Министерство образования и науки Украины

Севастопольский национальный технический университет

СТРУКТУРЫ ДАННЫХ И ФРАГМЕНТЫ ПРОГРАММНЫХ ПРОЦЕДУР РЕШЕНИЯ ЗАДАЧ ДИСКРЕТНОЙ МАТЕМАТИКИ

Методические указания
по КУРСОВОЙ РАБОТЕ
дисциплины
Дискретная математика

Для студентов дневной и заочной форм обучения

Севастополь, 2005

Цель курсовой работы - рассмотрение программных реализаций характерных изучаемых задач дисциплины *Дискретная математика. Особое внимание уделяется р*ассмотрению *форм представления данных в ЭВМ* и их влиянию на программные реализации процедур их обработки. Подсчитываются и сравниваются оценки сложности - число затрачиваемых операций при выполнении однотипных преобразований при различных формах представления данных и соответствующие *затраты памяти* представления данных. Более точно соответствующие *оценки быстродействия* можно получить, используя сами ПЭВМ, производя соответствующие замеры времени начала и конца преобразований. В рассматриваемых заданиях подчеркивается ориентация на *использовании битового представления* данных и применения *логических двоичных операций над словами ПЭВМ*, предоставляемых универсальными языками программирования и широко используемым в различных системах автоматизации программирования, обработки и поиска информации, проектирования. Таким образом формируются представления обучаемых о существенной зависимости быстродействия и объема внутренних форм описания данных от их структуры и средств обработки информации.

Студенты заочной формы обучения выполняют задания разделов 1 и 2; студенты дневной формы - все задания.

Методические указания составлены канд.физ.-матем.наук,
профессором Новоселовым В.Г.
Рецензент - канд. техн. наук, доцент Васильченко А.К.
Рассмотрены и утверждена на заседании кафедры КиВТ, протокол N 5 от 24 января 2005 года.
Зав. кафедрой _____ Скатков А.В.

Содержание

Общие замечания о программировании задач дискретной математики 3
1. Проведение комбинаторных построений 3
1.1. Преобразования над логическими векторами 4
1.2. Программная реализация построения сочетаний массивом характеристических векторов
Индивидуальные задания
13. Программная реализация построения сочетаний при их представлении двумерным массивом байт
Индивидуальные задания
2. Формы представления графов в программах на ЭВМ
2.1. Матрица отношений
Индивидуальные задания
2.2. Представление графа структурой
Индивидуальные задания
2.3. Структура без адресного массива
Индивидуальные задания
3. Преобразования интервальных форм булевых функций
Индивидуальные задания
4. Содержание и объем пояснительной записки к курсовой работе
Библиографический список

Общие замечания о программировании задач дискретной математики

Логические задачи во многом отличаются от задач вычислительной математики. Вычислительный аспект здесь выражается чаще всего не в формуле, а в алгоритмическом описании процесса решения задач. Решение обычно связано с перебором вариантов, сложной структурой представления данных, алгоритмы решения имеют разветвленную логическую структуру, объем перебора, время решения и требуемая память без специальных мер могут оказаться недопустимо большими.

При формулировании алгоритма для человека используются формы представления данных, удобные для восприятия человеком. В ЭВМ их необходимо заменить существенно другими (например, граф представить структурой, систему д.н.ф. представить в интервальной форме), удобными для реализации вычислительных процедур: формы данных должны обеспечить компактность записи этих процедур, сокращение времени их выполнения (отсутствие лишних циклов, замена поиска обращением по адресу). Выбор удачного представления данных при решении логических задач во многом определяет эффективность соответствующих программ, поиск такого представления требует интуиции, опыта, находится на основе рассмотрения нескольких вариантов. Конкретная реализация алгоритма и форм представления данных взаимосвязаны и их разработка осуществляется одновременно, окончательный выбор получается в результате нескольких итераций. Многие логические задачи по своей сути связаны с перебором вариантов, поэтому важно обеспечить быстродействие основных преобразований, проводящихся на внутренних циклах. Это иногда приводит к формированию нескольких форм представления одних и тех же данных, используемых на разных этапах алгоритма.

В очень многих программах встречаются логические фрагменты (организация перебора, формирование и использование масок, и т.д.). Преобразования логического характера составляют существенную часть программных систем автоматизации проектирования и программирования, оптимальное их решение во многом определяет эффективность реализации экспертных систем и программ распознавания образов. Логические преобразования и операции являются основой антивирусных программ.

Современные ПЭВМ предоставляют достаточно большие возможности для манипулирования затратами памяти и быстродействия. Затрачивая “лишнюю” память, удается повысить быстродействие или, наоборот, если задача требует слишком больших затрат памяти для хранения промежуточных результатов перебора, можно сэкономить память за счет временных затрат.

И, наконец, необходимо подчеркнуть не очень простой процесс отладки программ для логических задач. Необходимо уделить большое внимание разработке совокупности отладочных примеров, выбору последовательности их применения и обоснованию полноты контроля работы программы, например, по критерию С1.

1. Проведение комбинаторных построений

Ограничимся построением сочетаний - k элементных подмножеств n элементных множеств.

1.1. Преобразования над логическими векторами

В большинстве языков программирования высокого уровня наряду с операциями, предназначенными для выполнения арифметических вычислений, предусмотрены действия логического характера. Операции выполняются над n-разрядными двоичными векторами(*словами*), которыми удобнее всего представлять и интерпретировать целые

двоичные числа. Будем считать разряды *слова* пронумерованными справа налево, аналогично нумерации весов разрядов целого двоичного числа (n-1 ≥ i ≥ 0). Логические операции выполняются в словах поразрядно, над каждым разрядом слова параллельно и независимо. К логическим относятся операции логического умножения(AND) - /\(конъюнкции), логического сложения(OR) - V(дизъюнкции), исключительного ИЛИ(HOR) - ⊕(сложения по модулю 2), инверсии(NOT) - ¬(НЕ), сдвига влево(SHIFTL) - ⇐, сдвига вправо(SHIFTR) - ⇒ (с различными модификациями заполнения освобождающихся разрядов, РАССМАТРИВАЕТСЯ ЗАПОЛНЕНИЕ 0).

