ТЕОРЕТИЧНІ ОСНОВИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 51.681.3

Національний університет ім. Т.Шевченко
С. Л. Кривий

Абстрактні типи даних як многоосновні алгебраїчні системи I

Досліджуються властивості абстрактних типів даних, зображених у вигляді многоосновних алгебраїчних систем. Детально розглядаються такі абстрактні типи даних: стеки, черги, черги з пріоритетами, бінарні дерева та множини.

Рассматривается изображение абстрактных типов данных в виде многоосновных алгебраических систем. Детально рассматривается такие абстрактные типы данных: стэки, очереди, очереди с пріоритетами, бинарные деревья и множества.

The image of abstract data types is examined as manysorted algebraic systems. The abstract data tape as stack, queues, priority queues, binary trees and set are examined in details.

Ключові слова: абстрактні типи даних, многоосновні алгебри, алгебраїчні системи, повнота.

.

Вступ. Абстрактні типи даних (АТД) активно використовуються в процесі розробки програмного забезпечення, його обґрунтуванні та верифікації. АТД «натуральне число» та «список» розглядалися в роботі автора [1]. Дана робота є продовженням розгляду АТД, які часто зустрічаються в практичних застосуваннях. До цих АТД належать черги, черги з пріоритетами, стеки та множини. Стеки та черги являють собою окремі випадки списків і тому логічно інтерпретувати їх реалізацію у вигляді списків [2,3,4].

1. Алгебраїчна система АТД «стек»

Стеком називається список, операції додавання і вилучення елементів якого виконуються на одному з кінців списку, який визначається за допомогою операції *top*. А вся множина операцій в літературі називаються *push*, *pop* і *top*. Крім цих бінарної і двох унарних операцій визначається і нульарна операція *emptySt* - пустий стек, а також унарний предикат *is-emp* для перевірки пустоти стека. Алгебраїчна система «стек» має вигляд:

$$A = ((X, F(X), error), \Omega = \{push, pop, top, emptySt\}, \Pi = \{true, false, is\text{-}emp, =_l, =_n\}),$$

де

- *emptySt* - пустий стек,
- $push : X \times F(X) \to F(X)$,
- $pop, top : F(X) \to F(X) \cup \{error\}$,
- $is\text{-}emp : F(X) \to \{true, false\}$.

Формальне означення списка «стек» має вигляд.

Означення 1. Стеком називається список, який побудований за такими правилами:

а) *emptySt* - є стек,

б) якщо *p* - стек, то *push(x,p), pop(p), top(p)* - стек,

в) список, побудований не за правилами а) і б), не є стеком.

Означення 2. Формулами називаються вирази, побудовані за такими правилами:

а) *true, false* - формули,

б) якщо *p* - стек, то *is-emp(p)* - формула,

в) вирази, побудовані не за правилами а) і б), не є формулами.

Аксіоматика поняття ``стек'' має такі аксіоми:

$$is\text{-}emp(emptySt) =_n true$$

$$is\text{-}emp(push(x,p)) =_n false,$$

$$pop(push(x,p)) =_l p,$$

$$pop(emptySt) =_l error,$$

$$top(emptySt) =_l error,$$

$$top(push(x,p)) =_l x.$$

Те, що АС «стек» є окремим випадком АС «список» випливає з такої інтерпретації:

стек – список,
emptySt - e,
is-emp – isemp,
push – $\cdot$,
pop – tail,
top – head.

Доведення правильності цієї інтерпретації досить просте. Наприклад, доведення правильності третьої і шостої аксіом має вигляд:

$$pop(push(x,p)) = tail(x\cdot p) = p,$$
$$top(push(x,p)) = head(x\cdot p) = x.$$

Приклад 1. (Обчислення значень виразів, записаних в оберненій польській нотації). Нехай дано вираз *abc+ –*, значення якого потрібно знайти. Алгоритм обчислення значення виконує такі дії: спочатку за допомогою операції *push* в стек заносяться значення *a,b,c*; потім значення *b* і *c* за допомогою операції *pop* вилучаються із стека і заноситься в стек значення суми *(b+c)*; потім значення *a* і *b+c* вилучаються із стека і заноситься значення різниці *a-(b+c)*. Припустимо, що є функція *val*, яка отримує операцію і два операнди і повертає значення цієї операції на цих операндах.

