ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Τεχνολογίες Υλοποίησης Αλγορίθμων

Χρήστος Ζαρολιάγκης

Καθηγητής
Τμήμα Μηχ/κων Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
email: zaro@ceid.upatras.gr

Εισαγωγή στη Βιβλιοθήκη Λογισμικού LEDA – 1

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο **«Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών»** έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Περιεχόμενα

- Τί είναι η LEDA ;
- Τύποι Δεδομένων και Προδιαγραφές τους
- Βασικοί Σχεδιαστικοί Κανόνες στη LEDA

Η Βιβλιοθήκη LEDA

- Ευκολία στη χρήση (αλγόριθμος + LEDA = Πρόγραμμα).
- Επεκτασιμότητα.
- Ορθότητα.
- Αποδοτικότητα.
- Αποδοτική Διαχείριση Μνήμης.

Παράδειγμα 1

Πρόβλημα:

- Μέτρηση διπλοεγγραφών σε μία δεδομένη ακολουθία σειρών χαρακτήρων.

Παράδειγμα 1 (συνέχεια)

Η λύση μέσω της LEDA:

```
#include <LEDA/core/d_array.h>
using namespace leda;
using namespace std;

main()
{
  d_array<string,int> N(0);
  string s;
  while ( true )
  {
     cin >> s;
     if ( s=="end") break;
     N[s]++;
  }
  forall_defined (s,N)
    cout << s << " " << N[s] << endl;
}
```

Παράδειγμα 2

Πρόβλημα: Υλοποίηση αλγορίθμου Dijkstra (ψευδοκώδικας):

Algorithm Dijkstra
$S = 0; \bar{S} = V;$
for every $u \in V$ **do** $d(u) = \infty;$
$d(s) = 0;$
while $|S| < n$ **do**
    (*vertex selection*)
    let $u \in \bar{S}$ be the vertex for which $d(u) = min_{u \in \bar{S}}\{d(u)\};$
    $S = S \cup \{u\};\ \bar{S} = \bar{S} - \{u\};$
    (*distance update*)
    for each $(u, w) \in E$ **do**
        if $d(w) > d(u) + wt(u, w)$ **then**
            $d(w) = d(u) + wt(u, w);$
    od
od

Παράδειγμα 2 (συνέχεια)

```
#include <LEDA/graph/graph.h>
#include <LEDA/graph/node_pq.h>
using namespace leda;

void dijkstra(graph& G, node s,
              const edge_array<double>& cost,
              node_array<double>& dist)
{
    node_pq<double> PQ(G);
    node v; edge e;

    forall_nodes(v,G) {
       if (v==s)
          dist[v]=0;
       else
          dist[v] = MAXDOUBLE;
       PQ.insert(v,dist[v]);
    }
    while (! PQ.empty()) {
        node u = PQ.del_min();
        forall_out_edges(e,u) {
            v = G.target(e);
            double c = dist[u] + cost[e];
            if (c < dist[v]) {
                PQ.decrease_p(v,c);
                dist[v] = c;
            }
        }
    }
}
```

Τύποι Δεδομένων στη LEDA

- *Τύπος Δεδομένων T:*
  - $val(T)$: σύνολο *τιμών (values)*
  - $obj(T)$: σύνολο *αντικειμένων (objects)* (επώνυμα, ανώνυμα), ή *στοιχείων δεδομένων (items)* (πρωτογενή (ατομικά), μη πρωτογενή (σύνθετα))
  - σύνολο συναρτήσεων που μπορούν να εφαρμοσθούν στα αντικείμενα του τύπου T για δημιουργία, διαγραφή, χειρισμό και μετασχηματισμό τους.

- Όταν αναφερόμαστε στις τιμές ενός τύπου χωρίς αναφορά σε συγκεκριμένο αντικείμενο, θα χρησιμοποιούμε επίσης τους όρους *συνολοστοιχεία (elements)* ή *στιγμιότυπα (instances)*.

