Structuri de date

Laboratorul 4

Mihai Nan
mihai.nan.cti@gmail.com
Grupa 312aCC

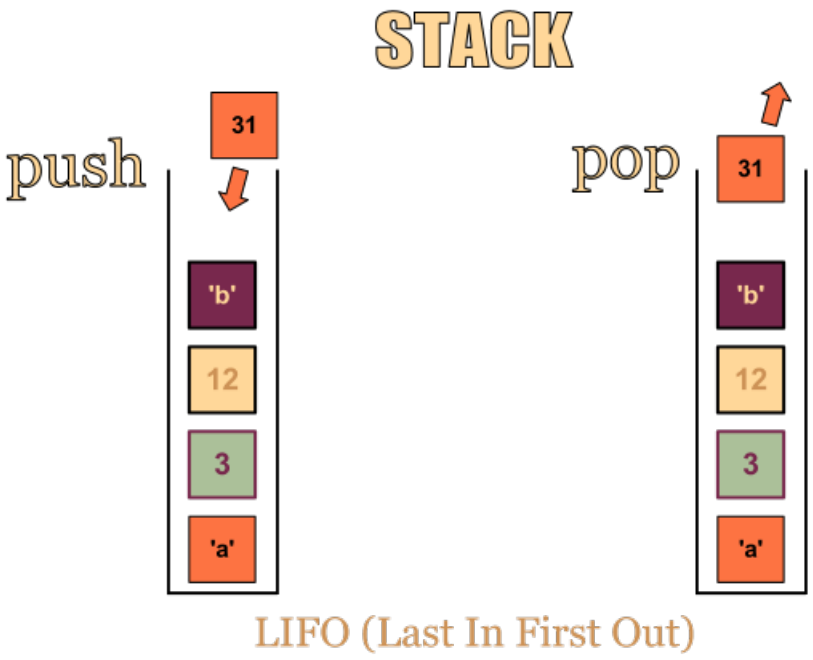
Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2016 - 2017

1 Stive

1.1 Introducere

O stiva este o structura de date de tip container (depoziteaza obiecte de un anumit tip) organizata dupa principiul **LIFO** (***Last In First Out***). Operatiile de acces la stiva (***push*** – adauga un element in stiva si ***pop*** – scoate un element din stiva) sunt create astfel incat ***pop*** scoate din stiva elementul introdus cel mai recent.

O stiva este un caz particular de lista, si anume este o lista pentru care operatiile de acces (inserare, stergere, accesare element) se efectueaza la un singur capat al listei.



⚠ IMPORTANT !

⚠ Accesul la elementele de pe stiva se poate face doar prin intermediul operatiilor ***init***, ***push***, ***pop***, ***top*** si ***isEmpty***, ceea ce inseamna ca stiva este un tip de date restrictionat.

1.2 Implementarea structurii de date

Stivele si cozile pot fi implementate in mai multe moduri. Cele mai utilizate implementari sunt cele folosind masive si liste. Ambele abordari au avantaje si dezavantaje:

	Avantaje	Dezavantaje
Masive	- implementare simpla; - consum redus de memorie; - viteza mare pentru operatii.	- numarul de elemente este limitat; - risipa de memorie in cazul in care dimensiunea alocata este mult mai mare decat numarul efectiv de elemente.
Liste	- numar oarecare de elemente	- consum mare de memorie pentru memorarea legaturilor

1.2.1 Implementarea cu vectori

Pentru implementarea unei stive folosind masive avem nevoie de un masiv V de dimensiune n pentru memorarea elementelor (acesta o sa fie alocat si realocat dinamic) si de o variabila care sa fie folosita pe post de contor (vom sti pe ce pozitie se afla in masiv ultimul element inserat in stiva).

Implementarea sub forma de tablou are dezavantajul lungimii finite a stivei, pentru ca se declara de la inceput dimensiunea maxima a stivei. Insa, pentru a mai diminua acest dezavantaj, putem uzita un vector alocat si realocat dinamic.