Определим все эти операции следующей таблицей:

x_1	x_2	$\neg x_1$	$\neg x_2$	$x_1 \wedge x_2$	$x_1 \vee x_2$	$x_1 \oplus x_2$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	0	1	0

В ЦВМ длина *n слова* различна, $n \in \{8, 16, 32, 64, ...\}$. В следующих примерах $n = 8$.

Выполним операции сдвига влево и вправо *слова* на один разряд с заполнением 0 освобождающегося разряда

a	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
a	0	1	0	1	1	1	0	0
$\Leftarrow 1$	1	0	1	1	1	0	0	0
$\Rightarrow 1$	0	0	1	0	1	1	1	0

Проиллюстрируем выполнение логических операций над *словами a* и *b*

a	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
a	0	1	0	1	1	1	0	0
b	0	1	1	0	0	1	1	0
$a \vee b$	0	1	1	1	1	1	1	0
$a \wedge b$	0	1	0	0	0	1	0	0
$a \oplus b$	0	0	1	1	1	0	1	0
$\neg a$	1	0	1	0	0	0	1	1

Слова могут интерпретироваться как целые числа и тогда в программе над ним можно выполнять целочисленные арифметические операции.

Наконец *слово* может быть интерпретировано как *характеристический* вектор *a:* если элемент a_i (n-1 ≥ i ≥ 0) принадлежит множеству A, то в i-ом разряде вектора *a* записывается 1, иначе - 0, Логические операции с характеристическими векторами можно интерпретировать как операции над множествами. Операция ∧ над характеристическими векторами эквивалентна пересечению, ∨ - объединению, инверсия ¬ - дополнению подмножества до *n* - элементного универсального (опорного) множества. Характеристический вектор, все элементы которого - нули, представляет *пустое* множество и одновременно такое слово представляет число нуль.

Желательно воспользоваться этими особенностями при программировании логических задач, тем более что большинство современных языков программирования (в том числе СИ и ПАСКАЛЬ) предоставляют соответствующие средства. Рассмотрим некоторые характерные простейшие преобразования и их реализации за счет логических операций над векторами.

Пересечение и объединение подмножеств

A⊆C, B⊆C, мощность $|C| \le n$, C - опорное(универсальное) множество.

C = { 7,6,5,4,3,2,1,0 } A = { 7,5,2,1 } B = { 7,6,4,2 }, пусть *A* и *B* - характеристические вектора соответствующих множеств.

C	c_7	c_6	c_5	C_4	c_3	c_2	c_1	c_0
A	1	0	1	0	0	1	1	0
B	1	1	0	1	0	1	0	0
$A \cup B$	1	1	1	1	0	1	1	0
$A \cap B$	1	0	0	0	0	1	0	0
$\neg A$	0	1	0	1	1	0	0	1

Проверка поглощения одного множества другим: A⊆B?. Обсудим эту простую задачу более подробно. По определению операции поглощения множество A поглощается множеством B, если каждый элемент, принадлежащий A, одновременно принадлежит B, т. е. все элементы A не принадлежат $\overline{B}$. Таким образом, проверка поглощения множества A множеством B сводится к проверке следующего соотношения - $\neg B \cap A = \varnothing$. На языке операций ЦВМ это выразится следующей последовательностью операций над соответствующими характеристическими векторами:

если ($\neg B \wedge A = 0$) **то**

Одновременное использование арифметических и логических операций. Например, осуществим проверку корректности некоторой двоичной матрицы D_{ij} размерности k x m при ее использовании в качестве таблицы покрытий. Каждый элемент опорного множества $C = \{c_0, ..., c_{m-1}\}$ должен быть покрыт, т.е. в каждом столбце матрицы D_{ij} должна быть хотя бы одна единица. Будем считать, что строки матрицы представляются *n*-разрядными словами и в случае m<n в словах заняты младшие m разрядов. Матрица D_{ij} представлена массивом D_j(j=1,..., k).

Характеристический вектор *c* опорного множества C может быть получен преобразованием: 2^m- 1. Величина 2^m представляется единственной единицей в m-ом разряде двоичного слова(*n=8,m=4)*

C	c_7	c_6	c_5	C_4	c_3	c_2	c_1	c_0
C	0	0	0	1	0	0	0	0
-1	0	0	0	0	0	0	0	1
C-1	0	0	0	0	1	1	1	1

и вектор *C* содержит значение 1 в *m* младших разрядах.

Проверка корректности таблицы покрытий сводится к циклу с дизъюнкцией *слов* массива D и сравнению результата с вектором C.

Формирование в слове Ц крайней справа 1, соответствующей элементу a_n.

Ц = 1 0...0

Ц $\Rightarrow$ *n-1*

Представление словом в крайней справа 1 в слове a

если($a \wedge$Ц = 0) **то**

$b = (a-1) \wedge a$

Пример

a	1	1	1	0	0	0	0	0	0
a-1	1	1	0	1	1	1	1	1	1
$b=(a-1) \wedge a$	1	1	0	0	0	0	0	0	0

Выделение в слове a крайней справа 1

$b = (a-1) \wedge a$

$c = a \oplus b$

Пример

a	1	1	1	0	0	0	0	0	0
$a-1$	1	1	0	1	1	1	1	1	1
$b=(a-1) \wedge a$	1	1	0	0	0	0	0	0	0
$c = b \oplus a$	0	0	1	0	0	0	0	0	0

Массив векторов с единственной 1.

Для оперирования с отдельными разрядами двоичных векторов полезно воспользоваться множеством $\{Ц_i = 2^i / 0 \le i \le n)\}$, каждый элемент $Ц_i$ - это двоичный вектор с единственной 1 в i-ом разряде, пусть $n=8$:

$Ц_i$	$ц_7$	$ц_6$	$ц_5$	$ц_4$	$ц_3$	$ц_2$	$ц_1$	$ц_0$
$Ц_0$	0	0	0	0	0	0	0	1
$Ц_1$	0	0	0	0	0	0	1	0
...				...				
$Ц_7$	1	0	0	0	0	0	0	0

Элемент $Ц_i$ позволяет либо посмотреть (проанализировать) значение i-го разряда двоичного вектора a (**если** ($a \wedge Ц_i = 0$ **то**), либо сформировать значение 1 в этом разряде ($a = a \vee Ц_i$), либо, наконец, сформировать значение 0 в этом разряде($a = \neg Ц_i \wedge a$). Так как массив Ц в дальнейшем будет использоваться неоднократно, то приведем алгоритм его построения.

