Генерирование k-элементных подмножеств

Выполнила Тюрина А.И
Группа 8307
2018-2019 уч.г

Содержание

Цели и задачи данной работы

Цель: Создать программную реализацию двух основных алгоритмов генерирования k-элементных подмножеств

Задачи:

- Исследовать статью 1.7 книги Липский В. «Комбинаторика для программистов»
- Разобрать алгоритм генерации всех подмножеств множества
- Разобрать алгоритм генерации k-элементных подмножеств в лексикографическом порядке
- Разобрать алгоритм генерации подмножеств множества за счет их минимального изменения(удаления и добавления одного элемента)

Введение

Алгоритмы теории множеств нередко применяются в программировании, поэтому я хотела бы рассказать о генерации подмножеств заданного множества. Данная Тема расположена на стыке комбинаторики и программирования, а значит мы с ней не раз сталкивались на нескольких дисциплинах.

Множество

Множество — это набор элементов. Все элементы множества различны, то есть один элемент не может встретиться в множестве дважды.

Программно реализовать множество можно разными способами: в виде класса, в виде массива с функциями для операций над ним, в виде структур, содержащих разные данные об элементе и т.д.

Генерация подмножеств множества может пригодиться, допустим, в игре, когда из набора персонажей необходимо выбрать случайную их группу.

Допустим, имеется множество M = {1,2, 3}. Запишем все его подмножества.

Первым в списке идет пустое множество, которое является подмножеством любого множества. Последний — исходное множество (множество является подмножеством самого себя).

Существует величина мощность множества, характеризующая количество элементов множества. Так вот, количество подмножеств множества можно вычислить по формуле 2^n, где n — мощность множества. У множества M количество подмножеств равно 2^3 = 8, в чем можно убедиться, посчитав их .

```
{}
{1}
{2}
{12}
{3}
{13}
{23}
{123}
```

Генерация всех подмножеств множества

Программно сгенерировать все подмножества, на самом деле, довольно просто. Например, можно воспользоваться Кодом Грея. Но есть способ еще проще, в основе которого лежит бинарный или двоичный код.

Рассмотрим числа в диапазоне 0..2^n-1 в двоичной системе счисления с разрядностью n (увеличив при необходимости незначащими нулями):

```
0: 000
1: 001
2: 010
3: 011
4: 100
5: 101
6: 110
7: 111
.
.
.
```

Каждый из этих двоичных кодов можно использовать как маску подмножества, то есть, единица в i-ом бите характеризует наличие i-го элемента множества в подмножестве, ноль — отсутствие. Например: 101 = {1, 3}

```
if (M)
{
    for (i=0;i<num;i++)
    {
        printf ("Enter your element:");
        scanf ("%i", &M[i]);
        getchar();
    }
}
n = pow(2, w);
for ( i = 0; i < n; i++ )
{
    printf("{");
    for ( j = 0; j < w; j++ )
        if ( i & (1 << j) )
           printf("%d", M[j]);
    printf("}\n");
}
```

Таким образом, для генерации подмножеств необходимо организовать цикл от 0 до 2^n-1, на каждой итерации представить значение счетчика в виде бинарного кода, на основе его составить подмножество.

Порядок, в данном случае, роли не играет. В данном случае мы просто выводим подмножества.

В целом алгоритм понятен, но считаю необходимым остановится на шаге : *if (i & (1 << j))* Напомню, что оператор “<<” — это логический сдвиг влево. Он используется, чтобы узнать значение j-го бита числа i.

Пример: i = 18 (10010), j = 2. После логического сдвига влево единицы (00001) на 2 получаем число 00100. Затем мы число i логически умножаем на полученную после сдвига маску. Маска представляет собой совокупность нулей с установленным в единицу j-м битом. При умножении, если j-й бит в числе i равен единице, мы получим число равное маске, если j-й бит равен нулю, то результат умножения будет нулем:

$$10010_2(18_{10})\ \&\ 10_2(2_{10}) = \mathbf{10_2(2_{10})}$$

Результаты реализации

```
Enter the number of elements in your set:4
Enter your element:2
Enter your element:5
Enter your element:8
Enter your element:9
{}
{2}
{5}
{25}
{8}
{28}
{58}
{258}
{9}
{29}
{59}
{259}
{89}
{289}
{589}
{2589}
```

```
Enter the number of elements in your set:3
Enter your element:1
Enter your element:4
Enter your element:9
{}
{1}
{4}
{14}
{9}
{19}
{49}
{149}
```

Генерация k-элементных подмножеств в лексикографическом порядке

Допустим, имеется множество X: {1, ..., n}. Тогда каждому k-элементному подмножеству взаимно однозначно соответствует возрастающая последовательность длины k с элементами из X: множеству {3,5,1} соответствует последовательность {1,3,5}.