Operatia de initializare a unei stive, implementata cu un vector, presupune alocarea dinamica a vectorului, alocarea memoriei pentru structura, setarea dimensiunii vectorului si stabilirea pozitiei (0 – indica faptul ca stiva este vida).

Un exemplu de structura ce poate reprezenta o stiva implementata folosind vectori, este urmatoarea:

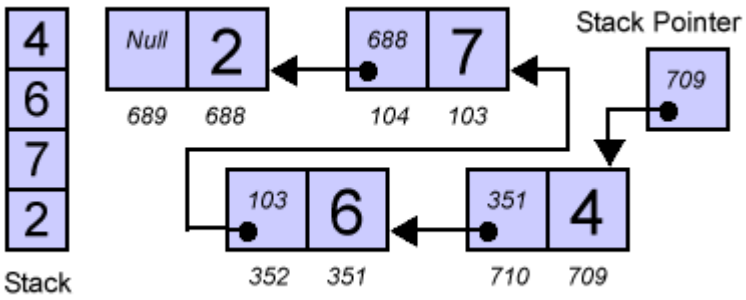
Cod sursa C

```c
typedef struct stack {
  int* data;
  int capacity, size;
}*Stack;
```

De asemenea, in arhiva laboratorului gasiti o implementare integrala pe care ar fi bine sa o parcurgeti si sa incercati sa o intelegeti.

1.2.2 Implementarea cu liste

Cea de-a doua modalitate de implementare a stivelor este cea folosind liste alocate dinamic si nu o vom mai analiza in detaliu in cadrul acestui laborator, deoarece seamana foarte mult cu abordarea prezentata pentru listele simplu inlantuite, insa aceasta este reprezentarea cu care vom lucra.



Operatia *init*

Algorithm 1 Init

```
procedure INIT(value)
    new ← malloc(sizeof(struct stack))
    new → value ← value
    new → next ← NULL
    return new
```

Operatia *push*

Algorithm 2 Push

```
procedure PUSH(stack, value)
    new ← init(value)
    new → next ← stack
    return new
```

Operatia *pop*

Algorithm 3 Pop

```
procedure POP(stack)
    if stack ≠ NULL then
        tmp ← stack → next
        stack ← stack → next
        free(tmp)
    return stack
```

Operatia *top*

Algorithm 4 Top

```
procedure TOP(stack)
    if stack ≠ NULL then
        value ← stack → value
        return value
```

Operatia *isEmpty*

Algorithm 5 IsEmpty

```
procedure ISEMPTY(stack)
    if stack ≠ NULL then
        return false
    else
        return true
```

Operatii	Masive	Liste
init	O(1)	O(1)
push	O(1)	O(1)
pop	O(1)	O(1)
top	O(1)	O(1)
isEmpty	O(1)	O(1)

Observatie

Pentru simplitate si omogenitate, exercitiile din cadrul acestui laborator se vor efectua uzitand reprezentarea, definita in blocul de cod de mai jos, pentru o stiva, uzitand pentru un nod reprezentarea din laboratorul trecut.

Cod sursa C

```c
typedef struct node {
  int value;
  struct node* next;
}*Node;
```

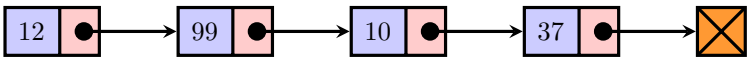
Cod sursa C

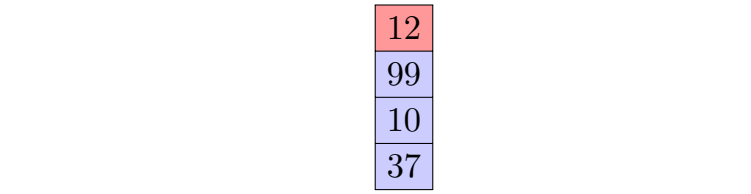
```c
typedef struct stack {
    Node top;
    int size;
}*Stack;
```

1.3 Exemple

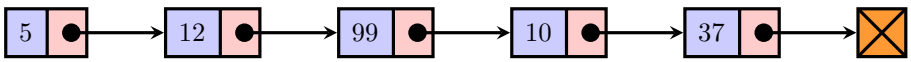
Orice browser Web memoreaza, pentru fiecare fila, adresele site-urilor vizitate recent intr-o stiva. De fiecare data cand utilizatorul acceseaza un site nou, adresa acestuia este adauga in stiva, iar cand este uzitat butonul ***back*** browserul elimina elementul din varful acestei stive si il afiseaza pe cel care se afla in varful stivei dupa operatia de eliminare. Cu alte cuvinte, va afisa site-ul precedent vizitat. Evident, in aceasta explicatie a fost simplificat modelul folosit de browser cu un scop pur didactic.

