Δομές Δεδομένων

ΓΛΩΣΣΙΚΉ ΤΕΧΝΟΛΟΓΊΑ

Επιλογή δομής δεδομένων

- Κριτήρια:
  - Μέγεθος του προβλήματος
  - Πως θα χρησιμοποιηθεί
- Ενέργειες που καθορίζουν το κόστος:
  - Lookup: αναζήτηση/έλεγχος ύπαρξης δεδομένων στη δομή.
  - Insert: εισαγωγή δεδομένων στη δομή.

Συνήθεις κλάσεις

- Οι νεότερες γλώσσες προγραμματισμού προσφέρουν συνήθως δύο βασικές κλάσεις:
  - Dictionary
    - Δυναμικού μεγέθους.
    - Κάθε εγγραφή είναι του τύπου <key, value>.
    - Το κλειδί εισάγεται σε hashtable.
    - Οι εγγραφές δεν ταξινομούνται.
  - List:
    - Δυναμικού μεγέθους.
    - Κάθε εγγραφή είναι μόνο value.
    - Δεν υπάρχει οργάνωση.
    - Υποστηρίζει κλήσεις ταξινόμησης.

Υπέρ-Κατά

- Dictionaries:
  - Πολύ γρήγορα lookups.
  - Πιο αργή προσθήκη εγγραφών (hashing process).
  - Για να ταξινομηθεί πρέπει να εξαχθεί η λίστα των κλειδιών.
- Lists:
  - Αργά lookups. Συνήθως γίνεται σειριακή αναζήτηση.
  - Γρήγορη προσθήκη εγγραφών.
  - Μπορεί να ταξινομηθεί.

Τι χρησιμοποιούμε?

- Κριτήρια:
  - Πόσες θα είναι οι εγγραφές? Πχ για < 10 δεν αξίζει το κόστος του hashing.
  - Πόσο συχνά θα ψάχνουμε στη δομή? Πχ αν σε κάθε insert αντιστοιχεί και ένα lookup ή περισσότερο το hashing συμφέρει.
  - Θα ταξινομήσουμε? Αν και οι περισσότερες γλώσσες υποστηρίζουν απευθείας μετατροπή των κλειδιών του Dictionary σε λίστα για να γίνει ταξινόμηση.
- Πως επιλέγουμε?
  - Μαντεύοντας. Ο γενικός κανόνας είναι “rule of thumb”. Κάθε πρόβλημα είναι διαφορετικό και μέχρι να το δεις να δουλεύει δεν ξέρεις με σιγουριά.

C# - Collection classes

- Namespace: System.Collection
- Βασικές κλάσεις:
  - Dictionary<TKey, TValue>: Η κλάση υλοποιεί ένα hashtable των κλειδιών, μέσω των οποίων είναι προσβάσιμο το value.
  - List<TValue>: Λίστα αντικειμένων δυναμικού μεγέθους.
- Υβριδικές Κλάσεις:
  - SortedDictionary<TKey, TValue>: Τα κλειδιά δεν είναι hashed, αλλά δέντρο. Διατηρούν σειρά, αλλά έχουν μεγαλύτερους χρόνους lookup O(logn) από τη hash συνάρτηση.
  - SortedList<TKey, TValue>: Διατηρεί τη λίστα ταξινομημένη σύμφωνα με το κλειδί. Έχει O(logn) χρόνο lookup, αλλά μεγαλύτερους χρόνους insertion, εκτός αν τα δεδομένα είναι ήδη ταξινομημένα.