  Π.χ. ο αριθμός 5 είναι μια τιμή, ένα συνολοστοιχείο, ή ένα στιγμιότυπο του τύπου `int`.

Τύποι Δεδομένων στη LEDA

1. **Ορισμός (definition):** περιγραφή ορισμού των στιγμιοτύπων του τύπου T.
2. **Δημιουργία (creation):** περιγραφή της δημιουργίας ενός αντικειμένου του τύπου T.
3. **Λειτουργίες (operations):** περιγραφή ορισμού των διαφόρων λειτουργιών (συναρτήσεων) που είναι διαθέσιμες για αντικείμενα τύπου T.
4. **Υλοποίηση (implementation):** πληροφορίες σχετικές με την υλοποίηση του τύπου δεδομένων.

Stacks (stack)

1. Definition

An instance S of the parameterized data type *stack*<E> is a sequence of elements of data type E, called the element type of S. Insertions or deletions of elements take place only at one end of the sequence, called the top of S. The size of S is the length of the sequence, a stack of size zero is called the empty stack.

2. Creation

stack<E> S;	creates an instance S of type *stack*<E>. S is initialized with the empty stack.

3. Operations

E	S.top( )	returns the top element of S. *Precondition*: S is not empty.
void	S.push(E x)	adds x as new top element to S.
E	S.pop( )	deletes and returns the top element of S. *Precondition*: S is not empty.
int	S.size( )	returns the size of S.
bool	S.empty( )	returns true if S is empty, false otherwise.
void	S.clear( )	makes S the empty stack.

4. Implementation

Stacks are implemented by singly linked linear lists. All operations take time $O(1)$, except clear which takes time $O(n)$, where n is the size of the stack.

Τύπος list

- Ένα στιγμιότυπο του τύπου list<E> είναι μια ακολουθία στοιχείων λίστας (προκαθορισμένου τύπου list_item). Κάθε στοιχείο της λίστας περιέχει ένα συνολοστοιχείο του τύπου E.
- Οι περισσότεροι τύποι της LEDA (π.χ. στοίβες, πίνακες, λίστες, σύνολα) είναι παραμετρικοί.
- Εμπράγματοι τύποι (concrete types) $\Leftarrow$ συγκεκριμενοποίηση των παραμετρικών τύπων.

```
list<int>
array<string>
stack<set<int> * >    // stack of pointers to
                      // sets of integers
```

Δημιουργία αντικειμένων

- *Επώνυμα αντικείμενα* ($\sim$ ορισμός μεταβλητών της C++)
  - string s;
    ορίζει μια μεταβλητή s τύπου string και την αρχικοποιεί με την κενή σειρά χαρακτήρων.
  - stack<E> S;
    ορίζει μια μεταβλητή S τύπου stack και την αρχικοποιεί με την κενή στοίβα.
  - list<E> L;
    ορίζει μια μεταβλητή L τύπου list και την αρχικοποιεί με την κενή λίστα.
  - array<E> A(int l, int u);
    ορίζει μια μεταβλητή A τύπου array<E> και την αρχικοποιεί με μια 1-1 συνάρτηση $a : [l, u] \rightarrow obj(E)$. Κάθε αντικείμενο του πίνακα αρχικοποιείται με την προεπιλεγμένη αρχικοποίηση των συνολοστοιχείων του τύπου E.
  - int i;
    ορίζει μια μεταβλητή i τύπου int και την αρχικοποιεί με κάποια τιμή τύπου int.

Παραδείγματα αρχικοποίησης αντικειμένων

```
string s("abc");          // intialized to "abc"

set<int> S;               // initialized to the empty
                          // set of integers

array<string> A(2,5);     // index set of A is [2..5];
                          // each entry is initialized
                          // to the empty string

b_stack<int> S(100);      // a bounded-size stack capable
                          // of storing a maximum of 100
                          // ints; initialized to the
                          // empty stack
```

Κανόνας αρχικοποίησης

Γενικά, η εντολή

```
T<T1,T2,...,Tk> y(x1,x2,...,xl)
```

ορίζει μια μεταβλητή `y` τύπου `T<T1,T2,...,Tk>` που χρησιμοποιεί τα ορίσματα `x1,x2,...,xl` για τον προσδιορισμό της αρχικής τιμής της `y`.