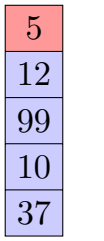
Editoarele de text au implementata comanda ***undo*** care anuleaza modificarile recente si readuc fisierul la o forma precedenta. In implementarea mecanismului pentru aceasta comanda se poate folosi o stiva pentru a memora modificarile de text.





push(5);





2 Cozi

2.1 Introducere

Coada este o structura de date abstracta, pentru care operatia de inserare a unui element se realizeaza la un capat, in timp ce operatia de extragere a unui element se realizeaza la celalalt capat.

Definitoriu pentru coada reprezinta modul de acces la elementele sale si nu modul specific de implementare. Coada functioneaza pe principiul *primul intrat primul iesit* - ***First In First Out*** (**FIFO**).

Ordinea de extragere din coada este aceeasi cu ordinea de introducere in coada, ceea ce face utila o coada in aplicatiile unde ordinea de servire este aceeasi cu ordinea de sosire: procese de tip „vanzator – client" sau „producator – consumator". In astfel de situatii, coada de asteptare este necesara pentru a acoperi o diferenta temporara intre ritmul de servire si ritmul de sosire, deci pentru a memora temporar cereri de servire (mesaje) care nu pot fi inca prelucrate.

2.2 Implementarea structurii de date

Cozile pot fi implementate in doua moduri: ***static***, folosind tablouri, si ***dinamic***, folosind pointeri. Implementarea sub forma de tablou are dezavantajul lungimii finite a cozii, pentru ca se declara de la inceput dimensiunea maxima a cozii.

Ca si stivele, cozile pot fi implementate in mai multe moduri. Cele mai utilizate implementari sunt cele folosind masive si liste. Ambele abordari au aceleasi avantaje si dezavantaje, ca in cazul unei stive.

2.2.1 Implementarea cu liste

In cazul alocarii dinamice, elementele (nodurile) cozii sunt structuri in componenta carora intra si adresa nodului urmator – exact ca la listele liniare simplu inlantuite; insa modul de accesare al elementelor este diferit.

2.2.2 Implementarea cu masive

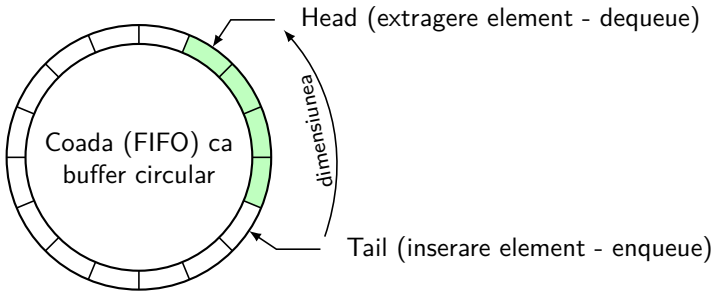
Observatie

In cadrul acestui laborator, vom implementa o coada folosind liste liniare simplu inlantuite, insa este interesant de analizat implementarea care foloseste un buffer circular. Intr-o astfel de implementare, vom lucra cu o coada infinita in care datele pot fi suprascrise. Prin buffer, intelegem un vector pentru care vom aloca memoria dinamic.

