Букачев Д.С. О реализации абстрактной граф-машины // Международный журнал информационных технологий и энергоэффективности. – 2017. – Т. 2 № 3(5) с. 14-22

Международный журнал информационных технологий и энергоэффективности

Сайт журнала: http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.42

О РЕАЛИЗАЦИИ АБСТРАКТНОЙ ГРАФ-МАШИНЫ

Букачев Д.С.

ФГБОУ ВО Смоленский государственный университет, Смоленск, Россия (21400, г. Смоленск, ул. Пржевальского, 4), e-mail: dsbuka@yandex.ru

Статья посвящена вопросам оптимизации разработки программного обеспечения. В работе предложена конструкция и описание интерфейса абстрактной граф-машины, которая позволяет строить динамические графовые модели в любой предметной области независимо от типов данных, сопоставляемых элементам графа. В качестве примера в статье приведена типонезависимая реализация алгоритма Дейкстры построения оптимального маршрута между вершинами нагруженного графа.

Ключевые слова: универсальная алгебраическая система, абстрактная граф-машина, объектно-ориентированный подход, алгоритм Дейкстры.

ABOUT IMPLEMENTATION OF THE ABSTRACT GRAPH MACHINE

Bukachev D.S.

Federal State Educational Institution of Higher Education Smolensk State University, Smolensk, Russia (21400, Smolensk, street Przewalski, 4), e-mail: dsbuka@yandex.ru

Article is devoted to questions of optimization of software development. In operation construction and interface description of graphs machine by the abstract which allows to build dynamic graph models in any data domain irrespective of the data types compared to elements of a graph is offered. Implementation of an algorithm of Dijkstra of creation of an optimum route, independent of data type, between tops of the loaded graph is given as an example in article.

Key words: the universal algebraic system, the abstract graph machine, object-oriented approach, Dijkstra's algorithm.

Достаточно часто возникает ситуация, когда модели задач, решаемых в системах обработки информации и управления, принципиально отличаются лишь тем, что они оперируют понятиями (и, как следствие, типами данных) различных предметных областей. Это делает возможным использование единой фундаментальной модели. Иными словами, существует естественная необходимость использования так называемой абстрактной алгебраической системы [4], которая позволит решать определенный класс задач вне зависимости от их принадлежности к конкретной предметной области.

Обращение к алгебраическим моделям не случайно, так как они позволяют строить формализованные модели процессов обработки данных и решать задачи оптимизации этих процессов. В основу построения алгебраических моделей положены понятия универсальной

алгебры и универсальной алгебраической системы, известные определения которых приведены ниже [3, 4].

Пусть A – некоторое непустое множество. Частично определенная функция $y = F(x_1, \ldots, x_n), (y, x_1, \ldots, x_n) \in A$, называется *n-арной частичной операцией* на *A*. Если функция всюду определена, говорят просто об *n*-арной операции.

Система $U_A =< A, \Omega >$, состоящая из основного множества *A* и определенной на нем совокупности частичных операций $\Omega = \{F_s^{n_s}\}(s = 1, 2, \ldots)$, называется *частичной универсальной алгеброй* с сигнатурой Ω.

Универсальные алгебры U_A и U_B , в которых заданы соответственно сигнатуры Ω и Ω', называются *однотипными*, если можно установить такое взаимно-однозначное соответствие между сигнатурами Ω и Ω', при котором любая операция $\omega \in \Omega$ и соответствующая ей операция $\omega' \in \Omega'$ будут *n*-арными с одним и тем же *n*.

Пусть даны две однотипные универсальные алгебры $U_A =< A, \Omega >$ и $U_B =< B, \Omega >$ с основными множествами *A* и *B*.

Отображение φ: $A \to B$ называется *гомоморфным отображением* алгебры U_A в алгебру U_B, если для любых элементов $a_1, \ldots, a_n \in A$ и произвольной *n*-арной операции $F \in \Omega$ выполняется соотношение

$$\varphi(F(a_1, \ldots, a_n) = F(\varphi(a_1), \ldots, \varphi(a_n)),$$

где $\varphi(a_i) = b_i$ и $b_i \in B(i = 1,..., n)$. Алгебры U_A и U_B называются гомоморфными.

Пусть $\pi(x_1,..., x_n), x_1,..., x_n \in A$ – *n*-местный предикат, $\Pi = \{\pi_s^{n_s}\}(s = 1, 2, \ldots)$ – сигнатура предикатов. Система $\boldsymbol{U}_A = < A; \Omega; \Pi >$ называется *универсальной алгебраической системой*.