Ορισμοί μεταβλητών με:

- προεπιλεγμένη αρχικοποίηση
- αρχικοποίηση αντιγράφου

Ορισμός με προεπιλεγμένη αρχικοποίηση

- δεν έχει ρητά ορίσματα
- η αρχικοποίηση πραγματοποιείται με μια προεπιλεγμένη τιμή
- η προεπιλεγμένη τιμή είναι συνήθως η «απλούστερη» τιμή ενός τύπου, π.χ. 0, κενή σειρά χαρακτήρων, κενό σύνολο, κλπ.
- *Στη LEDA η προεπιλεγμένη τιμή ορίζεται στο τμήμα που περιγράφει τη «Δημιουργία» του τύπου.*

Παρατήρηση: οι ενσωματωμένοι τύποι (π.χ. char, bool, int , float , double) και όλοι οι τύποι δεικτών δεν έχουν προεπιλεγμένη τιμή αρχικοποίησης.

Ορισμός με αρχικοποίηση αντιγράφου

`T<T1,T2,...,Tk> y(x)`

ή

`T<T1,T2,...,Tk> y = x`

ορίζει μια μεταβλητή `y` τύπου `T<T1,T2,...,Tk>` και την αρχικοποιεί με ένα αντίγραφο της τιμής της μεταβλητής `x` (που είναι του ίδιου τύπου).

```
stack<int> P(S);  // intialized to a copy of S
set<string> U(V); // intialized to a copy of V
string s = t;     // intialized to a copy of t
int i = j;        // intialized to a copy of j
int h = 5;        // intialized to a copy of 5
```

LEDA - Κανόνας 1

Σε κάθε τύπο της LEDA διατίθεται ένας ορισμός αρχικοποίησης μέσω αντιγράφου. Κάθε αντικείμενο που δημιουργείται μπορεί να αρχικοποιείται με ένα αντίγραφο που δίνεται ως όρισμα.

Τρόπος δημιουργίας αντικειμένων

- *Ανώνυμα αντικείμενα*
  - δημιουργούνται με τον τελεστή new

    new `T<T1,T2,...,Tk> (x1,x2,...,xl)`

    π.χ. `stack<int> * sp = new stack<int> ;`
    ορίζει μια μεταβλητή δείκτη και δημιουργεί ένα ανώνυμο αντικείμενο τύπου `stack<int>`. Η στοίβα αρχικοποιείται με την κενή στοίβα και η μεταβλητή `sp` με έναν δείκτη σε αυτή τη στοίβα.
  - καταργούνται με τον τελεστή `delete`

Λειτουργίες

- Ορισμός μιας λειτουργίας
  - ορισμός της διασύνδεσης (interface)
  - ορισμός της σημασιολογίας (semantics) της λειτουργίας

Διασύνδεση

- σύνταξη δηλώσεων συναρτήσεων:
  - `void U.insert(E x)`
    ορίζει τη διασύνδεση της λειτουργίας `insert` για τον τύπο `set<E>`.
  - `E& A[int i]`
    ορίζει τη διασύνδεση της λειτουργίας προσπέλασης για τον τύπο `array<E>`.
  - `E S.pop()`
    ορίζει τη διασύνδεση της λειτουργίας `pop` για τον τύπο `stack<E>`.
  - `int s.pos(string s1)`
    ορίζει τη διασύνδεση της λειτουργίας `pos` για τον τύπο `string`.

