Structuri de date

Rotaru Alexandru Andrei
rotarualexandruandrei94@gmail.com
2019-2020

University Politehnica of Bucharest

1 Stive & Cozi

1.1 Stiva (Last In First Out)

Stiva este o structura de date liniara care ofera urmatoarele operatii de baza:

- push: insereaza un element in stiva (intr-un capat al listei/vectorului)
- pop: elimina un element din acelasi capat unde se insereaza elemente
- top: returneaza elementul din varful stivei

Implementarea stivelor poate sa aiba la baza alte structuri de date (liste/vectori) atata timp cat se respecta functionarea operatiile de push si pop.

Listing 1: Stiva implementat pe baza de vector

```
1  typedef T double;
2  typedef struct {
3      T *buffer;
4      size_t size;
5      size_t capacity;
6  } stack_t;
7
8  int ensure_space(stack_t* stack) {
9      if (!stack)
10         return -1;
11     if (stack->size >= stack->capacity){
12         stack->buffer = (T *)realloc(stack->buffer,
13                                      stack->capacity * 2)
14         stack->capacity *= 2;
15     }
16     return 0;
17 }
18
19 int push(stack_t* stack, T value) {
20     int r = ensure_space(stack);
21     if (r)
22         return r;
23     stack->buffer[stack->size++] = value;
24     return 0;
25 }
26
27 int pop(stack_t* stack, T &value) {
28     if (!stack || !stack->size)
29         return -1;
30     value = stack->buffer[(stack->size--) - 1];
31     return 0;
32 }
```

1.2 Coada (First In First Out)

Coada este o structura de date liniara care ofera 3 operatii de baza:

- push: insereaza un element in stiva (intr-un capat al listei/vectorului)
- pop: elimina un element din capatul opus fata de cel unde se insereaza elemente
- top: returneaza elementul din varful cozii

O proprietate semnificativa a acestei structuri de date ar fi ca pastreaza ordinea de inserare a elementelor. Implementarea cozilor poate sa aiba la baza alte structuri de date (liste/vectori) atata timp cat se respecta functionarea operatiile de push si pop.

Listing 2: Coada implementat pe baza de vector

```
1  typedef T double;
2  typedef struct {
3      T *buffer;
4      size_t pop_index;
5      size_t push_index;
6      size_t capacity;
7  } queue_t;
8
9  // ensure space asemanator cu cel de la stive
10 int push(queue_t *queue, T value) {
11     int status = ensure_space(queue);
12     if (status)
13         return status;
14     queue->buffer[queue->push_index++] = value;
15 }
16
17 int pop(queue_t* queue, T &value) {
18     // diferenta intre push si pop index reprezinta
19     // cate elemente avem in queue
20     if (!queue || (queue->push_index < queue->pop_index))
21         return -1;
22     value = queue->buffer[(queue->pop_index++)];
23     return 0;
24 }
```

2 Discutie

Aceste structuri de date au o aplicabilitate larga, foarte multe probleme se rezolva folosind cozi/stive. Printre aplicabilitati enumeram:

Stive

- verificarea parantezarii corecte a unei expresii
- parcurgeri pe grafuri (DFS) iterativ
- procesarea unei expresii matematice

Cozi

- producator/consumator
- parcurgeri pe grafuri (BFS) iterativ
- determinarea unui subsecvente de sir cu o anumita proprietate
- radix sort

3 Precizari

Laborantul va preciza ce exercitii veti avea de rezolvat. Pentru rezolvarea fiecarui exercitiu veti folosi un fisier separat. **Optional**, va puteti verifica folosind checker-ul (daca este pus la dispozitie).

3.1 Vizualizare

Pentru a putea visualiza o parte din algoritmi si intelege mai bine cum functioneaza structurile de date accesati:
`https://www.cs.usfca.edu/~galles/visualization/Algorithms.html`

3.2 Link-uri catre platforme online cu probleme

- Hackerrank: `https://www.hackerrank.com/domains/data-structures`
- Leetcode: `https://leetcode.com/problemset/all/`

3.3 Tool-uri de debug

- GDB: `https://cs.baylor.edu/~donahoo/tools/gdb/tutorial.html`
- Valgrind: `http://valgrind.org/docs/manual/quick-start.html`

3.4 Feedback

Pentru a semnala probleme sau a oferi sugestii pentru laboratoarele urmatoare:
`https://forms.gle/8bRw7mPgqaJ9tRzd6`

4 Exercitii

4.1 Exercitiul 1 [5 puncte]

Sa se implementeze o structura de date care implementeaza atat operatiile de stiva cat si de coada. Structura de date se va numi "dequeue". Implementarea se va baza pe structura de lista dublu inlantuita. Tinand cont ca valorile din structura sunt transferate prin efect lateral, valorile de retur vor reprezenta coduri de eroare.

- ERROR_SUCCESS pentru cazul in care nu exista cazuri exceptionale (deq este null, etc)
- ERROR_DEQUEUE_EMPTY in cazul in care structura data ca parametru este null sau nu avem elemente
- ERROR_DEQUEUE_INTERNAL in cazul in care avem alte erori

Listing 3: Dequeue (double ended queue)

```
typedef struct {
    node_t *front;
    node_t *back;
    size_t size;
} dequeue_t;
dequeue_t init_deq(void);
dequeue_t *alloc_deq(void);
// acceseaza primul element
int front(dequeue_t *deq, double *value);
// acceseaza ultimul element
int back(dequeue_t *deq, double *value);
// insereaza un element in fata
int push_front(dequeue_t *deq, double value);
// insereaza un element in spate
int push_back(dequeue_t *deq, double value);
// scoate primul element
void pop_front(dequeue_t *deq);
// scoate ultimul element
void pop_back(dequeue_t *deq);
// intoarce numarul de elemente stocate in deq
size_t size(dequeue_t *deq)
```

4.2 Exercitiul 2 [5 puncte]

Folosind structura definita anterior SAU definind o structura de coada implementati algoritmul de sortare radix-sort.

Listing 4: Radix sort

```
// returneaza a kth cea mai nesemnificativa cifra
// sau 0 daca kth este mai mare decat numarul de cifre
char get_kth_digit(const int number, const char kth);
// sorteaza prin efect lateral vectorul
int radix_sort(const int vec[], const size_t length);
```

Listing 5: Radix sort

```
    q[10] = [init_queue(), ..];
    for digit in {0, 1, ... (max_digit)}
        for el in vector:
            q_insert(q[get_kth_digit(el, digit)], el)
        vector = []
        for qidx in {0, 1, ... 10}
            vector += q[qidx];
```

5 Extra / Intrebari de interviu

5.1 Exercitiul 4 [2 puncte]

Sa se implementeze o functie care primeste ca parametrii 2 siruri de caractere si returneaza cea mai lunga subsecventa a primului sir care contine jumatate din caracterele subsirului 2.

5.2 Exercitiul 5 [2 puncte]

Sa se implementeze o stiva care pe langa functionalitatile de baza ofera si functionalitatile de:

”

```
1 value min_element(<stack_structure> *stack);
2 value max_element(<stack_structure> *stack);
```