Цей алгоритм, який назвемо *post* має вигляд (*p* означає стек):

$$post(e,p) = top(p)$$
$$post(x\cdot t,\ p) = \text{if } x \text{ is an argument then } post(t,push(x,p))$$
$$\text{else } (x \text{ is an operator})\ post(t,eval(x,p)),$$

де

$$eval((op,push(a,push(b,p))) = push(val(b,op,a),p).$$

Наприклад, обчислення значення виразу *post((2,5,+),e)* має вигляд:

$$post((2,5,+),e) = post((5,+),(2)) = post(+,(5,2)) = post(e,(eval(+,(5,2))) =$$
$$= top(eval(+,(5,2))) = top(push(val(2,+,5),e)) = val(2,+,5) = 7.\ \spadesuit$$

2. Алгебраїчна система АТД «черга»

Чергою називається список, в якому операції додавання елементів виконуються на одному кінці, а операції їх вилучення - на другому кінці списка. Основні операції, які визначаються на чергах, в літературі називаються *add, del* і *front*. Крім цих бінарної і двох унарних операцій визначається і нульарна операція *emptyQ* – пуста черга, а також унарний предикат *is–empQ* для перевірки пустоти черги. Алгебраїчна система «черга» має вигляд:

$$A = ((X, F(X),\{true, false\}, error),\ \Omega = \{add, del, front, emptyQ\},\ \Pi = \{true, false, is\text{-}empQ\}),$$

де
$$emptyQ \text{ - пуста черга},$$
$$add : X \times F(X) \to F(X),$$
$$del : F(X) \to F(X) \cup \{error\},$$
$$front : F(X) \to X \cup \{error\}$$
$$is\text{-}emptyQ : F(X) \to \{true, false\}.$$

Аксіоматика операцій і предикатів цієї АС має вигляд:

$$is\text{-}emptyQ(emptyQ) = true,$$
$$is\text{-}emptyQ(add(x,p)) = false,$$
$$front(add(x,p)) = \text{if } is\text{-}emptyQ(p) \text{ then } x \text{ else } front(p),$$
$$del(add(x,p)) = \text{if } is\text{-}emptyQ(p) \text{ then } p \text{ else } add(x,del(p)),$$
$$del(emptyQ) = error,$$
$$front(emptyQ) = error.$$

Якщо інтерпретувати чергу як список, то має місце така відповідність:

$$p \in F(X) = список,$$
$$emptyQ = e,$$
$$is\text{-}emptyQ = isemp,$$
$$front = head,$$
$$del = tail,$$
$$add = putlast.$$

Доведення коректності цієї інтерпретації більш цікаве, ніж для стеків. Це пов'язано з тим, що аксіоматика черги включає дві умовні аксіоми і операцію *putlast*, яка визначається в термінах списків.

Третя аксіома при застосуванні до пустої черги *emptyQ* в даній інтерпретації виглядає так:

$$front(add(x,emptyQ)) = head(emptyQ\cdot x) = head(x) = x.$$

Якщо вона застосовується до непустої черги *p*, то

$$front(add(x,p)) = head(putlast(x,p)) = head(px) = head\ (p) = front(p)$$

Приклад 2.

а) Застосуємо засоби АС «черга» до означення функції *app(p,q)*, яка з'єднує дві черги *p* і *q* в одну чергу.

$$app(p,q) = \text{if } is\text{-}emptyQ(q) \text{ then } p \text{ else } app(add((front(q),p)),del(q))$$

Наприклад, якщо $p = ab$, $q = cd$, то

$$app(p,q) = app(ab,cd) = app(abc,d) = app(abcd,emptyQ) = abcd.\spadesuit$$

Неважко довести за допомогою математичної індукції такі властивості функцій *app* і *add:*

$app(x, add(y,q)) = add(y,app(x,q))$,
$app(p, app(q,r)) = app(app(p,q),r)$,
де $x, y \in X$, $p,q,r \in F(X)$.

