

3. COZI

Coada este o listă la care inserările se fac pe la un capăt – *baza* cozii, iar ștergerile se fac pe la celălalt capăt – *vârful* cozii.

Cozile sunt utile în aplicațiile de simulare, în traversarea grafurilor în lățime, în algoritmii cu arbori și grafuri.

Ordinea de extragere din coadă este aceeași cu ordinea de introducere în coadă, ceea ce sugerează și aplicațiile pentru o asemenea structură: simularea unor procese de servire de tip *producător - consumator* sau *vânzător - client*. În astfel de situații coada de așteptare este necesară pentru a acoperi o diferență temporară între ritmul de servire și ritmul cererilor solicitanților (clienților).

Cozi de servire se pot forma la comunicarea de date între un emițător și un receptor care au viteze diferite sau la stațiile de servire, pentru a memora temporar mesaje sau cereri de servire care nu pot fi încă satisfăcute, dar care nu trebuie pierdute.

3.1. Specificarea TAD Coadă

Domenii: $Coadă, Elem, int$

Semnături:

```
new      :          → Coadă_Vidă
front    :          Coadă → Elem
back     :          Coadă → Elem
enq      :  Coadă × Elem → Coadă
deq      :          Coadă → Coadă
empty    :          Coadă → int
```

Axiome:

```
front(enq(e, Coadă_Vidă)) = e
front(deq(q, e)) = front(q),          dacă q nu e vidă
front(Coadă_Vidă) = eroare
deq(enq(Coadă_Vidă, e)) = Coadă_Vidă
deq(enq(q, e)) = Enq(Deq(q), e),     dacă q nu e vidă
deq(Coadă_Vidă) = eroare
empty(Coadă_Vidă) = 1
empty(Enq(q, e)) = 0
```

3.2. Operații cu cozi

- Crearea unei cozi vide: `new ()`
- Copierea elementului din vârf fără a-l șterge din coadă; (dacă coada este vidă se produce eroare): `front(Q)`
- Inserarea unui element în coadă: `enq(Q, x)`
- Preluarea elementului de la începutul cozii și ștergerea lui din coadă; (dacă coada este vidă se produce eroare): `deq(Q)`
- Test coadă vidă: `empty(Q)`
- Test coadă plină: `full(Q)`
- Inspectarea elementului de la începutul cozii. `front(Q)`
- Inspectarea elementului de la sfârșitul cozii. `back(Q)`

3.3. Interfața TAD Coadă

```
// Fisierul coada.h
#ifndef _Q_H
#define _Q_H
struct coada;
typedef struct coada *Coadă;
typedef void* Elem;
Coadă Q_New(int); //constructor cu tablou circular
Coadă Q_New(); //constructor cu lista inlantuita
void Q_Delete(Coadă *pQ); //destructor
int Q_Empty(Coadă Q); //test coada vida
int Q_Full(Coadă Q); //test coada plina
void Enq(Coadă Q, Elem x); //pune din coada
Elem Front(Coadă Q); //citeste primul element
Elem Back(Coadă Q); //citeste ultimul element
Elem Deq(Coadă Q); //citeste si sterge
#endif
```

3.4. Aplicații cu cozi

1. *Problema lui Josephus:* n copii se așează în cerc, se numerotează în sens orar cu $1, 2, \dots, n$ și rostesc o poezie formată din c cuvinte (de tipul "ala bala portocala ..."). Fiecăruia, începând cu primul i se asociază un cuvânt din poezie. Cel care primește ultimul cuvânt este eliminat din cerc. Jocul continuă,

începând poezia de la următorul copil, până când se elimină toți copiii. Folosind numerotarea inițială, să se afișeze ordinea ieșirii copiilor din joc.

Rezolvare:

Se va folosi o coadă în care se introduc numerele $1, 2, \dots, n$.