Σημασιολογία (semantics)

- ορίζεται με χρήση μαθηματικών εννοιών και συμβολισμών (που συνήθως εισάγονται στα τμήματα του «Ορισμού» και της «Δημιουργίας» του τύπου).

void	U.insert(E x)	adds x to U.
E&	A(int i)	returns the variable $A(i)$. *Precondition:* $a \leq i \leq b$.
E	S.pop()	removes and returns the top element of S. *Precondition:* S is not empty.
int	s.pos(string s1)	returns -1 if $s1$ is not a substring of s and returns i, $0 \leq i \leq$ `s.length()` $- 1$, such that $s1$ occurs as a substring of s starting at position i, otherwise.

Προϋπόθεση (precondition) εφαρμογής μιας λειτουργίας

- Ορίζει ποιές συνθήκες (προϋποθέσεις) πρέπει να πληρούν τα ορίσματα μιας λειτουργίας. Π.χ. η λειτουργία `pop()` εφαρμόζεται μόνο σε μια μη κενή στοίβα.
- Παραβίαση προϋπόθεσης $\Rightarrow$ οτιδήποτε μπορεί να συμβεί.
- *Έλεγχος προϋποθέσεων στη LEDA:*
  Κάποιες φορές γίνεται και κάποιες όχι. Π.χ. γίνεται έλεγχος κενότητας στοίβας και αριθμοδείκτη πίνακα εκτός ορίων. Αλλά στην κλήση `D.inf(it)` δεν γίνεται έλεγχος αν το στοιχείο `it` ανήκει στο λεξικό `D`, διότι αυξάνεται σημαντικά ο χρόνος απόκρισης της λειτουργίας.

  Γενικά: *Οι προϋποθέσεις δεν ελέγχονται αν αλλάζουν τον ασυμπτωτικό χρόνο απόκρισης μιας λειτουργίας.*

  Οι έλεγχοι προϋποθέσεων μπορούν να απενεργοποιηθούν με χρήση της ακόλουθης επιλογής μεταγλωττιστή:
  `-DLEDA_CHECKING_OFF`

Τελεστής Καταχώρησης και Προαιρετικά Ορίσματα

- Ο τελεστής καταχώρησης `T& operator=(const T&)` παρέχεται από κάθε τύπο `T` της LEDA.
- προαιρετικό όρισμα (optional argument): παίρνει μια προεπιλεγμένη τιμή στην προδιαγραφή της λειτουργίας.
- Η C++ επιτρέπει μόνο τα τελευταία (πιο δεξιά) ορίσματα να ορισθούν ως προαιρετικά.

```
list_item L.insert(E x, list_item it, int dir = after)
```

εισάγει το `x` πριν (`dir == before`) ή μετά (`dir == after`) το στοιχείο `it` της `L`. Η προεπιλεγμένη τιμή της `dir` είναι `after`, δηλ.

L.insert(x,it)
L.insert(x,it,after)
} ισοδύναμα

Υλοποίηση Τύπου

- Πληροφορίες για την υλοποίηση ενός τύπου καθώς και για την πολυπλοκότητα χρόνου και χώρου των λειτουργιών.

- Παράδειγμα: Ο τύπος list

list_item	L += E x	same as $L.append(x)$; returns the new item.
ostream&	*ostream& out* $\ll$ L	same as $L.print(out)$; returns *out*.
istream&	*istream& in* $\gg$ *list<E>& L*	same as $L.read(in)$); returns *in*.

Iteration

forall_items(it, L) { "the items of L are successively assigned to it" }

forall(x, L) { "the elements of L are successively assigned to x" }

STL compatible iterators are provided when compiled with `-DLEDA_STL_ITERATORS` (see *LEDAROOT/demo/stl/list.c* for an example).

5. Implementation

The data type list is realized by doubly linked linear lists. All operations take constant time except for the following operations: search and rank take linear time $O(n)$, item(i) takes time $O(i)$, bucket_sort takes time $O(n + j - i)$ and sort takes time $O(n \cdot c \cdot \log n)$ where c is the time complexity of the compare function. n is always the current length of the list.