б) Скористаємося АС «черга» до означення функції *bin(n)* перетворення натурального числа $n \in N$ з десяткового зображення до двійкового зображення. Наприклад, $bin(4) = 100$, тобто старший розряд знаходиться на початку черги. Означення цієї функції має вигляд:

$$bin(n) = \text{if } isz(n) \vee n =_n 1 \text{ then } add(n,emptyQ) \text{ else } add(mod(n,2),\ bin(floor(n,2))).$$

Наприклад, зображення числа 11 двійковим словом у вигляді черги за допомогою функції *bin* виглядає таким чином:

$$bin(11) = add(mod(11,2), bin(floor(11,2))) = add(1,bin(5)) =$$
$$= add(1,add(mod(5,2),bin(floor(5,2)))) = add(1,add(1,bin(2))) =$$
$$= add(1,add(1,add(mod(2,2),bin(floor(2,2))))) =$$
$$= add(1,add(1,add(0,bin(1)))) = (1011).\ \spadesuit$$

3. Алгебраїчна система «черга з пріоритетами»

Черги з пріоритетами є списками, які задовольняють умові прийшов з найвищим пріоритетом, першим вийшов. Прикладами черг з пріоритетами можуть служити стеки і черги. Якщо для стека вважати, що елемент, який останнім був доданий до стека, має найвищий пріоритет, то стек перетворюється в чергу з пріоритетом. Якщо для черги вважати що її перший елемент має найвищий пріоритет, то черга стає чергою з пріоритетом.

Нехай $F(X)$ – множина черг з пріоритетами над алфавітом X, тоді основними операціями над чергами з пріоритетами є наступні:

– e – пуста черга,
– $insP : X \times F(X) \to F(X)$ - додавання елемента до черги,
– $delbest : F(X) \to F(X) \cup \{error\}$ - вилучення елемента з черги з найвищим пріоритетом,
– $best : F(X) \to X \cup \{error\}$ - видає елемент черги з найвищим пріоритетом,
– $isempP : F(X) \to \{true, false\}$ - предикат істинний тоді і тільки тоді, коли черга пуста,
– $better : X \times X \to \{true, false\}$ - предикат істинний тоді і тільки тоді, коли його перший аргумент має вищий пріоритет ніж другий.

Аксіоматика введених операцій і предикатів має вигляд:

$$isempP(e) = true,$$
$$isempP(insP(x,p)) = false,$$

$$best(insP(x, p)) = \begin{cases} x, \text{ якщо } isempP(p), \\ x, \text{ якщо } better(x, best(p)), \\ best(p), \text{ інакше.} \end{cases}$$

$$best(e) = error,$$

$$delbest(insP(x, p)) = \begin{cases} e, \text{ якщо } isempP(p), \\ p, \text{ якщо } better(x, best(p)), \\ insP(x, delbest(p)), \text{ інакше.} \end{cases}$$

$$delbest(e) = error.$$

Приклад 3. Розглянемо функцію сортування *sort(p,q)*, де p – черга з пріоритетом, виклик якої є таким *sort(p,e):*

$$sort(p,q)=\begin{cases} q, \text{ якщо } isempP(p), \\ sort(delbest(p), best(p)\cdot q), \text{ інакше.} \end{cases} \spadesuit$$

4. Алгебраїчна система АТД «бінарне дерево»

Дерева і зокрема бінарні дерева є графами спеціального вигляду. Розглянемо алгебраїчну систему АТД «бінарне дерево».

Нехай *B(U)* означає множину бінарних дерев над алфавітом *U*, де *U* - множина, з якої вибираються вершини бінарних дерев. Рекурсивне означення множини *B(U)* має вигляд.