Чтобы понять алгоритм генерации, достаточно заметить, что последовательностью, непосредственно следующей за последовательностью $\langle a_1, \dots, a_k \rangle$ является:

$$\langle b_1, \dots, b_k \rangle = \langle a_1, \dots, a_{p-1}, a_p + 1, a_p + 2, \dots, a_p + k - p + 1 \rangle$$

где $p = \max\{i: a_i < n - k + 1\}$

Более того, последовательностью, непосредственно следующей за $\langle b_1, \dots, b_k \rangle$, является:

$$\langle c_1, \dots, c_k \rangle = \langle \mathrm{b}_1, \dots, \mathrm{b}_{p'-1}, \mathrm{b}_{p'} + 1, \mathrm{b}_{p'} + 2, \dots, \mathrm{b}_{p'} + k - p + 1 \rangle$$

где $p' = \begin{cases} p - 1, & \text{если } b_k = n, \\ k. & \text{если } b_k < n \end{cases}$

Будем предполагать, что последовательности $\langle \mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k} \rangle$ и $\langle b_1, \dots, b_k \rangle$ отличаются от $\langle n - k + 1, \dots, n \rangle$ — последней последовательности в нашем порядке.

```
while( p >= 1 )
{
    my_out(a,k);
    if( a[k] == n )
        p = p - 1;
    else
        p = k;
    if( p >= 1)
        for( i=k; i>=p; i-- )
        {
            a[i] = a[p] + i - p + 1;
        }
}
```

Изначально множество a заполнено в лексикографическом порядке.
p — переменная, отвечающая за позицию в наборе, с которой мы начинаем менять его.
k — количество элементов в подмножестве
i — счетчик

Суть алгоритма: находим минимальный правый элемент (позиция [j]), и пробегаем по нему до n. Смещаемся в право [j-1], изменяем значение на позиции на +1, изменяем (j) на значение позиции [j-1] на +1 и т.д

Результаты реализации

```
Enter the number of elements in sequence:6
Enter the number of elements in set:4
The set is:{ 1 2 3 4 }
The set is:{ 1 2 3 5 }
The set is:{ 1 2 3 6 }
The set is:{ 1 2 4 5 }
The set is:{ 1 2 4 6 }
The set is:{ 1 2 5 6 }
The set is:{ 1 3 4 5 }
The set is:{ 1 3 4 6 }
The set is:{ 1 3 5 6 }
The set is:{ 1 4 5 6 }
The set is:{ 2 3 4 5 }
The set is:{ 2 3 4 6 }
The set is:{ 2 3 5 6 }
The set is:{ 2 4 5 6 }
The set is:{ 3 4 5 6 }
```

```
Enter the number of elements in sequence:5
Enter the number of elements in set:3
The set is:{ 1 2 3 }
The set is:{ 1 2 4 }
The set is:{ 1 2 5 }
The set is:{ 1 3 4 }
The set is:{ 1 3 5 }
The set is:{ 1 4 5 }
The set is:{ 2 3 4 }
The set is:{ 2 3 5 }
The set is:{ 2 4 5 }
The set is:{ 3 4 5 }
```

Генерация подмножеств множества за счет их минимального изменения. Код Грея.

Кодом Грея называется последовательность наборов, построенная таким образом, что каждый следующий набор отличается от предыдущего только в одном разряде.

Вообще, ***n*-разрядный код Грея** — это упорядоченная последовательность, состоящая из 2^n n-разрядных кодовых слов, каждое из которых отличается от соседнего в одном разряде.

С помощью бинарного кодирования рассмотрим алгоритм генерации подмножеств. Для удобства, рассмотрим множество B^m наборов из m нулей и единиц. Понятно, что множестве наборов из m нулей и единиц содержит ровно 2^m элементов.

Кроме рассмотренный способов перебора наборов, можно предложить другой алгоритм, который на каждом шаге меняет значение только одной компоненты. Этот алгоритм основан на идее рекурсии.

- Фиксируем нулевое значение m-й компоненты и перебираем все наборы длины $m-1$ для оставшихся компонент.
- Меняем значение m-й компоненты с 0 на 1. Перебираем наборы длины $m-1$ в обратном порядке.

В этом случае легко можно дать ответ, как пронумеровать наборы множества. Действительно, текущий набор множества — это двоичное представление некоторого числа, которое и является номером нашего набора. Таким образом, мы каждому набору можем сопоставить его численный эквивалент.

- Пример: Наборы длины 4.