Πολυπλοκότητες

- Χρονικές Πολυπλοκότητες: δεν περιλαμβάνουν τον χρόνο που απαιτείται για μεταβίβαση παραμέτρων.

  Χρόνος μεταβ. κατ' αναφορά = $O(1)$.
  Χρόνος μεταβ. κατ' αξία = χρόνος αντιγραφής ορίσματος.

- Πολυπλοκότητα Χώρου: δεν περιλαμβάνει τον επιπλέον χώρο που απαιτείται για τα συνολοστοιχεία που περιέχονται στο αντικείμενο. Περιλαμβάνει μόνο τον χώρο που απαιτείται από την ΔΔ που υλοποιεί τον τύπο.

$$\text{Επιπλέον χώρος} = \begin{cases} 0, & \text{αν το στοιχείο χωράει σε μια λέξη μηχανής} \\ \text{πραγματικός χώρος}, & \text{αλλιώς} \end{cases}$$

Υλοποίηση παραμετρικών τύπων στη LEDA

- ▷ τιμές που χωρούν σε μια λέξη μηχανής αποθηκεύονται άμεσα στη ΔΔ.
- ▷ τιμές που δεν χωρούν σε μια λέξη μηχανής, αποθηκεύονται έμμεσα στη ΔΔ μέσω δεικτών.

Πολυπλοκότητες

- Πληροφορία για τον χώρο $\Rightarrow$ ακριβής υπολογισμός των απαιτήσεων χώρου ενός τύπου.

  Π.χ. λίστα μεγέθους n: $16 + 12n$ bytes.

  $$\left.\begin{array}{l}\texttt{list<int>}\\ \texttt{list<list<int> * >}\end{array}\right\} 16 + 12n \text{ bytes}$$

  επειδή ακέραιοι και δείκτες χωρούν σε μια λέξη μηχανής.
  Όμως, μια λίστα του τύπου `list<list<int> >` απαιτεί

  $$16 + 12n + \sum_{i=1}^{n}(16 + 12n_i) \text{ bytes}$$

  όπου η i-οστη λίστα έχει n_i στοιχεία.

- Πληροφορία για χρονική πολυπλοκότητα : ασυμπτωτικός χρόνος.

Τύποι Στοιχείων Δεδομένων (Item Types)

- Στοιχεία δεδομένων: διευθύνσεις περιεχόντων τύπων (containers), δηλ. κλάσεων που περιέχουν άλλα αντικείμενα. Ουσιαστικά είναι τύποι δεικτών.

```
dic_item     // type of items in dictionaries
pq_item      // type of items in priority queues

node, edge                 // type of items in graphs
point, segment, line       // basic geometric items
```

Τύποι Στοιχείων Δεδομένων (Item Types)

- Η τιμή ενός `dic_item` είναι η διεύθυνση ενός `dic_container` και η τιμή ενός `point` είναι η διεύθυνση ενός `point_container`.

```
class dic_container
{
  K key;
  I inf;
  // ...
};

typedef dic_container * dic_item;

class point_container
{
  double x,y;
  // ...
};

typedef point_container * point;
// NOT the actual definition of point
```

Εξαρτημένοι-Ανεξάρτητοι Τύποι Στοιχείων

- **Εξαρτημένοι Τύποι Στοιχείων (εξαρτημένα στοιχεία)**

  αντιστοιχούν σε περιέχοντες τύπους που έχουν διάρκεια ζωής μόνο σαν τμήμα μιας συλλογής από περιέχοντες τύπους, π.χ. ένας dic_container υπάρχει μόνο σαν τμήμα ενός λεξικού, και ένας node_container υπάρχει μόνο σαν τμήμα ενός γραφήματος.

- **Ανεξάρτητοι Τύποι Στοιχείων (ανεξάρτητα στοιχεία)**

  είναι περιέχοντες τύποι που υπάρχουν αυτόνομα, δηλ. δεν χρειάζονται κάποιον "πατρικό τύπο" για να υπάρχουν. Π.χ. point , segment, line.

