Tehnici de programare

Stiva si coada

Structuri de date in programare

În programare, structurile de date sunt utilizate pentru a organiza și stoca date într-un mod structurat și eficient. Acestea sunt esențiale în dezvoltarea aplicațiilor și a algoritmilor, deoarece permit programatorilor să organizeze și să gestioneze seturi de date într-un mod logic și coerent.

O structură de date poate fi definită ca o colecție de date și o serie de operații care pot fi efectuate pe acestea. Structurile de date pot fi de diferite tipuri, cum ar fi vectori, matrice, liste, stive, cozi, arbori, grafuri și multe altele. Fiecare tip de structură de date este conceput pentru a rezolva probleme specifice în ceea ce privește stocarea, accesarea și manipularea datelor.

Doua structuri de date usor de implementat sunt stiva si coada.

Structurile de date Stiva si Coada

Stiva si coada sunt doua structuri de date importante in programare, utilizate pentru a stoca si organiza date intr-un mod specific.

Stiva este o structura de date LIFO (last in, first out), ceea ce inseamna ca ultimul element adaugat in stiva este primul element extras din stiva. Aceasta structura este asemanatoare cu o stiva de farfurii: se poate adauga o farfurie noua doar in varful stivei si nu se poate scoate decat ultima farfurie adaugata. Un exemplu de folosire a stivei in programare este pentru a urmari apelurile recursive.

Coada este o structura de date FIFO (first in, first out), ceea ce inseamna ca primul element adaugat in coada o sa fie primul element extras din coada. Aceasta structura este asemanatoare cu o coada la supermarket: primul client care ajunge la coada este primul care este servit si paraseste coada.
In programare, coada este folosita pentru a gestiona sarcinile in ordinea in care au fost adaugate sau pentru a stoca elementele in asteptare pana cand sunt procesate.

Structura de date STIVA

Operațiile fundamentale pe o stivă sunt push (adaugarea unui element în vârful stivei) și pop (scoaterea unui element din vârful stivei). O altă operație comună este peek, care permite vizualizarea valorii din vârful stivei fără a o scoate din stivă.

De exemplu, să presupunem că avem o stivă goală și adăugăm elementele 5, 8 și 3 în această ordine. Vârful stivei va fi 3, deoarece acesta este ultimul element adăugat. Dacă scoatem elementul din vârful stivei, vom obține valoarea 3, iar vârful stivei va fi actualizat la 8. Dacă scoatem încă un element, vârful stivei va fi actualizat la 5.

Stivele sunt folosite în diverse aplicații în programare, de exemplu, în evaluarea expresiilor matematice, în navigarea prin istoricul browserului sau în implementarea funcțiilor de undo/redo. Acestea sunt utile

în situațiile în care ordinea de procesare a datelor este importantă, iar accesul la date se face în ordinea inversă în care au fost adăugate în stivă.

O stiva poate fi implementata in C utilizand un vector sau o lista simpla inlantuita. In continuare, vom parcurge cum putem implementa o stiva utilizand un vector.

Definim o constanta care sa reprezinte dimensiunea maxima a stivei si o variabila care sa reprezinte varful stivei:

```
#define MAX_SIZE 100
int top = -1;
```

Definim vectorul care va reprezenta stiva:

```
int stack[MAX_SIZE];
```

Definim o functie care sa adauge un element in stiva:

```
void push(int item) {
   if(top == MAX_SIZE-1) {
      printf("Stiva este plina\n");
      return;
   }
   stack[++top] = item;
}
```

Aceasta functie verifica daca stiva este deja plina si afiseaza un mesaj de eroare in caz afirmativ. In caz contrar, adauga elementul in varful stivei si incrementeaza varful.

Definim o functie care sa extraga un element din stiva:

```
int pop() {
   if(top == -1) {
      printf("Stiva este goala\n");
      return -1;
   }
   return stack[top--];
}
```

Aceasta functie verifica daca stiva este goala si afiseaza un mesaj de eroare in caz afirmativ. In caz contrar, extrage elementul din varful stivei si decrementaza varful.

Definim o functie care sa afiseze elementele din stiva:

```
void display() {
   if(top == -1) {
      printf("Stiva este goala\n");
      return;
   }
   printf("Elementele din stiva sunt: ");
   for(int i=top; i>=0; i--)
      printf("%d ",stack[i]);
   printf("\n");
}
```

Aceasta functie verifica daca stiva este goala si afiseaza un mesaj de eroare in caz afirmativ. In caz contrar, afiseaza elementele din stiva de la varful stivei la baza stivei.

Exemplu de utilizare:

```
int main() {
  push(10);
  push(20);
  push(30);
  display();
  printf("Elementul extras din stiva: %d\n",pop());
  display();
  return 0;
}
```

Acest exemplu adauga trei elemente in stiva, afiseaza elementele din stiva, extrage un element din stiva si afiseaza elementele din stiva actualizate. Output-ul acestui exemplu va fi:

```
Elementele din stiva sunt: 30 20 10
Elementul extras din stiva: 30
Elementele din stiva sunt: 20 10
```

Aceasta implementare a unei stive in C poate fi imbunatatita prin utilizarea unei liste simplu inlantuite, dar ideea de baza ramane aceeasi. O sa revenim la stiva implementata cu ajutorul listelor cand vom parcurge listele simplu si dublu inlantuite.

