ТАРТУСКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

ТРУДЫ

ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА

35

ТАРТУ
1976

ТАРТУСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

ТРУДЫ ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА

ВЫПУСК 35

ТАРТУ 1976

РЕШЕНИЕ ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ С ФИКСИРОВАННЫМИ ДОПЛАТАМИ

Л.А.Кивистик, М.Й.Ури

§1. Постановка задачи

Часто (напр., при планировании новых производственных комплексов) необходимо решать задачу математического программирования следующего вида: минимизировать функцию

$$f(X) = \sum_{j=1}^{n} c_j(x_j), \tag{1.1}$$

где

$$c_j(x_j) = \begin{cases} 0, & \text{если } x_j = 0, \\ c_j x_j + d_j, & \text{если } x_j > 0, \end{cases} \tag{1.2}$$

при условиях

$$\sum_{j=1}^{n} a_{kj} x_j = b_k, \quad k=1,2,\ldots,m, \tag{1.3}$$

$$x_j \geqslant 0, \quad j=1,2,\ldots,n, \tag{1.4}$$

$$x_j \leqslant \beta_j, \quad j \in P, \tag{1.5}$$

где $P = \{ j : d_j > 0 \}$.

Предполагаем, что $c_j \geqslant 0$ при всех j и $d_j = 0$, если $j \notin P$.

Положительная величина d_j называется фиксированной до-

платой, а задача (1.1)-(1.5) задачей линейного программирования с фиксированными доплатами.

Задачу (1.1)-(1.5) можно свести к частично целочисленной задаче лчнейного программирования путем введения дополнительнх булевых переменнх y_j, определяемых при $j \in P$ следующим образом:

$$y_j = \begin{cases} 0, & \text{если } x_j = 0, \\ 1, & \text{если } x_j > 0. \end{cases} \qquad (1.6)$$

Тогда вместо функции $f(X)$, определяемой формулой (1.1), можно минимизировать функцию:

$$\bar{f}(X,Y) = \sum_{j=1}^{n} c_j x_j + \sum_{j \in P} d_j y_j .$$

Как легко показать (см. напр. [1], стр. 53), минимизация функции $\bar{f}(X,Y)$ при логических условиях (1.6) и условиях (1.5) эквивалентна минимизации этой функции при условиях

$$y_j = \begin{cases} 0 \\ 1 \end{cases}, \; x_j \leqslant \beta_j y_j, \quad \text{если} \quad j \in P.$$

Итак, мы получим новый вид задачи: минимизировать

$$\bar{f}(X,Y) = \sum_{j=1}^{n} c_j x_j + \sum_{j \in P} d_j y_j \qquad (1.7)$$

при условиях

$$\sum_{j=1}^{n} a_{kj} x_j = b_k, \quad k=1,2,\ldots,m, \qquad (1.8)$$

$$x_j \geqslant 0 , \quad j=1,2,\ldots,n, \qquad (1.9)$$

$$x_j \leqslant \beta_j y_j, \quad j \in P, \qquad (1.10)$$

$$0 \leqslant y_j \leqslant 1, \quad j \in P, \qquad (1.11)$$

$$y_j \text{ - целые}, \quad j \in P. \qquad (1.12)$$

Задачи (1.1)-(1.5) и (1.7)-(1.12) эквивалентны в том смысле, что минимальные значения их целевых функций совпадают и у них существуют оптимальные решения с одинаковыми значениями компонент $x_1, x_2, \ldots, x_n$.

Из-за разрывности целевой функции трудно найти эффективные точные методы для решения поставленной задачи в форме (1.1)-(1.5). Поэтому, многие авторы изучают приближенное решение этой задачи, особенно частного случая - транспортной задачи с фиксированными доплатами ([4]; [1], стр. 300-313; в [1] можно найти дополнительную литературу).

Рассматриваемую задачу в форме (1.7)-(1.12) можно решать методами целочисленного линейного программирования; например, методом отсечений Гомори. Но, в случае больших m и n и многих положительных d_j, большие размеры этой задачи превращают ее практическое решение трудной, даже при использовании ЭВМ ([3], стр. 256). Более перспективными окажутся комбинаторные методы, которые позволяют избавляться от дополнительных переменных y_j. Такие методы предлагаются в [5] и [6]. В настоящей заметке для решения рассматриваемой задачи применяется вариант метода ветвей и границ. При этом исходной является задача в форме (1.7)-(1.12), но применение метода требует решение ряда задач лишь с ограничениями (1.3)-(1.5) и линейными, отличными друг от друга целевыми функциями. От метода Грея [5] предлогаемый метод отличается правилами ветвления и способом оценки целевой функции.

§2. Решение поставленной задачи методом ветвей и границ

Рассмотрим решение задачи (1.7)-(1.12) методом ветвей и границ. При этом используем обозначения и схему метода, аналогичные обозначениям и общей схеме из [1], стр. 213-218, конкретизируя правило ветвления, метод нахождения оценок и допустимых решений и дополняя схему правилом заграждения ветвления.

Сделаем некоторые предварительные замечания.

Обозначим через G_0 множество, заданное условиями (1.8)-(1.12). В процессе решения задачи это множество определенным способом разбивается на конечное число подмножеств. На каждом таком подмножестве G_i найдем нижнюю границу (оценку) целевой функции, т.е. такое число $z(G_i)$, что для любого $(X,Y) \in G_i$ имеет место неравенство $\bar{f}(X,Y) \geqslant z(G_i)$. Если G_i пустое множество, берем $z(G_i) = +\infty$.

Оценка $z(G_0)$ получается как значение целевой функции (1.7) при оптимальном решении нецелочисленной задачи (1.7)-(1.11), а оценка $z(G_i)$ $(i > 0)$ - как значение функции (1.7) при оптимальном решении той же самой задачи с дополнительными условиями типа $y_r = 0$, $y_s = 1$. Как мы покажем ниже, последние задачи можно заменить более простыми, где переменные y_j отсутствуют. На вопросе определения оценок $z(G_i)$ будем останавливаться более подробно в следующем параграфе.

Для установления оптимальности допустимого решения используем следующий простой п р и з н а к о п т и м а л ь н о с т и ([1], стр. 216).

Пусть

$$G_0 = \bigcup_{i=1}^{s} G_i$$

и допустимое решение (X^*,Y^*) принадлежит некоторому подмножеству G_ν, $\nu \in \{1,2,\ldots,s\}$. Если

$$\bar{f}(X^*,Y^*) = \zeta(G_\nu) \leqslant \zeta(G_i), \quad i=1,2,\ldots,s,$$

то (X^*,Y^*) – оптимальное решение задачи (1.7)–(1.12).

Изложим общую схему алгоритма.

0 – й ш а г. Вычислим оценку $\zeta(G_0)$. Если $\zeta(G_0) = +\infty$, то задача не имеет допустимых решений. Если $\zeta(G_0) < +\infty$, то найдем допустимое решение $(X_0,Y_0) \in G_0$ и вычислим $\bar{f}(X_0,Y_0)$. Если

$$\bar{f}(X_0,Y_0) = \zeta(G_0), \tag{2.1}$$

то (X_0,Y_0) – оптимальное решение задачи (1.7)–(1.12), а X_0 – оптимальное решение задачи (1.1)–(1.5). Если равенство (2.1) не имеет место, включим G_0 в список S рассматриваемых подмножеств, присвоим переменной z значение $\bar{f}(X_0,Y_0)$ и перейдем к 1-му шагу.

k – й ш а г $(k \geqslant 1)$. 1. Из списка S исключим н а и б о л е е п е р с п е к т и в н о е множество G_{p_k} по условию

$$\zeta(G_{p_k}) = \min_{G_i \in S} \zeta(G_i). \tag{2.2}$$

Множество G_{p_k} разобьем на два подмножества, G_{2k-1} и G_{2k} (ветвление). При этом G_{2k-1} получается из G_{p_k} введением дополнительного условия $y_q = 0$ и G_{2k} – введением условия $y_q = 1$, где y_q – одна из переменных, получаемых дробное значение в

оптимальном решении задачи линейного программирования, решаемой при определении $\zeta(G_{p_k})$.

2. Вычислим оценки $\zeta(G_{2k-1})$ и $\zeta(G_{2k})$. Найдем допустимые решения $(X_{2k-1},Y_{2k-1}) \in G_{2k-1}$, $(X_{2k},Y_{2k}) \in G_{2k}$ и вычислим значения $\bar{f}(X_i,Y_i)$, $i=2k-1,2k$.

3. Проверим неравенство

$$\min_{i=2k-1,2k} \bar{f}(X_i,Y_i) < z.$$

Если оно выполняется, то присвоим

$$z := \min_{i=2k-1,2k} \bar{f}(X_i,Y_i),$$

иначе z остается без изменения.

4. То (те) из множеств G_i, $i=2k-1,2k$, которое удовлетворяет условию $\zeta(G_i) < z$, включим в список S.

5. Для множеств G_i из списка S проверим условие оптимальности:

$$\min_{G_i \in S} \zeta(G_i) = \zeta(G_\nu) = \bar{f}(X_\nu,Y_\nu) = z. \qquad (2.3)$$

Если оно выполняется, то (X_ν,Y_ν) - оптимальное решение задачи (1.7)-(1.12), а X_ν - оптимальное решение задачи (1.1)-(1.5). Если (2.3) не выполняется, то перейдем к следующему пункту.

6. Из списка S исключим множества G_i, для которых $\zeta(G_i) > z$ (заграждение ветвления) и перейдем к следующему шагу.

§3. Нахождение оценок и допустимых решений

Как следует из правила ветвления, множество G_i определяется условиями (1.8)–(1.12) и некоторыми дополнительными условиями типа

$$y_r = 0, \quad y_s = 1. \tag{3.1}$$

Обозначим

$$I_i^0 = \{ r : y_r = 0 \quad \text{в определении множества } G_i \},$$
$$I_i^1 = \{ s : y_s = 1 \quad \text{в определении множества } G_i \}.$$

Для нахождения оценки $\zeta(G_i)$ заменим множество G_i на множество $\bar{G}_i$, которое получается из G_i отбрасыванием условия целочисленности (1.12). Очевидно, $\bar{G}_i \supset G_i$. Оценку $\zeta(G_i)$ найдем как минимальное значение целевой функции (1.7) на множестве $\bar{G}_i$. Для этого нужно решить задачу линейного программирования $\{(1.7)$–$(1.11), (3.1)\}$. Последняя заменяется более простой, если применить следующую, почти очевидную теорему (ср. [1], стр. 304).