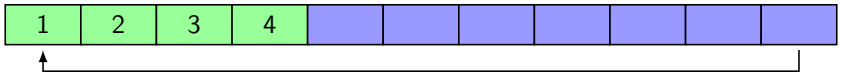
Cod sursa C

```c
typedef struct queue {
  int* data;
  int capacity, size;
  int head, tail;
}*Queue;
```

Buffer circular:



Buffer liniar:



Pentru a calcula indicii ***head*** si ***tail***, vom folosi aritmetica modulo ***N***, unde ***N*** este dimensiunea buffer-ului circular. Cand stergem un element din coada, adica aplicam operatia ***dequeue***, indexul ***head*** o sa devina ***(head + 1) % N***, iar atunci cand dorim sa introducem un element in coada, adica aplicam operatia ***enqueue***, indexul ***tail*** o sa devina ***(tail + 1) % N***.

Operatii	Masive	Liste
init	O(1)	O(1)
enqueue	O(1)	O(n)
dequeue	O(1)	O(1)
first	O(1)	O(1)
isEmpty	O(1)	O(1)

⚠ IMPORTANT !

⚠ Operatia ***enqueue*** se poate optimiza si in cazul unei implementari care uziteaza liste. Pentru acest lucru, vom retine si un pointer la sfarsitul listei pentru a nu fi nevoie de o parcurgere a listei de fiecare data cand dorim sa introducm un nod la sfarsitul ei.

Observatie

Pentru simplitate si omogenitate, exercitiile din cadrul acestui laborator se vor efectua uzitand reprezentarea, definita in blocul de cod de mai jos, pentru o coada implementata cu liste.

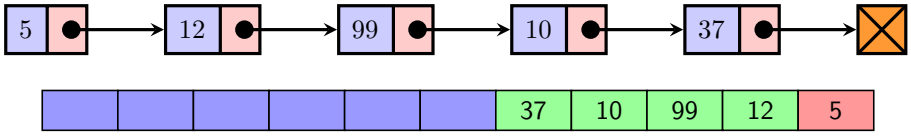
Cod sursa C

```c
typedef struct queue {
    Node head, tail;
    int size;
}*Queue;
```

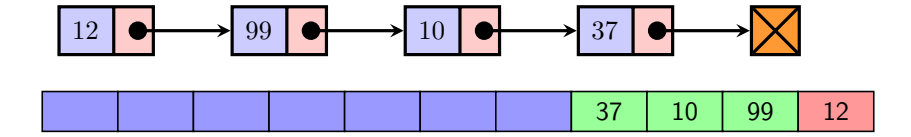
2.3 Exemple

In cazul sistemelor de operare multitasking, se uziteaza o coada pentru a retine procesele ce se afla intr-o stare de asteptare. Astfel, cand apare un nou proces,

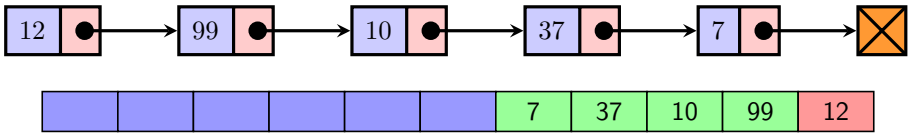
acesta este recunoscut de sistemul de operare si este introdus intr-o coada, asteptand sa ii vina randul pentru a putea fi executat. Inlocuirea unui proces cu un altul se realizeaza in cadrul unui procesor, iar operatia se numeste comutare de context. Aceasta operatie este realizata de un plainificator de procese care alege din coada de procese procesul pe care il considerat cel mai potrivit, pe baza anumitor criterii si ii ofera acestuia procesorul. Astfel, in acest caz, avem o coada cu prioritati, o structura despre care vom studia mai multe intr-un laborator viitor.



dequeue();



enqueue(7);