Случаи, когда можно обойтись единственным основным множеством, встречаются нечасто, поскольку объекты предметной области, как правило, являются сложными системами, для описания которых используется много разнотипных параметров. Для моделирования таких предметных областей используются многоосновные алгебры.

Система $U_{\boldsymbol{M}} =< \boldsymbol{M}; \Omega >$, состоящая из семейства основных множеств $\boldsymbol{M} = \{A_\alpha\}$ ($\alpha = 1, 2, \ldots$) и сигнатуры Ω операций, определенных на семействе $\boldsymbol{M}$ так, что каждая *n*-арная операция из Ω является отображением декартова произведения *n* множеств из семейства $\boldsymbol{M}$ в множество из того же семейства $A_{\alpha_1} \times \ldots \times A_{\alpha_n} \to A_{\alpha_r}$ называется *многоосновной алгеброй*.

Система $\boldsymbol{U}_{\boldsymbol{M}} =< \boldsymbol{M}; \Omega; \Pi >$, где Π – сигнатура *n*-местных предикатов $\pi : A_{\alpha_1} \times \ldots \times A_{\alpha_n} \to \{0,1\}$, называется многоосновной алгебраической системой.

Многоосновные алгебры и алгебраические системы играют важную роль в программировании. Они служат основой для создания пользовательских типов данных в современных языках программирования, ими также являются некоторые встроенные в языки типы данных. В частности, в [4] показано, что строковый тип данных, системы числовых матриц и нагруженных графов являются многоосновными алгебраическими системами.

На многоосновных алгебраических системах базируется современное теоретическое и практическое программирования. *Абстрактные типы данных* (они же *объекты* или *классы*), сокращенно АТД, определяются как многоосновные алгебраические системы.

Среди множества произвольных абстрактных типов данных выделяется один специфический вид. Эти АТД представляют собой двухосновные алгебраические системы вида $\boldsymbol{E}=<S, T; \Omega; \Pi>$. Основу S назовем *структурой*, а T – *типом*.

Важная особенность этих АТД состоит в том, что суть операций над элементами структуры S, не изменяется при изменении сути операций над элементами типа T. Такие АТД будем называть *абстрактными алгебраическими машинами* (ААМ). В работе [4] предлагается механизм построения абстрактной матричной машины, реализующей типовые операции над матрицами вне зависимости от типов данных элементов матриц, а также приводится листинг класса *TAbstractMatrixMachine* с подробным описанием его методов. На основе класса *TAbstractMatrixMachine* строится класс-наследник *TBooleanMatrixMachine*, определяющий тип элементов матриц и описывающий операции над элементами.

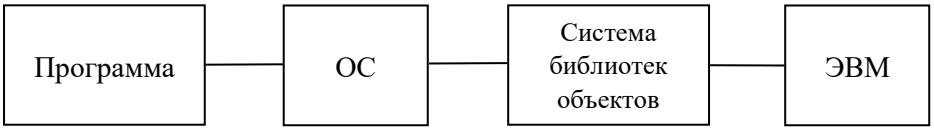
Абстрактные алгебраические машины составляют основу для решения задач оптимизации алгоритмов, реализующих процессы обработки данных. Оптимизация становится возможной благодаря отделению операций структуры от операций над элементами типа. При этом качество алгоритмов, реализующих операции структуры, не зависит от качества алгоритмов, реализующих операции типа.

Многие современные прикладные задачи часто бывает удобно формализовать с использованием теории графов. Такой подход позволяет, оперируя терминами этой теории, применять в ходе исследования известные алгоритмы на графах (см., например, [1]). Решая конкретную задачу, программист каждый раз должен:

1) описать структуры данных, которые он собирается использовать применительно к конкретной ситуации;
2) реализовать стандартные операции добавления/удаления элементов графа;
3) при необходимости модифицировать и реализовать стандартные алгоритмы на графах, используемые в процессе решения задачи.

Поскольку базовые операции над графами (такие как добавление/удаление элементов), а также реализация основных алгоритмов на графах фактически не зависят от характера данных, сопоставляемых элементам графа, и операций над этими данными, можно говорить о возможности создания такой алгебраической структуры, которая позволит работать с любыми графовыми моделями [2, 4].