Означення 3. Множина *B(U)* включає тільки такі елементи *T = (V,E)*, які побудовані за правилами:

а) *emptyT* – пусте бінарне дерево,

б) якщо $v \in U$, то *(emptyT, v, emptyT)* – бінарне дерево, в якому єдина вершина *v* є коренем,

в) якщо $l=(V_1,E_1)$, $r=(V_2,E_2)$ – бінарні дерева, коренями яких є вершини v_1^0 і v_2^0 відповідно і $v \in U$, $v \notin V_1$ і $v \notin V_2$, то *(l,v,r)* – бінарне дерево з коренем *v* і новими ребрами (v,v_1^0), (v,v_2^0).

З цього означення очевидним чином випливають такі властивості бінарних дерев: якщо

T = (V,E) $\in$ *B(U)*, то

а) *T* – ациклічний зв'язний граф,

б) кожна вершина дерева *T* досяжна з кореня v, причому цей шлях єдиний,

в) корінь *v* не досяжний з жодної вершини дерева *T*.

Основні операції і предикати, які визначаються на множині *B(U)*, є такими:

emptyT – пусте дерево,

root : B(U) $\rightarrow$ *U* $\cup$ *{error}*- корінь дерева,

tree : B(U) $\times$ *U* $\times$ *B(U)* $\rightarrow$ *B(U)* – дерево,

left : B(U) $\rightarrow$ *B(U)* $\cup$ *{error}* – ліве піддерево,

right : B(U) $\rightarrow$ *B(U)* $\cup$ *{error}* – праве піддерево,

isempT : B(U) $\rightarrow$ *{true, false}.*

Аксіоматика введених операцій і предикатів має вигляд:

isempT(emptyT) $=_n$ *true,*

isempT(tree(l,v,r)) $=_n$ *false,*

left(tree(l,v,r)) = l,

right(tree(l,v,r)) = r,

root(tree(l,v,r)) = v,

left(emptyT)=right(emptyT)=root(emptyT) = error.

Користуючись рекурсією, операціями та предикатами, введемо ще дві функції: функцію *count : B(U)* $\rightarrow$ *N*, значенням якої є кількість вершин бінарного дерева, і функцію *depth : B(U)* $\rightarrow$ *Z*, значенням якої є довжина найдовшого шляху з кореня бінарного дерева до його листків. Означення цих функцій є досить простим:

$$count(T)=\begin{cases} 0, \text{ якщо } isempT(T), \\ 1+count(left(T))+count(right(T)), \text{ інакше.} \end{cases}$$

$$depth(T)=\begin{cases} -1, \text{ якщо } isempT(T), \\ 1+\max(depth(left(T)), depth(right(T)), \text{ інакше.} \end{cases}$$

Приклад 4. Розглянемо функцію *inord*, яка при обході бінарного дерева *T* заносить його вершини до черги *Q*, тобто

inord : B(U) $\rightarrow$ *Q(U).*

Алгебраїчний вираз для цієї функції в термінах основних операцій і предикатів має вигляд:

$$inord(T)=\begin{cases} emptyQ, \text{ якщо } isempT(T), \\ app(add(root(T), inord(left(T))), inord(right(T))), \text{ інакше.} \end{cases}$$

де *app* - функція з'єднання двох черг в одну, яка була визначена в прикладі 2 ♠.

