УДК 004.021

Informatics and Mathematical Methods in Simulation
Vol. 6 (2016), No. 4, pp. 385-396

ВЫЧИСЛИТЕЛЬНЫЕ МОДЕЛИ АЛГОРИТМОВ ПОКРЫТИЯ

О.Н. Паулин

Одесский Национальный Политехнический Университет,
пр. Шевченко, 1, Одесса, 65044, Украина; e-mail: paolenic@yandex.ua

В статье ставится и решается проблема построения вычислительных моделей для класса комбинаторных задач. Практически важной в этом классе является задача о покрытии, использующая переборный механизм. Такого рода задачи возникают, например, при необходимости оптимального выбора поставщиков при сборке сложного изделия. Вычислительные процессы решения задачи о покрытии имеют много общих функционально законченных компонент, названных нами макрооперациями, которые могут быть выделены как вычислительные модели из этого процесса. Такое выделение позволит собрать библиотеку макроопераций для разных классов задач, что упростит и ускорит анализ программ ещё на стадии построения алгоритмов (вычислительных процессов). Рассматриваются 2 метода и соответственно 2 алгоритма решения задачи о покрытии: полного перебора подмножеств и граничного перебора по вогнутому множеству. Приводятся словесные описания алгоритмов, их схемы, а также описания и схемы вычислительных модулей. Выделяются макрооперации как вычислительные модели, которые частично обобщаются.

Ключевые слова: комбинаторные задачи, вычислительные процессы, вычислительные модели, макрооперации, алгоритмы покрытия, полный перебор, граничный перебор, вычислительные модули.

Введение

В настоящее время широко распространён класс задач, в которых осуществляется поиск оптимальных решений. Такая задача обладает некоторым множеством решений, на котором задаётся функция стоимости; критерием оптимальности в таких задачах является минимальное значение функции стоимости решения. Этот класс задач является по сути комбинаторным [1]. Задачу покрытия [2] можно рассматривать как вариант задачи поиска в некотором конечном множестве определённых подмножеств с заданными свойствами. Она возникает при необходимости выбора таких компонент из множества возможных, которые покрывают заданную их совокупность, при условии, что выбор будет оптимальным. Например, в производстве сложного изделия, скажем, авиадвигателя, может понадобиться некоторое множество деталей от различных поставщиков с разными комплектами деталей и с разными ценами. При этом производителю надо минимизировать либо общую цену поставок, либо количество поставщиков. Другой пример – логические задачи о покрытии множеств, о поиске минимальных разбиений и др. [3] при синтезе и анализе цифровых схем на программируемых логических матрицах.

Расширение понятия «покрытия» выполнено в [4]; здесь синтез логического модуля проводится на основе специальным образом построенной таблицы покрытия, в которой отношение принадлежности отмечается непосредственным указанием элементов, которые входят в множество термов фундаментальных симметрических функций [5].

Существуют различные методы и алгоритмы для решения задачи покрытия (РЗП), отличающиеся точностью и сложностью вычислительного процесса (ВП); зачастую оказывается, что множество решений – подмножества с заданными свойствами – является выпуклым по отношению включения. В таком случае нахождение множества всех решений сводится к нахождению его нижней границы, что может иногда существенно сократить объём производимых при поиске вычислений.

В этих ВП имеется немало общего и задача состоит в выделении этого общего в виде макроопераций (МО) как вычислительных моделей компонент ВП. Первый опыт подобного выделения приведен в [6]. Отлаженные с помощью сети Петри МО [7] могут быть собраны в библиотеку МО для разных классов задач, что позволит «собирать» новые ВП. В свою очередь, это позволит упростить и ускорить анализ программ, реализующих разнообразные вычислительные процессы, ещё на стадии построения алгоритмов (организации вычислительных процессов) [8, 9]. Последнее особенно важно в свете того, что программы усложняются, растёт число ошибок, и их отладка часто занимает в разы большее время, чем само написание программ [10].

Целью работы является ускорение анализа программ за счёт выделения макроопераций из вычислительного процесса покрытия. Для достижения цели необходимо решить следующие задачи:

- составить словесные описания и построить схемы алгоритмов перебора;
- проанализировать словесные описания и выявить макрооперации;
- составить общий список макроопераций.

Задача о покрытии и методы её решения

Постановка задачи о покрытии. Пусть $B=\{b_1,\ldots,b_n\}$ – опорное множество. Имеется множество *A,* состоящее из *m* подмножеств A_i ($A=\{A_1,\ldots,A_m\}$) множества *B* таких, что $A_i \subseteq B, \bigcup_{i=1}^{m} A_i = B$. Каждому подмножеству A_i сопоставлено число c_i, называемое *ценой.* Множество $P=\{A_{k_1},\ldots,A_{k_l}\}$ ($k_j \in \{1,\ldots,m\}, l \le m, k$ – номер варианта выборки подмножеств) называется *решением задачи о покрытии* или просто *покрытием*, если выполняется условие

$$\bigcup_{j=1}^{l} A_{k_j} = B, \qquad (1)$$

при этом цена $C=\sum_{j=1}^{l} c_{k_j} \to \min$.