Теорема. Если $X_i = (x_j^i)$, $\bar{Y}_i = (\bar{y}_j^i)$ – оптимальное решение задачи $\{(1.7)$–$(1.11), (3.1)\}$, то

$$x_j^i = \beta_j \bar{y}_j^i \tag{3.2}$$

для всех y_j, не встречающихся в условиях (3.1), т.е. для $j \in P \setminus (I_i^0 \cup I_i^1)$.

Из теоремы следует, что оптимальное решение задачи $\{(1.7)$–$(1.11), (3.1)\}$ не изменяется, если в ней ограничения (1.10) для $j \in P \setminus (I_i^0 \cup I_i^1)$ заменить ограничениями

$$x_j = \beta_j y_j, \quad j \in P \setminus (I_i^0 \cup I_i^1). \tag{3.3}$$

При указанных в (3.3) индексах j проделаем в целевой

функции замену $y_j = x_j / \beta_j$. При этом освободимся также от соответствующих ограничений (1.11), так как они всегда выполняются, если имеют место (3.3) и неравенства (1.4), (1.5). Рассмотрим еще переменные y_r и y_s ($r \in I_i^0$, $s \in I_i^1$), встречающиеся в условиях (3.1). Если имеем условие $y_s = 1$, то соответствующее ограничение (1.10) заменяется ограничением $x_s \leqslant \beta_s$, а в целевой функции (1.7) слагаемое $d_s y_s$ – константой d_s. Если $y_r = 0$, то, в силу (1.10), также $x_r = 0$. Соответствующую замену можно проделать в целевой функции (1.7) и ограничениях (1.8).

Обозначим

$$\bar{c}_j = \begin{cases} c_j & \text{при } j \notin P \text{ или } j \in I_i^1 \\ c_j + d_j / \beta_j & \text{при } j \in P \setminus (I_i^0 \cup I_i^1). \end{cases}$$

Тогда из сказанного следует, что оценка $z(G_i)$ находится как оптимальное значение целевой функции следующей задачи линейного программирования: минимизировать

$$\sum_{j \in I_i^1} d_j + \sum_{j \notin I_i^0} \bar{c}_j x_j \qquad (3.4)$$

при условиях

$$\sum_{j \notin I_i^0} a_{kj} x_j = b_k, \quad k = 1,2,\ldots,m, \qquad (3.5)$$

$$x_j \geqslant 0, \quad j \in \{1,2,\ldots,n\} \setminus I_i^0, \qquad (3.6)$$

$$x_j \leqslant \beta_j, \quad j \in P \setminus I_i^0 . \qquad (3.7)$$

Иногда, например при решении транспортной задачи с фиксированными доплатами, более удобна следующая форма задачи (3.4)–(3.7): минимизировать

$$\sum_{j\in I_i^1} d_j + \sum_{j=1}^{n} \bar{c}_j x_j \qquad (3.8)$$

при условиях

$$\sum_{j=1}^{n} a_{kj} x_j = b_k, \quad k=1,2,\ldots,m, \qquad (1.3)$$

$$x_j \geqslant 0, \quad j=1,2,\ldots,n, \qquad (1.4)$$

$$x_j \leqslant \beta_j, \quad j \in P, \qquad (1.5)$$

где дополнительно определено

$$\bar{c}_j = M \quad \text{при} \quad j \in I_i^0,$$

а M – достаточно большое положительное число, условно, $M = +\infty$. Здесь нужно только учитывать, что оптимальному значению $+\infty$ целевой функции последней задачи соответствует отсутствие допустимых решений в задаче (3.4)-(3.7). В обоих случаях берем $\zeta(G_i) = +\infty$.

Если $\zeta(G_i) < +\infty$, то нахождение точки $(X_i, Y_i) \in G_i$ не представляет трудности. В качестве $X_i = (x_j^i)$ возьмем оптимальное решение задачи {(3.8), (1.3)-(1.5)} (или задачи (3.4)-(3.7)), а координаты вектора $Y_i = (y_j^i)$ найдем по формулам

$$y_j^i = \begin{cases} 0, \text{ если } j \in I_i^0 \text{ или } j \in P \setminus (I_i^0 \cup I_i^1) \text{ и } x_j^i = 0, \\ 1, \text{ если } j \in I_i^1 \text{ или } j \in P \setminus (I_i^0 \cup I_i^1) \text{ и } x_j^i > 0. \end{cases}$$

После этого найдем значение целевой функции $\bar{f}(X_i, Y_i)$ по формуле (1.7).

Кроме вектора Y_i при ветвлении требуется вектор $\bar{Y}_i = (\bar{y}_j^i)$ такой, что $(X_i, \bar{Y}_i)$ является решением задачи {(1.7)-(1.11),

(3.1)}. Очевидно, координаты вектора $\bar{Y}_i$ получаются при $j \in P \setminus (I_i^0 \cup I_i^1)$ из соотношений (3.2), а при $j \in I_i^0$ и $j \in I_i^1$ – из условий (3.1), имеющихся в определении множества G_i.

§4. Применение алгоритма для транспортной задачи с фиксированными доплатами

Рассмотрим теперь частный случай задачи (1.1)-(1.5) – транспортную задачу с фиксированными доплатами (см.[4]): минимизировать функцию

$$f(X) = \sum_{k=1}^{m} \sum_{j=1}^{n} c_{kj}(x_{kj}), \tag{4.1}$$

где

$$c_{kj}(x_{kj}) = \begin{cases} 0, & \text{если } x_{kj} = 0, \\ c_{kj}x_{kj} + d_{kj}, & \text{если } x_{kj} > 0; \end{cases} \tag{4.2}$$

при условиях

$$\sum_{j=1}^{n} x_{kj} = a_k, \quad k=1,2,\ldots,m, \tag{4.3}$$

$$\sum_{k=1}^{m} x_{kj} = b_j, \quad j=1,2,\ldots,n, \tag{4.4}$$

$$x_{kj} \geqslant 0, \; k=1,2,\ldots,m; \; j=1,2,\ldots,n. \tag{4.5}$$

Предполагаем, что

$$c_{kj} \geqslant 0, \quad d_{kj} \geqslant 0.$$

Алгоритм, приведенный во втором параграфе, применим на этом частном случае, если учитывать следующее.

1. Величинам x_j, y_j, c_j и d_j в задаче (1.1)-(1.5) соответствуют величины x_{kj}, y_{kj}, c_{kj} и d_{kj} в транспортной задаче

(4.1)-(4.5) и верхним границам β_j - величины $\beta_{kj} = \min\{a_k, b_j\}$. (Если уже в формулировке задачи к условиям (4.3)-(4.5) прибавлены условия $x_{kj} \leqslant \beta_{kj}$, где $\beta_{kj} < \min\{a_k, b_j\}$, то имеем дело с задачей с ограниченными пропускными способностями коммуникаций, которая, разумеется, также решается рассматриваемым методом.)

2. Если a_k, b_j, c_{kj}, d_{kj} - целые, то значение целевой функции (4.1) при базисных решениях также целое. Поэтому, в качестве $\zeta(G_1)$ можно взять наименьшее ц е л о е число, не меньшее чем минимальное значение функции

$$\sum_{(k,j)\in I_1^1} d_{kj} + \sum_{k=1}^{m}\sum_{j=1}^{n} \bar{c}_{kj} x_{kj} \qquad (4.6)$$

при условиях (4.3)-(4.5), где

$$\bar{c}_{kj} = \begin{cases} c_{kj}, & \text{если } (k,j) \in I_1^1 \\ M, & \text{если } (k,j) \in I_1^0 \\ c_{kj} + d_{kj} / \beta_{kj}, & \text{если } (k,j) \in I_1^0 \cup I_1^1 \end{cases}$$

(ср. с задачей {(3.8), (1.3)-(1.5)} из предыдущего параграфа).

Отметим, что $I_0^0 = I_0^1 = \emptyset$ и задача {(4.6), (4.3)-(4.5)}, решаемая при определении $\zeta(G_0)$, совпадает с задачей Балинского [4], дающей обычно достаточно хорошее приближение к оптимальному решению задачи (4.1)-(4.5).

Для решения транспортной задачи с фиксированными доплатами методом ветвей и границ составлена программа для ЭВМ "Минск-32". Эта программа использует стандартную программу ОПТПЛ [2], которая требует от решаемой задачи невырожденнос-

ти. Но, к сожалению, это требование часто не выполняется. Для решения задачи 5×7 из статьи К.Мурти [6] при помощи этой программы на ЭВМ "Минск-32" потребовалось 33 секунды.

П р и м е р. Решим транспортную задачу с фиксированными доплатами, данные которой приведены в таблице 4.1. В этой таблице в левом углу каждой клетки находятся величины c_{kj} и в правом - величины d_{kj}. В таблицах 4.2 - 4.10 указаны оптимальные решения $X_i = (x^i_{kj})$ задач $\{(4.6), (4.3)-(4.5)\}$, решаемых при определении оценок $\xi(G_i)$ $(i=0,1,2,\ldots,8)$. При этом ненулевые компоненты x^i_{kj} размещены в клетках в правых нижних углах, а соответствующие коэффициенты $\bar{c}_{kj}$ целевой функции (4.6) - в левых верхних углах. По описанному алгоритму на 0-ом шагу вычислим $\xi(G_0) = 27$. Соответствующее до-

a_k \ b_j	2	3	3
4	1 3	2 2	3 1
4	4 2	3 6	2 5

Таблица 4.1.

	2	3	3
4	5/2 2	8/3 2	10/3
4	5	5 1	11/3 3

Таблица 4.2. (X_0)

пустимое решение $(X_0, Y_0) \in G_0$ дает $z = \bar{f}(X_0, Y_0) = 31$. Компоненты X_0 даны в таблице 4.2. Так как $z = 31 > \xi(G_0) = 27$, то включим G_0 в список S и переходим к шагу 1. Следующие шаги не требуют особых объяснений. Ход решения иллюстрируется деревом на рис. 4.1.