5. Алгебраїчна система АТД «множина»

Одним з основних є АТД «множина», оскільки множини є тією базовою структурою, яка лежить в основі як математики, так і інформатики. В зв'язку з цим при розробці алгоритмів множини виступають як основа багатьох АТД і тому багато методів і технічних прийомів грунтується саме на множинах. Базовими операторами, які пов'язуються з множинами і складають основу при реалізації різного роду АТД, виступають такі оператори:

1. Пуста множина *emptyset*, та *un(A,B), int(A,B), dif(A,B)* – оператори, які реалізують три основні операції на множинах *об'єднання, перетин* і *різницю* відповідно. Вхідними аргументами даних операторів є множини *A* і *B*, а результатом – множина $A \cup B$, $A \cap B$, $A \backslash B$ відповідно.
2. Оператор *merge(A,B)* називається злиттям множин-аргументів *A* і *B*, які не перетинаються (тобто $A \cap B = \varnothing$), і генерує множину-результат $A \cup B$. Цей оператор відрізняється від оператора *un* тим, що результат його виконання є невизначений, коли множини *A* і *B* мають спільні елементи.
3. Предикат *memb(x, A)* має аргументами множину *A* і елемент *x* того ж типу, що і елементи множини *A*. Результатом обчислення цього предиката буде *true*, якщо $x \in A$ і *false*, якщо $x \notin A$.
4. Оператори *ins(x, A)* і *del(x, A)* виконують відповідно додавання і усування з множини *A* елемента *x* того ж самого типу, що і елементи множини *A*. Тобто, $ins(x, A) = A \cup \{x\}$ і $del(x, A) = A \backslash \{x\}$. Зрозуміло, що коли $x \in A$, $ins(x, A) = A$ і коли $x \notin A$, то $del(x, A) = A$.
5. Предикат *eq(A,B)* приймає значення *true*, якщо $A = B$ і *false*, якщо $A \neq B$.
6. Оператор *find(x)*, визначений на сукупності множин, що попарно не перетинаються, дає множину, елементом якої є *x*. Якщо такої множини немає, то значенням цього оператора є 0.
7. Предикат *isemp(A)* приймає значення *true*, якщо множина *A* є пустою, і значення *false* в протилежному випадку.
8. Потужність множини *A:* $|A|$ – кількість елементів множини *A*.
9. Декартовий добуток множин *A* і *B* – $A \times B = \{(a,b) \mid a \in A, b \in B\}$.

В термінах цих операцій симетрична різниця $\div$ для множин записується у вигляді виразу

$$A \div B = un(dif(A,B),dif(B,A)).$$

5.1 АТД «множина» як аксіоматична теорія

Серед розглянутих вище операцій і предикатів, які визначені на множинах, виділяються три операції: об'єднання, перетин і різниця множин. Розглянемо алгебру множин над деякою універсальною множиною *U* як аксіоматичну теорію.

Означення 4 Елементами множини *Un* є ті і тільки ті елементи, які побудовані за такими правилами:
а) пуста множина є елементом *Un*
б) довільна одноелементна множина $\{a\}$, де $a \in Un$ є елементом множини *Un*,
в) якщо *A* і *B* є елементами *Un*, то $un(A,B), int(A,B), dif(A,B), dif(Un,A) = A''$ (доповнення множини *A*) є елементами *Un*,
г) ніяких інших елементів, крім тих, які побудовані за правилами а) - в), немає.

Аксіоматика введених операцій має вигляд: для довільних $A, B, C \in Un$

$un(A,B) = un(B,A);$
$int(A,B) = int(B,A);$
$un(un(A,B),C = un(A,un(B,C));$
$int(int(A,B),C) = int(A,int(B,C));$
$un(A,int(B,C)) = int(un(A,B),un(A,C));$
$int(A,un(B,C)) = un(int(A,B),int(A,C));$
$un(A,emptyset) = A;$
$int(A, Un) = A;$
$un(A,A') = U;$
$int(A,A') = emptyset,$

де $A' = dif(Un,A)$ - доповнення множини A в множині *Un*.

Поповнимо дану алгебру предикатами *true, false, memb, eq, isemp* та традиційними булевими операціями над ними $\neg, \vee, \wedge$. Після такого поповнення отримуємо таку алгебраїчну систему для цього АТД:

$$A = (C = (U, \{true, false\}), error), \ \Omega = \{un, int, dif, find, \neg, \vee, \wedge\}, \ \Pi = \{true, false, memb, eq, isemp\}),$$

де *Un* – універсальна множина, над якою розглядаються множини.

