



Camp Προετοιμασίας
Πανελλήνιος Διαγωνισμός Πληροφορικής
Αθήνα, 11-15 Απριλίου 2011

Δομές δεδομένων και C++ STL

Νίκος Παπασπύρου

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. και Μηχ. Υπολογιστών

Γλώσσα προγραμματισμού

- **Pascal:**
 - + σχετικά απλή και εύρωστη
 - πεπαλαιωμένη, σχετικά αργές υλοποιήσεις
- **C:**
 - + πολύ γρήγορες υλοποιήσεις
 - εύκολο να κάνεις λάθος
- **C++:**
 - + σχετικά γρήγορες υλοποιήσεις, βιβλιοθήκη
 - υπερβολικά πολύπλοκη

Γλώσσα προγραμματισμού

- Τα παραδείγματα θα είναι σε C
- Όχι εξεζητημένη!
σαφήνεια αντί ταχύτητας!
- Θα χρησιμοποιούμε τις δομές δεδομένων που ορίζονται στη βιβλιοθήκη της C++
 - σωστές
 - καλές, αποδοτικές
 - γενικά πιο πολύπλοκες απ' ότι χρειαζόμαστε

Βασικές δομές δεδομένων

- Πίνακας (array)
- Εγγραφή (struct/record)
- Απαρίθμηση (enum)
- Ένωση (union)
- Δείκτης (pointer)

Συμβολοσειρά

- **string**
 - length
 - c_str
 - empty
 - s[i], substr
 - append, +=
 - compare, ==, <, κ.λπ.
 - insert, replace
 - find, rfind

Άλλες χρήσιμες δομές δεδομένων

- Διάνυσμα (vector)
- Λίστα (list)
- Στοίβα (stack)
- Ουρά (queue)
- Ουρά προτεραιότητας (priority queue)
- Σύνολο (set)
- Χάρτης (map)

Διάνυσμα

- `vector<T>`
 - παρόμοια συμπεριφορά με `array`
 - μεταβλητού μεγέθους
(εισαγωγή και αφαίρεση στο τέλος)
 - ειδική αποδοτική υλοποίηση: `vector<bool>`
- Παράδειγμα: `vector1.cpp`
- Παράδειγμα: `vector2.cpp`

Λίστα

- `list<T>`
 - εισαγωγή και αφαίρεση σε αρχή, τέλος
 - γραμμική διάσχιση
 - εισαγωγή και διαγραφή ενδιάμεσα
- Παράδειγμα: `list1.cpp`
- `iterator`
- Παράδειγμα: `list2.cpp`
- Παράδειγμα: `list3.cpp`

Στοίβα και ουρά

- `stack<T>`
 - εισαγωγή και αφαίρεση στην κορυφή
- Παράδειγμα: `stack.cpp`
- `queue<T>`
 - εισαγωγή στο τέλος, αφαίρεση από την αρχή
- Παράδειγμα: `queue.cpp`

Ουρά προτεραιότητας

- `priority_queue<T>`
- `priority_queue< T, container, compare >`
 - εισαγωγή σε χρόνο $O(\log n)$
 - εύρεση του μέγιστου σε χρόνο $O(1)$
 - αφαίρεση του μέγιστου σε χρόνο $O(\log n)$
- Παράδειγμα: `prqueue1.cpp`
- Παράδειγμα: `prqueue2.cpp`

Σύνολο

- `set<T>`
- `set< T, compare >`
 - μοναδικά στοιχεία (αλλιώς multiset)
 - εισαγωγή, εύρεση και αφαίρεση σε χρόνο $O(\log n)$
- Παράδειγμα: `set.cpp`

Χάρτης

- `map<K, T>`
- `map< K, T, compare >`
 - μοναδικές τιμές (αλλιώς `multimap`)
 - εισαγωγή, εύρεση και αφαίρεση σε χρόνο $O(\log n)$
- Παράδειγμα: `map1.cpp`
- Παράδειγμα: `map2.cpp`
- Παράδειγμα: `map3.cpp`

Χρήσιμοι αλγόριθμοι

- Αναζήτηση
- Ταξινόμηση
- Πράξεις συνόλων
- Λεξικογραφική διάταξη
- Γεννήτρια μεταθέσεων