$Ц_0 = 1$

Цикл по i **от** 1 **до** n

$Ц_i = Ц_{i-1} \Leftarrow 1)$

Можно использовать другую модификацию алгоритма.

$Ц_0 = 1$

Цикл по i **от** 1 **до** n

$Ц_i = 2 * Ц_{i-1}$

(ни в коем случае не $Ц_i = 2^i$) !!.

Ввод и вывод двоичных векторов.

В большинстве языков программирования ввод и вывод исходных данных можно осуществлять не только в десятичной, но и в других системах счисления, в частности, в двоичной. Однако, в ряде трансляторов ПАСКАЛ'я ввода и вывода данных в двоичной системе счисления нет, поэтому необходима разработка соответствующих процедур ввода и вывода двоичных векторов. Можно, конечно, перевести двоичное число в десятичное, особенно если число разрядов не велико. Например:

$11010111_2 = 256 - 32 - 8 - 1 = 215.$

Если возможен ввод чисел в 16-ной системе, то можно вручную свернуть двоичный вектор в 16-ное число. Например, $11010111_2 = D7_{16}$. Но такие представления не наглядны и поэтому могут привести к ошибкам.

Удобно воспользоваться промежуточным представлением вектора в виде массива символов (0 и 1).Для этого никакого перевода не требуется, достаточно при вводе двоичного вектора предусмотреть соответствующую интерпретацию каждого его разряда.

Символьный массив

A	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)
A	1	1	0	1	0	1	1	1

Перевод массива A в двоичный вектор a осуществляется следующим преобразованием.

a = 0

цикл по i от 0 до n

если A(I) = 1, то a =a V Ц(i).

Вывод двоичного вектора может быть выполнен аналогично с использованием символьного массива A:

цикл по i от 0 до n

если a Λ Ц(i) = 0,то A(i) = 0 , иначе A(i) = 1.

Далее выполняется вывод символьного массива A.

Если число разрядов n большое, а число единичных значений в каждом - мало, то можно сократить объем ввода, заменив символьный массив A на числовой массив номеров единичных разрядов каждого вектора.

Для нашего вектора a числовой массив A = {0, 1, 2, 4, 6, 7}.

Индивидуальные задания

1.1.1. Перевести число, представленное двумя ненулевыми цифрами, расположенными справа в номере Вашей зачетки, в двоичную систему счисления.

1.1.2. Составить процедуру на языке Паскаль удаления двух крайних справа 1 в слове a.

1.1.3. Составить отладочные примеры для этой процедуры, один из них - на основе слова задания 1.1.1

1.2. Программная реализация построения сочетаний массивом характеристических векторов

Подмножества достаточно наглядно представляются характеристическими векторами. Исходное n элементное множество $A = \{a_1,a_2,...,a_n\}$ называется опорным. Характеристический n компонентный двоичный вектор имеет единичными компоненты, сопоставленные элементам подмножества, остальные - нулевые. Начальное k элементное сочетание формируется и представляется характеристическим вектором - *словом* ПЭВМ, первые k компонент которого равны 1, остальные - нули:

1	2	...	*k-1*	*k*	*k+1*	...	*n*
1	1	...	1	1	0	...	0

Построение следующего сочетания можно выполнить преобразованием:

находится крайняя справа 1 в начальном или очередном векторе и сдвигается на один разряд вправо, остальные 1 остаются на своих местах.

№	1	2	...	*k-1*	*k*	*k+1*	...	*n*
1	1	1	...	1	1	0	...	0
2	1	1	...	1	0	1	...	0
...				...				

n-k 1 1 ... 1 0 0 ... 1

Возникла новая ситуация - 1 дошла до конечного элемента опорного множества. Нужно удалить все подряд идущие крайние справа 1, подсчитав их количество, затем снова найти крайнюю справа 1, сдвинуть ее вправо на один элемент и добавить подряд столько 1, сколько было удалено. Перебор закончен, когда все *k* единиц окажутся крайними справа.

Пример перебора сочетаний при $n = 6,\ k = 4$.

N	1	2	3	4	5	6
1	1	1	1	1		
2	1	1	1		1	
3	1	1	1			1
4	1	1		1	1	
5	1	1		1		1
6	1	1			1	1
7	1		1	1	1	
8	1		1	1		1
9	1		1		1	1
10	1			1	1	1
11		1	1	1	1	
12		1	1	1		1
13		1	1		1	1
14		1		1	1	1
15			1	1	1	1

$$\mathbf{C_6^4 = 15}$$

Фрагменты основных преобразований алгоритма с использованием логических операций над двоичными целыми числами без знака(AND - $\wedge$, OR - $\wedge$, HOR - $\oplus$, NOT - инверсия(замена 1 на 0 и 0 на 1), $\Rightarrow 1$ - сдвиг вправо на один разряд, $\Leftarrow 1$ - сдвиг влево на 1 разряд, все операции выполняются одновременно над всеми разрядами *слова*). Предполагаем, что $n \le 32$, такова разрядность слов современных ПЭВМ.

Формирование в слове Ц крайней справа 1, соответствующей элементу a_n.