Аксіоматика предикатів має вигляд: для довільних $A,B,C \in Un$

$eq(A,A) = true$ (рефлексивність),
$eq(A,B) = eq(B,A)$ (симетричність),
$eq(A,B) \wedge eq(B,C) \Rightarrow eq(A,C)$ (транзитивність),
$eq(A,B) \Rightarrow eq(un(A,C),un(B,C))$(правило підстановки),
$eq(A,B) \Rightarrow eq(int(A,C),int(B,C))$ (правило підстановки),
$memb(a, emptyset) = false,$
$memb(a,un(a,A)) = true,$
$isemp(emptyset) = true,$
$isemp(un(a,A)) = false.$

Користуючись основними предикатами даної АС, введемо відношення включення для множин:

$A \subseteq B \Leftrightarrow eq(un(A,B),B) \vee eq(int(A,B),A).$

Відомо, що введене відношення включення є відношенням часткового порядку і тоді отримуємо таку АС:

$$A = (C = (U, \{true, false\}, error),\ \Omega = \{un, int, dif, find, \neg, \vee, \wedge\},\ \Pi = \{true, false, memb, eq, isemp, \subseteq\}).$$

Для того, щоб переконатися у коректності даної аксіоматики, скористаємося її інтерпретацією на АС «список». Множини за даної інтерпретації зображуються у вигляді списка своїх елементів. Якщо A, B - множини із Un, то нехай p_A і p_B означають списки елементів множин A і B відповідно. Тоді маємо відповідність:

$eq(A,B) \Leftrightarrow eq1(p_A, p_B) =$ if $p_A =_l p_B$ then *true* else *false;*

$memb(a, A) \Leftrightarrow memb(a, p_A)$; $isemp(A) \Leftrightarrow isemp(p_A);$

$un(A,B) \Leftrightarrow con1(p_A, p_B) =$ if $isemp(p_B)$ then p_A else
if $memb(head(p_B), p_A)$ then $con1(p_A$,tail(p_B) else
$con1(putlast(head(p_B), p_A), tail(p_B))$;

$int(A,B) \Leftrightarrow con2(p_A, p_B, p_C) =$ if $isemp(p_A)$ then p_C else
if $memb(head(p_A), p_B))$ then $con2(tail(p_A), p_B, head(p_A) \cdot p_C)$ else
$con2(tail(p_A), p_B, p_C)$;

(Перший виклик *con2* виконується за пустого списка p_C).

$ins(a,A) \Leftrightarrow ins1(a, p_A) =$ if $memb(a, p_A)$ then p_A else $a \cdot p_A$;

$del(a,A) \Leftrightarrow del1(a, p_A) =$ if $memb(a, p_A)$ then
if $head(p_A) =_l a$ then $tail(p_A)$ else $head(p_A) \cdot del1(a, tail(p_A))$;

$dif(A,B) \Leftrightarrow dif1(p_A, p_B) =$ if $memb(head(p_B), p_A)$ then
$dif1(del(head(p_B), p_A), tail(p_B))$else $dif1(p_A, tail(p_B))$;

$merge(A,B) \Leftrightarrow merge1(p_A, p_B) =$ if $isemp(p_B)$ then p_A else
if $memb(head(p_B), p_A))$ then *error* else
$merge1(putlast(head(p_B), p_A), tail(p_B))$;

$find(a, A_1, A_2, ..., A_n) =$ if $memb(a, A_1) = true$ then A_1 else
if $memb(a, A_2) = true$ then A_2 else
…… *else*
if $memb$(a, A_n) = *true* then A_n else 0;

$|A| = l(p\_A);$

$A \times B \Leftrightarrow$ dec(p_A, p_B),

де

$$dec(p,q) = \begin{cases} empty,\ якщо\ isemp(p), \\ empty,\ якщо\ isemp(q), \\ (dec1(head(p), q)), dec(tail(p), q)), \end{cases}$$

а *dec1(x,q)*, має вигляд

$$dec1(x,q) = \begin{cases} empty,\ якщо\ isemp(q), \\ ((x, head(q)), dec1(tail(q))),\ інакше. \end{cases}$$

5.2 Методи реалізації множин

Найчастіше при зображенні множин в пам'яті комп'ютера використовуються два способи: зображення характеристичними векторами та зв'язними (упорядкованими) списками.