Αναζήτηση

- `binary_search(first, last, value)`
- `binary_search(first, last, value, compare)`
 - σε χρόνο $O(\log n)$
- Παράδειγμα: `search.cpp`

Ταξινόμηση

- `sort(first, last)`
- `sort(first, last, compare)`
 - σε χρόνο $O(n \log n)$ στη μέση περίπτωση
- `stable_sort(first, last)`
- `stable_sort(first, last, compare)`
 - σε χρόνο $O(n \log n)$ στη χειρότερη περίπτωση
 - διατηρεί τη σειρά ίσων στοιχείων
- Παράδειγμα: `sort1.cpp`

Ταξινόμηση

- `partial_sort(first, middle, last)`
- `partial_sort(first, middle, last, compare)`
 - σε χρόνο $O(n \log m)$ στη χειρότερη περίπτωση ($n = \text{last} - \text{first}$, $m = \text{middle} - \text{first}$)
 - ταξινομεί μόνο τα m πρώτα στοιχεία
- Παράδειγμα: `sort2.cpp`

Ταξινόμηση

- `nth_element(first, nth, last)`
- `nth_element(first, nth, last, compare)`
 - σε χρόνο $O(n)$ στη μέση περίπτωση
 - το ζητούμενο στοιχείο στη θέση του
 - τα προηγούμενα μικρότερα,
τα επόμενα μεγαλύτερα
- Παράδειγμα: `sort3.cpp`

Ταξινόμηση

- `min_element(first, last)`
- `min_element(first, last, compare)`
- `max_element(first, last)`
- `max_element(first, last, compare)`
 - σε χρόνο $O(n)$

Πράξεις συνόλων

- `includes(first1, last1, first2, last2)`
 - έλεγχος υποσυνόλου σε χρόνο $O(n_1+n_2)$
- Παράδειγμα: `setop1.cpp`
- `set_union(first1, last1, first2, last2, out)`
- `set_intersection(first1, last1, first2, last2, out)`
- `set_difference(first1, last1, first2, last2, out)`
 - ένωση, τομή και διαφορά σε χρόνο $O(n_1+n_2)$
- Παράδειγμα: `setop2.cpp`

Λεξικογραφική διάταξη

- `lexicographical_compare`
`(first1, last1, first2, last2)`
- `lexicographical_compare`
`(first1, last1, first2, last2, compare)`
 - σε χρόνο $O(\min(n_1, n_2))$

Γεννήτρια μεταθέσεων

- `next_permutation(first, last)`
- `next_permutation(first, last, compare)`
- `prev_permutation(first, last)`
- `prev_permutation(first, last, compare)`
 - γεννά την επόμενη/προηγούμενη μετάθεση κατά λεξικογραφική διάταξη
 - σε χρόνο $O(n)$
- Παράδειγμα: `permut.cpp`

Συμβουλές και στρατηγική

- Πρώτα σκεφτείτε, μετά προγραμματίστε
- Πριν αρχίσετε να γράφετε πρέπει να έχετε ολόκληρη τη λύση στο μυαλό σας
- Χρησιμοποιείτε σχόλια, με μέτρο αλλά συστηματικά
- Όσο λιγότερο copy-paste γίνεται
- Χρησιμοποιείτε υποπρογράμματα

Συμβουλές και στρατηγική

- Ο χρόνος είναι **περιορισμένος**
- Μοιράστε τον και αξιοποιήστε τον σωστά
- Καλύτερα μια **μέτρια λύση** που δουλεύει σχετικά σωστά, παρά μια **τέλεια λύση** που δε δουλεύει καθόλου!
- Χρησιμοποιείτε τον **debugger**, με μέτρο αλλά συστηματικά

Συμβουλές και στρατηγική

- Χρησιμοποιείτε assertions!

```
#include <assert.h>
```

```
...
```

```
assert(i < j && x[i] < x[j]);
```

- Γιατί;
 - προλαβαίνουν σφάλματα
 - σας βοηθούν να αναγνωρίσετε αναλλοίωτες (invariants) στο πρόγραμμά σας
 - στο τέλος δεν κοστίζουν τίποτα

```
#define NDEBUG
```