Ιδιότητες Εξαρτημένων και Ανεξάρτητων Στοιχείων

- Κατηγορήματα (attributes) στοιχείου: τιμές που αποθηκεύονται στον περιέχοντα τύπο.

  Π.χ. ένα `point` έχει *x* και *y* συντεταγμένες, ένα `dic_item` έχει ένα κλειδί και μια πληροφορία.

- Συναρτήσεις που επιτρέπουν ανάγνωση των κατηγορημάτων.

```
p.xcoord() // returns the x-coordinate of point p

s.start()  // returns the starting point of segment s

D.key(it)  // returns the key of dic_item it
           // which belongs to dictionary D
```

Υλοποίηση Στοιχείων στη LEDA

- *Εξαρτημένοι τύποι στοιχείων:* τύποι δεικτών.
  Π.χ. ο τύπος `dic_item` ορίζεται σαν `dic_container *`.

- *Ανεξάρτητοι τύποι στοιχείων:* κλάσεις των οποίων το μοναδικό μέλος δεδομένων είναι ένας δείκτης στην αντίστοιχη κλάση του περιέχοντος τύπου.

Διαχείριση μνήμης στη LEDA

- *Διαφοροποίηση:* καλύτερη διαχείριση μνήμης.
  - Ένας dic_container απελευθερώνει τη μνήμη που καταλαμβάνει αν διαγραφεί από το λεξικό που ανήκει, ή η διάρκεια ζωής του λεξικού λήξει (και τα δύο εύκολα αναγνωρίσιμα).
  - Ένας point_container απελευθερώνει τη μνήμη που καταλαμβάνει όταν δεν υπάρχει άλλο point που να δείχνει σε αυτό.

Μετρητής Αναφορών: κάθε point_container ξέρει πόσα αντικείμενα αναφέρονται σε αυτόν.

Ενθυλάκωση του δείκτη που υλοποιεί το point σε μία κλάση και επαναορισμός των τελεστών καταχώρησης και προσπέλασης.

Τελεστής Καταχώρησης

- *Όλοι οι τύποι στοιχείων παρέχουν τον τελεστή καταχώρησης (=) και ελέγχου ισότητας (==).*
  - `it1 = it2; //it1, it2 items of item type T` καταχωρεί την τιμή του `it2` στο `it1` και επιστρέφει μια αναφορά στο `it1` (ακριβώς όπως γίνεται η καταχώρηση δεικτών).

    Στην περίπτωση των ανεξάρτητων στοιχείων έχει την επιπλέον παρενέργεια της ενημέρωσης των μετρητών αναφοράς των αντικειμένων που αναφέρονται από τα `it1` και `it2`.

Τελεστής Ελέγχου Ισότητας

- `bool operator==(const& T, const& T)`
  - *Εξαρτημένα στοιχεία:* ισότητα μεταξύ τιμών (δηλ. δεικτών)
  - *Ανεξάρτητα στοιχεία:* ορίζεται διαφορετικά για κάθε τύπο.

```
point p(2.0,3.0);
point q(2.0,3.0);
p == q;                  // evaluates to true
```

Αλλά, τα `p` και `q` δεν είναι ίσα σαν δείκτες

Τα ανεξάρτητα στοιχεία είναι εφοδιασμένα με μια συνάρτηση ελέγχου ταυτόσημων τιμών (identity predicate):

```
bool identical (const& T, const& T);
identical(p,q);    // evaluates to false
```

LEDA - Κανόνας 2

(α) Για ανεξάρτητα αντικείμενα ο έλεγχος ταυτοσημότητας είναι έλεγχος ισότητας μεταξύ των τιμών τους. Ο έλεγχος ισότητας ορίζεται ξεχωριστά για κάθε τύπο στοιχείου. Συνήθως είναι ισότητα μεταξύ κατηγορημάτων.