Термин покрытие означает, что совокупность множеств $A_{k_1},\ldots,A_{k_l}$ содержит все элементы множества B, т.е. «покрывает» множество B.

Определение 1. *Безизбыточным* называется покрытие, если при удалении из него хотя бы одного элемента оно перестает быть покрытием. Иначе – покрытие избыточно.

Определение 2. Покрытие P называется *минимальным*, если его цена $C=\sum_{j=1}^{l} c_{k_j}$ – наименьшая среди всех покрытий данной задачи.

Определение 3. Покрытие P называется *кратчайшим*, если l – наименьшее среди всех покрытий данной задачи.

Отметим, что в этом случае цены всех подмножеств A_i приняты по умолчанию одинаковыми и равными 1.

Теорема 1. Минимальные и кратчайшие покрытия – *безизбыточны*.

Удобным и наглядным представлением исходных данных и их преобразований в задаче о покрытии является *таблица покрытий.* Таблица покрытий – это матрица T отношения принадлежности элементов множеств $A_i \in A$ опорному множеству B; столбцы матрицы T сопоставлены элементам b_j множества B, строки – элементам A_i множества A:

$$T = \begin{cases} 1, \text{если } b_j \in A_i \\ 0, \text{если } b_j \notin A_i \end{cases}. \qquad (2)$$

Нули в матрице T не проставляются.

Пример 1. Некий писатель выпустил множество сборников $A = \{A_1, \ldots, A_7\}$, содержащее в совокупности все множество $B = \{b_1, \ldots, b_9\}$, его сочинений (табл. 1). Каждый сборник A_i содержит некоторое подмножество сочинений из B ($A_i \subseteq B$) и имеет некоторую цену c_i. Необходимо найти такое множество $P = \{A_{k_1}, \ldots, A_{k_7}\}$, ($i_j \in \{1, \ldots, 7\}, l \le 7$) сборников, чтобы в их совокупности содержались все сочинения данного автора $\bigcup_{j=1}^{l} A_{k_j} = B$ и чтобы при этом либо цена $C = \sum_{j=1}^{l} c_{k_j}$ такого полного собрания сочинений была *наименьшей*, либо количество l сборников было *наименьшим.*

Таблица 1.

Матрица отношения принадлежности элементов множества $A_i \in A$ опорному множеству B

T	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	C
A_1							1	1	1	1.5
A_2	1	1		1				1		2
A_3			1							2
A_4				1	1	1	1			1
A_5	1				1			1	1	3
A_6		1				1				1
A_7			1		1	1	1		1	5

Возможные решения примера 2 – следующие покрытия:

$P_1 = \{A_2, A_7\}$, $A_2 \cup A_7 = B$; $C_1 = 2 + 5 = 7$; $l_1 = 2$ – кратчайшее покрытие;

$P_2 = \{A_1, A_3, A_4, A_6\}$, $A_1 \cup A_3 \cup A_4 \cup A_6 = B$; $C_2 = 1.5 + 2 + 1 + 1 = 5.5$; $l_2 = 4$ – минимальное покрытие;

$P_3 = \{A_3, A_4, A_5\}$, $A_3 \cup A_4 \cup A_5 = B$; $C_3 = 2 + 1 + 3 = 6$; $l_3 = 3$ – неоптимальное покрытие.

Табл. 1 может рассматриваться как таблица покрытия в сборочном производстве изделия, состоящего из комплектующих $b_1, \ldots, b_9$, которые могут быть частично поставлены разными производителями $A_1, \ldots, A_7$ с разной ценой c (ТП), чтобы в совокупности строк, входящих в P_i, во всех столбцах были 1 (единицы в некотором столбце могут повторяться). Такое подмножество строк ТП называется покрытием.

Имеются следующие варианты формулировки задачи о покрытии.

1. Требуется найти *все* покрытия. Для решения задачи необходимо выполнить полный перебор всех подмножеств множества A.

2. Требуется найти *только* безизбыточные покрытия. Не существует простого и эффективного алгоритма, не требующего построения всех избыточных покрытий; хорошо, если уменьшается их количество. Используется граничный перебор либо разложение по столбцу ТП.

3. Требуется найти *одно* безизбыточное покрытие. Решение задачи основано на сокращении ТП.

Задачи о покрытии могут быть решены *точно* (при небольшой размерности) либо *приближенно*.

Для нахождения точного решения используются следующие алгоритмы:

1. *Алгоритм полного перебора* основан на методе упорядочения перебора подмножеств множества *A*.

2. *Алгоритм граничного перебора по вогнутому множеству* основан на одноименном методе сокращения перебора.

3. *Алгоритм разложения по столбцу ТП* основан на методе сокращения перебора, который состоит в рассмотрении только тех строк ТП, в которых имеется 1 в выбранном для разложения столбце.

4. *Алгоритм сокращения ТП* основан на использовании теорем о свойствах ТП. Результатом работы алгоритма является множество ядерных строк и циклический остаток ТП, покрытие для которого далее строится методами граничного перебора либо разложения по столбцу.

