то, что в них вместо комбинаций самих объектов генерируются комбинации из их номеров. Очевидно, что такие расстановки легко преобразовать в манипуляции, например, с названиями объектов, введя в программу массив из строк символов соответствующей размерности.

Многие из комбинаторных конфигураций удобно генерировать с помощью алгоритма поиска с возвратом.

***Общая схема рекурсивного поиска с возвратом.*** Пусть дано n упорядоченных множеств В1, В2,… Вn. Требуется найти набор векторов вида Х = (х1, х2,… хn), где xi∈Вi, удовлетворяющих некоторым дополнительным условиям. Иными словами Х ∈ В1×В2 ×… ×Вn.

В алгоритме поиска с возвратом каждый вектор Х строится покомпонентно слева направо. Предположим, что уже найдены первые k – 1 компонент х1, х2,… хk–1. Тогда их значения определяют некий набор условий, ограничивающих выбор k-й компоненты некоторым множеством Sk ⊆ Вk. Если Sk не пусто, мы вправе выбрать в качестве хk первый по порядку элемент из Sk, присоединить его к уже выбранным компонентам, и перейти к выбору хk+1 и так далее. Однако, если набор условий таков, что Sk пусто, то мы возвращаемся к выбору k–1-й компоненты. При этом мы отбрасываем хk–1 и выбираем в качестве новой k–1-й составляющей вектора Х тот элемент из Sk–1, который непосредственно следует за только что отброшенным хk–1.

Предположим теперь, что в процессе поиска новых компонент мы дошли до конца, т.е. выбрали хn. Теперь надо как-то использовать полученный вектор Х, в соответствие с условием задачи. Например, если в задаче требуется найти лучший из векторов, то нам следует сравнить построенный вектор Х с решением, которое до данного момента считалось оптимальным и если Х окажется лучше, то надо обновить текущее оптимальное решение, вернуться на шаг назад и продолжить поиски новых решений.

Этот процесс продолжается, пока не будут рассмотрены все возможные вектора решений.

Данная схема легко реализуется посредством рекурсивной процедуры.

**procedure Solve ( k );**

{ k – номер подбираемой координаты (номер глубины рекурсии) }

{ Х – вектор текущего решения }

**begin if 〈 Х – решение 〉 then 〈 Использовать Х 〉**

**else**

**begin 〈 Определить Sk 〉 ;**

**for у ∈ Sk do**

**begin Xk:= y; Solve(k+1 ) end;**

**end;**

**end;**

Конкретный набор требований, которым должны удовлетворять сгенерированные векторы, определяются множествами Sk, а также правилом проверки того, что Х – решение.

**1. *Алгоритм генерации r-размещений с повторениями.*** В размещениях с повторениями на k-месте должно стоять число из набора 1,…, n независимо от чисел, стоящих на предыдущих позициях. То есть для любого k всегда будет Sk= {1, …, n }.

**Пример 2.1.** Пусть r = 3; n =2. Надо получить следующие векторы Х: (1 1 1), (1 1 2), (1 2 1), (1 2 2), (2 1 1), … (2 2 2).

Обозначим А[j] – номер предмета, находящегося на j-м месте, j = 1,…r. Предлагается следующий алгоритм.

**Procedure Razm\_P(k);**

**begin**

**if k=r+1 then write(A[1],…A[r]);**

**else**

**for y:=1 to n do**

**begin A[k]:=y; Razm\_P(k+1);**

**end;**

**end;**

**begin**

**Razm\_P(1);**

**end.**

Назначение процедуры **Razm\_P(k)**– получение k-й компоненты вектора Х =(A[1],…A[r]). Если k = r+1 – это означает, что вектор полностью получен, и надо его использовать (вывести на экран). В противном случае надо получить следующую компоненту. Так как на ее месте может стоять любое число от 1 до n, то Sk = {1, … n}, перебор элементов этого множества обеспечивается циклом **for y:=1 to n do**

**2. *Алгоритм генерации r-размещений без повторений.*** Отличие данного алгоритма от предыдущего в том, что каждое из возможных значений 1, … n элементов размещений можно использовать только один раз. Поэтому в процедуре **Razm\_BP** используется массив **dop[j], j = 1,…n,** в котором **dop[j] = 1**, если значение j не было использовано и **dop[j] = 0**, в противном случае. При продвижении вглубь рекурсии значение j блокируется, чтобы его нельзя было использовать повторно, а при откате – восстанавливается.