Ц = 1 0...0

Ц $\Rightarrow$ *n-1*

Поиск и сдвиг вправо на один разряд крайней 1 в слове a

если($a \wedge$Ц = 0) **то**

выполнить

$b = (a-1) \wedge a$

$c = a \oplus в$

$d = c \Rightarrow 1$

$a = b \vee d$

конец

Пример при $n = 6,\ k = 3$

a	1	1	1	0	0	0
a-1	1	1	0	1	1	1
$b=(a-1)\wedge a$	1	1	0	0	0	0
$c = b \oplus a$	0	0	1	0	0	0
$d = c \Rightarrow 1$	0	0	0	1	0	0
$a = b \vee d$	1	1	0	1	0	0

Удаление конечных подряд идущих 1 в слове a, представляющем сочетание, с одновременным подсчетом их числа f

если($a \wedge$Ц = 1) **то**
выполнить
e = Ц
f = 0
$m = a$
цикл по $j = n$ **шаг** -1 **до** 1
выполнить
если($a \wedge e = 1$) **то**
выполнить
$f = f + 1$
$m = m \oplus e$
$e = e \Leftarrow 1$(*$\Leftarrow$ - сдвиг влево)*
конец
иначе (выйти из цикла)
конец
если(m = 0) **то(**заканчивается перебор сочетаний - все построены)
$a = m$
конец

Добавление в слово j следующих 1 за сдвинутой 1

a - слово с сдвинутой крайней 1,
c - слово с 1 крайней справа в *слове a*

цикл по i **от** 1 **до** j
выполнить
$c = c \Rightarrow 1$
$a = a \vee c$
конец

Индивидуальные задания

1.1.1. Написать процедуру на языке Паскаль построения массива слов, представляющих характеристические вектора сочетаний из n по k.

1.1.2. Подготовить отладочный пример массива сочетаний для Вашего варианта i(для заочников - значение последней цифры зачетки) по следующей таблице

i	n	k
0	5	3
1	8	6
2	6	2
3	6	3
4	7	2
5	7	3
6	7	4
7	7	5
8	8	2
9	5	2

Численно убедиться, что все варианты построены по формуле числа сочетаний

$$C_n^k = \frac{n!}{\text{----------}}$$

$$\frac{n!}{k!\,(n-k)!}$$

1.1.3. Провести оценку быстродействия процедуры, учитывая самый внутренний цикл, число его повторений и количество операций в нем.

1.2. Программная реализация построения сочетаний при их представлении двумерным массивом байт

Ограничив число *n* элементов массива (*n≤ 255)*, можно любой номер элемента представить одним байтом. Число элементов в каждом сочетании известно - поэтому все множество сочетаний можно представить двумерным байтовым массивом размерности (C_n^k, k).

Пример построения C_6^4:

N				
1	*1*	*2*	*3*	*4*
2	*1*	*2*	*3*	*5*
3	*1*	*2*	*3*	*6*
4	*1*	*2*	*4*	*5*
5	*1*	*2*	*4*	*6*
6	*1*	*2*	*5*	*6*
7	*1*	*3*	*4*	*5*
8	*1*	*3*	*4*	*6*
9	*1*	*3*	*5*	*6*
10	*1*	*4*	*5*	*6*
11	*2*	*3*	*4*	*5*
12	*2*	*3*	*4*	*6*
13	*2*	*3*	*5*	*6*
14	*2*	*4*	*5*	*6*
15	*3*	*4*	*5*	*6*

Сравнивая примеры двух реализаций построения сочетаний при различных формах их представления необходимо самим сформулировать алгоритм построения сочетаний для последнего варианта. Это сделать проще, так как алгоритм в последнем случае реализуется обычными в вычислительной практике операциями.

Индивидуальные задания

1.2.1. Описать алгоритм

1.2.2. Составить процедуру на Паскале реализации построения массива байт, представляющего сочетания.

1.2.3. Подготовить соответствующие примеры для Вашего варианта задания предыдущего раздела

1.2.4. Подсчитать оценку быстродействия процедуры и сравнить с оценкой предыдущей процедуры

2. Формы представления графов в программах на ЭВМ

2.1. Матрица отношений

По определению, граф - это отношение, поэтому матрица отношений используется для представления графов в ЭВМ. Действительно, наглядная для человека графическая форма представления совершенно не удобна для обработки программой.

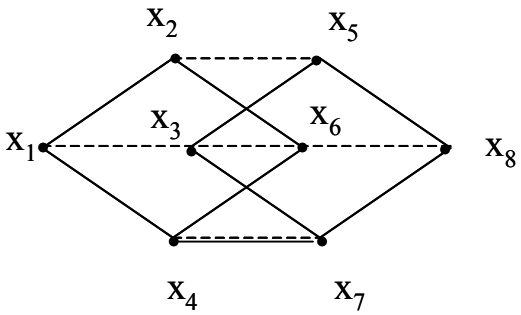

Рисунок 1 – Наглядное для человека графическое представление

Граф рисунка 1, рассматриваемый как направленный, все дуги которого ориентированы слева направо, представляется матрицей отношений:

ij	1	2	3	4	5	6	7	8
1		1	1	1				
2					1	1		
3					1	1	1	
4						1	1	
5								1
6								1
7								1
8								

Строки и столбцы матрицы отношений соответствуют вершинам графа. Единичное значение элемента матрицы отношений в *i*-ой строке и и *j*-ом столбце представляет дугу (i,j) графа. Тогда выход $G(x_3)$ вершины 3 определяется номерами ненулевых элементов 5, 6, 7 третьей строки этой матрицы. Для человека при небольшом числе вершин ЭВМ может представить граф рисунком на экране дисплея. Во внутреннем представлении графа матрицу отношений естественно представить двумерным массивом. Например при решении задачи построения кратчайшего пути необходимо для очередной выбранной вершины находить множество вершин ее выхода. Обработка рисунка - весьма сложная программная процедура, в то же время на матрице отношений такое преобразование элементарно: цикл по элементам строки матрицы и запоминание номеров не нулевых элементов строки.