(β) Για εξαρτημένα αντικείμενα ο έλεγχος ισότητας είναι έλεγχος ισότητας μεταξύ των τιμών τους.

Εξαρτημένοι Τύποι Στοιχείων

Συνήθως οι πολύπλοκοι τύποι δεδομένων ορίζονται σαν μια συλλογή στοιχείων.

- Π.χ. ένα στιγμιότυπο του τύπου `dictionary<K, I>` είναι μια συλλογή στοιχείων τύπου `dic_item`, κάθε ένα από τα οποία έχει ένα συσχετιζόμενο κλειδί τύπου `K` και μία πληροφορία τύπου `I`. Τα κλειδιά διαφορετικών στοιχείων είναι επίσης διαφορετικά μεταξύ τους.
- Η συλλογή των στοιχείων έχει μια συνδυαστική δομή. Π.χ. λίστες είναι ακολουθίες στοιχείων, κόμβοι και πλευρές ενός γραφήματος ορίζουν το γράφημα.
- Στοιχείο: περιέχει ≥ 0 κατηγορήματα. Ένα κατηγόρημα:
  είτε βοηθάει να ορίσουμε τη συνδυαστική δομή (γραφήματα),
  είτε συσχετίζει επιπρόσθετη πληροφορία με ένα στοιχείο (λεξικά).
- *Πως γίνεται η προσπέλαση στα στοιχεία ενός τύπου δεδομένων;*

Προσπέλαση στοιχείων

Είναι χρήσιμο να προσπελαύνουμε ένα στοιχείο (k, i) όχι μόνο μέσω του κλειδιού του k, αλλά επίσης και μέσω της θέσης που αποθηκεύεται.

- Έστω ότι θέλουμε να δούμε το στοιχείο με πληροφορία i που σχετίζεται με το κλειδί k ($\Rightarrow$ διερεύνηση στο λεξικό), να υπολογίσουμε με χρήση της i μια νέα τιμή i' και να την συσχετίσουμε με το k .
  Αυτό επιτυγχάνεται:
  είτε με μία επιπλέον διερεύνηση στη ΔΔ, είτε μπορεί να γίνει με μία απευθείας προσπέλαση αν η προηγούμενη διερεύνηση είχε επιστρέψει τη θέση αποθήκευσης του στοιχείου.
- Η δεύτερη λύση είναι πιο αποδοτική και υποστηρίζεται από την LEDA.

  $\Rightarrow$ πολλές λειτουργίες σε λεξικά και άλλους πολύπλοκους τύπους δεδομένων έχουν στοιχεία στη διασύνδεσή τους.

Αυτή η χρήση δεικτών δεν δημιουργεί κινδύνους;

Προσπέλαση στοιχείων

Όχι, γιατί η προσπέλαση στους περιέχοντες τύπους είναι περιορισμένη.

Π.χ. η προσπέλαση του κλειδιού *k* σε ένα `dic_container` είναι read-only.

Παράδειγμα

```
dic_item D.lookup(K k);
```

επιστρέφει ένα στοιχείο `it` (`nil` αν δεν υπάρχει) με κλειδί `k`.

```
I D.inf(dic_item it);
```

εξάγει την πληροφορία του στοιχείου `it`.

```
void D.change_inf(dic_item it, I j);
```

συσχετίζει μια νέα πληροφορία `j` με το στοιχείο `it`.

- Μόνο η πρώτη λειτουργία απαιτεί διερεύνηση στο λεξικό `D`, ενώ οι άλλες δύο προσπελαύνουν το στοιχείο απευθείας.

Κατασκευή και κατάργηση εξαρτημένων στοιχείων

```
dic_item D.insert(K k, I i);
```

Αν το `D` περιέχει στοιχείο `it` με κλειδί `k`, τότε η πληροφορία του `it` αλλάζει σε `i` και το στοιχείο `it` επιστρέφεται. Αλλιώς, ένας νέος περιέχων τύπος προστίθεται στο `D` του οποίου τα κατηγορήματα είναι `k` και `i` αντίστοιχα, και επιστρέφεται η διεύθυνσή του.