**Procedure Razm\_BP(k);**

**begin**

**if k=r+1 then write(A[1],…A[r]);**

**else**

**for y:=1 to n do**

**if dop[y] > 0 then**

**begin A[k]:=y; dop[y]:=dop[y]-1;**

**Razm\_BP(k+1); dop[y]:=dop[y]+1;**

**end;**

**end;**

**begin for i:=1 to n do dop[i]:= 1;**

**Razm\_BP(1);**

**end.**

Данный алгоритм можно также использовать для генерации перестановок без повторений при r = n.

**3. *Алгоритм генерации перестановок с повторениями.*** Этот алгоритм похож на предыдущий, только первоначально **dop[j]=n[j]**– число повторений j-го значения, j = 1,…m. По мере использования этого значения переменная **dop[j]** уменьшается, пока не станет равной 0. Это будет означать, что данное значение уже нельзя использовать. Предлагается следующий алгоритм генерации перестановок из m объектов при .

**Procedure Perest\_P(k);**

**begin if k=n0+1 then write(P[1],…P[n0]);**

**else**

**for y:=1 to m do**

**if dop[y] > 0 then**

**begin P[k]:=y; dop[y]:=dop[y]-1;**

**Lex(k+1); dop[y]:=dop[y]+1;**

**end;**

**begin n0=0;**

**for j:=1 to m do**

**begin dop[j]:=n[j]; n0:=n0+n[j]; end;**

**Lex(1);**

**end.**

**4. *Алгоритм генерации r-сочетаний без повторений.*** Алгоритм похож на генерацию размещений без повторений, отличие в том, что в каждом сочетании не допускаются перестановки элементов, т.е. каждый следующий элемент больше предыдущего.

**Пример 2.2.** Подобные 3-сочетания из 5 выглядят так: (1 2 3), (1 2 4), (1 2 5), (1 3 4), … (1 4 5), (2 3 4), … – т.е. генерируется возрастающая последовательность 3-значных чисел.

Для обеспечения этого условия в генерации сочетаний используется цикл **for y:=t to n do,** а не **for y:=1 to n do,** как в размещениях, где переменная **t** подбирается по правилу **if k<=1 then t:=1 else t:=А[k-1]+1**.

**Procedure Sochet\_BP(k);**

**begin if k=r+1 then write(А[1],…А[r]);**

**else**

**begin if k<=1 then t:=1 else t:=А[k-1]+1;**

**for y:=t to n do**

**if dop[y] > 0 then**

**begin А[k]:=y; dop[y]:=dop[y]-1;**

**Sochet\_BP(k+1);**

**dop[y]:=dop[y]+1;**

**end;**

**end;**

**end;**

**begin for i:=1 to n do dop[i]:= 1;**

**Sochet\_BP(1);**

**end.**

**Технология выполнения работы.**

Рассмотрим различные типы вариантов заданий.

1. Разгадка числовых ребусов.

В числовых ребусах зашифровывают некоторое арифметическое действие, буквами обозначают цифры. Разным цифрам соответствуют разные буквы. Требуется разгадать, какая цифра скрывается за каждой буквой. Это возможно, если воспользоваться программой генерации размещений без повторений.

**Пример.** ТОРГ **×** Г = ГРОТ. Имеем 4 различных цифры: Т, О, Р, Г. Следовательно, необходимо сгенерировать 4-размещения из 10 (всего 10 цифр), т.е. массив а[1], … , а[4], удовлетворяющий следующим условиям:

1) а[1] ≠ 0, а[4] ≠ 0 – т.к. число не может начинаться с 0.

2) а[1]⋅1000 + а[2]⋅100 + а[3]⋅10 + а[4] = а[4]⋅1000 + а[3]∙100 + а[2]∙10 + а[1] – это запись зашифрованного действия.

В процессе генерации каждую полученную комбинацию проверяют на выполнение перечисленных условий и отбирают подходящие. В общем случае решений может быть несколько. В данном примере ТОРГ = 1089 (1089⋅9 = 9801).

1. Генерация m-последовательностей 0 и 1.