Матрица отношений может не только представлять наличие дуги между вершинами графа, но и различные характеристики этой дуги, используемые в различных задачах, например длину дуги, ее пропускную способность или протекающий по данной дуге поток. Естественно различные свойства дуги представляются соответствующими элементами различных матриц отношений

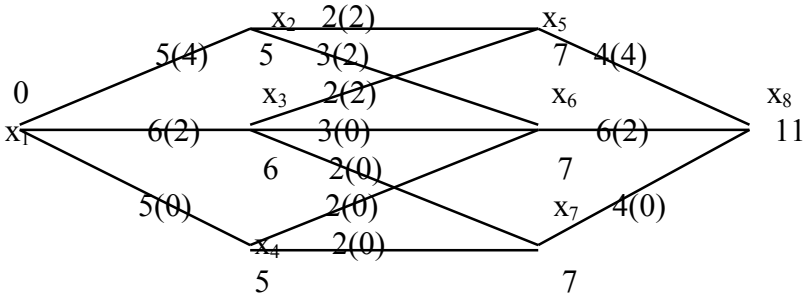

Рисунок 2 - Граф для задачи построения максимального потока в сети

Например граф рисунка 2 описывается двумя матрицами - C_{ij} и φ_{ij}:

C_{ij}	1	2	3	4	5	6	7	8
1		5	6	5				
2					2	3		
3					2	3	2	
4						2	2	
5								4
6								6
7								4
8								

φ_{ij}	1	2	3	4	5	6	7	8
1		4	2	0				
2					2	2		
3					2	0	0	
4						0	0	
5								4
6								2
7								0
8								

Индивидуальные задания к разделу

2.1.1. По матрице отношения несовместности вашего варианта задания контрольной работы 2 о максимальных совместимых подмножествах[2] построить соответствующие графы

2.1.2. Построить матрицы отношений $\boldsymbol{C_{ij}}$ и $\boldsymbol{\varphi_{ij}}$ для Вашего варианта графа задания при построении максимального потока[1] контрольной работы 3

2.1.3. Написать по матрице отношений графа с n вершинами фрагмент программы (процедуру) построения массива номеров вершин выхода для *i*-ой вершины

2.1.4. На подготовленном примере описания графа матрицы отношений $\boldsymbol{C_{ij}}$ выполнить подробное моделирование работы Вашего фрагмента программы для двух произвольно взятых вершин графа

2.1.5. Подсчитать количество *s* операций языка Паскаль в процедуре вычисления выхода одной вершины при заданном параметрах *n* - числе вершин графа и степени *k* выхода вершины

2.1.6. Подсчитать объем памяти для представления матрицы отношений графа с *n* вершинами двумерным массивом

2.2. Представление графа структурой

Представление графа матрицей отношений не всегда эффективно ни по объему памяти, ни по времени выполнения соответствующих преобразований. В алгоритмах построения кратчайшего пути и максимального потока поиск дуг, инцидентных вершине i, связан с просмотром всех элементов строки i матрицы отношений. Если степени вершин графа значительно меньше числа вершин, то матрица отношений состоит в основном из нулей. Более компактна модель представления графа множеством ребер или дуг, но при этом поиск каждой дуги связан с перебором всех дуг. Весьма эффективным является упорядочение дуг по начальной вершине с формированием адресного A и информационного И массивов. Элемент i адресного массива задает начало описания в информационном массиве подмножества дуг, начинающихся в вершине i. Элементы информационного массива содержат номера конечных вершин этих дуг. Представление графа числом n вершин (определяющим, в частности, мощность массива A, $|A| = n+1$) и

двумя массивами - адресным А и информационным И назовем *структурой*, понимая под этим понятием совокупность данных, описывающих некоторый объект. Отдельные компоненты структуры часто не имеют самостоятельного смысла и приобретают его только в составе структуры.

Число элементов адресного массива на 1 больше числа вершин графа. Каждый элемент i, i+1 адресного массива определяет адрес начала описания части информационного массива, содержащей номера вершин выхода для вершины x_i, разность пары элементов i+1 и i адресного массива определяет число элементов выхода вершины x_i, как это показано в таблице 1:

Таблица 1 - Описание структуры

Адресный массив А		Информационный массив И		Пояснения
N i элемента a_i и вершины графа x_i	Адрес начал подмассивов $G(x_i)$	N элемента массива И	Конечные вершины дуг для $G(x_i)$ - множества вершин выхода вершины x_i	
1	1	1	подмассив $G(x_1)$ выхода вершины x_1	Пусть *к* - степень вершины x_1
2	*к*	...	...	
3	*к+m*	*к*	конец подмассива $G(x_1)$	
4		*к*+1	начало подмассива $G(x_2)$	Пусть *m* - степень вершины x_2
...		...	...	
n+1		*к+m*	конец подмассива $G(x_2)$	

n - 8

Например, граф, представленный на рисунке 1, описан адресным (А) и информационным (И) массивами:

N A	N И	
1 1	1 2	$G(x_1)$
2 4	2 3	
3 6	3 4 /	
4 9	4 5	$G(x_2)$
5 11	5 6 /	
6 12	6 5	$G(x_3)$
7 13	7 6	
8 14	8 7 /	
9 14	9 6	$G(x_4)$
	10 7 /	
	11 8	$G(x_5)$
	12 8	$G(x_6)$
	13 8	$G(x_7)$

Параллельно адресному массиву можно создать массивы числовых характеристик вершин графа(например, массив весов $V(x_i)$ вершин в алгоритме поиска кратчайшего

пути), параллельно информационному - массивы числовых характеристик дуг(например массивы пропускных способностей дуг $C(xi\ xj)$ и потоков $\varphi(xi\ xj)$ по ним).

На рисунке 2 представлен граф, у которого каждой вершине x_i приписан вес $v(x_i)$, заданный числом при этой вершине, каждой дуге (x_i,x_j) сопоставлены две функции: $C(x_i,x_j)$, записанная на дуге вне скобок, и $\varphi(x_i,x_j)$, представленной числом в скобках. Все функции - целочисленные. Граф и все функции представлены следующими массивами.