Реалізація характеристичними векторами. Нехай $A=\{a_1, a_2, ..., a_n\}$ – деяка універсальна *n*-елементна множина. Поставимо у відповідність елементам множини *A* елементи $\{1,2,..., n\}$, а кожній підмножині $B \subseteq A$ характеристичний вектор $v(B) = (v_1, v_2, ..., v_n)$, де

$$v_i = \begin{cases} 1, \text{ якщо } a_i \in B, \\ 0, \text{ якщо } a_i \notin B. \end{cases}$$

Тоді пустій множині відповідатиме нульовий характеристичний вектор, а універсальній множині *A* – характеристичний вектор, всі координати якого дорівнюють 1.

Основна перевага такого подання множин полягає в тому, що оператори *memb, ins* і *del* виконуються у фіксованому часі, який не залежить від потужності множини *A*, шляхом прямої адресації до відповідного біта. Очевидно, що пам'ять, необхідна для такої реалізації, пропорціональна потужності множини *A*, що є недоліком такої реалізації. Операції *un, int* і *dif* виконуються в часі *O(|A|) = O(n)*. Якщо потужність універсальної множини *A* не перевищує довжини машинного слова, то ці оператори реалізуються на характеристичних векторах за допомогою покомпонентних булевих операцій $\vee, \wedge, \neg$.

Дійсно,

якщо $B, C \subseteq A$, то $\upsilon(B \cup C) = \upsilon(B) \vee \upsilon(C)$,

$\upsilon(B \cap C) = \upsilon(B) \wedge \upsilon(C)$,

$\upsilon(B \setminus C) = \upsilon(b) \wedge \neg\upsilon(C)$,

Підсумкова таблиця часової складності реалізації операторів цього АТД має вигляд:

Таблиця 1 Часові характеристики операцій

Операція	Час реалізації	Операція	Час реалізації
un	*0(n)*	*int*	*0(n)*
dif	*0(n)*	*memb*	*0(1)*
ins	*0(1)*	*del*	*0(1)*
merge	*0(n)*	*empty*	*0(n)*
eq	*0(n)*	*find*	*0(n)*

Реалізація зв'язними списками. При реалізації множини за допомогою зв'язних списків, елементи списка являють собою елементи множини. На відміну від попередньої реалізації за допомогою характеристичних векторів, при такій реалізації необхідна пам'ять пропорціональна потужності відповідних підмножин, а не потужності універсальної множини *A*. Крім того, множини при такому зображенні не обов'язково мусять бути підмножинами деякої універсальної множини.

Для реалізації операції *int(A,B)* (перетин) при таких структурах даних є декілька альтернатив. Якщо універсальна множина лінійно упорядкована, то в цьому випадку множини можна подати у вигляді посортованих списків. Це означає, що елементи $a_1, a_2, ..., a_n$ будуть знаходитися в списку в порядку $a_1 \le a_2 \le ... \le a_n$. Перевага посортованого списка полягає в тому, що для пошуку конкретного елемента немає необхідності переглядати всі елементи списка.

У випадку посортованих списків, які мають не більше *n* елементів, реалізація операції *int(A,B)* виконуються в часі *O(n)*, оскільки елементи списків переглядаються лише один раз.

У випадку непосортованих списків, які мають не більше *n* елементів, реалізація операції *int(A,B)* виконуються в часі $O(n^2)$, оскільки потребує *n* кратного перегляду елементів списків.

Аналогічні оцінки справедливі і для операцій *un(A, B)* і *dif(A, B)*.