	2	3	3
4	5/2 2	∞	10/3 2
4	5	5 3	11/3 1

Таблица 4.3. (X_1)

	2	3	3
4	5/2 1	2 3	10/3
4	5 1	5	11/3 3

Таблица 4.4. (X_2)

	2	3	3
4	∞	2 3	10/3 1
4	5 2	5	11/3 2

Таблица 4.5. (X_3)

	2	3	3
4	1 2	2 2	10/3
4	5	5 1	11/3 3

Таблица 4.6. (X_4)

	2	3	3
4	1 1	2 3	10/3
4	5 1	∞	11/3 3

Таблица 4.7. (X_5)

	2	3	3
4	1 2	2 2	10/3
4	5	3 1	11/3 3

Таблица 4.8. (X_6)

	2	3	3
4	1	2	10/3
4	∞	∞	11/3

Таблица 4.9. (X_7)

	2	3	3
4	1 1	2 3	10/3
4	4 1	∞	11/3 3

Таблица 4.10. (X_8)

На шагу k = 4, проверяя условие оптимальности, получаем, что

$$\min_{G_i \in S} \xi(G_i) = \xi(G_8) = \bar{f}(X_8, Y_8) = z = 29$$

т.е. X_8 - оптимальное решение исходной задачи.

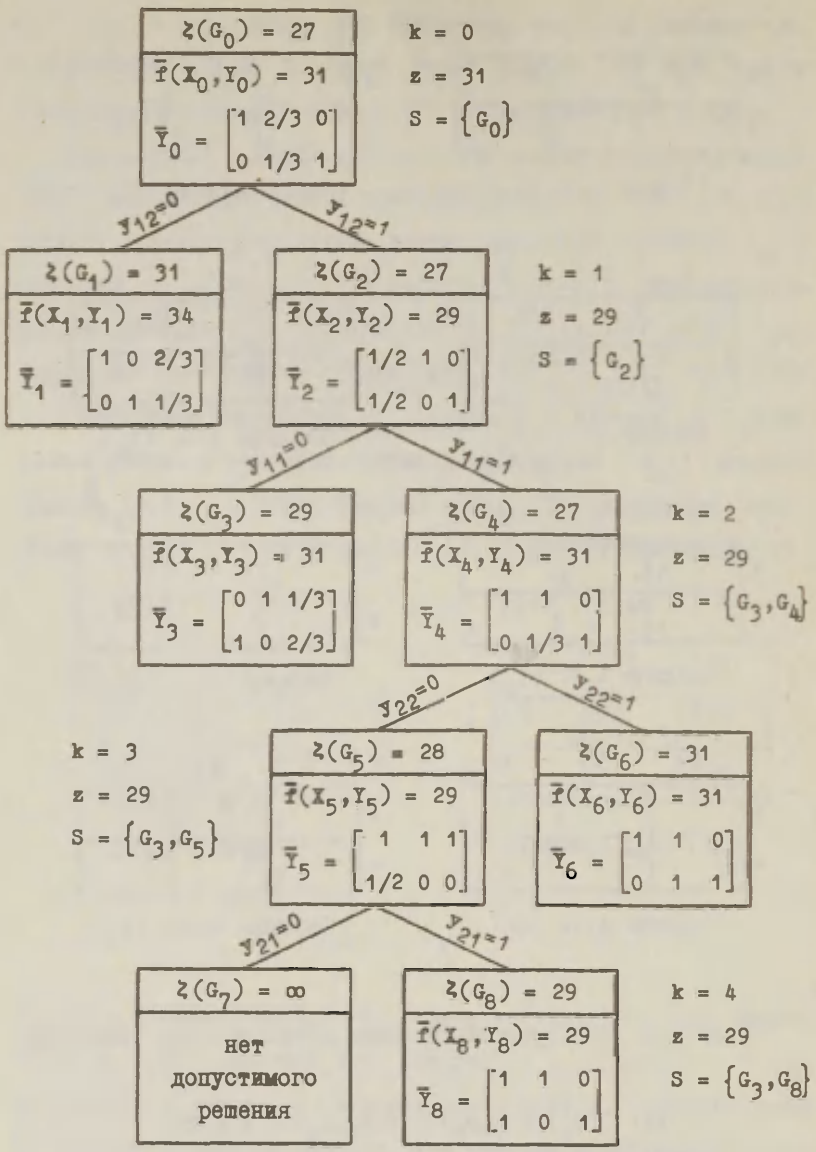

$z(G_0) = 27$
$\bar{f}(X_0, Y_0) = 31$
$\bar{Y}_0 = \begin{bmatrix} 1 & 2/3 & 0 \\ 0 & 1/3 & 1 \end{bmatrix}$
k = 0
z = 31
S = $\{G_0\}$
$y_{12}=0$
$y_{12}=1$
$z(G_1) = 31$
$\bar{f}(X_1, Y_1) = 34$
$\bar{Y}_1 = \begin{bmatrix} 1 & 0 & 2/3 \\ 0 & 1 & 1/3 \end{bmatrix}$
$z(G_2) = 27$
$\bar{f}(X_2, Y_2) = 29$
$\bar{Y}_2 = \begin{bmatrix} 1/2 & 1 & 0 \\ 1/2 & 0 & 1 \end{bmatrix}$
k = 1
z = 29
S = $\{G_2\}$
$y_{11}=0$
$y_{11}=1$
$z(G_3) = 29$
$\bar{f}(X_3, Y_3) = 31$
$\bar{Y}_3 = \begin{bmatrix} 0 & 1 & 1/3 \\ 1 & 0 & 2/3 \end{bmatrix}$
$z(G_4) = 27$
$\bar{f}(X_4, Y_4) = 31$
$\bar{Y}_4 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1/3 & 1 \end{bmatrix}$
k = 2
z = 29
S = $\{G_3, G_4\}$
$y_{22}=0$
$y_{22}=1$
k = 3
z = 29
S = $\{G_3, G_5\}$
$z(G_5) = 28$
$\bar{f}(X_5, Y_5) = 29$
$\bar{Y}_5 = \begin{bmatrix} 1 & 1 & 1 \\ 1/2 & 0 & 0 \end{bmatrix}$
$z(G_6) = 31$
$\bar{f}(X_6, Y_6) = 31$
$\bar{Y}_6 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$
$y_{21}=0$
$y_{21}=1$
$z(G_7) = \infty$
нет
допустимого
решения
$z(G_8) = 29$
$\bar{f}(X_8, Y_8) = 29$
$\bar{Y}_8 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$
k = 4
z = 29
S = $\{G_3, G_8\}$

Рис. 4.1.

Литература

1. Корбут А.А., Финкельштейн Ю.Ю., Дискретное программирование. М., "Наука", 1969.
2. Математическое обеспечение ЭВМ "МИНСК-32". вып. 4,1972, 107-109.
3. Хедли Дж., Нелинейное и динамическое программирование. М., "Мир", 1967.
4. Balinski, M.L., Fixed-cost transportation problems. Naval Res. Log. Quart., 1961, 8, № 1, 41-54.
5. Gray, P., Exact solution of the fixed-charge transportation problem. Oper. Res., 1971, 19, № 6, 1529-1538.
6. Murty, K.G., Solving the fixed-charge problem by ranking the extreme points. Oper. Res., 1968, 16, № 2, 268-279.

ВВОД ОГРАФОВ В ЭВМ

Ю.К.Кихо

В статье излагается способ линейного кодирования размеченных графов, т.н.ографов [1] и дается описание соответствующего языка ЛИНКС.

Первое сообщение об этом языке было опубликовано в 1970 году [2]. В течении последующих лет язык ЛИНКС интенсивно использовался при подготовке к вводу в ЭВМ структурных данных о химических реакциях. Был также создан транслятор с языка ЛИНКС для ЭВМ "Минск-32", т.е. программа, которая преобразует ЛИНКС-код во внутримашинное представление соответствующегоографа. В ходе практического применения языка и создания транслятора обнаружилась целесообразность небольшого изменения первого варианта языка, а также необходимость более строгого описания его синтаксиса. Настоящая статья посвящена, в основном, описанию окончательного варианта языка ЛИНКС. Кроме того, приводятся некоторые сведения о конкретной системе вводаографов в ЭВМ "Минск-32".

§1. Кодирование в системе ЛИНКС

В настоящем параграфе рассматриваются принципы составления ЛИНКС-кодов, т.е. дается семантика языка ЛИНКС. Точные определения всех конструкций этого языка можно найти в следующем параграфе.

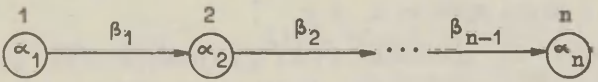
Кодируемые графы называются терминальными графами, а метки их вершин и дуг - терминальными метками. В системе ЛИНКС фиксированы некоторые терминальные метки, т.н. стандартные метки. Остальные используемые (нестандартные) терминальные метки - произвольные идентификаторы; их список составляется и задается пользователем.

Совокупность правил кодирования состоит из двух частей: правила кодирования блоков и правила представления графов. Первые из них позволяют, по-существу, составлять коды графов, вторые - кодировать графы "по частям" и, тем самым, облегчать процесс кодирования сложных графов.

Блоком называется граф с пронумерованными вершинами.

Основные коды блоков определяются следующим образом.

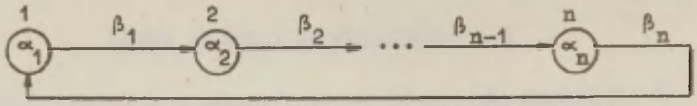
А1. Цепь вида



кодируется словом

$$< \alpha_1 [\beta_1] \alpha_2 [\beta_2] \cdots [\beta_{n-1}] \alpha_n > \quad .$$

А2. Цикл вида



кодируется словом

$$<\alpha_1[\beta_1]\alpha_2[\beta_2]\ \cdots\ [\beta_{n-1}]\alpha_n[\beta_n]> \quad .$$

А3. Направленное от корня n-вершинное дерево[1] ($n \geqslant 2$), корень которого имеет номер 1, кодируется словом вида

$$<\gamma_1;\gamma_2;\ \cdots\ ;\gamma_n> \quad ,$$

где $\gamma_1 = \alpha_1;0$,

$\gamma_i = \alpha_i[\beta_i]\eta_i \quad (i=2,3,\ldots,n)$,

α_i есть метка i-ой вершины, η_i – номер предка i-ой вершины, β_i – метка дуги, входящей в i-ую вершину.

А4. Произвольный n-вершинный и m-дужный блок ($n \geqslant 2$, $m \geqslant 1$), который не содержит дуг с меткой ";", можно кодировать словом вида

$$<\gamma_1;\gamma_2;\ \cdots\ ;\gamma_n> \quad ,$$

где $\gamma_i = \alpha_i[\beta_{i1}]\eta_{i1}[\beta_{i2}]\eta_{i2}\ \cdots\ [\beta_{im}]\eta_{im}$,

α_i есть метка i-ой вершины (x_i),

η_{ij} – номер j-ого соседа ($x_{\eta_{ij}}$) вершины x_i,

β_{ij} – метка дуги из x_i в $x_{\eta_{ij}}$.