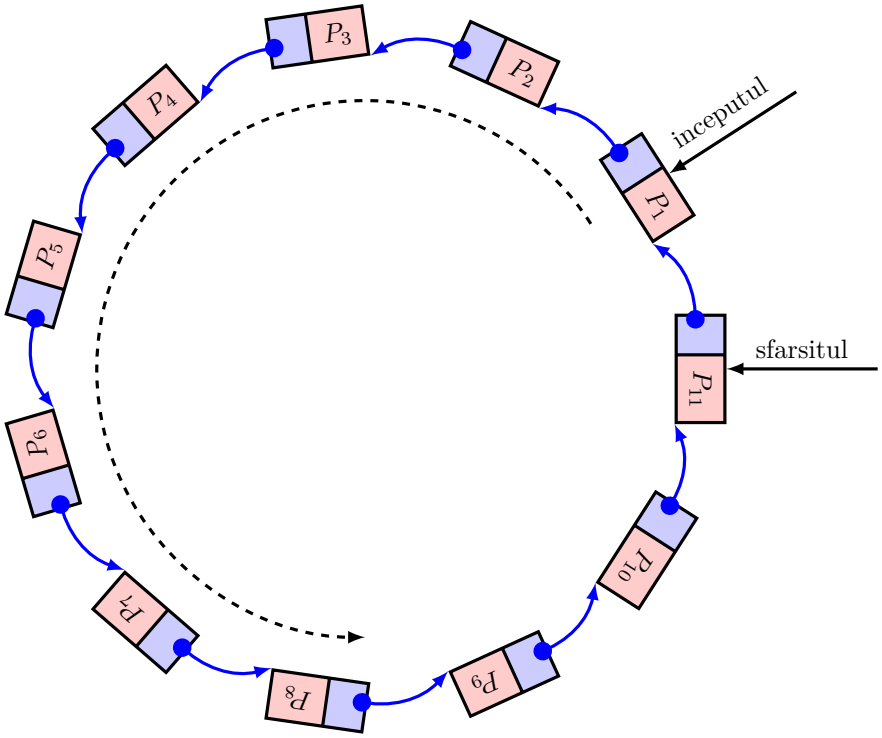


3 Liste circulare

In laboratorul precedent, am lucrat cu o lista liniara simplu inlantuita, implementand operatiile de baza pentru aceasta structura de date. De aceasta data, vom vedea ca putem avea si o lista simplu inlantuita care sa nu aiba o forma liniara. Mai exact, vom implementa o lista simplu inlantuita circulara. Daca in cazul listei simplu inlantuite liniare ultimul nod pointa catre valoarea NULL, pentru a indica sfarsitul listei, de aceasta data, ultimul nod va pointa catre primul nod al listei.

Observatie

Pentru a putea descoperi usor daca nodul curent este capul listei, o sa adaugam, in structura noastra, care va defini lista circulara, un nou camp pe care il vom seta cu o anumita valoare pentru primul nod din lista circulara si cu o alta valoare pentru toate celelalte noduri. Astfel, vom evita cazul in care parcurgem lista la infinit, deoarece nu ne putem da seama cand am ajuns la finalul ei.



Observatie

Pentru simplitate si omogenitate, exercitiile din cadrul acestui laborator se vor efectua uzitand reprezentarea, definita in blocul de cod de mai jos, pentru o lista circulara simplu inlantuita.

Cod sursa C

```c
typedef struct list {
  int value;
  struct list* next;
  int first;
}*CircularList;
```

4 Probleme de laborator

Observatie

In arhiva laboratorului, gasiti un schelet de cod de la care puteti porni implementarea functiilor propuse, avand posibilitatea de a le testa functionalitatea.
Punctajul maxim pentru acest laborator este 10 si se va acorda doar daca rezolvarea este insotita si de explicatii.

4.1 Probleme standard

Problema 1 - 3 puncte

Pornind de la definitia structurii de date ***Stack*** care foloseste liste simplu inlantuite, implementati urmatoarele operatii: ***initStack***, ***push***, ***pop***, ***top***, ***isEmptyStack***, ***freeStack***.

Problema 2 - 3 puncte

Pornind de la definitia structurii de date ***Queue*** care foloseste liste simplu inlantuite, implementati urmatoarele operatii: ***initQueue***, ***enqueue***, ***dequeue***, ***first***, ***isEmptyQueue***, ***freeQueue***.

Problema 3 - 2 puncte

Sa se scrie o functie care verifica daca o expresie are parantezele imbricate corect. Expresia contine doar () si [].

Exemplu

(([]([()()]))(([[()]])) - corect;

()([)[(]()([(]())() - incorect.

Problema 4 - 2 puncte

In ultima ecranizare a celebrei piese shakespeariene Romeo si Julieta traiesc intr-un oras modern, comunica prin e-mail si chiar invata sa programeze. Intr-o secventa tulburatoare sunt prezentate framantarile interioare ale celor doi eroi incercand fara succes sa scrie un program care sa determine un punct optim de intalnire.

Ei au analizat harta orasului si au reprezentat-o sub forma unei matrice cu n linii si m coloane, in matrice fiind marcate cu spatiu zonele prin care se poate trece (strazi lipsite de pericole) si cu 'X' zonele prin care nu se poate trece. De asemenea, in matrice au marcat cu 'R' locul in care se afla locuinta lui Romeo, iar cu 'J' locul in care se afla locuinta Julietei.

Ei se pot deplasa numai prin zonele care sunt marcate cu spatiu, din pozitia curenta in oricare dintre cele 8 pozitii invecinate (pe orizontala, verticala sau diagonale). Cum lui Romeo nu ii place sa astepte si nici sa se lase asteptat, ei au hotarat ca trebuie sa aleaga un punct de intalnire in care atat Romeo, cat si Julieta, sa poata ajunge in acelasi timp, plecand de acasa. Fiindca la intalniri amandoi vin intr-un suflet, ei estimeaza timpul necesar pentru a ajunge la intalnire prin numarul de elemente din matrice care constituie drumul cel mai scurt de acasa pana la punctul de intalnire. Si cum probabil exista mai multe puncte de intalnire posibile, ei vor sa il aleaga pe cel in care timpul necesar pentru a ajunge la punctul de intalnire este minim.

Exemplu

```
5 8
XXR  XXX
 X  X  X
J X X  X
       XX
XXX XXXX
```

Explicatie

Traseul lui Romeo poate fi: `(1, 3), (2, 4), (3, 4), (4, 4)`. Asadar, timpul necesar lui Romeo pentru a ajunge de acasa la punctul de intalnire este 4.

Traseul Julietei poate fi: `(3, 1), (4, 2), (4, 3), (4, 4)`. Timpul necesar Julietei pentru a ajunge de acasa la punctul de intalnire este, de asemenea, 4. In plus `(4, 4)` este punctul cel mai apropiat de ei cu aceasta proprietate.

Rezolvare

Pornim alternativ din pozitiile lui Romeo si a Julietei. La fiecare pas `K` generam in doua cozi separate punctele din matrice aflate la distanta `K` de pozitiile initiale pentru Romeo si Julieta.

Comparam cele doua liste de puncte. Daca gãsim un punct comun algoritmul se incheie, daca nu, generam punctele aflate la distante `K + 1` s.a.m.d.

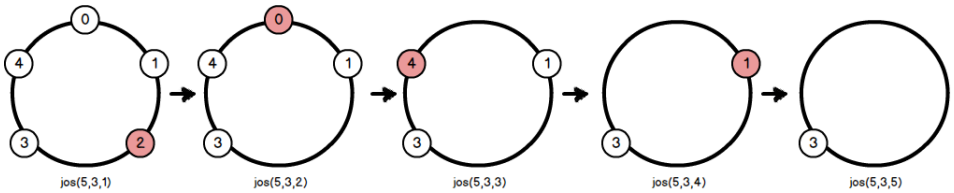
In literatura de specialitate, algoritmul folosit se numeste parcugere in latime, iar in algoritmica romaneasca este conuscut ca si algoritmul lui Lee.

Algoritmul are la baza o metoda iterativa. Fiecare vecin va fi bagat, iar la extragerea unuia din coada vom baga in coada toti vecinii nevizitati ai nodului curent (avand grija sa eliminam nodul curent). Algoritmul se repeta pana cand coada devine vida sau gasim solutia asteptata.

4.2 Problema bonus

Problema 5 - 4 puncte

n copii se aseaza in cerc, primind fiecare un numar, in sensul acelor de ceasornic. Acestia rostesc o poezie formata din **c** cuvinte (de tipul „*ala bala portocala...*"). Fiecaruia, incepand cu primul, i se asociaza un cuvant din poezie. Cel care primeste ultimul cuvant este eliminat din cerc. Acest joc continua, incepand poezia de la urmatorul copil, pana cand se elimina toti copiii. Folosind numerotarea initiala, sa se afiseze ordinea iesirii copiilor din joc.



