Δομές Δεδομένων

Υλοποίηση Δυναμικού Πίνακα σε γλώσσα C

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Δυναμικός Πίνακας

ως ΑΤΔ

Ένα πίνακας που περιλαμβάνει τις εξής λειτουργίες:

- Εισαγωγή ενός νέου στοιχείου στο τέλος του πίνακα, $ADD(V, x)$.
- Εισαγωγή ενός νέου στοιχείου πριν την θέση i του πίνακα, $ADD\_AT(V, i, x)$.
- Διαγραφή του στοιχείου στην θέση i του πίνακα, $REMOVE\_AT(V, i, x)$.
- Επιστροφή του i στοιχείου του πίνακα, $GET(V, i)$.
- Αλλαγή του i στοιχείου του πίνακα, $PUT(V, i, x)$.
- Έλεγχος κενού πίνακα, $ISEMPTY(V)$.
- Επιστροφή του αριθμού τον στοιχείων που είναι αποθηκευμένα στον πίνακα, $SIZE(V)$.
- κ.τ.λ

Δυναμικός Πίνακας

Ο δυναμικός πίνακας αυξομειώνει δυναμικά την χωρητικότητα του έτσι ώστε να μπορούμε να προσθέτουμε και να αφαιρούμε στοιχεία, και ταυτόχρονα να διατηρούμε την δυνατότητα της τυχαίας πρόσβασης.

Ως υλοποίηση υπάρχει σε πολλές γλώσσες προγραμματισμού:

- στην C++ είναι η κλάση `std::vector`
- στην Java υπάρχουν δύο υλοποιήσεις, η κλάση `ArrayList` και η κλάση `Vector` που έχει το επιπλέον χαρακτηριστικό πως είναι thread-safe

Δυναμικός Πίνακας

Μερικές προϋποθέσεις:

- Ξεχωριστή υλοποίηση με διασύνδεση
- Οι χρήστες δεν επιτρέπεται να πειράζουν την εσωτερική αναπάσταση
- Οι χρήστες πρέπει να μπορούν να φτιάξουν πολλούς δυναμικούς πίνακες ταυτόχρονα

Δυναμικός Πίνακας

διασύνδεση σε C (vector.h)

```c
#ifndef _VECTOR_H
#define _VECTOR_H

typedef struct _vector* vector;
typedef int value_type;

vector vector_create();
void vector_destroy(vector);

value_type vector_get(vector, int);
void vector_put(vector, int, value_type);

void vector_add(vector, value_type);

void vector_add_at(vector, int, value_type);
value_type vector_remove_at(vector, int);

int vector_is_empty(vector);
int vector_size(vector);
void vector_clear(vector);

#endif
```

Υλοποίηση Δυναμικού Πίνακα

Αναπαράσταση (vector.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "vector.h"

#define INITIAL_CAPACITY 64
#define min(x,y) (((x)<(y))?(x):(y))

struct _vector {
    value_type* array;
    int size;
    int capacity;
};
```

Υλοποίηση Δυναμικού Πίνακα

Δημιουργία (vector.c)

```
vector vector_create() {
    vector v = (vector) malloc(sizeof(struct _vector));
    if (v == NULL) {
        fprintf(stderr, "Not enough memory!");
        abort();
    }
    v->size = 0;
    v->capacity = INITIAL_CAPACITY;
    v->array = (value_type*) malloc( \
        sizeof(value_type)* v->capacity);
    if (v->array == NULL) {
        fprintf(stderr, "Not enough memory!");
        abort();
    }
    return v;
}
```

Υλοποίηση Δυναμικού Πίνακα

Καταστροφή (vector.c)

```
void vector_destroy(vector v) {
    assert(v);

    free(v->array);
    free(v);
}
```

Υλοποίηση Δυναμικού Πίνακα

Διπλασιασμός Χωρητικότητας (vector.c)