C# - Dictionary<TKey, TValue>

```
public static void DictionaryExample()
{
    Dictionary<string, int> term_frequences = new Dictionary<string, int>();

    int i;
    string[] terms = { "term1", "term2", "term3", "term4", "term5" };
    string term;

    for (i = 0; i < 10000; i++)
    {
        //dummy term
        term = terms[i % 5];
        if (term_frequences.ContainsKey(term)) //lookup
        {
            term_frequences[term]++; //lookup
        }
        else
        {
            term_frequences.Add(term, 1); //insert
        }
    }

    /*Dictionary contains:
     * <term1, 2000>
     * <term2, 2000>
     * <term3, 2000>
     * <term4, 2000>
     * <term5, 2000>
    */
}
```

- Σε αυτή την περίπτωση συμφέρει γιατί έχουμε 2*9995 lookups και 5 insertions

C# - List<TValue>

```
public static void ListExample()
{
    Dictionary<string, int> term_frequences = new Dictionary<string, int>();


    int i;
    string[] terms = { "term5", "term2", "term1", "term4", "term3" };
    string term;

    for (i = 0; i < 10000; i++)
    {
        //dummy term
        term = terms[i % 5];
        if (term_frequences.ContainsKey(term)) //lookup
        {
            term_frequences[term]++; //lookup
        }
        else
        {
            term_frequences.Add(term, 1); //insert
        }
    }
    /*Dictionary contains:
     * <term5, 2000>
     * <term2, 2000>
     * <term1, 2000>
     * <term4, 2000>
     * <term3, 2000>
    */
    //Ανάθεση των κλειδιών στη λίστα
    List<string> lstTerms = new List<string>(term_frequences.Keys);
    lstTerms.Sort();

    /*List contains:
     * <term1>
     * <term2>
     * <term3>
     * <term4>
     * <term5>
    */

}
```

- Για να ταξινομήσουμε αναθέτουμε σε λίστα.
- Κόστος: Μνήμη. Τα κλειδιά αποθηκεύονται * 2.

Python - Lists

- list.sort()
  - Ταξινόμηση λίστας
- list.reverse()
  - Αντιστροφή στοιχείων λίστας

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> a
[66.25, 333, 333, 1, 1234.5]
>>> a.reverse()
>>> a
[1234.5, 1, 333, 333, 66.25]
>>> a.sort()
>>> a
[1, 66.25, 333, 333, 1234.5]
>>>
```

Python - Lists as stacks

- list.append()
  - Προσθήκη στο τέλος της λίστας
- list.pop()
  - Αφαίρεση από το τέλος της λίστας

```
>>> stack = [1, 2, 3, 4, 5]
>>> stack
[1, 2, 3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack.append(8)
>>> stack
[1, 2, 3, 4, 5, 6, 7, 8]
>>> stack.pop()
8
>>> stack.pop()
7
>>> stack
[1, 2, 3, 4, 5, 6]
>>>
```

Python – List Comprehensions

```
>>> a = [1, 2, 3, 4, 5]
>>> b = [6, 7, 8]
>>> [x*2 for x in a]
[2, 4, 6, 8, 10]
>>> [x*3 for x in a if x > 2]
[9, 12, 15]
>>> [x+y for x in a for y in b]
[7, 8, 9, 8, 9, 10, 9, 10, 11, 10, 11, 12, 11, 12, 13]
>>> [a[i]*b[i] for i in range(len(b))]
[6, 14, 24]
>>> [[x,y] for x in a for y in b]
[[1, 6], [1, 7], [1, 8], [2, 6], [2, 7], [2, 8], [3, 6], [3, 7], [3, 8], [4, 6], [
4, 7], [4, 8], [5, 6], [5, 7], [5, 8]]
>>>
```

Python - Dictionaries

```
>>> dict = {'term1':5, 'term2':3, 'term3':0, 'term4':4}
>>> dict.keys()
['term4', 'term3', 'term2', 'term1']
>>> a = dict.keys()
>>> a.sort()
>>> a
['term1', 'term2', 'term3', 'term4']
>>> b = dict.items()
>>> b
[('term4', 4), ('term3', 0), ('term2', 3), ('term1', 5)]
>>> b.sort()
>>> b
[('term1', 5), ('term2', 3), ('term3', 0), ('term4', 4)]
>>>
```