In rezolvarea acestei probleme, veti utiliza o lista circulara simplu inlantuita.

5 Interviu

Observatie

Aceasta sectiune este una optionala si incearca sa va familiarizeze cu o serie de intrebari ce pot fi adresate in cadrul unui interviu tehnic. De asemenea, aceasta sectiune poate fi utila si in pregatirea pentru examenul final de la aceasta disciplina.

Intrebari interviu

1. Avem la dispozitie un singur vector alocat dinamic si dorim sa implementam 3 stive. Este acest lucru posibil? Argumentati!
2. Afirmatia de mai sus se poate generaliza pentru **n** stive? Argumentati!
3. De ce in cazul transmiterii parametrilor apelului unei functii este utilizata o structura de date de tip stiva?
4. Presupunand ca avem o coada ce contine un numar mare de elemente, coada neputand fi tinuta in memorie. Prezentati o modalitate de a implementa operatiile enqueue si dequeue.
5. Cate structuri de date de tip stiva sunt necesare pentru implementarea unei cozi?
6. Este posibila implementarea unei functii care sa inverseze ordinea elementelor dintr-o coada? Argumentati!

5.1 Probleme interviu

Problema 1: Tirbi tocmai a invatat la un curs de Silverlight despre paranteze rotunde "(,)", drepte "[,]" si acolade ", ". Un sir este parantezat corect daca este construit conform regulilor:

- <sir parantezat corect> = <sirul vid>
- <sir parantezat corect> = "(" + <sir parantezat corect> + ")"
- <sir parantezat corect> = "[" + <sir parantezat corect> + "]"
- <sir parantezat corect> = "" + <sir parantezat corect> + ""
- <sir parantezat corect> = <sir parantezat corect> + <sir parantezat corect>

Ca la orice curs, se dau teme de casa, iar Tirbi a primit urmatoarea problema: Avand un sir cu N paranteze, sa se afle lungimea maxima a unei secvente parantezata corect.

Sursa infoarena

Problema 2: Implementati o structura de date de tip stiva care sa permita definirea operatiei ***getMinimum*** in **O(1)**.

Problema 3: Pe o masa se afla N stive (numerotate de la 1 la N). Stiva i contine i jetoane ($1 \leq i \leq n$). La o mutare se poate alege o multime de stive si se pot extrage din fiecare stiva care face parte din multimea aleasa acelasi numar de jetoane. Sa se determine o secventa cu numar minim de mutari care sa goleasca toate cele N stive.

Sursa infoarena

Problema 4: Ca tema pentru acasa, Gigel are de scris multe egalitati.

Profesorul de matematica este insa prea lenes ca sa verifice corectitudinea calculelor asa ca prefera sa se asigure numai de faptul ca parantezele sunt asezate corect. Astfel, el scrie pe un singur rand toate parantezele din egalitatile lui Gigel (impreuna cu semnul de egalitate dintre ele - "="), in ordinea in care Gigel le-a scris in tema pentru acasa. Egalitatile sunt despartite intre ele prin ";". La finalul sirului se va gasi caracterul ".".

Din cauza faptului ca acest sir este foarte mare, profesorul va cere ajutorul: trebuie sa ii spuneti daca aceste parantezari sunt corecte sau nu.

Sursa infoarena

Problema 5: Ioana tocmai a invatat la scoala despre paranteze rotunde si despre siruri parantezate corect. Andrei i-a furnizat Ioanei un sir format din N paranteze inchise sau deschise si ea se gandeste acum sa inverseze unele paranteze (sa schimbe o paranteza deschisa cu una inchisa sau invers) astfel

incat la final sirul sa fie parantezat corect. Ajutati-o pe Ioana si determinati numarul minim de inversari care trebuie efectuat astfel incat la final sirul sa fie parantezat corect.