Приближенное решение задачи о покрытии основано на следующем соображении. Даже сокращённый перебор приводит к очень трудоёмкому процессу решения, поэтому для получения результата приходится отказываться от гарантий построения оптимального решения (минимального либо кратчайшего); однако при этом целесообразно получить не самый худший результат – хотя бы безизбыточное покрытие, удовлетворяющее необходимому условию (1). Тогда, в ущерб качеству, можно значительно упростить процесс решения.

Алгоритм построения покрытия, близкого к кратчайшему, основан на методе минимального столбца – максимальной строки.

Ниже рассматриваются первые два из упомянутых алгоритмов точного решения задачи о покрытии.

Алгоритм полного перебора. Будем считать, что подмножества множества A пронумерованы $A_1,\ldots,A_m$ и таким образом упорядочены. Естественный на первый взгляд метод перебора *всех* подмножеств D строк таблицы покрытий строится так: пустое подмножество строк, подмножества из одной строки, из двух строк, …, из всех строк, т.е. всего 2^m подмножеств:

$$D = \{\varnothing, \{A_1\},\ldots,\{A_m\},\{A_1, A_m\},\ldots,\{A_1,\ldots,A_m\}\}.$$

Этот метод перебора громоздкий, поскольку требует большое число циклов. Более эффективный алгоритм требует всего 2 цикла.

Описание алгоритма перебора по вогнутому множеству.

Структуры данных:

T – таблица покрытия (ТП), содержащая n строк и m столбцов;

D – текущее подмножество (список), элементами которого являются номера строк ТП;

P, C – покрытие и список покрытий.

Словесное описание алгоритма полного перебора:

1. Вводим в текущее подмножество номер 1-й строки ТП ($D = \{A_1\}$), $i := 0$.

2. $i := i+1$. Реализуем модуль «Формирование признака p покрытия» (ФПП); если $p=1$ (имеет место покрытие), запоминаем множество D, как очередное покрытие P_i и вносим его в список покрытий.

3. Находим наибольший номер j элемента $A_j \in D$.

3.1. Если $j \neq n$ (n – количество подмножеств A_j), то переходим к п.4.

3.2. Если $j = n$, то удаляем A_n из D ($A_n \leftarrow D$) и если $D = \varnothing$, то заканчиваем построение подмножеств и переходим на п. 5.; если $D \neq \varnothing$, то находим наибольший номер j элемента $A_j \in D$ и удаляем A_j из D ($A_j \leftarrow D$).

4. $j := j+1$. Вводим в D элемент A_j ($A_j \rightarrow D$). Переходим к п.2.

5. Вывод на печать всех покрытий.

6. Конец.

В качестве элемента в этом алгоритме рассматривается номер строки ТП. По этому описанию построена схема алгоритма (рис.1а). Модуль ФПП описан далее.

Пример 2. Для $n=4$ определить список всех подмножеств, не выделяя покрытий. Имеем: $D_1 = \{A_1\}$, $D_2 = \{A_1, A_2\}$, $D_3 = \{A_1, A_2, A_3\}$, $D_4 = \{A_1, A_2, A_3, A_4\}$, $D_5 = \{A_1, A_2, A_4\}$, $D_6 = \{A_1, A_3\}$, $D_7 = \{A_1, A_3, A_4\}$, $D_8 = \{A_1, A_4\}$, $D_9 = \{A_2\}$, $D_{10} = \{A_2, A_3\}$, $D_{11} = \{A_2, A_3, A_4\}$, $D_{12} = \{A_2, A_4\}$, $D_{13} = \{A_3\}$, $D_{14} = \{A_3, A_4\}$, $D_{15} = \{A_4\}$.

Метод и алгоритм граничного перебора по вогнутому множеству. Алгоритм основан на методе генерации подмножеств и их целенаправленном отборе. Алгоритм генерации подмножеств должен гарантировать, что при его последовательном применении можно построить, начиная с некоторого начального подмножества, все возможные подмножества; при этом не должно быть повторений и должен существовать критерий окончания перебора. В алгоритме полного перебора упорядочение перебора произведено сначала по числу i элементов, а затем для фиксированного i – по лексикографическому правилу относительно номеров подмножеств.

Более удобный для дальнейшего упрощения процесса решения алгоритм полного перебора заключается в следующем: сначала строятся все подмножества D_i, содержащие A_1, затем – содержащие A_2, но не содержащие A_1; если построено подмножество D_i, то за ним строится подмножество D_{i+j}, целиком содержащее D_i ($D_i \subset D_{i+j}$).

Во многих приложениях достаточно находить только безизбыточные покрытия (теорема 1), что естественным образом сокращает перебор. Однако не удаётся найти простой и эффективный алгоритм, не требующий построения всех избыточных покрытий. Идея улучшения алгоритма перебора (генерации) подмножеств заключается в следующем: если текущее подмножество – покрытие, то в это подмножество не нужно вводить новые элементы.

Теорема 2. Если P покрытие, то и P', $P' \supseteq P$, тоже покрытие, т.е. множество всех возможных покрытий *вогнуто*.

Безизбыточные покрытия – это граница вогнутого множества всех покрытий, поэтому модифицированный алгоритм носит название *«Алгоритм граничного перебора по вогнутому множеству»*.

Описание алгоритма перебора по вогнутому множеству.

Структуры данных:

T – таблица покрытия (ТП), содержащая n строк и m столбцов;

D – текущее подмножество (список), элементами которого являются номера строк ТП;

P – покрытие (список номеров строк, дающих покрытие).

Относительно списка примем следующие соглашения:

– имеется специальный указатель **first** на первый элемент списка;

– связность элементов списка обеспечивается указателем i на следующий элемент $(x.i)$;

– имеется указатель **free** на первую свободную ячейку списка;

– для доступа к срединному элементу списка просматривается цепочка элементов от первого до искомого;

– операционная система регулярно осуществляет чистку списков от мусора (вычищает удалённые элементы) и проводит перенумерацию элементов.

Операции над списком L:

– **вставить** элемент x ($x \to L$);

– **удалить** элемент x ($x \leftarrow L$);

– **считать** значение элемента x ($x \leftrightarrow L$) с сохранением его в списке.

Отметим, что при вставке/удалении элемента x модифицируется указатель, а сам список упорядочивается (происходит перенумерация элементов).

Словесное описание алгоритма граничного перебора:

1. Текущее подмножество $D := \{A_1\}$, $i := 0$.

2. Выясняем, является ли D покрытием, для чего формируем признак p покрытия (ФПП). Если очередное построение в D – покрытие ($p = 1$), то упорядочиваем список покрытий (УСП).

3. Находим наибольший номер j элемента в подмножестве D.

3.1. Если $j \neq n$ и D – не покрытие ($p = 0$), то выполняем п.5.

3.2. Если $j \neq n$ и D – покрытие ($p = 1$), то выполняем п.4.

3.3. Если $j = n$, то удаляем A_n из D ($A_n \leftarrow D$), и если $D = \varnothing$, то заканчиваем построение всех безизбыточных покрытий и переходим на п. 6, иначе – снова находим наибольший номер j элемента в D.

4. Удаляем элемент A_j из D ($A_j \leftarrow D$).

5. $j := j + 1$. Вводим элемент A_j в D ($A_j \to D$) и выполняем п.2.

6. Выводим на печать список безизбыточных покрытий.

7. Конец.

Из полученных безизбыточных покрытий можно выбрать покрытия с минимальным количеством строк (кратчайшее покрытие) либо покрытие с минимальной ценой (минимальной покрытие) в соответствии с (2).

Пример 3. Задана таблица покрытий T (табл. 2), в которой $B = \{b_1, b_2, b_3, b_4, b_5\}$ – опорное множество, $A = \{A_1, A_2, A_3, A_4\}$ – множество подмножеств D множества B. Работа алгоритма иллюстрируется последовательностью построения подмножеств D:

$D = \{A_1\}$, $D = P_1 = \{A_1, A_2\}$, $D = \{A_1, A_3\}$, $D = P_2 = \{A_1, A_3, A_4\}$, $D = \{A_1, A_4\}$, $D = \{A_2\}$, $D = \{A_2, A_3\}$, $D = P_3 = \{A_2, A_3, A_4\}$, $D = \{A_2, A_4\}$, $D = \{A_3\}$, $D = P_4 = \{A_3, A_4\}$ - упорядочение списка покрытий: $P_2 \supset P_4$, $P_3 \supset P_4 \Rightarrow P_1 = \{A_1, A_2\}$, $P_2 = \{A_3, A_4\}$; $D = \{A_1\}$; $D = \varnothing$. P_1 и P_2 - кратчайшие покрытия.

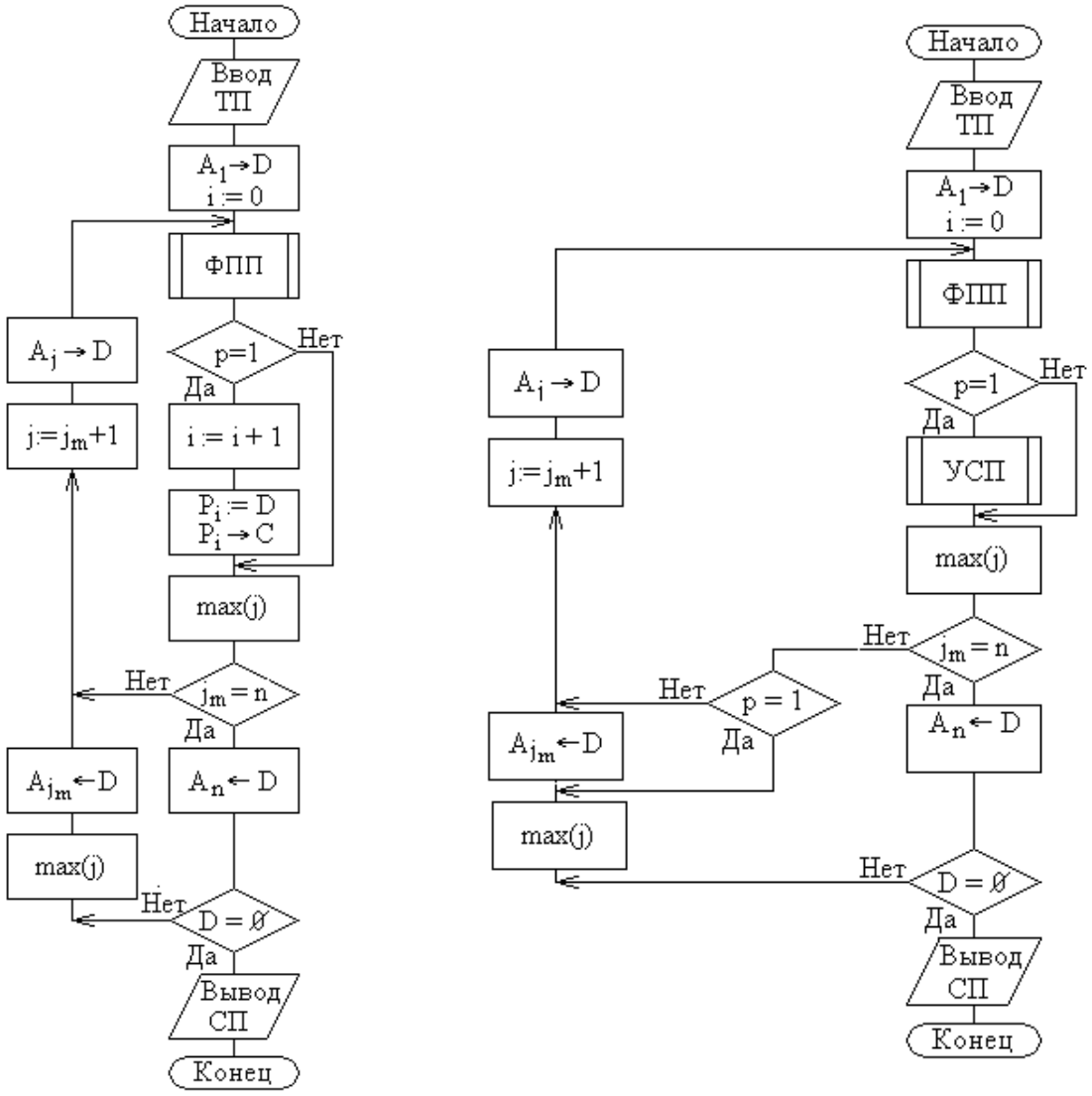
На рис. 1, б представлена схема алгоритма граничного перебора, построенная в соответствии с его словесным описанием.

Рассмотрим алгоритмические модули ФПП и УСП.

Таблица 2.

Матрица отношения принадлежности элементов множества $A_i \in A$ опорному множеству B

T	b_1	b_2	b_3	b_4	b_5
A_1	1		1		1
A_2	1	1		1	
A_3	1			1	1
A_4		1	1		



а б

Рис. 1. Схема алгоритма перебора: а – полного; б – граничного

Описание модуля ФПП. Сформулируем признак покрытия: $p = 1$, если в процессе перебора строк ТП, заданных подмножеством (списком) D, выяснится, что все 0-позиции покрыты. Покрытие может наступить до исчерпания строк, входящих в подмножество D.

Структуры данных:

T – таблица покрытия (ТП), содержащая *n* строк и *m* столбцов;

D – текущее подмножество (список) из *s* номеров строк ТП;

L – список из r позиций, на которых находятся нули (0-позиции);
V – двоичный вектор;
t – номер 0-позиции, $t = L[j]$;
q – номер строки в ТП.

Словесное описание вычислительного процесса в модуле ФПП.

1. Если в D всего одна строка (принимаем по умолчанию, что одна строка не образует покрытия), то переход на п. 8 (возврат значения признака $p = 0$).

2. Определяется номер первой в D строки ($q := D[1]$) и считывается сама строка A_q из ТП ($A_q \leftrightarrow T$). Она представляется в виде двоичного вектора (ДВ) V, ($V \div A_q$).

3. Просматриваются позиции V; номера 0-позиций заносятся в список L ($V[j] = 0 \Rightarrow j \to L$).

4. Определяется номер следующей строки из D и, соответственно, считывается сама строка, которая далее представляется в виде ДВ, $V \div A_q$.

5. Просматриваются в V все те позиции t, которые занесены в L; если в текущей позиции находится 1, то номер этой позиции удаляется из L ($t \leftarrow L$).

6. Если $L = \varnothing$, то $p := 1$ (D – покрытие); переход на п. 8;

7. Если не все строки в D обработаны, то переход на п.4, иначе $p := 0$.

8. Возврат значения p в основную программу.

Пример 4. Пусть $D = \{A_2, A_7, A_8\}$. Это означает, что D содержит номера 2, 7 и 8 строк из ТП. Вначале выбирается 2-я строка, в которой определяются номера 0-позиции; эти номера заносятся в список L. В последующих строках нас интересуют только номера 0-позиций и значения вектора на них. Если некоторые из этих значений равны 1, то соответствующие номера исключаются из списка L.

Пусть далее первая в D строка A_2 содержит 10 элементов с таким распределением значений: $A_2 = 1101001111$. Тогда $L = \{3, 5, 6\}$. Рассмотрим очередную строку со следующим распределением значений: $A_7 = 1000100111$. Для нахождения результирующего вектора (РВ) надо рассматривать в A_7 только позиции 3, 5, 6. Видно, что РВ принимает значение $V = 010$. Это означает, что номер пятый 0-позиции может быть исключён из L. Результат: $L = \{3, 6\}$.

На рис. 2, а представлена схема алгоритма вычислительного модуля ФПП, построенная по его словесному описанию.

Описание вычислительного модуля УСП. Идея упорядочения списка покрытий состоит в удалении по очереди из ранее построенных покрытий тех покрытий, которые поглощают (включают в себя) D, т.е. избыточных покрытий, при этом уменьшая i каждый раз на 1; запоминаем последнее D как покрытие P_i ($i := i + 1; P_i := D$).

Отметим, что возможны 4 варианта результата проверки на включение множеств A и B:

1. $A \subset B$.
2. $A \supset B$.
3. $A = B$.
4. A и B несравнимы. В данном случае возможны по построению подмножеств D только 1-й и 4-й варианты. Следовательно, достаточно убедиться в том, что все элементы B входят/не входят в множество A.

Примем по умолчанию, что множества A и B представлены списками и упорядочены по возрастанию.

Структуры данных:

C – список покрытий;

P^* – текущее покрытие;
i – номер покрытия, $i = 1 \ldots q$.

Исходные данные: список покрытий C, очередное покрытие P^*, не вошедшее ещё в список C, номер последнего в списке покрытия – q .

Словесное описание вычислительного процесса в модуле УСП.

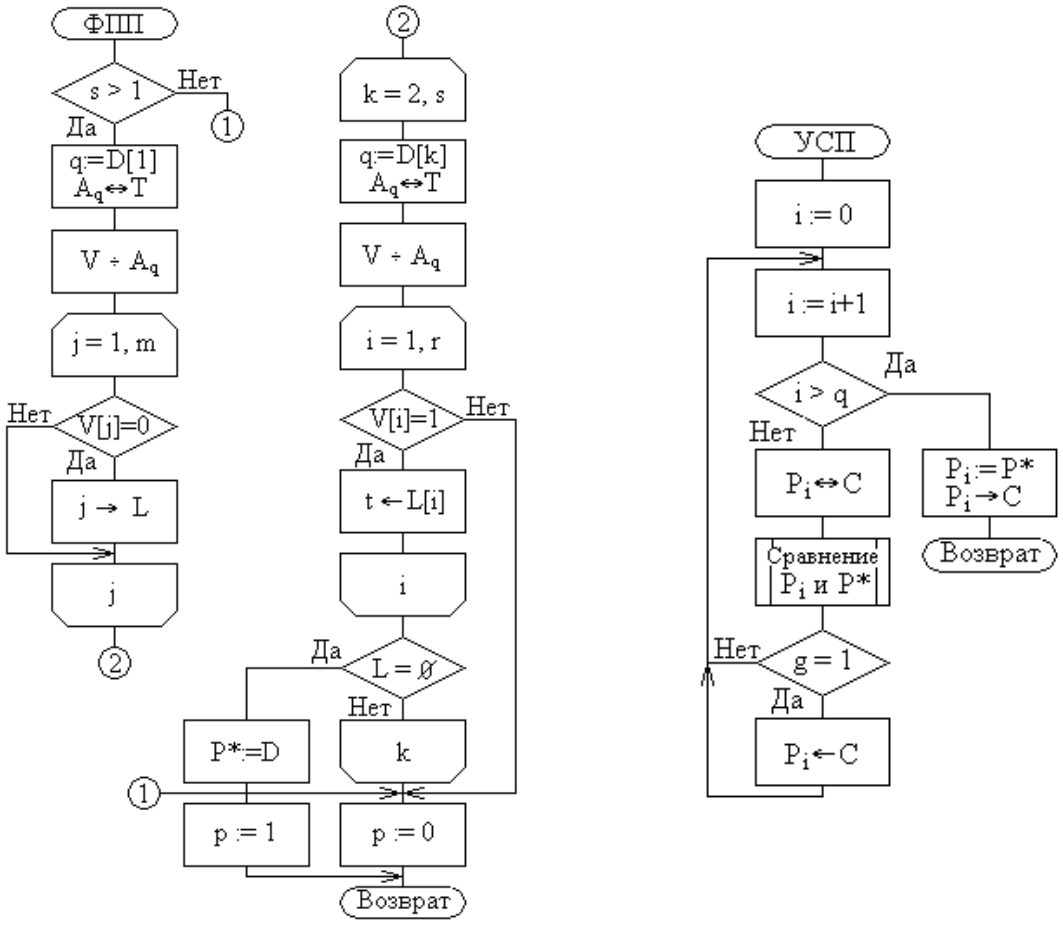
1. Организуем цикл по i перебора покрытий в списке C ($i := i + 1$). Если $i > q$, то переходим на п. 5.

2. Считываем очередное покрытие из C ($P_i \leftrightarrow C$).

3. Определяем, не является ли оно избыточным относительно P^* (сравнение покрытий P_i и P^* на избыточность – модуль СПИ). Если P_i не является избыточным ($g = 0$), то переходим на п.1.

4. Удаляем избыточное покрытие P_i из списка C ($P_i \leftarrow C$), переходим на п.1.

5. Процедура оканчивается и возвращается результирующий список покрытий C' .



а б

Рис. 2. Выделенные алгоритмические модули: а – ФПП; б – УСП

Описание вычислительного модуля СПИ. Переобозначим: $A := P_i$, $B := P^*$. Отметим, что количество элементов в P_i всегда больше, чем в P^*, что определяется методом построения подмножеств D; при этом число вариантов сравнения уменьшается до двух: избыточно/неизбыточно. Обозначим g признак избыточности. Тогда $g = 1$ означает избыточность P_i относительно P^*.

Идея решения состоит в исключении из обоих множеств их пересечения $A \cap B$; в случае $B = \varnothing$ это будет означать, что P_i избыточно относительно P^*.

Словесное описание алгоритма для модуля СПИ

1. Перебираем элементы y_j множества B.

2. Просматриваем элементы x_k множества A.

3. Если элементы x_k и y_j совпадают, то они удаляются из обоих множеств; переход к п.2. Иначе множества несравнимы ($g := 0$) – переход на п. 5.

4. Проводится модификация указателей множеств и перенумерация элементов (по умолчанию – автоматическая)

5. Если $B = \varnothing$, то $g := 1$, иначе переход на п. 1.

6. Процесс заканчивается; возвращается значение g.

Пример 5. Пусть $C = \{P_1, P_2, P_3, P_4\}$, очередное покрытие – P^*. Требуется упорядочить список покрытий C.

Поочерёдно сравниваем P_i с P^* ($i = 1 \ldots 4$). Пусть покрытия P_2 и P_3 – избыточны относительно покрытия P^*. Тогда P_2 и P_3 удаляются из списка C, а у оставшегося элемента списка (P_4) номер уменьшится на 2. Таким образом, результирующий список примет вид: $C' = \{P'_1, P'_2, P'_3\}$, где $P'_1 := P_1$, $P'_2 := P_2$, $P'_3 := P_3$.

Выделение макроопераций из рассмотренных ВП и ВМ

Анализ вычислительного процесса позволяет выделить из него макрооперации как функционально законченные объекты.

Сводный список МО в ВП решения задачи о покрытии. В результате просмотра словесных описаний основных алгоритмов и их вычислительных модулей были выявлены МО для каждого из них, а затем они были сведены в единый список:

1. Счётный цикл.
2. Условный цикл.
3. Увеличение значения счётчика на 1.
4. Проверка на истинность отношения.
5. Считывание строки таблицы покрытия и преобразование типа: строки в вектор.
6. Проверка множества (списка) на пустоту.
7. Считывание значения элемента множества (списка).
8. Вставка элемента в множество (список).
9. Удаление элемента из множества (списка).
10. Определение максимального номера элемента множества (списка).
11. Перенумерация элементов множества (списка).
12. Присвоение номера текущему подмножеству и его запоминание в виде элемента списка.

Проведём анализ сводного списка. Видно, что первые 3 МО определяют разные варианты повторения фрагмента ВП. Четвёртая МО является стандартной проверкой истинности предиката с отношениями разного типа (=, ≠, > … и т.д.). Пятая МО может быть организована поэлементно. МО 12 простая, но важная операция. МО 6-11 определяют работу со списком, причём МО 6 и 11 выполняются операционной системой, МО 8 и 9 – забота пользователя, МО 10 реализуется с использованием указателя **free** ($j_{\max} := \text{free} - 1$); МО 7, если это не предусмотрено в языке высокого

уровня, может быть выполнена в 3 шага: удаление элемента, присвоение его значения некоторой переменной, вставка в список.

Проведенный анализ позволил построить следующие обобщённые МО.

Обобщённая МО *«Формирование подмножества* (при $j_{тек} = n$)» включает в себя следующие операции:

- удаление последнего элемента из подмножества ($A_n \leftarrow D$);
- определение номера предпоследнего элемента (j_m);
- удаление элемента с номером j_m;
- определение номера нового элемента $j = j_m + 1$.

Обобщённая МО «Упорядочение списка покрытий» включает в себя следующие операции:

- сравнение покрытия P^* из таблицы покрытия с текущим покрытием P_i в списке;
- удаление избыточного покрытия;
- перенумерация элементов списка;
- введение в список текущего покрытия.

Выводы

Рассмотрены 2 метода и алгоритма перебора при решении задачи о покрытии: полный и граничный. Отметим, что полный перебор элементов некоторого множества пригоден не только для решения задачи о покрытии – он имеет широкий круг применения; при этом модуль ФПП заменяется на другой модуль со своим конкретным признаком отбора элементов с особыми свойствами. Метод граничного перебора используется при малом числе строк ТП и, в основном, в связке с методом сокращения ТП с помощью теорем, в результате применения которых получается т.н. циклический остаток (ЦО). Для дальнейшей обработки ЦО как раз и предназначен метод граничного перебора.

Для каждого алгоритма в ВП выделены МО, которые затем сведены в общий список с незначительными модификациями обобщающего плана.

Реализацию МО можно осуществить различными способами (различными ВП), отличающимися сложностью. Сравнение вариантов алгоритмов по сложности не проводилось – автор полагался на свой достаточно большой опыт проведения практических занятий и курсового проектирования по теории алгоритмов.