Под абстрактной граф-машиной (АГМ) будем понимать ЭВМ и связанную с ней систему библиотек объектов:



В работе [2] рассмотрены основные принципы построения АГМ. В настоящей статье предлагается один из способов реализации АГМ на базе связных списков (см., например, [1]). Такой подход позволит использовать АГМ для создания динамических графовых моделей.

Листинг 1. Описание класса TVGraph (АГМ)

```
type
  TGrNodePtr=^TGrNode;                  //указатель на элемент списка вершин
  TListAdr=^TListNode;                  //указатель на элемент списка связей (рёбер)
  TGrNode=record
```

```
          N:integer;                //номер вершины в графе
          VData:pointer;            //указатель на данные в вершине
          Next:TGrNodePtr;          //указатель на следующую вершину
          Links:TListAdr;           //список связей
        end;

TListNode=record
           Node:TGrNodePtr;         //связь
           VWeight:pointer;         //ссылка на вес ребра
           Next:TListAdr;           //следующая связь
           VMark:pointer;           //ссылка на метку
         end;
 TRegIndex=1..3;                    //индексы регистров АГМ
 TObjectProc=procedure of object;
 TBoolFunc=function:boolean of object;
 TVGraph=class (TObject)
          private
            Graph:TGrNodePtr;       //список вершин
            FCount:integer;         //счетчик вершин
            FDVD:boolean;           //значение DestroyVirtualData
            //вспомогательные методы
            function GetCount:integer;
            function GetDVD:boolean;
            procedure SetDVD(Value: boolean);
          public
            RegResult:TRegIndex; //индекс регистра результата
            Regs:array[TRegIndex] of pointer;          //регистры данных
            VDCreate,VWCreate,VMCreate:TObjectProc;    //конструкторы данных
            VDDestroy,VWDestroy,VMDestroy:TObjectProc; //деструкторы данных
            //алгебра "абстрактных данных" вершин
            VDZero,VDAdd:TObjectProc;
            //алгебра "абстрактных весов"
            VWZero,VWInfinity,VWAdd:TObjectProc;
            VWOrder:TBoolFunc; //абстрактное отношение порядка
            //алгебра "абстрактных меток"
            VMZero,VMAdd:TObjectProc;
            property Count:integer read GetCount; //счетчик вершин
            //уничтожение (да/нет) связанных данных при удалении
            //элементов графа (вершин, рёбер)
            property DestroyVirtualData:boolean read GetDVD
                                        write SetDVD default true;
            //создание и уничтожение экземпляра АГМ
            constructor Create;
            destructor Destroy;
            procedure Reset; //сброс: p(G)->0, q(G)->0
            //инициализация деструкторов
            procedure InitConstructors(iVDCreate,iVWCreate,
                                       iVMCreate:TObjectProc);
            procedure InitDestructors(iVDDestroy,iVWDestroy,
                                      iVMDestroy:TObjectProc);
            //инициализация операций над данными вершин, рёбер(дуг), меток
            procedure InitVDataAlgebr(iVDZero,iVDAdd:TObjectProc);
            procedure InitVWeightAlgebr(iVWZero,iVWInf,iVWAdd:TObjectProc;
                                        iVWOrder:TBoolFunc);
            procedure InitVMarkAlgebr(iVMZero,iVMAdd:TObjectProc);
```

```
    //методы АГМ
    //операции с вершинами
    procedure AddNode(iData:pointer);
    procedure DelNode(Num:integer);
    function  GetNode(iNum:integer):TGrNodePtr;
    function  GetNodeProp(iNum:integer):pointer;
    procedure SetNodeProp(iNum:integer; iData:pointer);
    //операции с рёбрами/дугами
    function  YesLink(inN,toN:integer):boolean;
    procedure AddLink(inNum,toNum:integer; Double:boolean);
    procedure DelLink(inNum,toNum:integer; Double:boolean);
    function  GetLink(inNum,toNum:integer):TListAdr;
    function  GetLinkList(iNum:integer):TListAdr;
    function  GetLinkProp(inNum,toNum:integer):pointer;
    function  GetLinkMarker(inNum,toNum:integer):pointer;
    procedure SetLinkProp(inNum,toNum:integer; iWeight:pointer);
    //операции с метками
    procedure SetLinkMarker(inNum,toNum:integer; iMark:pointer);
    procedure ResetLinkMarker(ResetValue:pointer=nil);
    procedure ResetLink;
    //вычисление степени вершины графа
    function  DegIn(iNode:integer):integer;
    function  DegOut(iNode:integer):integer;
    function  DegDouble(iNode:integer):integer;
    function  Deg(iNode:integer):real;
  end;{TVGraph}
```

Класс не хранит и не обрабатывает данные, сопоставляемые элементам графа, в явном виде. Для работы с абстрактными данными класс содержит три регистра-указателя, абстрактные конструкторы, деструкторы, абстрактные операции сложения, умножения, присваивания нейтрала, абстрактное отношение порядка на множестве весов.