Sursa infoarena

Problema 6: Fiind data o stiva care contine **n** elemente, implementati o functie care inverseaza ordinea elementelor din stiva, folosind numai operatii de tip ***push*** si ***pop***, fara a uzita o structura de date auxiliara.

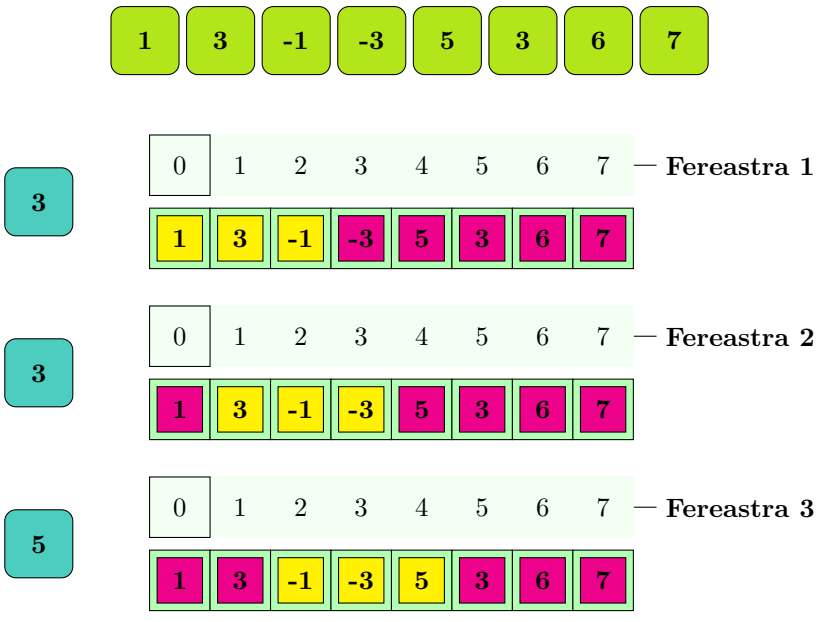
Problema 7: Dandu-se un sir de **n** numere, consideram cele **n - k + 1** subsecvente de **k** numere consecutive in sir (denumite si *ferestre* de lungime **k**). Putem vedea aceste *ferestre* ca pe o singura *fereastra glisanta* care ne da acces la **k** elemente din vector in acelasi timp. Sa se implementeze un program care determina maximul pentru fiecare astfel de fereastra.

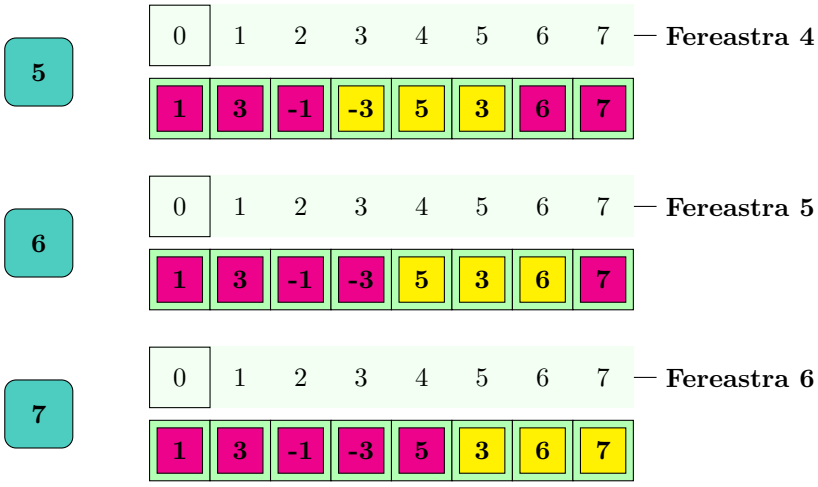
⚠ IMPORTANT !

⚠ Utilizati o structura de date de tip coada in implementarea rezolvarii acestei probleme.

Exemplu

k = 3





Feedback

Pentru imbunatatirea constanta a acestui laborator, va rog sa completati formularul de feedback disponibil aici.

De asemenea, va rog sa semnalati orice greseala / neclaritate depistata in laborator pentru a o corecta.

Va multumesc anticipat!