Заметим, что правилами А1-А3 предусматривается кодирование только антисимметричных подографов.

Основные коды блоков упрощаются по следующим правилам.

---

[1] Имеется в виду связный ограф, в котором в каждую вершину кроме одной (корня) входит одна и только одна дуга; в корню дуг не входит.

Б1. Единицу в качестве номера вершины можно опустить.

Б2. Скобки "[ ]" вокруг стандартных меток дуг можно опустить.

Б3. Повторяющиеся (в коде) метки вершин можно опустить, за исключением последней вершины цепи.

Б4. Каждый код блока, а также любое его подслово (фрагмент кода), заключенное между ограничителями, можно заменить соответствующим прямым кодом. Под последним подразумевается обозначение, введенное пользователем в целях получения более коротких или более наглядных кодов графов.

Таблица прямых кодов, в которой каждому прямому коду сопоставлен некоторый код блока или фрагмент кода блока, составляется пользователем. В качестве прямого кода можно использовать любой идентификатор, отличный от терминальной метки. Поскольку прямые коды являются, по-существу, обозначениями (кодов или фрагментов), то один прямой код может в таблице встречаться лишь один раз, т.е. должен быть описан однозначно.

Слово, полученное при помощи замены по Б4, называется сокращенным кодом. Правило Б4 применимо и для сокращенных кодов.

В системе ЛИНКС разрешается кодировать блок и в некоторой другой системе при условии, что кодами блоков - спецкодами - являются балансированные слова. Под последним подразумевается слово в алфавите языка ЛИНКС, в котором символы "< " и " >" использованы как скобки. Спецкодам приписывают пятисимвольный признак для отличия их от основных кодов блоков. Признак начинается символом " ‘ ", четыре последующих

символа образуют название спецсистемы.

В принципе, правилами кодирования блоков обеспечивается возможность кодирования любых терминальных графов. Для этого достаточно пронумеровать вершины данного графа и применить одно из правил А1-А4 (или некоторое правило составления спецкода). Но, поскольку простые правила А1 и А2 для сложных графов неприменимы, а кодирование по правилам А3 и А4 значительно сложнее, то предусматривается возможность разбиения исходного графа на более мелкие блоки, которые можно кодировать отдельно. Процедура такого разбиения называется процедурой представления. В результате применения процедуры представления получается нетерминальный граф, т.н. представление исходного графа. Метками вершин в представлении служат коды выделенных в исходном графе блоков.

Правила кодирования блоков (А1-А4 и Б1-Б4) применимы и для нетерминальных графов. Поэтому, полученное представление может быть либо пронумеровано и закодировано по этим правилам (если оно достаточно просто), либо, в свою очередь, представлено в виде более простого нетерминального графа и т.д.

Процедура представления состоит из четырех этапов (В1-В4) и также применима как для терминальных так и для нетерминальных графов.

В1. Множество вершин (X) представляемого графа (G) разбивают на непустые подмножества $X_1, X_2, \ldots, X_m$ $(m \geqslant 1)$. Разбиение должно удовлетворять следующему условию: если два различных подмножества X_i и X_j связаны несколькими дугами в исходном графе, то все дуги между X_i и X_j должны иметь одно

и то же направление (либо из X_i в X_j либо из X_j в X_i). Кроме того, если G нетерминальный ограф, то выделенные подмножества должны быть попарно непересекающимися. Множество $X^* = \{X_1, X_2, \ldots, X_m\}$ считается множеством вершин представления.

В2. Множество дуг (U^*) представления определяют так: $(X_i, X_j) \in U^*$ тогда и только тогда, когда в исходном графе существует дуга, направленная из X_i в X_j или $X_i \cap X_j \neq \emptyset$ ($i \neq j$, $X_i \in X^*$, $X_j \in X^*$).

В3. Определяют метки вершин представления. Для этого произвольным образом нумеруют вершины подографа, порожденного множеством X_i, получая блок $G(X_i)$ ($i=1,2,\ldots,n$). Блок $G(X_i)$ кодируется по правилам кодирования блоков (по А1-А4, Б1-Б4 или по спецсистеме). Меткой вершины X_i в представлении считается код блока $G(X_i)$.

В4. Определяют метки дуг представления. Метка дуги, направленной от вершины X_i к вершине X_j, есть последовательность характеристик всех дуг, направленных из X_i в X_j в исходном графе G. Характеристики отделены друг от друга запятыми. Характеристика дуги (x,x'), где $x \in X_i$ и $x' \in X_j$, имеет вид $\eta[\beta]\eta'$, где η есть номер вершины x в блоке $G(X_i)$, η' – номер вершины x' в блоке $G(X_j)$ и β – метка дуги (x,x') в G. В том случае, когда $X_i \cap X_j \neq \emptyset$, для каждой вершины x'' из $X_i \cap X_j$ в метку дуги (X_i, X_j) прибавляется еще характеристика вида $\eta \uparrow \eta'$, где η – номер вершины x'' в блоке $G(X_i)$ и η' – номер этой же вершины в $G(X_j)$. Метка "$\uparrow$" есть стандартная нетерминальная метка дуги. Порядок расположения характеристик в последовательности произволен.

Метка дуги может быть упрощена по правилам Б1 и Б2 (см.

выше). Например, метка

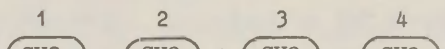
2[YE]1,1[.]1,3[=]2

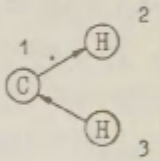
сводится к виду

2[YE],.,3=2 .

С формальной точки зрения, процесс кодирования заданного терминального графа G_0 в системе ЛИНКС заключается в составлении последовательности $G_1, G_2, \ldots, G_k$ ($k \geqslant 1$), где G_i есть представление графа G_{i-1} ($i=1,2,\ldots,k$) и G_k есть одновершинный граф. ЛИНКС-кодом графа G_0 называется метка единственной вершины графа G_k.

Стандартная нетерминальная метка ";" используется для удобства кодирования. Именно, в ходе кодирования можно графу прибавить произвольное количество дуг с этой меткой, т.н. пустых дуг. Например, графы

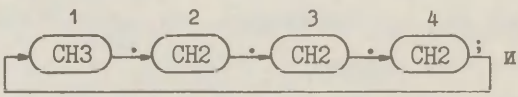


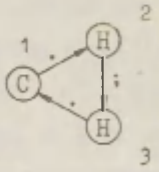
и


непосредственно кодируются словами

<CH3 . CH2 . . CH2> (по A1) и <C.2;H;.1> (по A4)

соответственно. Но после превращения их в циклы (добавлением пустой дуги), т.е. в



и


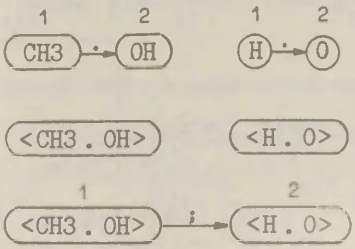
эти графы кодируются (по A2) словами

<CH3 . CH2 . . ;> и <C.H;.>

соответственно.

При помощи пустой дуги можно "связать" компоненты связности графа, чтобы получать связный граф. Обычно компоненты связности несвязного графа представляются отдельно в виде одновершинных графов, которые затем объединяются в цепь (или цикл) при помощи пустых дуг. Кодом исходного графа является код такой цепи (такого цикла).

Пример.



<<СН3 . ОН> ; <Н . О>>

Из правила Б1 вытекает возможность кодировать отдельно и висячие подграфы[2]: каждый такой подграф можно заменить на его одновершинное представление независимо от других частей графа. Необходимо только соблюдать, чтобы при нумерации вершин на этапе В3 (процедуры представления) номер "1" присваивался всегда выходу подграфа или блоку,содержащему этот выход.

---

[2] Висячим подграфом называется подграф, который имеет один и только один выход. Выходом подграфа считается вершина, в которую входит или из которой исходит дуга, непринадлежащая этому подграфу.

Пример. Нетерминальный граф G_1 на рис. 1 получен из G_0 после замен висячих подографов I, II, III на их одновершинные представления. Подографы указаны пунктирной линией. На рис. 2 изображены последовательные представления подографа III при его отдельном кодировании. Представления III-1 и III-2 получены также путем замены висячих подографов.

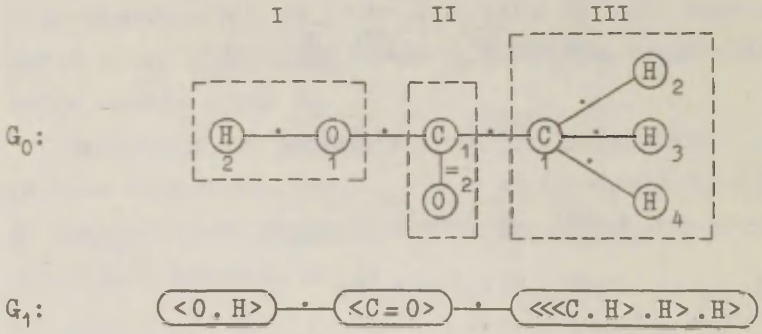


Рис. 1.

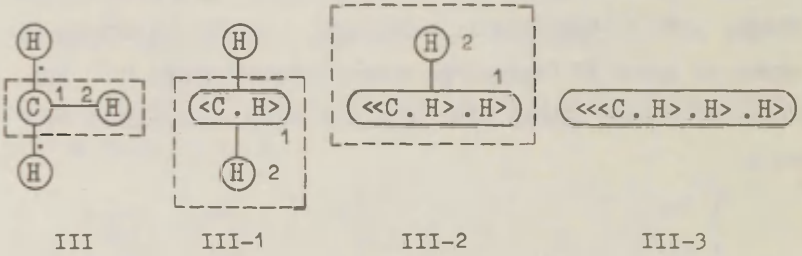


Рис. 2.

Из G_1 получается ЛИНКС-код графа G_0 –

<<O . H>.<C = O>. <<<C . H>. H>. H>> .

Замечание. Терминальным графом G_0 на рис. 1 изображается структура с неориентированными связями: на рисунке дуги даны линиями, а не стрелками. Такие структуры кодируются в виде ориентированных антисимметричных графов, причем ориентацию терминальных дуг определяет (обычно подразумевает) кодировщик по своему усмотрению, непосредственно перед кодированием блока.