N	А	*V(xi)*	*N*	И	*C*(*xi xj*)	φ (*xi xj*)
1	1	0	1	2	5	4
2	4	5	2	3	6	2
3	6	6	3	4	5	0
4	9	5	4	5	2	2
5	11	7	5	6	3	2
6	12	7	6	5	4	2
7	13	6	7	6	3	0
8	14	10	8	7	2	0
9	14		9	6	2	0
			10	7	1	0
			11	8	4	4
			12	8	6	2
			13	8	4	0

n - 8

Такое представление взвешенного графа тоже является структурой, и она составляет большую совокупность данных: (n,A,V,U,C,φ). Это представление называется *прямым* представлением дуг графа. Для некоторых преобразований удобно *обратное* представление: в информационном массиве указаны начала дуг, вершины которых инцидентны вершине xi. Для графа рисунке 1 обратное представление имеет следующий вид:

N	А	N	И	N	АДР
1	1	1	1	1	1
2	1	2	1	2	2
3	2	3	1	3	3
4	3	4	2	4	4
5	4	5	3	5	6
6	6	6	2	6	5
7	9	7	3	7	7
8	11	8	4	8	9
9	14	9	3	9	8
		10	4	10	10
		11	5	11	11
		12	6	12	12
		13	7	13	13

n - 8

Структуры прямого и обратного представления графа, введенные таким образом, полностью соответствуют эффективному представлению данных для алгоритмов построения кратчайших путей и максимального потока. Для вершины x_i вычисляется по значению i адрес $a = A_i$ начала записи в информационном массиве И списка номеров вершин выхода $G(x_i)$ вершины x_i и число $k=A_{i+1}-A_i$ этих вершин - $G(x_i) = \{И_a,И_{a+1},...,И_{a+к-1}\}$. Для каждой дуги $(x_i,И_j)$ можно получить любую интересующую нас характеристику - C_j или φ_j. Однако существуют задачи, для которых только этой информации недостаточно.

Индивидуальные задания к разделу

2.2.1. Построить структуру представления графа, заданного для задачи построения максимального потока в сети

2.2.2. Написать фрагмент программы (процедуру) построения по структуре графа с *n* вершинами массива номеров вершин входа для *i*-ой вершины

2.2.3. Повторить для тех же вершин моделирование работы фрагмента программы пункта 2.2.2

2.2.4. Подсчитать количество *s* операций в процедуре вычисления выхода одной вершины при заданном параметрах *n* - числе вершин графа и степени *k* выхода вершины

2.2.5. Подсчитать объем памяти для представления структуры графа с *n* вершинами адресным и информационным массивами

2.2.6. Написать фрагмент программы пересчета весов вершин выхода заданной *i*-ой вершины в задаче построения кратчайшего пути по структуре графа

2.2.7. Провести моделирование работы этого фрагмента программы для двух вершин графа по структуре, построенной в пункте 2.2.1

2.3. ***Структура без адресного массива***

Структура с использованием адресного и информационного массивов - является наиболее компактной и удобной формой представления для внутренних преобразований в различных задачах теории графов. Однако такая форма имеет существенный недостаток при модификации графа - введение новой дуги приводит к сдвигу информации в информационном массиве и пересчету адресов в адресном. Чтобы как то исправить положение изменяем представление структуры - информационный массив представляем двумерным: строки сопоставляем вершинам, столбцы - элементам множеств $G(x_i)$ их выходов. Размерность второго измерения двухмерного массива должна быть не меньше максимальной степени выходов вершин графа. Тогда введение большинства новых дуг вершин не изменяет структуру двухмерного массива, достаточно изменить значение одного элемента. И только если новая дуга изменяет максимальную степень выхода вершин графа, тогда нужно изменять описание размерности двумерного массива. Естественно, что адресный массив при таком представлении не нужен. Например, для графа рисунка 2 структура выглядит следующим образом:

n=8,k=3

$G(x_i)$	1	2	3
1	2	3	4
2	5	6	
3	5	6	7
4	6	7	
5	8		
6	8		
7	8		
8			

Аналогично могут быть представлены массивы любых характеристик этих дуг.

Индивидуальные задания раздела 2.3

2.3.1. Построить прямую и обратную структуры представления графа, заданного для задачи построения максимального потока в сети

2.3.2. Написать фрагмент программы (процедуру) построения по двухмерному массиву представления графа с n вершинами массива номеров вершин выхода для *i*-ой вершины

2.3.3. Повторить для тех же вершин, что и в пункте 2.2.2 моделирование работы фрагмента программы пункта 2.3.2

2.3.4. Подсчитать количество *s* операций в процедуре вычисления выхода одной вершины при заданном параметрах *n* - числе вершин графа и степени *k* выхода вершины

2.3.5. Подсчитать объем памяти для представления структуры графа с *n* вершинами двухмерным массивом

2.3.6. По двухмерным массивам прямого и обратного представления графа с n вершинами написать фрагмент программы (процедуру) для *i*-ой вершины разметки вершин входа и выхода согласно алгоритму Форда-Фалкерсона

2.3.7. Провести моделирование работы этой процедуры для трех лежащих на одном пути вершин графа, последовательно активизирующих друг лруга в алгоритме Форда-Фалкерсона

3. Операции над интервальными формами

Дизъюнктивные нормальные формы булевых функций в ЭВМ представляются множеством интервалов. При записи интервала для значений каждой переменной используется троичная система {1,0,-}. Для кодирования этих значений удобно представить их двумя *словами*, обозначим их через *а* и *в*. Переменная x_i сопоставляется одноименным разрядам a_i и b_i этих векторов. Для определенности разряды *слов* будем нумеровать справа налево, что соответствует уже принятой нами нумерации элементов массива Ц и правилам формирования их значений. Двумя битами a_i и $в_i$ можно кодировать четыре значения, в записи переменной в интервале - три значения. В литературе используются несколько вариантов кодирования, например, Закревский А.Д. вектором *а* представляет минимальный элемент интервала, вектором *в* - максимальный. В данной работе будем использовать другое тоже часто применяемое кодирование, которое, как нам кажется, приводит к более понятным выражениям в применяемых преобразованиях. Вектор *в* является характеристическим для внешних переменных интервала, вектор *а* - минимальным набором интервала, т. е. в векторе *в* единицами обозначены те переменные, которые равны 1 или 0 в наборе интервала, в векторе *а* указаны значения этих переменных, и все внутренние переменные приравнены нулю.

Например, конъюнкция $\bar{x}_5 x_3 \bar{x}_1$:

	x_5	x_4	x_3	x_2	x_1	x_1
u	0	-	1	-	0	-
а	0	0	1	0	0	0
в	1	0	1	0	1	0

Множество интервалов (д.н.ф.) представляется двумя массивами А и В, соответственно.

Рассмотрим основные операции над интервалами.

Проверка пересечения интервалов. Пусть u_1 представлен наборами a_1, $в_1$; интервал u_2 - a_2, $в_2$, соответственно.

Основу алгоритма проверки интервалов на пересечение является следующее свойство - общие внешние переменные не ортогональны (т.е. равны).

$c = в_1 \Lambda в_2$

если $((a_1 \oplus a_2) \Lambda c = 0)$ **то** (интервалы пересекаются)

при естественном порядке выполнения операции интервалы не пересекаются.

Поглощение интервала u_2 интервалом u_1.

Основой алгоритма является определение операции поглощения - все внешние переменные интервала u_1 входят в множество внешних переменных интервала u_2 и имеют каждая одинаковые значения в обоих интервалах.

если $(\neg в_2 \Lambda в_1 = 0)$ **то**

выполнить

$c = в_1 \Lambda в_2$

если $((a_1 \oplus a_2) \Lambda c = 0)$ **то (** интервал u_2 поглощается интервалом u_1)

конец

интервал u_2 не поглощается. интервалом u_1

Склеивание интервалов

Как и в предыдущем случае, опираемся на определение операции склеивания - множества внешних переменных должны совпадать, их значения - различаться для одной и только одной переменной.

если $(в_1 \oplus в_2 = 0)$ **то**

выполнить

$c = a_1 \oplus a_2$ далее необходимо провести анализ переменной *c* на единственность 1 и в случае положительного ответа сформировать результат склеивания, иначе операцию склеивания провести нельзя.

конец

Проверка единственности 1 в слове

Можно предложить несколько вариантов решения этой задачи. В цикле по i анализируем с помощью массива Ц значения разрядов вектора *c*, прервем цикл при первом же $a \wedge Ц_I = 1$ и затем выполнить $\neg Ц_i \wedge a$; если получится 0...0, то 1 была единственной, иначе - не единственной.

Но лучше обойтись вообще без цикла, выполнив следующую последовательность вычислений:

в = *а* - 1, **если** (*в* отрицательно), **то (**в *а* вообще нет 1), **иначе**

$в = a \wedge в$ (убирается крайняя справа в слове *а* единица)

если(*в* = 0) **то** (1 была единственной в *а)* **иначе** - не единственной.

Например:

a	0	0	1	0	0	0
a-1	0	0	0	1	1	1
$b=(a-1)\wedge a$	0	0	0	0	0	0

вывод - единица в *a* была единственной.

Рассмотрим пример со словом *a* с тремя единичными разрядами.

a	1	1	1	0	0	0
a-1	1	1	0	1	1	1
$b=(a-1)\wedge a$	1	1	0	0	0	0

вывод - число единиц в *a* больше одной.

Подсчет числа единиц в слове. Можно воспользоваться массивом Ц и в цикле, используя операцию **если (***a* Λ $Ц_i = 1$), подсчитать число единиц в слове *a*. Каждый такой подсчет занимает *n* выполнений цикла. Можно сократить число циклов до числа единиц в слове *a*, используя операцию вычитания в следующей последовательности:

в= 0

M1: *c* = *a* - 1 , **если**(результат отрицателен) **то (***в* представляет число 1 в *a)* **иначе** (*в* = *в* + 1, *a*= *a* Λ *c* и осуществляется переход к метке M1)

Пример.

в = 0 *a* = 1 0 0 1 1 0 0 0

c = *a* - 1 = 1 0 0 1 0 1 1 1≥ 0, *в* = 0 + 1 = 1, *a* = *a* Λ *c* = 1 0 0 1 0 0 0 0

c = *a* - 1 = 1 0 0 0 1 1 1 1≥ 0, *в* = 1 + 1 = 2, *a* = *a* Λ *c* = 1 0 0 0 0 0 0 0

c = *a* - 1 = 0 1 1 1 1 1 1 1≥ 0, *в* = 2 + 1 = 3, *a* = *a* Λ *c* = 0 0 0 0 0 0 0 0

c = *a* - 1 ≤ 0. Подсчет закончен, *в* = *3*.

Неполное склеивание интервалов

d = $в_2$ $в_1$ Λ ¬$в_2$ o→ M1 d = $в_1$ ¬$в_1$ Λ $в_2$ o→ M1(обозначение o→ M1 означает анализ текущего значения результата предыдущей операции на 0 и при положительном результате анализа переход к метке M1, иначе сохранение естественной последовательности операций.

M1 : *c* = $в_1$ Λ $в_2$ *c* = $a_1 \oplus a_2 \Lambda c$ далее необходимо провести анализ слова *c* на единственность 1 и в случае положительного ответа сформировать результат склеивания. В зависимости от значения переменной d далее преобразуется либо интервал a_1,$в_1$, если d = $в_1$, иначе - интервал a_2 ,$в_2$.

Например, для второго интервала это построение выглядит так:

$в_2$ = ¬ c Λ $в_2$ a_2 = ¬ c Λ a_2.

Например, $x_1 \bar{x}_3 x_4 \vee \bar{x}_1 \bar{x}_3 x_4 x_5 = x_1 \bar{x}_3 x_4 \vee \bar{x}_3 x_4 x_5$

x_5 x_4 x_3 x_2 x_1 x_5 x_4 x_3 x_2 x_1

a_1 = 0 1 0 0 1 a_2 = 1 1 0 0 0

$в_1 = 0\ 1\ 1\ 0\ 1$ $в_2 = 1\ 1\ 1\ 0\ 1$

$d = в_2$ $c = в_1 \Lambda в_2 = 0\ 1\ 1\ 0\ 1$ $\neg в_2 = 0\ 0\ 0\ 1\ 0$

$e = a_1 \oplus a_2 \Lambda c = 0\ 0\ 0\ 0\ 1$, 1 одна и $d = в_2$, поэтому преобразуется второй интервал

$a_2 = \neg\, c \Lambda a_2 = 1\ 1\ 0\ 0\ 0$, $в_2 = \neg\, c \Lambda в_2 = 1\ 1\ 1\ 0\ 0$.