Rotirea unui cuvânt din poezie se traduce printr-o permutare circulară (o scoatere a unui element urmată de o inserare a lui în coadă). După c permutări circulare, elementul scos nu mai este pus la loc în coadă, ci afișat. Programul se termină în momentul în care coada ajunge vidă.

```
#include "coada.h"
#include <stdio.h>
#define MAX 20
int main(){
    char nume[MAX][30];
    int np, i, *pj, *pi, c, n;
    scanf("%d", &n);
    printf("Numele celor %2d persoane\n", n);
    for (i=0; i<n; i++)
        scanf("%s", nume[i]);
    //citeste numar de cuvinte poezie
    scanf("%d", &c);
    //pune persoanele in coada
    Coada Joc = Q_New(n);
    for (i=0; i<n; i++){
        pi = (int*)malloc(sizeof(int));
        *pi = i;
        Enq(Joc, pi);
    }
    while(!Q_Empty(Joc)) {
        for (i=0; i<c-1; i++) {
            pj = Deq(Joc);    //permutare circulara
            Enq(Joc, pj);
        }
        pj = Deq(Joc);        //scoate ultimul din joc
        printf("%s\n", nume[*pj]);
        free(pj);
    }
}
```

2. Sortarea pe ranguri (radix sort): Pentru a sorta un șir de numere întregi x prin metoda "radix sort" se folosesc 10 cozi corespunzătoare cifrelor $0, 1, \dots, 9$, cozi inițial vide.

Sortarea are loc în următorii pași:

1. COLECȚII DE DATE

- 1.1. Operații specifice colecțiilor
- 1.2. Exemple de colecții de date
- 1.3. Parcurgerea colecțiilor
- 1.4. Criterii de clasificare a colecțiilor
- 1.5. Criterii de alegere a colecțiilor
- 1.6. Tipuri abstracte de date
 - 1.6.1. Specificarea Tipurilor Abstracte de Date
 - 1.6.2. Implementarea Tipurilor Abstracte de Date
 - 1.6.3. Contractul client – furnizor
 - 1.6.4. Criterii de proiectare ale interfețelor TAD
 - 1.6.5. Eficiența utilizării structurilor de date
- 1.7. Compilare separată
 - 1.7.1. Utilitarul make
- 1.8. Complexitatea algoritmilor
 - 1.8.1. Notății asimptotice

2. STIVE

- 2.1. Specificarea TAD Stivă
- 2.2. Interfața TAD Stivă
- 2.3. Aplicații cu stive
- 2.4. Implementarea stivelor
 - 2.4.1. Implementare cu tablouri
 - 2.4.2. Implementare cu liste înlănțuite
- 2.5. Probleme propuse

3. COZI

- 3.1. Specificarea TAD Coadă
- 3.2. Operații cu cozi
- 3.3. Interfața TAD Coadă
- 3.4. Aplicații cu cozi
- 3.5. Implementarea cozilor
 - 3.5.1. Implementare cu tablou circular
 - 3.5.2. Implementare cu liste înlănțuite
- 3.6. Probleme propuse

4. LISTE

- 4.1. Generalități
- 4.2. Operații specifice listelor
- 4.3. Specificarea TAD Listă

- 4.4. Interfața TAD Listă
- 4.5. Aplicații cu liste
- 4.6. Implementarea listelor
- 4.6.1. Implementare cu tablouri
- 4.6.2. Implementare cu liste dublu înlănțuite
- 4.7. Probleme propuse

5. ARBORI BINARI

- 5.1. Definiții și generalități
- 5.2. Operații specifice TAD Arbore Binar
- 5.3. Traversarea arborilor binari și aplicații ale traversărilor
- 5.3.1. Traversarea în preordine
- 5.3.2. Traversarea în postordine
- 5.3.3. Traversarea în inordine
- 5.3.4. Traversarea în lățime
- 5.4. Implementarea arborilor binari
- 5.5. Probleme propuse

6. ARBORI GENERALI (MULTICĂI)

- 6.1. Interfața TAD Arbore General
- 6.2. Transformarea unui arbore general într-un arbore binar
- 6.3. Implementarea arborilor generali
- 6.3.1. Prin tabel de cursori la predecesori
- 6.3.2. Cu tablouri, prin liste de adiacențe
- 6.3.3. Prin tablourile PrimFiu și UrmătorFrate
- 6.3.4. Cu pointeri, cu listă de succesori și pointer la predecesor
- 6.4. Traversarea arborilor generali