Если рассматриваемые терминальные графы содержат много одинаковых висячих подграфов, то введением для каждого такого подграфа обозначения (прямого кода) существенно упрощается процесс кодирования. Например, если кодам блоков <O.H>, <C=O> и <<<C.H>.H>.H> соответствовали бы в таблице прямых кодов идентификаторы OH, CO и ME соответственно, то вместо G_1 непосредственно получился бы граф (OH)–(CO)–(ME) и ЛИНКС-кодом графа G_0 было бы слово <OH.CO.ME> .

Разумеется, роль, которую играют прямые коды при облегчении кодирования, далеко не ограничивается ролью обозначений висячих подграфов. Некоторые примеры кодирования структур в системе ЛИНКС с использованием прямых кодов можно найти, например, в [3].

§2. Синтаксис языка ЛИНКС

Под языком ЛИНКС подразумевается формальный язык, слова в котором – ЛИНКС-коды. Алфавит языка ЛИНКС состоит из печатных символов ЭВМ "Минск-32" (ГОСТ 10859-64, см. [4], стр. 234-235).

Синтаксис языка описывается в нормальной форме Бэкуса, но традиционная нотация Бэкуса слегка изменена. Это сделано потому, что скобки "<" и ">" входят в алфавит языка ЛИНКС и их неудобно использовать в качестве метасимволов. Для обозначения синтаксических единиц используются слова, состоящие из русских строчных букв и символа "_" . В правой части формулы Бэкуса такие обозначения отделяются друг от друга либо символом "|" (в обычном значении "или") либо символом "‖", означающим конкатенацию соответствующих конструкций. Вместо термина "ЛИНКС-код" в нижеприведенных формулах используется обозначение "л_код" .

л_код ::= основной_код | признак_спецкода ‖ спецкод |
сокращенный_код

основной_код ::= <цепь> | <цикл> | <дерево> | <таблица>

цепь ::= проявленная_вершина | проявленная_вершина ‖ открытая_
цепь ‖ проявленная_вершина

проявленная_вершина ::= терминальная_метка_вершины | л_код

терминальная_метка_вершины ::= стандартная_метка_вершины |
нестандартная_метка_вершины

нестандартная_метка_вершины ::= идентификатор

идентификатор ::= 0 | буква_знак | натуральное_число ‖ буква_
знак | идентификатор ‖ цифра | идентификатор ‖ буква_знак

буква_знак ::= А|Б|В|Г|Д|Е|Ж|З|И|Й|К|Л|М|Н|О|П|Р|С|Т|У|Ф|Х|Ц|
Ч|Ш|Щ|Ы|Ь|Э|Ю|Я|D|F|G|I|J|L|M|N|Q|R|S|U|V|W|Z|+|-|/|
,|␣|$_{10}$|(|)|×|*|'

цифра ::= 0|1|2|3|4|5|6|7|8|9

натуральное_число ::= 1|2|3|4|5|6|7|8|9|натуральное_число ‖
цифра

стандартная_метка_вершины ::= R|R1|R2|R3|R4|R5|R6|R7|R8|R9|
X|X1|X2|X3|X4|X5|X6|X7|X8|X9|Y|Y1|Y2|Y3|Y4|Y5|Y6|Y7|
Y8|Y9|Z|Z1|Z2|Z3|Z4|Z5|Z6|Z7|Z8|Z9|S0|S1|S2|S3|S4|S5|
S6|S7|S8|S9|T|K|P0

открытая_цепь ::= связь | связь ↓ вершина ↓ связь

связь ::= ; | [;] | непустая_связь

непустая_связь ::= стандартная_непустая_метка_дуги |
[непустая метка дуги]

стандартная_непустая_метка_дуги ::= ↑ | стандартная_
терминальная_метка_дуги

стандартная_терминальная_метка_дуги ::= · | = | ≠ | :

непустая_метка_дуги ::= стандартная_непустая_метка_дуги |
нестандартная_терминальная_метка_дуги | нетерминальная_
метка_дуги

нетерминальная_метка_дуги ::= номер_вершины ↓ связь ↓ номер_
вершины | нетерминальная_метка_дуги , нетерминальная_
метка_дуги

нестандартная_терминальная_метка_дуги ::= идентификатор

номер_вершины ::= пусто | натуральное_число

пусто ::=

вершина ::= пусто | проявленная_вершина

цикл ::= проявленная_вершина ↓ открытая_цепь

дерево ::= проявленная_вершина ; 0 ; крона

крона ::= описание_вершины_дерева | крона ; крона

описание_вершины_дерева ::= вершина ↓ непустая_связь ↓
номер_вершины

таблица ::= первая_строка ; внутренние_строки ; последняя_
строка

первая_строка ::= проявленная_вершина ‖ окружность_вершины

последняя_строка ::= вершина ‖ непустая_окружность_вершины

окружность_вершины ::= пусто | окружность_вершины ‖ непустая_
связь ‖ номер_вершины

непустая_окружность_вершины ::= окружность_вершины ‖
непустая_связь ‖ натуральное_число

внутренние_строки ::= пусто | внутренние_строки ; строка

строка ::= вершина ‖ окружность_вершины

признак_спецкода ::= ' имя_спецсистемы

имя_спецсистемы ::= буква_знак ‖ буква_знак ‖ буква_знак ‖
буква_знак

спецкод ::= идентификатор | <балансированное_слово>

балансированное_слово ::= чистое_слово | чистое_слово ‖
<чистое_слово> ‖ чистое_слово

чистое_слово ::= пусто | чистое_слово ‖ внутренний_символ

внутренний_символ ::= [|] | · | = | ≠ | : | ; | ↑ | цифра | буква_
знак

сокращенный_код ::= идентификатор | <вторичное_слово>

вторичное_слово ::= идентификатор | вторичное_слово ‖
ограничитель | вторичное_слово ‖ ограничитель ‖
идентификатор | вторичное_слово ‖ ограничитель ‖
натуральное_число

ограничитель ::= < | > | · | = | ≠ | : | ; | ↑

§3. О системе ЛИНКС-1

Система ЛИНКС-1 есть совокупность программ для ЭВМ "Минск-32", обеспечивающих трансляцию ЛИНКС-кодов. К этой системе относятся, кроме транслятора основных и сокращенных кодов, еще декодер спецкодов, а также обслуживающие программы, предусмотренные для ввода и преобразования вспомогательных данных (списков терминальных меток и таблиц прямых кодов).

В результате трансляции получается связный код [1] графа, соответствующего исходному ЛИНКС-коду. Связный код является достаточно гибкой формой внутримашинного представления графа, позволяющий выполнить многие операции над графами, а также сравнительно легко перейти к другим типам представления графа.

Вспомогательные данные сохраняются на магнитной ленте: список нестандартных терминальных меток в виде массива с именем TERM, а таблица прямых кодов - в виде массивов DC2 и DC2V. В массиве DC2 расположен список прямых кодов, в DC2V - список соответствующих им слов (ЛИНКС-кодов или фрагментов ЛИНКС-кодов). Все массивы выведены стандартными зонами.

Во время трансляции ЛИНКС-кода вспомогательные данные должны находиться в оперативной памяти. Поэтому, перед (первым) обращением к транслятору эти данные считываются с магнитной ленты в память при помощи соответствующей программы из системы ЛИНКС-1.

В системе ЛИНКС-1 не различают нестандартных терминальных меток вершин и дуг. Любая метка из TERM может быть в терминальном графе использована для метки и вершины и дуги.

Списком в массиве TERM доопределяется множество всех терминальных меток и, тем самым, класс терминальных графов. Последним обычно тесно связана таблица прямых кодов. Следовательно, всем пользователям, которые занимаются одним и тем же классом графов, необходимы одни и те же вспомогательные данные и, наоборот, группы пользователей, имеющие дело с разными классами графов, используют разные массивы TERM, DC2 и DC2V, т.е. разные магнитные ленты.

Системой ЛИНКС-1 на вспомогательные данные налагаются следующие ограничения:

- количество нестандартных терминальных меток $\leqslant 123$;
- длина нестандартной терминальной метки $\leqslant 5$ символов;
- длина прямого кода $\leqslant 125$ символов;
- длина соответствующего прямому коду слова (длина элемента массива DC2V) $\leqslant 1595$ символов;
- элемент массива DC2V должен быть балансированным словом, т.е. правилу Б4 (см. §1) добавляется требование о балансированности заменяемого прямым кодом подслова.

Процесс трансляции ЛИНКС-кодов распадается на три последовательные этапа: декодирование спецкодов, переход от сокращенного кода к основному коду и трансляция основных кодов.

В системе ЛИНКС-1 допускаются спецкоды с признаком 'COND, которые получаются при специальном кодировании мозаичных структур. (К последним относятся структурные формулы полициклических конденсированных химических соединений. Кодирование в системе COND детально описано в [5], стр. 122-129, принципами кодирования таких структур можно познакомиться и

по [6].) Программа декодирования спецкодов, т.н. декодер[3], заменяет каждый встретившийся в исходном ЛИНКС-коде спецкод блока соответствующим несокращенным ЛИНКС-кодом типа А4 (см. §1). Таким образом, после всех замен получается ЛИНКС-код того же графа, уже не содержащий спецкодов.

Следующие два этапа выполняются подпрограммами одной и той же программы, называемой главным транслятором[4].

Подпрограммой приведения кода к несокращенному виду просматривается (многократно) исходный ЛИНКС-код. Каждый найденный прямой код заменяется на соответствующее ему слово по таблице прямых кодов. Подпрограмма заканчивает свою работу, если на очередном просмотре не произошло ни одной замены, т.е. если получился основной код. Эта подпрограмма использует списки терминальных меток и таблицу прямых кодов. Если в просматриваемом коде обнаружился идентификатор, которого нет ни среди терминальных меток, ни среди прямых кодов, то печатается соответствующее сообщение об отсутствии прямого кода и работа заканчивается аварийным выходом из главного транслятора.

Отметим, что при проверке идентификатора сначала просматривается список нестандартных терминальных меток, затем список стандартных меток вершин и, наконец, список прямых кодов. Если обнаруживается вхождение проверяемого идентификатора в некоторый список, то проверка прекращается (тип

---

3 Программу составила А.Р. Ялас.

4 Программу составил Я.Ю. Каазик.

установлен). Таким образом, упомянутые списки могут иметь общие элементы, это не приводит к ошибке. Но необходимо иметь в виду, что включением некоторого идентификатора в список нестандартных терминальных меток, этот идентификатор как бы исключается из двух остальных списков; нельзя также использовать прямые коды, совпадающие со стандартными метками вершин.

На последнем этапе применяются подпрограммы трансляции основного кода. Здесь происходят действия, обратные кодированию: после декодирования нетерминальных меток вершин и дуг представления, составляется связный код "предыдущего представления"; процесс продолжается, пока не получается связный код терминального графа.

В результирующем связном коде метки вершин и дуг изображаются следующим образом.

Метка вершины имеет вид mn, где $m \in \{1,2\}$, $n \in \{5,6,7,\ldots,127\}$; m имеет значение 1 у нестандартной метки и 2 у стандартной метки; $n = k + 4$, где k есть порядковый номер метки в соответствующем списке ($k \geqslant 1$). Упорядоченность списка нестандартных меток определяется пользователем при составлении этого списка, т.е. при записи массива TERM на магнитную ленту. Упорядоченность же списка стандартных меток вершин фиксирована (в этом порядке приведен их список и в §2). В ячейке величины m и n записывают как двоичные целые без знака в разрядах 26, 27 и 28–34 соответственно.

Метка дуги имеет вид n, $n \in \{1,2,\ldots,127\}$. Первые четыре значения n используются для обозначения стандартных терминальных меток

соответственно. Бо́льшие значения n имеют тот же смысл, что и в случае метки вершины. В ячейке запись метки дуги (n) занимает разряды 14-20.

Неиспользованные разряды ячейки, которые предусмотрены в общем связном коде тоже для хранения метки вершины или дуги, заполняют нулями.

Приводим, наконец, в виде таблицы данные, которые характеризуют скорость работы транслятора. В эксперименте использовалась таблица прямых кодов, содержащая 203 прямых кода. В частности, прямые коды, входящие в состав протранслированных ЛИНКС-кодов, были описаны следующим образом.

Прямой код	Соответствующее слово
0-	<0[-]>
B	<C:<C.H>:::::>
BB1,2	<C:2:9;:3;<C.H>:4;:10;:10;:5;:6;:7;C:8;:9>[2.1

(Метки "C", "H", "O" и "-" - терминальные.)

ЛИНКС-код	Кол-во вершин	Кол-во дуг	Время трансляции в сек. (±0,02 сек.)
0-	1	1	0,10
B	11	11	0,32
<B.B>	22	23	0,84
<BB1,2]H>	17	18	0,82
<BB1,2]BB1,2]>	32	36	2,00

Литература

1. Кихо Ю.К., Машинные форматы ографов. Труды ВЦ ТГУ, 1975, № 33, 40-52.
2. Кихо Ю.К., Система ЛИНКС I. "Реакционная способность органических соединений", 1970, № 1(23), 94-109.
3. Кихо Ю.К., Система ЛИНКС II. "Реакционная способность органических соединений", 1970, № 3(25), 547-555.
4. Кушнерев Н.Т., Неменман М.Е., Цагельский В.И., Программирование для ЭВМ "Минск-32". "Статистика", М., 1973.
5. Кихо Ю.К., Диссертация на соискание ученой степени кандидата технических наук. Тарту, 1971.
6. Гейвандов Э.А., Система кодирования органических соединений. "НТИ", 1970, № 5(25).

НЕКОТОРЫЕ ОБЩИЕ ПРИНЦИПЫ ПОСТРОЕНИЯ И ПРОЕКТИРОВАНИЯ ПРИКЛАДНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

В.А.Крахт

Создание информационной базы для больших автоматизированных систем обработки управленческой и экономической информации считается наиболее важной и сложной задачей при проектировании таких систем (см. предисловия в [1], а также [2, 3]). Проблема заключается в создании многоцелевой и интегрированной системы хранения и обработки информации, обеспечивающей автоматическую организацию хранения больших массивов данных, их оперативное обновление, поиск и выдачу данных в ответ на различные запросы пользователей, оперативное обеспечение данными различных расчетных программ в ходе их выполнения и пр. В [2] подчеркивается, что такие системы не могут быть созданы методами проектирования, принятыми при разработке автоматизированных систем управления (АСУ).

Путь к решению этой проблемы указывается в ряде работ [1, 8 – 12] и заключается в разработке т.н. систем управления базами данных (СУБД) и использовании их как инструмента при создании различных прикладных систем. Такая точка зрения, основанная на зарубежном опыте [8 – 12], поддерживается редакционной коллегией сборника [1] и опубликованными там ра-

ботами [1-6].

СУБД, как инструментальные системы для создания конкретных прикладных систем, представляют большой интерес для разработчиков АСУ. С уверенностью можно сказать, что применение таких систем позволяет в существенной мере ускорить и упростить проектирование и внедрение задач и подсистем, легко учитывать особенности организации, оперативно изменить и приспосабливать внедренные уже задачи и подсистемы к изменяющимся условиям работы организации.

С другой стороны, однако, нужно учитывать, что разработка АСУ на основе СУБД требует существенного изменения установившихся представлений о структуре информационной базы и программного обеспечения задач (в АСУ), а также о методике проектирования и внедрения их. До сих пор, однако, эта сторона проблемы во всем мире еще мало затронута, хотя сущность и важность проблемы подчеркивается в ряде работ [10-12]. Непосредственно проблемы проектирования прикладных информационных систем на основе СУБД рассматриваются, например в [11, 12].

В данной работе также рассматриваются некоторые основные принципы построения и проектирования информационных систем на основе СУБД. Автор поставил себе задачу показать основные и общие исходные принципы, которыми нужно руководствоваться при проектировании прикладных систем. Настоящая работа во многом опирается на работы [11, 12].

Проблемы проектирования прикладных информационных систем ниже рассматриваются безотносительно к каким-то конкретным инструментальным системам (СУБД). Но при этом предполагает-

ся, что читатель знаком с общими принципами построения и применения СУБД. Общее и в то же время достаточно подробное представление об этих системах можно получить из [8, 9]. Некоторые конкретные СУБД описываются в [4, 5, 6].

§1. Построение прикладных информационных систем на основе СУБД

1.1. Информационная система административного типа создается в интересах какой-то группы (организации) пользователей для накопления и хранения информации об интересующих пользователей объектах и выдачи им обработанную надлежащим образом информацию об этих объектах. При этом мы имеем в виду хранение и обработку управленческой и экономической информации. Тем самым, назначение рассматриваемых нами систем то же самое, что и у АСУ административного типа. Но, учитывая совершенно различные принципы построения и методику их проектирования, нам представляется необходимым использовать и различную терминологию.

С точки зрения пользователя информационная система должна выполнять две основные функции: хранение данных и обработку данных. В соответствии с этим при проектировании информационной системы также можно выделить две основные проблемы. Первая из них связана с определением организации хранимых данных (базы данных), вторая - определением организации процесса обработки данных и взаимосвязи системы с пользователем.

Первой из этих проблем уделено до сих пор гораздо больше внимания, чем второй, хотя с точки зрения пользователя процесс обработки имеет такое же (или даже большее) значение, чем организация базы данных (БД). Это, конечно, связано с тем, что именно концепция БД революционировала принципы построения информационных систем и придавала им новое качество.

Современная концепция БД учитывает процесс обработки данных в основном лишь в той мере, как он обеспечивается самой СУБД. Как правило, в таком случае обработка ограничивается загрузкой и обновлением базы, реорганизацией и изменением базы и выводом данных из базы. Но такие функции как генерация различных сводных справок и отчетов, выполнение расчетов и пр. системными операциями СУБД уже не обеспечиваются. В данной работе мы постараемся учитывать и этот аспект проблемы.

Основные концепции по построению и проектированию информационных систем лучше всего, на наш взгляд, выводятся в работе [12]. В этой работе информационная система (у автора - база данных) рассматривается с четырех аспектов - как источник информации, как модель, как система и как результат процесса отображения. В данной работе мы в соответствующих местах очень сжато освещаем некоторые основные идеи автора работы [12]. Здесь уместно лишь отметить, что первые два аспекта касаются сущности самой системы (базы данных), а два последних - методики проектирования (у автора такого разделения нет).

К этим четырем аспектам мы хотим прибавить еще один, на наш взгляд такой же важности аспект, а именно - информацион-

ная система как процесс (обработки информации).

1.2. Одной функцией информационной системы, как уже было сказано, является хранение информации об объектах, интересующих пользователя системы. Такие объекты могут представлять собой любые абстрактные или конкретные предметы (явления), о которых можно собирать, регистрировать или производить информацию, например, служащие организации, продукция предприятия, производственные планы, студенты, учебные предметы и пр. Все они вместе образуют предметную область информационной системы.

По отношению к объектам предметной области предполагается, что каждый из них может быть характеризован некоторым множеством признаков (свойств) и что каждая из них может находится в некоторых отношениях с другими объектами данной предметной области. Объекты с одинаковыми признаками считаются однотипными, причем все однотипные объекты образуют класс объектов, соответствующий данному множеству признаков. В состав предметной области может войти множество различных классов.

Множество признаков данного класса объектов по сути дела является информационной характеристикой объектов. Но сами эти признаки могут быть двух типов: исходные (первичные) и производные (вторичные). Производные признаки могут быть определены на основе других признаков (того же или других объектов), исходные - нет.

Предполагаем еще, что каждый объект имеет по крайней мере один признак, значение которого однозначно определяет

данный объект среди других. Такой признак называется ключевым.

В общем случае, если информационная система предполагается быть использованной в разных сферах деятельности данной пользовательской организации, то в предметной области может быть выделен ряд подобластей, соответствующих этим сферам деятельности.

1.3. Перейдем теперь к рассмотрению того, как информация о предметной области хранится в информационной системе.

Каждому объекту предметной области в информационной системе сопоставляется некоторый набор данных, называемый записью (реляционные базы данных в данной работе не рассматриваются). Этот набор можно также рассматривать как информационный объект, соответствующий данному объекту предметной области.

Логическая и физическая структура записи может быть различной в зависимости от принятой инструментальной системы [8, 9]. Но в данной работе это не имеет значения. Важно лишь то, что каждому признаку данного объекта предметной области в соответствующей записи сопоставляется некоторый элемент данных (поле данных, если иметь в виду физический уровень) и то, что логическая структура всех записей объектов одного и того же класса одинакова. Элементам данных в записях присваиваются значения соответствующих признаков объектов, чем и записи станут информационным отображением представляемых ими объектов.

Элемент данных в записи, сопоставляемый ключевому при-

знаку объекта, называется ключем записи. По значению ключа в системе идентифицируются записи и тем самым определяется доступ к данным соответствующего объекта.

Все записи, соответствующие объектам данного класса, вместе образуют файл, что, тем самым, является информационным отображением класса однотипных объектов.

Все файлы вместе взятые, в свою очередь образуют структуру, которую и принято называть базой данных [8]. Следовательно, база данных является информационным отображением всей предметной области или ее подобласти, если вся область делится на подобласти.

1.4. В подразделе 1.2 было сказано, что объекты предметной области могут находится между собой в разных отношениях (например, сотрудники организации могут находится в отношениях сотрудничества и подчинения и т.д.). Кроме того, было еще сказано, что одни признаки объектов могут быть определены на основе других признаков того же или других объектов. Все эти отношения между объектами и их признаками должны быть отражены и в базе данных.

Мы считаем целесообразным выделить в базе данных два класса отношений: структурные и функциональные.

Структурные отношения отражают отношения между объектами предметной области и могут быть двух видов - внутрифайловые и межфайловые, отображающие отношения между объектами одного и того же или разных классов предметной области соответственно. Отношение объекта с другими объектами учитывается введением в записи этого объекта специальных элементов (ука-

зателей), значения которых и указывают на объекты, с которыми данный элемент находится в рассматриваемых отношениях. В СУБД с сетевой организацией данных [4, 5, 9] связанные таким образом записи образуют специальный тип структуры данных, определяемый при описании данных и целиком доступный при обработке. В других системах [6, 9] такая возможность отсутствует, вследствие чего отношения могут быть обработаны лишь составленными для этого проблемными программами.

Функциональные отношения в базе данных отражают зависимость одних признаков объектов от других и вообще зависимость одних (производных) данных от других (исходных или производных). Мы выделим два подкласса функциональных отношений: внутренние и внешние.

Под внутренними отношениями понимаются функциональные зависимости между хранимыми в базе данными. Они могут быть внутризаписные, внутрифайловые (межзаписные в пределах одного файла) и межфайловые.

Под внешними отношениями понимаются функциональные зависимости между данными в терминальных структурах (файлах) с одной стороны и хранимыми в базе данными с другой. При этом под терминальными структурами (файлами) понимаются структуры (файлы) вводимых в базу и выводимых из базы данных. В этой связи структуры хранимых в базе данных можно называть базовыми.

Все функциональные зависимости в основном могут быть реализованы только процедурно при помощи соответствующих программ обработки и тем самым относятся к процессу обработки данных, а не организации базы.

1.5. Информационная система создается не только для того, чтобы хранить информацию, а в основном для того, чтобы обрабатывать информацию. С точки зрения пользователя обработка заключается в выполнении различных операций по организации базы и в решении прикладных (проблемных) задач.

Различные задачи и операции в информационной системе реализуются по разному. Некоторые из них (ввод данных в базу, вывод их из базы и пр.) реализуются при помощи СУБД (непосредственно или обращением из прикладных программ), другие - полностью прикладными программами, составленными на поддерживаемых данной инструментальной системой языках. Много зависит и от того, как в СУБД предусматривается организация передачи данных из базы к программам обработки и обратно [8].

Независимо от способа реализации и возможностей принятой за основу СУБД, задачи обработки, инициализируемые пользователями, целесообразно делить на два класса:

(1) прикладные (проблемные) задачи, непосредственно интересующие пользователя - генерация справок и отчетов, выполнение расчетов и пр.;

(2) поддерживающие задачи, необходимые для поддержания базы в актуальном состоянии.

Последние в свою очередь целесообразно делить на два подкласса задач: загрузка и корректировка базы вводимыми извне исходными данными (трансакция), реорганизация базы и регенерация (изменение и перерасчет) хранимых в базе данных.

Следует отметить, что в базе данных в принципе можно хранить лишь исходные данные. Но в случае прикладных задач

это может привести к очень сложным программам обработки и существенному увеличению времени реакции системы. Поэтому, весьма часто может оказаться целесообразным расширение базы промежуточными производными данными, необходимыми в различных прикладных задачах. Именно поэтому и необходимо отдельно рассматривать задачи реорганизации базы и регенерации данных.

С другой стороны, предварительная генерация и хранение всех запрашиваемых потребителями данных также не оправдано, так как это обходится слишком дорого, хотя время реакции при этом будет минимальным. Таким образом, здесь требуется некоторое компромиссное решение.

1.6. Задачи обработки данных, рассмотренные в предыдущем подразделе с точки зрения пользователя, очевидно также относятся к предметной области. Подобно тому, как объектам предметной области и отношениям между ними в информационной системе сопоставляются соответствующие структуры данных (см. 1.3 и 1.4), задачам обработки мы сопоставляем функциональные отношения. Другими словами, мы считаем, что задачи обработки в системе отображаются функциональными отношениями.

Функциональные отношения, как уже было сказано, устанавливаются между разными информационными структурами - базовыми или терминальными. В этой связи мы выделим три типа информационных структур:

(1) хранимые в базе данные - базовые структуры (файлы) как основные,

(2) вводимые в систему данные - входные (трансакционные) структуры (файлы),

(3) выводимые из системы данные - выходные (генерированные) структуры (файлы).

Все задачи обработки данных в информационной системе связаны с некотором переносом данных из одной структуры в другую. Это очевидно в случае задач ввода данных в базу и генерации отчетов. Но и в случае реорганизации и регенерации базы на самом деле происходит перенос данных из одной базовой структуры в другую. В частном случае, конечно, данные после обработки могут быть записаны обратно в исходную структуру, т.е. входная и выходная структуры совпадают, но перенос данных как такой остается. В общем случае перенос данных сопровождается их логической и арифметической обработкой.

В обобщенной форме функциональная структура информационной системы схематически изображается на рис. 1. На схеме выделены три обобщенные функции системы по обработке: трансакция (transaction), переструктурирование (restructuring), генерация (generation).

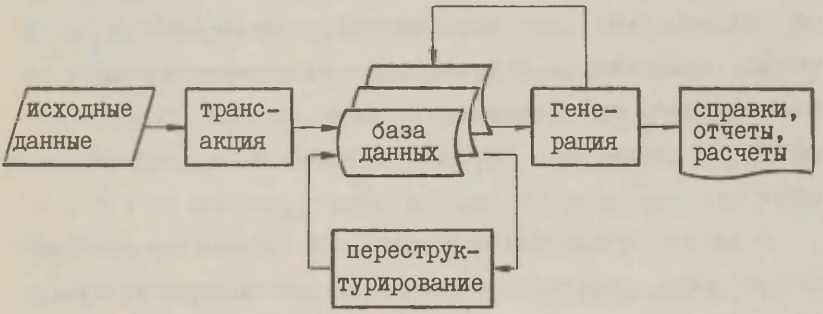


Рис. 1.

Трансакцией отображается перенос данных из входных файлов в БД, т.е. прибавление, замена и удаление данных, но не изменение логической структуры базы. Переструктурированием, наоборот, учитывается перенос данных из одной структуры базы в другую, т.е. изменение структуры хранимых в базе данных. Генерацией представляется вывод из системы данных (справок, отчетов, расчетов), составленных на основе хранимых в базе данных, а также изменение и корректировка хранимых данных результатами обработки (расчета), но опять без изменения логической структуры базы.

§2. Проектирование информационной системы

2.1. Информационная система создается в интересах пользователя. С этой точки зрения система должна: быть хорошоорганизованной, надежной и служить достоверным хранилищем информации (базой данных) об объектах данной предметной области; предоставить пользователю всю необходимую информацию об этих объектах; дать пользователю возможность просто и удобно поддерживать базу данных в актуальном состоянии и контролировать это состояние. Создание системы, удовлетворяющей этим требованиям, и является конечной целью проектирования системы.

Создание прикладной информационной системы административного типа представляет собой сложную задачу, требующую участия большого числа соответствующих специалистов - разработчиков системы. Однако, это не значит, что пользователи

системы могли бы остаться в стороне. Как и в других задачах проектирования пользователь должен определить - что делать?, в то время как разработчик системы должен ответить на вопрос - как делать? В этой связи решающую роль должен играть пользователь, а не разработчик системы.

Основными исходными вопросами, которые нужно решить при участии пользователя, являются следующие:

(1) Информация о каких объектах предметной области должна храниться в БД системы?

(2) Какую информацию об этих объектах следует хранить в БД?

(3) Какие функциональные связи должны устанавливаться между данными и какие задачи нужно решать в системе?

2.2. База данных и вся информационная система в целом являются в определенном смысле моделью предметной области [2]. Но следует учитывать, что сама предметная область, также как организация пользователей постоянно изменяется и развивается. Ясно, что в результате этого соответствующим образом должна изменяться и развиваться также и информационная система, а поэтому она уже в принципе не может быть завершенной. Это, как нам представляется, необходимо учитывать сразу с того момента, когда принимается решение о создании такой системы.

В процессе создания и развития можно выделить несколько стадий. Опираясь на работу [12] мы выделим пять стадий (образующих цикл, указанный на рис. 2) - анализ, системирование

(systemeering), реализация, внедрение, эксплуатация (в [12] первая из указанных стадий отдельно не выделяется).

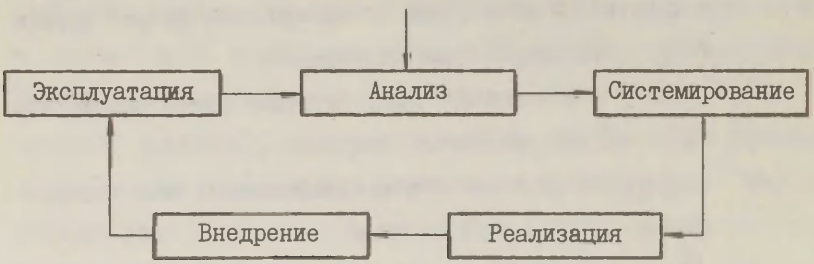


Рис. 2.

При первоначальном создании системы анализ заключается в выявлении требований пользователей к создаваемой системе.

Системирование является основным этапом проектирования, в ходе которого определяется информационная и функциональная структура разрабатываемой версии системы. Для этого в [12] предлагается т.н. информационно-логический подход (infological approach). Согласно этому, на наш взгляд весьма интересному и плодотворному подходу, в системировании выделяются две фазы, которые мы называем семантической и логической фазой соответственно (у автора [12] эти фазы именуются соответственно infological и datalogical). В семантической фазе на концептуальном уровне определяются информационные и функциональные зависимости в рассматриваемой версии системы. В логической фазе определяется структура базы данных (на языке описания данных) и составляются программы обработки (на под-

держиваемых данной СУБД языках программирования).