Расчет расстояния между пересекающимися интервалами (расстояние 1 - O_1, расстояние больше 1 - O_2)

$c = в_1 \Lambda в_2$ *в* $= (a_1 \oplus a_2) \Lambda$ с далее осуществляется для *в* проверка единственности 1, если эта проверка дает удовлетворительный результат, то O_1, иначе O_2.

Выполнение операции вычитания (#) интервалов. Операция вычитания достаточно подробно описана в [21], однако в связи с ее необычностью рассмотрим еще раз подробно алгоритм вычитания (#) с представлением разности непересекающимися интервалами:

$u_1 \# u_2 =$ { совокупность ортогональных интервалов разности }

1) Проверяется ортогональность интервалов уменьшаемого и вычитаемого, в случае ортогональности интервалов разность равна уменьшаемому. Проверка ортогональности состоит в анализе значений общих внешних переменных интервалов u_1 и u_2, если значения хотя бы одной из них различаются - интервалы ортогональны, иначе - нет.

$c = в_1 \Lambda в_2$ **если (**$(a_1 \oplus a_2) \Lambda c) \neq 0$ **то(** интервалы ортогональны)

M2 : интервалы не ортогональны.

2) Проверяется поглощение интервала u_1 интервалом u_2 , в положительном случае разность пуста. Алгоритм уже приведен при описании операции поглощения интервалов.

3) Если пункты 1 и 2 алгоритма дают отрицательный ответ, то необходимо проводить не тривиальное выполнение операции вычитания, которое выполняется в два этапа - сначала вычисляется число *k* интервалов разности, которое равно числу внутренних переменных уменьшаемого, являющихся внешними в вычитаемом. Затем строятся сами интервалы разности.

Подсчет числа k.

$c = d = \neg\, в_1 \Lambda в_2$ $k = 0$

M1: c o$\rightarrow$ M2 $c = (c - 1) \Lambda c$ $k = k + 1 \rightarrow$ M1

M2 : Подсчет *к* закончен

Построение k ортогональных интервалов разности.

В разделе 5.1 при проверке единственности единицы в двоичном слове *c*, уже использовалась операция вычитания 1 из слова , как способ выделения крайней правой 1 в слове *c* (формирования слова *в*, содержащего одну эту 1):

$в = (c - 1) \oplus c \Lambda c$.

Множество интервалов разности представим в двух массивах MA и MB, хранящих элементы *a* и *в* каждого интервала. Воспользуемся значением переменной *d*, уже вычисленным характеристическим вектором переменных внутренних для уменьшаемого интервала и внешних для вычитаемого.

Цикл по i от 1 до *k*

$c = (d - 1) \oplus d \Lambda d \quad d = d \oplus c$

$MA_i = \neg\, a_2 \Lambda c \vee a_1 \quad Mb_i = в_1 = в_1 \vee c \quad a_1 = a_2 \Lambda c \vee a_1$

Индивидуальное задание к разделу

3.1. Составить процедуры подсчета числа 1 в слове раздела 1.1.1

3.2. Провести решение по программе для подсчитанного в 1.1.1 двоичного числа

3.3. Составить процедуру выполнения операций

а) склеивания для вариантов $i \in \{1,4,7,10,13, 16,19,22,25\}$

б) поглощения для вариантов $i \in \{2,5,8,11,14, 17,20,23,26\}$

в) неполного склеивания для вариантов $i \in \{3,6,9,12, 15,18,21,24\}$

3.4. Подготовить отладочные примеры для соответствующих процедур

4. Содержание и объем пояснительной записки к курсовой работе

Пояснительная записка к курсовой работе оформляется по ГОСТу оформления технической документации на бумаге формата А4, с титульным листом. На титульном листе указываются данные Исполнителя - Выполнил студент группы м - 1хз Ф.И.О. и подпись. Проверил - данные о преподавателе, Ф.И.О.

№	Наименование раздела	Приблизительный объем в страницах
	Введение	0.5
1	Построение множества сочетаний на основе логических операций над словами ПЭВМ	3
2	Построение множества сочетаний на основе арифметических операций	3
3	Сравнение двух подходов	0.5
4	Представление графа матрицей отношений, процедура построения множества вершин выхода, расчет времени решения и объема памяти	2-4
5	Представление графа структурой, процедура построения множества вершин выхода, расчет времени решения и объема памяти	3-4
6	Представление графа двухмерным массивом, процедура построения множества вершин выхода, расчет времени решения и объема памяти	3-4

7	Сравнение форм представления графа	1
8	Описание процедуры подсчета числа 1 в слове и отладочного примера	2
9	Описание процедуры выполнения операции и отладочного примера	2
	Заключение	0.5
	Библиографический список	0.2

Библиографический список

Для студентов-заочников допускается получение в библиотеке методических указаний [1,2] в библиотеке - указаны номера(№229 и 229а) указаний - или соответствующих файлов в аудитории Г208.

1. Новоселов В.Г. Методические указания Прикладная математика для системотехников. Дискретная математика в задачах и примерах. Часть 2. Задачи оптимизации на конечных графах / В.Г. Новоселов -Севастополь: изд. СевНТУ, 1991. -70 с.(№229)
2. Новоселов В.Г. Методические указания Прикладная математика для системотехников. Дискретная математика в задачах и примерах. Часть 1. Задачи оптимизации на конечных множествах / В.Г. Новоселов -Севастополь: изд. СевНТУ, 1991. -68с.(229а)
3. Новоселов В.Г. Прикладная математика для системотехников. Дискретная математика в задачах и примерах / В.Г. Новоселов, А.В. Скатков - Киев: УМК ВО, 1992.-200с.
4. Новоселов В.Г. Дискретная математика в задачах и примерах. Часть 2. /Учебное пособие./ В.Г. Новоселов, А.В. Скатков -Севастополь: изд. СевНТУ, 2002. -124 с.