7. ARBORI DE CĂUTARE

- 7.1. Arbori binari de căutare
- 7.1.1. Definiții și generalități
- 7.1.2. Operații specifice TAD Arbore Binar de Căutare
- 7.1.2.1. Căutarea unei chei
- 7.1.2.2. Inserarea unui nod
- 7.1.2.3. Cheia maximă dintr-un arbore binar de căutare
- 7.1.2.4. Succesorul și predecesorul unui nod
- 7.1.2.5. Ștergerea unui nod
- 7.2. Arbori echilibrați
- 7.2.1. Rotații în arbori binari de căutare
- 7.2.2. Arbori AVL
- 7.2.2.1. Definiții și generalități
- 7.2.2.2. Calculul factorului de echilibrare pentru o rotație simplă
- 7.2.2.3. Inserarea unui nod într-un arbore AVL
- 7.2.3. Arbori bicolori (roșii-negri)

- 7.2.3.1. Definiții și generalități
- 7.2.3.2. Funcții suplimentare pentru arbori bicolori
- 7.2.3.3. Inserarea unui nod într-un arbore bicolor
- 7.2.3.4. Ștergerea unui nod dintr-un arbore bicolor
- 7.2.4. Structuri de date pentru memoria externă
 - 7.2.4.1. Arbori 2-3
 - 7.2.4.1.1. Inserarea unei chei într-un arbore 2-3
 - 7.2.4.2.2. Ștergerea unei chei dintr-un arbore 2-3
 - 7.2.4.2. Arbori B
 - 7.2.4.2.1. Creerea unui arbore B
 - 7.2.4.2.2. Căutarea unei chei într-un arbore B
 - 7.2.4.2.3. Inserarea unei chei într-un arbore B
 - 7.2.4.2.4. Variante de arbori B
- 7.3. Probleme propuse

8. COZI PRIORITARE

- 8.1. Specificarea TAD Coadă Prioritară
- 8.2. Interfața TAD Coadă Prioritară
- 8.3. Exemple
- 8.4. Arbori parțial ordonați (heapuri binare)
 - 8.4.1. Definiții și terminologie
 - 8.4.2. Transformarea unui tablou într-un heap
 - 8.4.3. Sortare prin metoda heapurilor (heapsort)
 - 8.4.4. Implementarea cozilor cu priorități folosind heapuri binare
 - 8.4.5. Aplicații ale cozilor prioritare
- 8.5. Probleme propuse

9. TABELE DE DISPERSIE

- 9.1. Definiții și terminologie
- 9.2. Funcții de dispersie
- 9.3. Strategii de rezolvare a coliziunilor
 - 9.3.1. Dispersie deschisă
 - 9.3.1.1. Interfață dispersie deschisă
 - 9.3.1.2. Implementare dispersie deschisă
 - 9.3.2. Dispersie închisă
 - 9.3.2.1. Verificare liniară
 - 9.3.2.2. Verificare pătratică
 - 9.3.2.3. Dispersie dublă
 - 9.3.2.4. Interfață dispersie închisă
 - 9.3.2.5. Implementare dispersie închisă
 - 9.3.2.6. Redispersare
- 9.4. Eficiența operațiilor în tabelele de dispersie
- 9.5. Probleme propuse

10. GRAFURI

- 10.1. Elemente de teoria grafurilor: definiții și terminologie
- 10.2. Operații asociate vârfurilor
- 10.3. Operații asociate arcelor
- 10.4. TAD Graf
- 10.5. Iterarea vârfurilor și arcelor grafului
- 10.6. Implementarea operațiilor independente de reprezentarea grafului
- 10.7. Implementarea grafurilor cu matrice de adiacențe
- 10.8. Reprezentarea grafurilor prin liste de adiacențe
- 10.9. Implementarea iteratorilor
- 10.10. Metode de explorare a grafurilor
 - 10.10.1. Traversarea în adâncime a unui graf
 - 10.10.2. Traversarea în lățime a unui graf
- 10.11. Sortare topologică
- 10.12. Determinarea componentelor tare conexe
- 10.13. Colecții de mulțimi disjuncte
- 10.14. Determinarea arborelui de acoperire de cost minim
 - 10.14.1. Algoritmul Kruskal
 - 10.14.2. Algoritmul Prim
- 10.15. Algoritmi pentru drumuri minime în grafuri
 - 10.15.1. Algoritmul Dijkstra
 - 10.15.2. Algoritmul Bellman-Ford
 - 10.15.3. Drumuri minime între toate perechile de vârfuri
- 10.16. Probleme propuse