Для решения можно использовать алгоритм генерации размещений с повторениями, где xi = 0 и yi = 1.

Однако если в условии задано ограничение на количество 0 или 1, то при таком подходе заведомо будут генерироваться лишние последовательности. В этом случае лучше использовать алгоритм генерации сочетаний без повторений для номеров мест, на которые будут расставляться 0 или 1. При получении каждой сгенерированной комбинации номеров мест для 1 (0) в массиве С, нужно обнулить (заполнить 1) массив А требуемых m-последовательностей, а затем расставить в нём на места из массива С единицы (нули).

1. Вычисление кратного интеграла.

Для приближенного вычисления кратного интеграла использовать формулу левых, средних или правых прямоугольников



,

где , *k* – кратность интеграла, *h* – шаг разбиения (вводится с клавиатуры), а  для формулы левых прямоугольников,  для формулы средних прямоугольников и  для формулы правых прямоугольников. Для решения используется алгоритм генерации размещений с повторениями для номеров интервалов по каждой переменной.

1. Раскрытие полиномиальной формулы.

Для раскрытия полиномиальной формулы используется алгоритм генерации размещений с повторениями для получения степеней разложения, в котором для каждой сгенерированной комбинации проверяется условие . На экран монитора выводится формула разложения полинома с использованием знака ^ для операции возведения в степень.

1. Использование принципа включений и исключений.

Для вычисления по формуле включений и исключений генерируются сочетания без повторений для номеров проверяемых свойств. Для каждого числа свойств генерируются различные сочетания, а для каждой сгенерированной комбинации вычисляется (или извлекается) и суммируется количество элементов, удовлетворяющих соответствующему свойству или набору свойств, затем вычисление суммы добавляются в формулу включений и исключений с нужным знаком.

1. Решение задачи коммивояжера.

Используется алгоритм генерации перестановок без повторений, в котором для каждой комбинации вычисляется суммарное расстояние, пройденное коммивояжером, и определяется минимальное из этих расстояний.

1. Генерация числовых комбинаций.

Для генерации используется алгоритм генерации размещений с повторениями или без повторений (в зависимости от условия варианта). Каждая сгенерированная комбинация должна проверяться на выполнение условия и выводиться при выполнении условия.

Если требуется исключить заданные *p*  цифр из набора, то нужно определить константный массив из 10 – *p*  элементов, не включающий эти цифры, и генерировать номера элементов в этом массиве, а выводить сами цифры.

Если требуется включить заданные *p*  цифр в число выводимых для чисел, не содержащих одинаковых цифр, то нужно записать эти цифры в результирующий массив сочетаний на первые *p* мест, определить константный массив из 10 – *p*  элементов, не включающий эти цифры. Затем сгенерировать номера для *k - p* элементов в этом массиве, записать цифры с этими номерами в массив сочетаний на места с *p+1*-го по *k*-е. Наличие заданных цифр в числе с повторяющимися цифрами проверяется непосредственно по сгенерированному массиву.

1. Генерация различных слов из заданного набора букв.

Для генерации используется алгоритм генерации размещений с повторениями или без повторений (в зависимости от условия варианта). Каждая сгенерированная комбинация должна проверяться на выполнение условия и выводиться при выполнении условия.