Отметим, что обобщенное представление программ в виде классов и методов скрывает количество реально выполняемых операторов и затрудняет анализ. Нами предлагается создание библиотеки отлаженных с помощью сетей Петри МО разного уровня сложности.

Список литературы

1. Рейнгольд, Э. Комбинаторные алгоритмы. Теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М.: Мир, 1980. – 480 с.
2. Кнут, Д. Искусство программирования. Т.3. – Сортировка и поиск / Д. Кнут. – М.: Изд. Дом «Вильямс», 2000. – 832 с.
3. Закревский, А.Д. Логический синтез каскадных схем / А.Д. Закревский. – М.: Наука, 1981. – 416 с.
4. Паулин, О.Н. Расширение принципа покрытия при построении универсальных логических модулей на основе симметрических функций [Текст] / О.Н. Паулин / Электротехнические и компьютерные системы. – 2013. – № 10 (86). – с. 105-110.
5. Паулин, О.Н. Основы теории симметрических булевых функцій / О.Н. Паулин – Саабрюкен (Германия): Lambert Academic Publisher, 2013. – 66 с.

6. Паулин, О.Н. О выделении макроопераций из вычислительных процессов сортировки массивов данных [Текст] / О.Н. Паулин, Н.О. Комлевая, С.Ю. Марулин // Проблеми програмування. – 2016. – № 2-3. – С. 87 – 95. (Scopus).
7. Паулин, О.Н. Представление вычислительных процессов сетями Петри [Текст] / О.Н. Паулин, Ю.С. Поляков, В.А. Шульгин // Научные труды SWorld. – 2015. – Вып. 4 (41), Т. 2. – С. 64-70.
8. Паулин, О.Н. Об управлении вычислительными процессами [Текст] / О.Н. Паулин, Н.О. Комлевая, С.Ю. Марулин // Труды XVII МНПК «Современные информационные и электронные технологии», Киев, 2016. – С. 20-21.
9. Паулин, О.Н. О функциональной полноте элементов управления вычислительными процессами [Текст] / О.Н. Паулин // Научные труды SWorld. – 2016. – Вып. 4, Том 4. – С. 4-8.
10. Yang, Y. Phase Distribution of Software Development Effort / Y. Yang, M. He, M. Li, Q. Wang, B. Boehm // International Symposium on Empirical Software Engineering and Measurement (ESEM), Kaiserslautern, Germany, 2008. – Pp. 384–392.

ОБЧИСЛЮВАЛЬНІ МОДЕЛІ АЛГОРИТМІВ ПОКРИТТЯ

О.Н. Паулін

Одеський національний політехнічний університет,
просп. Шевченка, 1, Одеса, 65044, Україна; e-mail: paolenic@yandex.ua

У статті ставиться і вирішується проблема побудови обчислювальних моделей для класу комбінаторних задач. Практично важливою в цьому класі є задача про покриття, що використовує переборний механізм. Такого роду задачі виникають, наприклад, при необхідності оптимального вибору постачальників при збиранні складного виробу. Обчислювальні процеси розв'язання задачі про покриття мають багато загальних функціонально закінчених компонент, названих нами макрооперaціями, які можуть бути виділені як обчислювальні моделі цього процесу. Таке виділення дозволить зібрати бібліотеку макрооперацій для різних класів задач, що спростить і прискорить аналіз програм ще на стадії побудови алгоритмів (обчислювальних процесів). Розглядаються 2 метода і відповідно 2 алгоритма розв'язання задачі про покриття: повного перебору підмножин і граничного перебору по увігнутій множині. Наводяться словесні описи алгоритмів, їх схеми, а також описи і схеми обчислювальних модулів. Виділяються макрооперації як обчислювальні моделі, які частково узагальнюються.
Ключові слова: комбінаторні задачі, обчислювальні процеси, обчислювальні моделі, макрооперації, алгоритми покриття, повний перебір, граничний перебір, обчислювальні модулі.

COMPUTATIONAL MODELS OF COVERAGE ALGORITHMS

O.N. Pauline

Odessa National Polytechnic University,
av. Shevchenko, 1, Odessa, 65044, Ukraine; e-mail: paolenic@yandex.ua

The article raises and solves the problem of building computational models for a class of combinatorial tasks. Important for practice in this class is the coverage task using mechanism of iterates through of the elements. Such tasks arise, for example, on necessity of optimal selection of suppliers for the assembly of complex products. The computational process of solving the task of the coverage have a lot in common functionally complete component called us macrooperation, which can be picked out as computational models of this process. This detachment will allow to build the library of macrooperations for different classes of tasks, which will simplify and speed up the analysis of programs at the stage of construction of algorithms (computational processes). Discusses 2 method and 2 respectively of algorithm for solving the task of covering: complete iterates through of the subsets and boundary iterates through in the concave set. Given the verbal descriptions of algorithms, their schemes, and descriptions and schemes of the computing modules. Pick out macrooperation as computational models, which are partially summarized.
Keywords: combinatorial tasks, computational processes, computational models, macrooperation, the algorithms of cover, full choice iterating through, boundary choice iterating through, compute modules.