Описание классических алгоритмов на графах (волновой алгоритм, алгоритм Дейкстры, поиск циклических маршрутов, алгоритм Прима-Краскала, тест на эйлеровость) с абстракцией относительно типов данных элементов графа предлагается реализовать в виде отдельных классов. В работе [4] в качестве примера типонезависимого алгоритма приводится алгоритм Дейкстры поиска кратчайшего расстояния между вершинами нагруженного графа. В данной статье рассмотрим реализацию метода *Way* класса *TDijkstra*, реализующего данный алгоритм на базе класса *TVGraph* (АГМ).

Листинг 2. Описание метода Way класса TDijkstra

```
function TDijkstra.Way(inNode, toNode: integer; var WayArr: TIntArray): pointer;
type
  TDRec=record
          outNode:integer;  //номер вершины
          Len:pointer;      //указатель на длину пути до текущей вершины
          ConstLb:boolean;  //метка
        end;
var
  YL:boolean;
  DArr:array of TDRec;
  i,k,Lb:integer;
  e:pointer;
begin
```

```
WayArr:=nil;
//начальное заполнение массива DArr
if G.Count>0 then
  begin
    SetLength(DArr,G.Count);
    for i:=1 to G.Count do
      if i=inNode then       //если текущая вершина = стартовая
        with DArr[i-1] do
          begin
            outNode:=inNode; //запоминаем номер
            G.RegResult:=1;  //устанавливаем номер регистра-результата
            G.VWCreate;      //выделяем место для веса в регистре
            G.VWZero;        //обнуляем вес в регистре
            Len:=G.Regs[1];  //запоминаем указатель на длину пути
            ConstLb:=true;   //помечаем вершину
          end
      else                   //в противном случае
        with DArr[i-1] do
          begin
            outNode:=inNode; //запоминаем номер
            G.RegResult:=2;  //устанавливаем номер регистра-результата
            G.VWCreate;      //выделяем место для веса в регистре
            if G.YesLink(inNode,i) then //если есть ребро (дуга) inNode->i
              begin
                G.VWZero;    //обнуляем вес в регистре
                //получаем указатель на вес ребра (дуги) inNode->i
                G.Regs[1]:=G.GetLinkProp(inNode,i);
                if G.Regs[1]<>nil then
                  G.VWAdd; //складываем значения в регистрах 1 и 2
              end
            else //если нет ребра (дуги) inNode->i
              //устанавливаем в регистре-результате значение «бесконечность»
              G.VWInfinity;
            Len:=G.Regs[2]; //запоминаем указатель на длину пути
            ConstLb:=false; //деактивируем метку
          end;
    //вычисляем расстояние до финальной вершины
    while not DArr[toNode-1].ConstLb do
      begin
        Lb:=0;
        //находим первую непомеченную вершину в массиве DArr
        while DArr[Lb].ConstLb do
          Inc(Lb);
        //выбираем непомеченную вершину, наиболее близкую к стартовой
        for i:=Lb to (G.Count-1) do
          if not DArr[i].ConstLb then
            begin
              G.Regs[1]:=DArr[i].Len;
              G.Regs[2]:=DArr[Lb].Len;
              if ((G.Regs[1]=nil) and (G.Regs[2]<>nil)) or
                 ((G.Regs[1]<>nil) and (G.Regs[2]<>nil) and G.VWOrder) then
                Lb:=i;
            end;
        //активируем метку
        DArr[Lb].ConstLb:=true;
```

```
    Inc(Lb);
    //если не добрались до конечной вершины
    if Lb<>toNode then
      //пересчитываем расстояние до всех непомеченных вершин
      G.VWAdd;
      for i:=0 to (G.Count-1) do
        if not DArr[i].ConstLb then
          if G.YesLink(Lb,i+1) then
            begin
              e:=G.GetLinkProp(Lb,i+1);
              G.RegResult:=3;  G.VWCreate;
              G.Regs[1]:=DArr[Lb-1].Len;  G.Regs[2]:=e;
              if G.Regs[2]<>nil then
                G.VWAdd
              else
                begin
                  G.RegResult:=2; G.VWCreate;  G.VWZero;
                  G.RegResult:=3; G.VWAdd;
                  G.RegResult:=2; G.VWDestroy; G.RegResult:=3;
                end;
              G.Regs[1]:=G.Regs[3];  G.Regs[2]:=DArr[i].Len;
              //если пересчитанная длина через вершину Lb меньше текущей
              if G.VWOrder then
                //обновляем указатель на длину пути
                begin
                  DArr[i].outNode:=Lb;  G.RegResult:=1;
                  G.Regs[1]:=DArr[i].Len;
                  if G.Regs[1]<>nil then
                    G.VWDestroy;
                  DArr[i].Len:=G.Regs[3];
                end
              else
                G.VWDestroy;
            end;
  end;
G.RegResult:=2;  G.VWCreate;  G.VWInfinity;
G.Regs[1]:=DArr[toNode-1].Len;
//если маршрут до финальной вершины не построен
if not G.VWOrder then
  begin
    G.VWDestroy;  Result:=nil;
  end
else //если путь до финальной вершины существует
  begin
    //формируем массив, содержащий вершины маршрута
    G.VWDestroy; k:=0; i:=toNode;
    while i<>inNode do
      begin
        Inc(k); i:=DArr[i-1].outNode;
      end;
    if k>0 then
      begin
        Result:=DArr[toNode-1].Len;  SetLength(WayArr,k+1);  i:=toNode;
        repeat
          WayArr[k]:=i; i:=DArr[i-1].outNode; Dec(k);
```

```
            until (k<0);
          end
        else
          Result:=0;
      end;
    G.RegResult:=1;
    //освобождаем память
    for i:=1 to G.Count do
      if (i<>toNode) or (Result=nil) then
        begin
          G.Regs[1]:=DArr[i-1].Len;  G.VWDestroy;
        end;
    Finalize(DArr);
  end
 else
   Result:=nil;
end;{Way}
```