Столбец *it* показывает номер текущей итерации, а столбец k_{it} — номер компоненты, которая подлежит обновлению.

x_4	x_3	x_2	x_1	it	k_{it}
0	0	0	0	1	1
0	0	0	1	2	2
0	0	1	1	3	1
0	0	1	0	4	3
0	1	1	0	5	1
0	1	1	1	6	2
0	1	0	1	7	1
0	1	0	0	8	4
1	1	0	0	9	1
1	1	0	1	10	2
1	1	1	1	11	1
1	1	1	0	12	3
1	0	1	0	13	1
1	0	1	1	14	2
1	0	0	1	15	1
1	0	0	0	16	—

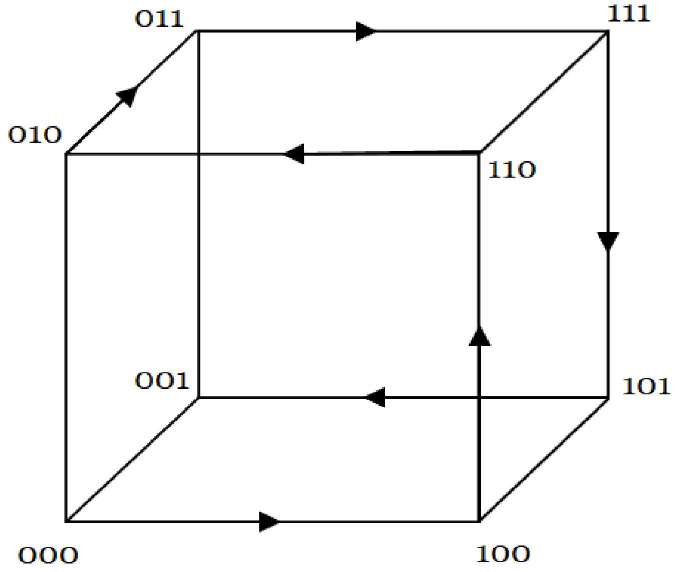
```
if (b)
{
    i=1;
    for (;i<=n;i++)
    {
        b[i]=0;
    }
    i=0;
    do
    {
        set_out(b,n);
        i = i + 1;
        p = 1;
        j = i;
        while ((j%2)==0)
        {
            j = j/2;
            p ++;
        }
        if (p <= n)
         {
             b[p] = 1 - b[p];
         }
    }while (p < n);
}
```

- Приведем нерекуррентный алгоритм генерации кодов Грея. Будем рассматривать бинарные коды Грея порядка *n*. Итак, на вход алгоритма подается единственное число *n*, которое указывает порядок кода Грея.

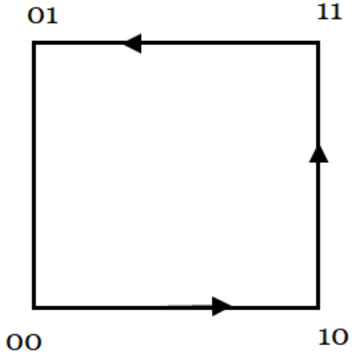
По ходу выполнения алгоритма мы получим последовательность всех подмножеств n-элементного множества, в которой каждое последующее подмножество получается из предыдущего добавлением или удалением единственного элемента. При этом каждое подмножество будет представляться бинарной последовательностью $B[1], \dots, B[n]$.

Последовательность полученных подмножеств можно проиллюстрировать на графе (n-мерном кубе), где вершины —бинарные последовательности, которые соединены ребром, если их пос-ти отличаются в 1-ой позиции. Тогда эта последовательность соответствует **гамильтонову пути** в графе, т. е. пути, содержащему каждую вершину графа только один раз.

Пример гамильтонова пути: n=3



Пример гамильтонова пути: n=2



Результаты реализации

```
The number of elements in set: 4
Our set is:
Our set is: 1
Our set is: 12
Our set is: 2
Our set is: 23
Our set is: 123
Our set is: 13
Our set is: 3
Our set is: 34
Our set is: 134
Our set is: 1234
Our set is: 234
Our set is: 24
Our set is: 124
Our set is: 14
Our set is: 4
```

```
The number of elements in set: 3
Our set is:
Our set is: 1
Our set is: 12
Our set is: 2
Our set is: 23
Our set is: 123
Our set is: 13
Our set is: 3
```

Заключение

В итоге проведенной работы были последовательно выполнены все поставленные задачи и достигнута цель.

Полный код программ можно найти по ссылке: https://github.com/AlexandraTyurina/alternative-examination

Использованная литература

- Сачков В.Н. Введение в комбинаторные методы дискретной математики. – 2-е изд., испр. и доп. – М.: МЦНМО, 2004
- Липский В. Комбинаторика для программистов. М.:“Мир”, 1988.
- Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. М.: МЦНМО, 2000.
- Романовский И.В. Дискретный анализ. — 3-е изд. — СПб: Невский Диалект; БХВ Петербург, 2003.