***Варианты заданий***

1. Написать программу разгадки числового ребуса

**П Ч Ё Л К А \* 7 = Ж Ж Ж Ж Ж Ж**

1. Написать программу разгадки числового ребуса

**П Р О П : О = Р Ц И Я**

1. Написать программу разгадки числового ребуса

**С С С Р = Р Ф**

1. Написать программу разгадки числового ребуса

**(М + О + С + К + В + А) 4 = М О С К В А**

1. Написать программу разгадки числового ребуса

****

1. Написать программу разгадки числового ребуса

**С А Р = А Т О В**

1. Написать программу генерации *m*-последовательностей 0 и 1, удовлетворяющих обоим требованиям:
   1. число единиц должно быть чётно (включая 0 единиц);
   2. число нулей должно быть не меньше числа единиц.
2. Написать программу генерации *m*-последовательностей 0 и 1, удовлетворяющих обоим требованиям:
   1. число единиц должно быть не меньше *m* / 2 - 2;
   2. хотя бы 2 единицы шли подряд.
3. Написать программу генерации *m*-последовательностей 0 и 1, удовлетворяющих обоим требованиям:
   1. число нулей должно быть не больше *m* / 2 + 2;
   2. никакие 2 нуля не шли подряд.
4. Написать программу генерации *m*-последовательностей 0 и 1, удовлетворяющих обоим требованиям:
   1. число нулей должно быть нечётно;
   2. число нулей должно быть меньше числа единиц не больше, чем на 3.
5. Написать программу генерации *m*-последовательностей 0 и 1, удовлетворяющих обоим требованиям:
   1. никакие 3 единицы не стоят рядом;
   2. число единиц превосходит число нулей.
6. Написать программу генерации *m*-последовательностей 0 и 1, удовлетворяющих обоим требованиям:
   1. хотя бы 3 единицы стоят рядом;
   2. число единиц кратно 3..
7. Написать программу, которая генерирует все *k*-значные числа, не содержащие одинаковых цифр, кратные 2 и 3 (*k*≤10).
8. Написать программу, которая генерирует все *k*-значные числа, не содержащие цифр 5 и 7 (*k*≤10).
9. Написать программу, которая генерирует все *k*-значные числа, не содержащие одинаковых цифр и цифры 4 и 8 (*k*≤10).
10. Написать программу, которая генерирует все *k*-значные числа не кратные 3 и 4 (*k*≤10).
11. Написать программу, которая генерирует все *k*-значные числа, не содержащие одинаковых цифр и имеющие цифры 3, 5 и 7 (*k*≤10).
12. Написать программу, которая генерирует все *k*-значные числа, имеющие цифры 2 и 6 и кратные 3 и 5 (*k*≤10).
13. Составить программу вычисления определённого интеграла произвольной кратности по прямоугольной области по формуле средних прямоугольников. С её помощью вычислить , где  
14. Составить программу вычисления определённого интеграла произвольной кратности по прямоугольной области по формуле левых прямоугольников. С её помощью вычислить .
15. Составить программу вычисления определённого интеграла произвольной кратности по прямоугольной области по формуле правых прямоугольников. С её помощью вычислить .
16. Написать программу раскрытия полиномиальной формулы .
17. Написать программу для формулы включения и исключения. С её помощью определить количество натуральных чисел меньше или равных *m*, не делящихся ни на одно из заданных *k* чисел .
18. Написать программу, которая генерирует слова, содержащие *k* различных букв из последних *k*+4 букв алфавита таких, что сумма их порядковых номеров четна.
19. Написать программу, которая генерирует слова, содержащие *k* букв из последних *k* букв алфавита, допускающие повторения букв, но никакие 2 соседние буквы не повторяются и 1-я не совпадает с последней.
20. Написать программу, которая генерирует слова, содержащие *k* различных букв из первых *k*+5 букв алфавита таких, что код 1-й буквы не превосходит кода последней.
21. Написать программу, которая генерирует слова, содержащие *k* букв (от «а» до «о»), допускающие повторения букв, но никакая буква не должна встречаться более *k*-2 раз.
22. Дана матрица расстояний между населенными пунктами. Решить задачу коммивояжера, т. е. построить маршрут, проходящий точно 1 раз через все населенные пункты и возвращающийся в начальный, при котором суммарное расстояние минимально.



1. Дана симметричная матрица расстояний между населенными пунктами. Решить задачу коммивояжера, т. е. построить маршрут, проходящий точно 1 раз через все населенные пункты и возвращающийся в начальный, при котором суммарное расстояние минимально.



1. Переставить буквы слова «ЛОГАРИФМ» так, чтобы 2-е, 4-е и 6-е места были заняты гласными буквами.
2. Переставить буквы слова «ПАСТУХИ» так, чтобы как гласные, так и согласные шли в алфавитном порядке.
3. Составить из цифр числа 53694 все возможные нечетные числа (каждую цифру можно использовать не более одного раза).
4. Составить все возможные четырёхзначные числа из цифр числа 132132.
5. Составить слова из 9 согласных и 7 гласных слова, в которые входят 4 различные согласные и 3 различные гласные.
6. Расположить 5 львов и 4 тигров так, чтобы никакие 2 тигра не шли друг за другом.
7. Выбрать 5 книг из 12 стоящих на полке так, чтобы никакие 2 выбранные книги не стояли рядом.