```
static
void vector_double_capacity(vector v) {
    assert(v);
    int new_capacity = 2 * v->capacity;
    value_type* new_array = (value_type*) malloc( \
        sizeof(value_type)*new_capacity);
    if (new_array == NULL) {
        fprintf(stderr, "Not enough memory!");
        abort();
    }

    for(int i = 0; i < v->size; i++) {
        new_array[i] = v->array[i];
    }

    free(v->array);
    v->array = new_array;
    v->capacity = new_capacity;
}
```

Υλοποίηση Δυναμικού Πίνακα

Υποδιπλασιασμός Χωρητικότητας (vector.c)

```
static
void vector_half_capacity(vector v) {
    assert(v);
    if (v->capacity <= INITIAL_CAPACITY) {
        return;
    }

    int new_capacity = v->capacity / 2;
    value_type* new_array = (value_type*) malloc( \
        sizeof(value_type)*new_capacity);
    if (new_array == NULL) {
        fprintf(stderr, "Not enough memory!");
        abort();
    }

    for(int i = 0; i < min(v->size, new_capacity); i++) {
        new_array[i] = v->array[i];
    }

    free(v->array);
    v->array = new_array;
    v->capacity = new_capacity;
    v->size = min(v->size, new_capacity);
}
```

Υλοποίηση Δυναμικού Πίνακα

Προσθήση στοιχείου στο τέλος (vector.c)

```
void vector_add(vector v, value_type value) {
    assert(v);

    if (v->size >= v->capacity) {
        vector_double_capacity(v);
    }
    v->array[v->size++] = value;
}
```

Υλοποίηση Δυναμικού Πίνακα

Διάβασμα και Γράψιμο(vector.c)

```
value_type vector_get(vector v, int i) {
    assert(v);
    if (i < 0 || i >= v->size) {
        fprintf(stderr, "Out of index!");
        abort();
    }
    return v->array[i];
}

void vector_put(vector v, int i, value_type value) {
    assert(v);
    if (i < 0 || i >= v->size) {
        fprintf(stderr, "Out of index!");
        abort();
    }
    v->array[i] = value;
}
```

Υλοποίηση Δυναμικού Πίνακα

Προσθήση στην μέση(vector.c)

```
void vector_add_at(vector v, int i, value_type value) {
    assert(v);

    if (i < 0 || i >= v->size) {
        fprintf(stderr, "Out of index!");
        abort();
    }

    if (v->size >= v->capacity) {
        vector_double_capacity(v);
    }

    for(int j = v->size; j>i; j--) {
        v->array[j] = v->array[j-1];
    }
    v->array[i] = value;
    v->size++;
}
```

Υλοποίηση Δυναμικού Πίνακα

Διαγραφή (vector.c)

```
value_type vector_remove_at(vector v, int i) {
    assert(v);

    if (i < 0 || i >= v->size) {
        fprintf(stderr, "Out of index!");
        abort();
    }

    value_type ret = v->array[i];
    for(int j = i+1; j < v->size; j++) {
        v->array[j-1] = v->array[j];
    }
    v->size--;

    if (4 * v->size < v->capacity) {
        vector_half_capacity(v);
    }

    return ret;
}
```

Υλοποίηση Δυναμικού Πίνακα

Is-Empty και Size (vector.c)

```c
int vector_is_empty(vector v) {
    assert(v);
    return v->size == 0;
}

int vector_size(vector v) {
    assert(v);
    return v->size;
}
```

Υλοποίηση Δυναμικού Πίνακα

Clear (vector.c)

```
void vector_clear(vector v) {
    assert(v);

    v->size = 0;
    while (v->capacity > INITIAL_CAPACITY) {
        vector_half_capacity(v);
    }
}
```

Δοκιμάζοντας τον Δυναμικό Πίνακα

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "vector.h"

int main() {
    vector v;
    v = vector_create();

    for(int i = 0; i < 1000000; i++) {
        vector_add(v, i);
        assert(vector_size(v) == i+1);
        assert(vector_get(v, i) == i);
    }

    assert(vector_size(v) == 1000000);

    while(!vector_is_empty(v)) {
        vector_remove_at(v, vector_size(v)-1);
    }

    assert(vector_is_empty(v));
    assert(vector_size(v) == 0);

    vector_destroy(v);

    return 0;
}
```