возможна лишь активизация перехода *Execute*(исполнение программы в защищенном режиме). После активизации перехода *Adjust*(подготовка к эксплуатации) уничтожаются следы процесса подготовки рабочей версии программы, включая код функций переходов *Evaluate*, *Prepare* и *Adjust*. Позиция *Control* соответствует необходимым условиям активизации перехода *Execute*, включая, например, значение ключа, считанного из аппаратного носителя.

Практическая реализация защиты не требует от разработчика функциональной части прикладной программы специальных действий. Декларация объекта защиты может выполняться независимым лицом, осведомленным о технологии обработки данных.

Таким образом, представляемая схема защиты на логическом уровне утилизирует потенциальные возможности автономных аппаратных средств криптографии для упреждающей проверки программного канала доступа к защищаемым данным.

Литература

1. Ревотюк М.П. Шаблоны систем обеспечения безопасности разрабатываемых программ в вычислительных средах с открытой архитектурой//Компьютерные технологии в обеспечении безопасности электронной информации: Материалы межд. конф.(Минск, 4-9 ноября 2002 г.) - Мн.: БелИСА, 2002. – с. 107-117.

2. Ховард М., Лебланк Д. Защищенный код/Пер. с англ. – М.: Издательско-торговый дом "Русская редакция", 2003.–704 с.

УДК 004.056.5

Генераторы подмножеств комбинаторных объектов

Кишкевич А.П., Ревотюк М.П.*
ИП "Центр компетенции "Эвклид", БГУИР*

Объект рассмотрения – вычислительные схемы решения комбинаторных задач методом перебора вариантов. В случае, когда множество вариантов порождается алгоритмически, возникает проблема построения их подмножеств с целью независимого анализа на вычислительной сети.

Централизованное порождение вариантов агентом-диспетчером приводит к повышенной загрузке сети. Предлагается для генерации вариантов из множества сочетаний и перестановок экономный алгоритм выделения подмножеств без потерь времени на порождение и фильтрацию пропускаемых вариантов.

Цель исследования – построить абстрактные производные классы-итераторы сочетаний и перестановок для заданного подмножества всех перестановок и сочетаний.

Ключевой вопрос решения задачи порождения подмножества сочетаний/перестановок – фиксация начального состояния итератора. Необходимо по номеру сочетания/перестановки восстановить состав его элементов. По существу, требуется решать задачу, обратную задаче порождения сочетаний/перестановок.

В качестве исходной основы для построения итератора сочетаний и перестановок используем известные алгоритмы порождения сочетаний и перестановок в лексикографическом порядке.

Базовые классы-итераторы генерации всех перестановок и сочетаний будут имеют схожую структуру. Конструктор базового класса создает внутреннее отображение множества элементов перестановки и сочетания соответственно.

Оператор вызова функции выполняет проверку условий продолжения итераций, в случае ее успеха, организует обработку текущей перестановки/сочетания (вызов виртуальной функции doit()) и подготовку представления следующей перестановки/сочетания. Таким образом достаточно в конструкторе производного класса указать начальную перестановку/сочетание и количество итераций. [1].

Порождение подмножества сочетаний

Начальное значение сочетания – множество $\overline{\{n-m,\ n-1\}}$.

Так как общее количество сочетаний целочисленное

$$C_m^n = \frac{n!}{m! \cdot (n-m)!}, \qquad (1)$$

то при разбиении на k не пересекающихся подмножеств размер последних - $]C_m^n / k[$ или $\lfloor C_m^n / k \rfloor$.

Сочетания, порождаемые в лексикографическом порядке, связаны рекуррентным соотношением для индекса [2]

$$I_{n,m}(k_1, k_2, \ldots k_m) = C_m^n - \sum_{i=1}^{m} C_{m-i+1}^{n-k_i}, \qquad (2)$$

позиции сочетания $(k_1, k_2, \ldots k_m)$ в списке всех сочетаний.

Индекс последнего сочетания определяется (1):

$$I_{n,m}(n-m-1,\ n-m-2,\ldots\ n) = C_m^n .$$

Анализируя (1) и (2), достаточно легко придти к идее рекурсивного восстановления сочетания $(k_1, k_2, \ldots k_m)$ из значения $I_{n,m}(k_1, k_2, \ldots k_m)$.

На первом шаге множество S сочетаний из n элементов по m можно разбить на две непересекающиеся части:

S_1 – сочетания из $n-1$ элемента по $m-1$ с позицией m, установленной в n;

S_2 – сочетания из из $n-1$ элемента по m, такие, что $S_2 = S \setminus S_1$.

Мощности множеств сочетаний связаны соотношениями

$$|S| = C_m^n,\ |S_1| = |S| \cdot m / n,\ |S_2| = |S| - |S_1|. \qquad (3)$$

Зафиксировав известный максимально допустимый элемент $k_m = n$ в позиции m, и учитывая, что для сохранения лексикографического порядка всех элементов любого сочетания должно выполняться условие $(0 \le k_1 < k_2 < \ldots < k_m \le n-1)$, процедуру разбиения можно последовательно продолжить. Выражения (3) при этом применяются для уменьшающихся значений m и n.

Алгоритм порождения любого из k подмножеств сочетаний из n элементов по m штук, указанного номером i, $i = \overline{0, k-1}$ представлен ниже производным классом, где конструктор выполняет фиксацию начального сочетания и счетчика итераций:

```
template <class Type> // Итератор подмножеств сочетаний
class nth_combination: public combination<Type> {
public:
 nth_combination(Type N,Type M,Type K,Type I):
  combination<Type>(N,M) {
  Type cs=count;
  count=cs/K;
  Type ost=cs%K;
  Type ps=(ost)? ((I<ost)? ++count*I: count*I+ost): count*I;
  for (Type i=m, cp=0, cn=n; i>0; ) {
   Type nb=(cs*i)/cn;
   if (cp+nb<=ps){
    cp+=nb; cs-=nb; cn--;
   } else {
    cs=nb; v[--i]=--cn;
   }
  }
 }
};
```

Порождение подмножества перестановок

Так как общее количество перестановок целочисленное

$$C^n = n!, \qquad (1)$$

то при разбиении на k непересекающихся подмножеств размер последних - $]C^n / k[$ или $\lfloor C^n / k \rfloor$.

На первом шаге множество S перестановок из n элементов можно разбить на n непересекающихся частей:

$S_0\ S_1\ S_i \ldots S_{n-1}$ – сочетания из $n-1$ элемента с позицией 0, установленной в i;

Мощности множеств перестановок связаны соотношениями

$$|S| = C^n,\ |S_i| = |S|/n. \qquad (3)$$

Зафиксировав известный максимально допустимый элемент $k_0 = i$ в позиции *0*, и учитывая, что для сохранения лексикографического порядка всех элементов любой перестановки должно выполняться условие $(0 \le k_1 < k_2 < \ldots < k_n \le n-1)$, процедуру разбиения можно

последовательно продолжить. Выражения (3) при этом применяются для уменьшающегося значения n.

Алгоритм порождения любого из k подмножеств перестановок из n, указанного номером i, $i = \overline{0, k-1}$ представлен ниже производным классом:

```
template <class Type> // Итератор подмножеств сочетаний
class nth_permutation: public permutation<Type> {
int I, K, N;
public:
 nth_permutation (int i, int K, int N): permutation <Type>(N),
I(i), K(K), N(N) {
    Type cs=count; count/=K;
    Type ost=cs%K;
    Type ps=(ost)? ((I<ost)? ++count*I: count*I+ost): count*I;
    vector<bool> tv(N, true);
    std::vector<int>::iterator it=first;
    for(int cn=0, itn; cn<N; cn++, ps-=itn*cs){
     int cd=0, j=0;
     for(cs/=(N-cn), itn=ps/cs;;j++)
      if(tv[j])
       if(cd++==itn) break;
     tv[j]=false, *it++=j;
    }
  }
};
```

Таким образом, для координации процессов анализа вариантов на сети агентами системы выбора достаточно назначения номеров участвующим в решении задачи узлам сети. Передача элементов перестановок и сочетаний и холостой пропуск неиспользуемых вариантов не требуется. Порождаемые подмножества не пересекаются.

Литература

1. Липский,В. Комбинаторика для программистов: Пер. с польск. – М.: Мир, 1988. – 218 с.
2. Рейнгольд,Э., Нивергельт,Ю., Део,Н. Комбинаторные алгоритмы. Теория и практика: Пер с агл. – М.: Мир, 1980. – 476 с.