- Για κάθε εξαρτημένο τύπο στοιχείων T, το $val(T)$ περιέχει την τιμή `nil`. Η τιμή αυτή δεν ανήκει σε καμία συλλογή και δεν έχει κατηγορήματα. Π.χ.:
  `dic_item D.lookup(K k);`
  επιστρέφει `nil` αν δεν υπάρχει στοιχείο με κλειδί `k` στο `D`.

```
void D.del_item(dic_item it);
```

Διαγράφει από το `D` τον περιέχοντα τύπο που αναφέρεται από το `dic_item it`. Το στοιχείο `it` πρέπει να ανήκει στο `D`.

LEDA - Κανόνας 3

Δεν επιτρέπεται η προσπέλαση των κατηγορημάτων ενός στοιχείου το οποίο αναφέρεται σε έναν περιέχοντα τύπο ο οποίος έχει καταργηθεί, ή η προσπέλαση των κατηγορημάτων του στοιχείου nil .

Ανεξάρτητοι τύποι στοιχείων

Παράδειγμα: τύπος point (αναπαράσταση σημείων).

- point: στοιχείο με (δηλ. δείκτης σε έναν περιέχοντα τύπο ο οποίος περιέχει) δύο κατηγορήματα τύπου double (x- και y-συντεταγμένη).

  - Καταχώρηση μεταξύ σημείων παίρνει $O(1)$ χρόνο.
  - Μπορεί να γίνεται πολύ γρήγορα έλεγχος ταύτισης (ίδια τιμή δείκτη), ή ισότητας (ίδια τιμή κατηγορημάτων).

Λειτουργίες ποιντ

- Ερωτήσεις για τιμές κατηγορημάτων. Π.χ. p.xcoord() και p.ycoord().
- Κατασκευή νέων σημείων από υπάρχοντα.
  point p. translate (double a, double b)
  επιστρέφει ένα νέο σημείο με συντεταγμένες
  (p.xcoord()+a, p.ycoord()+b).
  Η λειτουργία translate *δεν* αλλάζει το σημείο p.

LEDA - Κανόνας 4

Οι ανεξάρτητοι τύποι στοιχείων δεν παρέχουν λειτουργίες που επιτρέπουν την αλλαγή των κατηγορημάτων.

Αλλιώς, θα μπορούσαν να δημιουργηθούν διάφορα προβλήματα λόγω παρενεργειών.

```
q = p;
p.change_x(a);   // changes x-coord of p to a
                 // and does the same for q!
```

LEDA - Κανόνας 5

Τα κατηγορήματα ενός ανεξάρτητου αντικειμένου ορίζονται πάντοτε. Συγκεκριμένα, ο ορισμός με προεπιλεγμένη αρχικοποίηση αρχικοποιεί όλα τα κατηγορήματα. Ένας τύπος μπορεί να προσδιορίζει αρχικές τιμές για τα κατηγορήματα, αλλά όχι υποχρεωτικά.

- Ορισμός ανεξάρτητων στοιχείων.

```
point p(2.0,3.0);
point q;     // defines a point q with coordinates
             // whose exact value is undetermined
```

Παρατήρηση: Ίδια συμπεριφορά με εκείνη των ενσωματωμένων τύπων της C++.

Θα μπορούσε να υπάρξει εναλλακτική λύση;

Π.χ. αρχική τιμή = προεπιλεγμένη τιμή τύπου

`point` $\rightarrow (\mathbf{undetermined}, \mathbf{undetermined})$, ενώ `rat_point` $\rightarrow (0, 0)$

Τέλος Ενότητας

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση **1.0**.

Σημείωμα Ιστορικού Εκδόσεων Έργου

Σημείωμα Ιστορικού Εκδόσεων Έργου

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει :

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους