3. COZI

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

Cozile sunt utile în aplicaţiile de simulare, în traversarea grafurilor în lăţime, în algoritmii cu arbori şi grafuri.

Ordinea de extragere din coadă este aceeaşi cu ordinea de introducere în coadă, ceea ce sugerează şi aplicaţiile pentru o asemenea structură: simularea unor procese de servire de tip *producător - consumator* sau *vânzător - client*. În astfel de situaţii coada de aşteptare este necesară pentru a acoperi o diferenţă temporară între ritmul de servire şi ritmul cererilor solicitanţilor (clienţilor).

Cozi de servire se pot forma la comunicarea de date între un emiţător şi un receptor care au viteze diferite sau la staţiile 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: **Coada, Elem, int**

Semnături:

```
new   :                → Coada_Vidă
front :          Coada → Elem
back  :          Coada → Elem
enq   :  Coada × Elem  → Coada
deq   :          Coada → Coada
empty :          Coada → int
```

Axiome:

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

3.2. Operații cu cozi

- Crearea unei cozi vide: `new ()`
- Copierea elementului din vârf fără a-l şterge din coadă; (dacă coada este vidă se produce eroare): `front(Q)`
- Inserarea unui element în coadă: `enq(Q, x)`
- Preluarea elementului de la inceputul cozii şi ştergerea 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 sfârşitul cozii. `back(Q)`

3.3. Interfața TAD Coadă

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

3.4. Aplicații cu cozi

1. ***Problema lui Josephus***: `n` copii se aşează în cerc, se numerotează în sens orar cu `1,2,...,n` şi 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 primeşte 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 afişeze ordinea ieşirii copiilor din joc.

Rezolvare:

Se va folosi o coadă în care se introduc numerele `1,2,...,n` .

Rostirea 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 afişat. Programul se termină în momentul în care coada ajunge vida.

```
#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 şir de numere întregi `x` prin metoda "radix sort" se folosesc 10 cozi corespunzătoare cifrelor `0,1,...,9` , cozi inițial vide.

Sortarea are loc în următorii paşi:

V-248

Colecții de Date

Valeriu Iorga

10. GRAFURI