Для реалізації оператора присвоювання необхідна операція *copy* – операція копіювання елементів списка. Рекурсивне означення цієї операції має вигляд:

$$copy(p) = \text{ if } l(p) =_n 1 \text{ then } p \text{ else } copy(head(p)) \cdot copy(tail(p)).$$

Тепер можна ввести і сам оператор присвоювання *assign*. Оператор *assign(p,q)* копіює список *p* в список *q*.

Зазначимо, що цей оператор не можна реалізувати простим переозначенням заголовка списка *q* на заголовок списка *p*, оскільки при подальших змінах в списку *q* необхідно буде робити аналогічні зміни в списку *p*, що може призвести до помилок.

Підсумкова таблиця для операторів цього АТД у випадку реалізації посортованими списками наведена нижче.

Таблиця 2 Часові характеристики операцій

Операція	Час реалізації	Операція	Час реалізації
un	*0(n)*	*int*	*0(n)*
dif	*0(n)*	*memb*	*0(n)*
ins	*0(1)*	*del*	*0(1)*
merge	*0(n)*	*empty*	*0(n)*
eq	*0(n)*	*Find*	*0(n)*
assing	*0(n)*		

Описані два способи реалізації АТД «множина» – це найпростіші методи зображення цього АТД. Існують спеціальні методи подання множин такі, як дерева двійкового пошуку, зважені дерева, збалансовані дерева тощо. Ці подання детально описані в монографіях [5,6], де зацікавлений читач може з ними познайомитися ближче. Зокрема, для операцій типу *union-find* відповідні структури даних добре описані в монографії [6] і тому тут ці структури не розглядаються. Зауважимо тільки, що деякі додаткові властивості цих структур і їх реалізацію в мові програмування C++ можна знайти в книзі [7].

Заключне слово

На завершення зазначимо, що описані алгебраїчні системи можуть породжувати окремі випадки алгебраїчних систем та спеціальні способи реалізації. Наприклад, якщо відомо, що над чергами виконуватимуться лише операції модифікації елементів без їх додавання і усування, то це накладає умови на їх властивості і реалізацію. Оскільки рекурсія не завжди дає самий ефективний алгоритм, то можливість вибору способу реалізації дає певну свободу і часто ця свобода приводить до побудови ефективних алгоритмів. В загальному випадку задана алгебраїчна система деякого АТД породжує підсистеми, які виникають в результаті обмеження сигнатури операцій чи предикатів. Ці обмеження складають основу властивостей предметної області, на яких грунтується побудова ефективного алгоритму і відповідної програми.

Список літератури

1. *Кривий С. Л.* Абстрактні типи даних як многоосновні алгебраїчні системи. ж. Інженерія програмного забезпечення. - 2010. - N 3. - С. 5-18.
2. *Кривий С.Л.* Дискретна математика: вибрані питання. Київ : Видавничий дім "Києво-Моги-лянська академія. - 2007. - 570 с.
3. *Фостер Дж.* Обработка списков. - М. : Мир. - 1979. - 535 с.
4. *Hein J.L.* Discrete Mathematics - Sudbury, Massachusetts: Jones and Bartlett Publishers: -1995. -656 p.
5. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов - М. : Мир. - 1979. - 535 с.
6. *Ахо А., Хопкрофт Дж., Ульман Дж.* Структуры данных и алгоритмы - М.: Издат. дом «Вильямс» - 2000. - 382 с.
7. *Седжвик Р.* Фундаментальные алгоритмы на C++ Части 1--4. - М.: DiaSoft, 2002. – 687 с.

Відомості про автора

Кривий Сергій Лук'янович – професор кафедри інформаційних систем Національного університету ім. Т.Шевченко, докт.фіз.-мат.наук, професор. Наукові інтереси: дискретна математика, теорія автоматів, мережі Петрі, методи розробки та верифікація програмного забезпечення.

E-mail: krivoi@i.com.ua

Стаття надійшла до редакції 05.11.2010 р.