Предложенный вариант реализации алгоритма Дейкстры оперирует регистрами АГМ и абстрактными методами работы с регистрами, следовательно, не зависит от типа данных, используемых в качестве весовых коэффициентов рёбер (дуг) графа.

Процесс создания реальной граф-машины состоит в следующем: программисту необходимо построить класс-наследник, описать алгебры используемых данных, после чего инициализировать виртуальную машину уже описанными методами, реализующими алгебры данных (например, $< R_+ \cup \{0\}, \cdot >$ – алгебра неотрицательных действительных чисел с обычной операцией сложения, которая может быть использована для работы со взвешенными графами).

Предложенный вариант реализации абстрактной граф-машины на базе связных списков позволяет строить динамические графовые модели и решать типовые задачи теории графов независимо от типов данных, сопоставляемых элементам графа. Для работы в рамках определенной предметной области требуется лишь описать алгебру элементов и инициализировать АГМ. Данная оптимизация стала возможной благодаря отделению операций структуры от операций над элементами типа.

Список литературы

1. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. – М.: Издательский дом «Вильямс», 2000. – 384 с.
2. Букачев Д.С., Мунерман В.И. О принципах реализации виртуальных алгебраических машин. //Системы компьютерной математики и их приложения: Материалы международной конференции. – Смоленск, СмолГУ, 2006. – с. 55-56.
3. Бурбаки Н. Теория множеств. – М.: Мир, 1968.
4. Левин Н.А., Мунерман В. И. Алгебраический подход к оптимизации обработки информации. – Системы и средства информатики. Спецвыпуск. Математические модели и методы информатики, стохастические технологии и системы. Москва: ИПИ РАН 2005. – с. 279-294.

References

1. Aho A., Hopkroft Dzh., Ulman Dzh. Struktury dannyh i algoritmy. – M.: Izdatel'skij dom «Vil'jams», 2000. – 384 s.
2. Bukachev D.S., Munerman V.I. O principah realizacii virtual'nyh algebraicheskih mashin. //Sistemy komp'juternoj matematiki i ih prilozhenija: Materialy mezhdunarodnoj konferencii. – Smolensk, SmolGU, 2006. – s. 55-56.
3. Burbaki N. Teorija mnozhestv. – M.: Mir, 1968.
4. Levin N.A., Munerman V. I. Algebraicheskij podhod k optimizacii obrabotki informacii. – Sistemy i sredstva informatiki. Specvypusk. Matematicheskie modeli i metody informatiki, stohasticheskie tehnologii i sistemy. Moskva: IPI RAN 2005. – c. 279-294.