В стадии реализации создается или реорганизуется БД, компилируются и отлаживаются программы обработки. Далее уже следует внедрение и эксплуатация.

Процесс заканчивается анализом результатов внедрения и эксплуатации и сравнением их с предъявленными требованиями. Этот анализ в то же время является началом нового цикла, теперь уже цикла усовершенствования или реконструкции системы. Такими итерациями добиваются максимальной адаптации информационной системы к изменяющимся условиям в предметной области и в организации пользователей.

Описанный итеративный подход к созданию и расширению информационной системы требует применения методов проектирования и средств генерации системы, обеспечивающих достаточно быстрое и оперативное выполнение итераций. Основой такого метода, на наш взгляд, может стать уже упомянутый нами информационно-логический подход [12], а средством генерации - современные СУБД. Можно сказать, что СУБД и создаются для того, чтобы обеспечить быстрое создание, внедрение и развитие интегрированных информационных систем вообще и баз данных особенно.

2.3. Ядром информационной системы несомненно является база данных. Поэтому, первичной задачей проектирования информационной системы является проектирование БД.

В [12] база данных рассматривается как результат информационного отображения некоторой предметной области (системы по [12]) или подобласти. Но сам процесс отображения рассмат-

ривается состоящим из четырех компонентных отображений, каждое из которых заканчивается созданием некоторой модели. На рис. 3, заимствованном (в упрощенной форме) из [12], стрелками указываются компоненты отображения, а ромбоидами - модели. Компоненты отображения и соответствующие им результаты

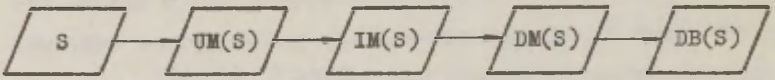


Рис. 3.

(модели) следующие (S - предметная система):

1. Построение модели, соответствующей представлениям пользователя о предметной области; получается UM(S) - user model.
2. Семантическое проектирование; получается IM(S) - infological model.
3. Логическое проектирование; получается DM(S) - datalogical model.
4. Реализация и внедрение; получается DB(S) - data base.

Представление проектирования БД в виде отображения является одной из основных положений информационно-логического подхода [12]. Но нам представляется, что такой подход с успехом можно распространить и на проектирование функциональной структуры информационной системы. Объектом отображения в этом случае являются задачи обработки данных, решение которых пользователь требует от системы. В таком случае со-

держание компонентов отображения (моделей) следующие (Т - задача обработки):

1. Представление пользователя о задаче - UM(T).
2. Описание задачи и связанных с ней информационных структур в виде функционального соотношения - IM(T).
3. Формальное описание задачи (точнее программы ее решения) на соответствующем входном языке (манипулирования данными) СУБД и/или на поддерживаемых СУБД языках программирования - DM(T).
4. Получение отлаженной программы, реализующую данную задачу - P(T).

2.4. Процесс проектирования прикладной системы в общих чертах схематически изображается на рис. 4. Эта схема разработана по примеру аналогичной схемы, приведенной в [11]. Оттуда же заимствована на наш взгляд весьма удачная нотация описания процесса проектирования. Согласно этой нотации прямоугольниками отображается сам процесс проектирования (по отдельным этапам или фазам), а ромбоидами - информация, производимая в ходе проектирования. Текстом внутри прямоугольников сокращенно указывается задача соответствующей фазы процесса проектирования, а внутри ромбоидов - к чему относится соответствующая информация. Стрелками на схеме указывается, какую роль - входную или выходную - играет изображенная ромбоидами информация по отношению к фазам процесса проектирования.

В процессе проектирования информационной системы нами выделяется четыре фазы:

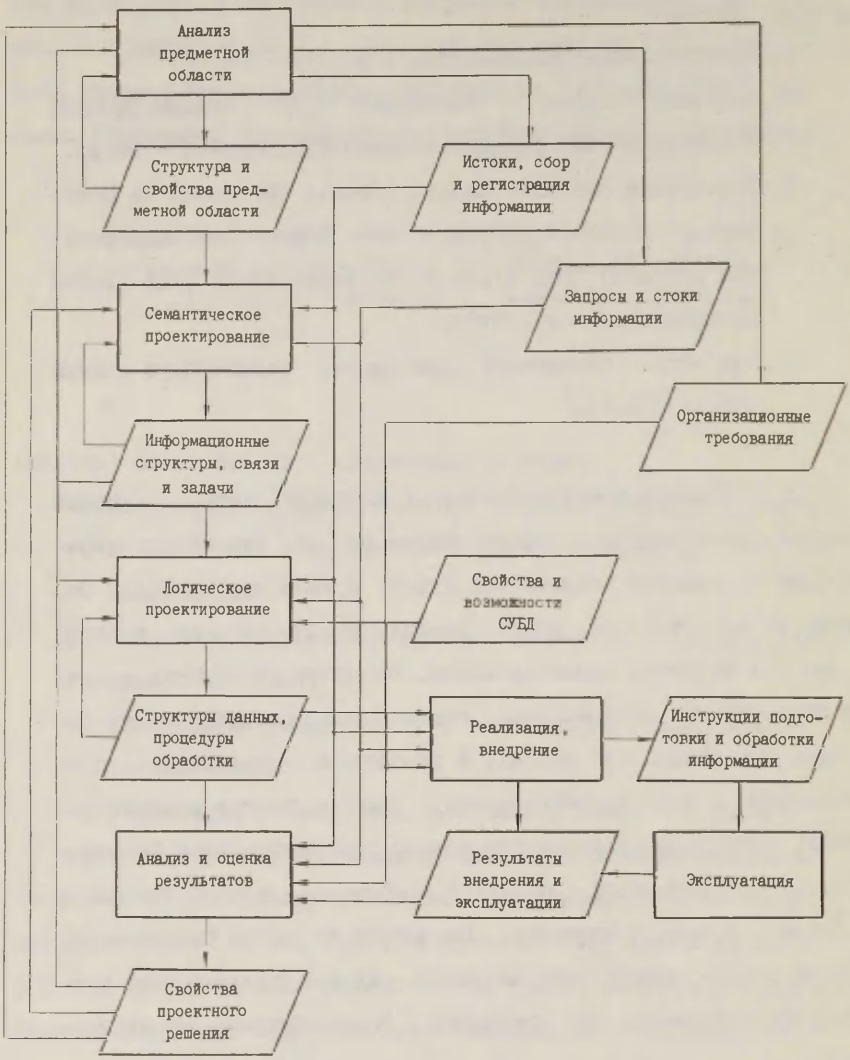

Анализ предметной области
Структура и свойства предметной области
Истоки, сбор и регистрация информации
Семантическое проектирование
Запросы и стоки информации
Информационные структуры, связи и задачи
Организационные требования
Логическое проектирование
Свойства и возможности СУБД
Структуры данных, процедуры обработки
Реализация, внедрение
Инструкции подготовки и обработки информации
Анализ и оценка результатов
Результаты внедрения и эксплуатации
Эксплуатация
Свойства проектного решения

Рис. 4.

1. Анализ объектной системы, в результате которого определяются:
   - структура, свойства, информационные и функциональные связи в предметной области;
   - истоки и условия сбора и регистрации исходной информации для информационной системы;
   - информация, запрашиваемая потребителями от системы;
   - требования к организации обработки информации в системе (сроки и частота выполнения задач, функции пользователя БД и пр.).
2. Семантическое проектирование, т.е. проектирование в семантических (содержательных) категориях, целью которого является выделение информационных объектов, определение их признаков, отображаемых в БД, информационных (структурных) и функциональных связей между ними, а также перечня и сущности задач обработки (поддерживающих и прикладных).
3. Логическое проектирование, т.е. проектирование в логических (формальных) категориях, целью которого является определение логической структуры базы и спецификация поддерживающих базу задач на соответствующих входных языках инструментальной системы и программирование прикладных (проблемных) задач. Исход этой фазы определяется свойствами и возможностями СУБД.
4. Анализ и оценка результатов проектирования.

Дополнительно к этому на схеме (рис. 4) отдельными блоками изображаются внедрение и эксплуатация разработанной системы. В ходе реализации и внедрения системы отлаживается

программное обеспечение, составляются инструкции по кодированию информации, заполнению и оформлению входных форм, перфорации (ввода) данных из них, а также организации контроля над состоянием БД и процессом выполнения задач.

Нетрудно заметить, что схема на рис. 4 является результатом компиляции, детализации и некоторой модификации схем, приведенных на рис. 2 и 3.

Литература

1. Савинков В.М. (редактор), Сборник "Алгоритмы и организация решения экономических задач", 1974-75, вып. 4,5,6.
2. Глушков В.М., Проблемы ОГАС на современном этапе. Тот же сборник, вып. 6.
3. Беззаботнов Ю.И., Артамонов Б.А., Вопросы организации информационного обслуживания АСУ. Там же, вып. 5.
4. Столяров Г.К., Обзор предложений Рабочей группы КОДАСИЛ по базам данных. Там же, вып. 4.
5. Фридлендер Ф.Л., Мамаев Э.А., Основные принципы построения СУБД НАБОБ. Там же, вып. 5.
6. Брусенков И.В., Михалкин В.Г., Пурвин Ю.В., Окулов М.С., Программная система управления базами данных СИНБАД. Там же, вып. 6.
7. Столяров Г.К., Дрибас В.П., Основные требования к банкам данных. Управляющие системы и машины, 1974, № 2.

8. CODASYL Systems Commitee, Feature Analysis of Generalized Data Base Management Systems, Report, May, 1971.

9. Date, C.J., An Introduction to Data Base Systems. Addison-Wesley, 1975.

10. Lundeberg, M., Bubenko, J. jr. (Eds), Systemeering 75, Studentlitteratur, Lund, 1975.

11. Bubenko, J.A., Some Theoretical and Practical Observations in a Data Base Design Case. Systemeering 75, Studentlitteratur, Lund, 1975.

12. Sundgren, B., Using the Infological Approach in the Design of Data Bases. Systemeering 75, Studentlitteratur, Lund, 1975.

Содержание

ТРУДЫ ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА. Выпуск 35. На русском языке. Тартуский государственный университет. ЭССР, г. Тарту, ул. Юликооли, I8. Ответственный редактор Ю. Тапфер. Бумага ротаторная 30x42 I/4. Печ. листов 3,75 (условных 3,49). Учетно-изд. листов 2,42. Тираж 300. МВ 04051. Типография ТГУ. ЭССР, г.Тарту, ул. Пялсони, I4. Зак. № 372. Цена I6 коп.

I6 коп.