Structura de date COADA

Operațiile fundamentale pe o coadă sunt enqueue (adaugarea unui element la capătul cozii) și dequeue (scoaterea unui element din capul cozii). O altă operație comună este qpeek, care permite vizualizarea valorii din capul coadei fără a o scoate din coadă.

De exemplu, să presupunem că avem o coadă goală și adăugăm elementele 5, 8 și 3 în această ordine. Capul cozii va fi 5, iar coada cozii va fi 3. Dacă scoatem elementul din capul cozii, vom obține valoarea 5, iar capul cozii va fi actualizat la 8. Dacă scoatem încă un element, capul cozii va fi actualizat la 3.

Cozile sunt folosite în diverse aplicații în programare, de exemplu, în simularea evenimentelor (unde evenimentele sunt adăugate în coadă în ordinea în care apar) sau în procesarea cererilor într-un sistem de procesare a datelor (unde cererile sunt procesate în ordinea în care au fost adăugate în coadă). Acestea sunt utile în situațiile în care ordinea de procesare a datelor este importantă, dar accesul la date se face în ordinea în care au fost adăugate în coadă.

O coada poate fi implementata in C utilizand un vector sau o lista simpla inlantuita. In continuare, vom parcurge cum se poate implementa o coada utilizand un vector.

Definim o constanta care sa reprezinte dimensiunea maxima a cozii si doua variabile care sa reprezinte inceputul si sfarsitul cozii:

```
#define MAX_SIZE 100
int front = -1, rear = -1;
```

Definim vectorul care va reprezenta coada:

```
int queue[MAX_SIZE];
```

Definim o functie care sa adauge un element in coada:

```
void enqueue(int item) {
  if(rear == MAX_SIZE-1) {
    printf("Coada este plina\n");
    return;
  }
  if(front == -1) front = 0;
  queue[++rear] = item;
}
```

Aceasta functie verifica daca coada este deja plina si afiseaza un mesaj de eroare in caz afirmativ. In caz contrar, adauga elementul in sfarsitul cozii si actualizeaza variabila "rear". In cazul in care coada este goala, actualizeaza si variabila "front".

Definim o functie care sa extraga un element din coada:

```
int dequeue() {
  if(front == -1 || front > rear) {
    printf("Coada este goala\n");
    return -1;
  }
  return queue[front++];
}
```

Aceasta functie verifica daca coada este goala sau daca variabila "front" depaseste variabila "rear" si afiseaza un mesaj de eroare in caz afirmativ. In caz contrar, extrage elementul din inceputul cozii si incrementeaza variabila "front".

Definim o functie care sa afiseze elementele din coada:

```
void display() {
  if(front == -1 || front > rear) {
    printf("Coada este goala\n");
    return;
  }
  printf("Elementele din coada sunt: ");
  for(int i=front; i<=rear; i++)
    printf("%d ",queue[i]);
  printf("\n");
}
```

Aceasta functie verifica daca coada este goala sau daca variabila "front" depaseste variabila "rear" si afiseaza un mesaj de eroare in caz afirmativ. In caz contrar, afiseaza elementele din coada de la inceputul cozii la sfarsitul cozii.

Exemplu de utilizare:

```
int main() {
  enqueue(10);
  enqueue(20);
  enqueue(30);
  display();
  printf("Elementul extras din coada: %d\n",dequeue());
```

```
  display();
  return 0;
}
```

Acest exemplu adauga trei elemente in coada, afiseaza elementele din coada, extrage un element din coada si afiseaza elementele din coada actualizate. Output-ul acestui exemplu va fi:

```
Elementele din coada sunt: 10 20 30
Elementul extras din coada: 10
Elementele din coada sunt: 20 30
```

Aceasta implementare a unei cozi in C poate fi imbunatatita prin utilizarea unei liste simplu inlantuite, dar vom reveni cand vom studia listele.

Cozi "hardware"

O coadă poate fi utilizată într-un buffer hardware pentru a gestiona datele care sunt transmise între dispozitivele hardware sau software. Un buffer este o zonă de memorie temporară care este utilizată pentru a stoca datele înainte de a fi transmise mai departe la dispozitivul receptor. Un buffer poate fi reprezentat de un registru, o memorie cache sau un alt tip de memorie.

În cazul unui buffer hardware, o coadă poate fi utilizată pentru a asigura o transmitere eficientă a datelor între dispozitive. Datele sunt stocate într-o coadă în ordinea în care sunt primite de la sursă. Apoi, aceste date sunt transferate în buffer și ulterior transmise la dispozitivul receptor.

Utilizarea unei cozi într-un buffer hardware poate fi utilă în situațiile în care viteza de transmitere a datelor este mai mare decât viteza de procesare a acestora. În astfel de cazuri, o coadă poate fi utilizată pentru a stoca temporar datele și pentru a evita pierderea acestora.

De exemplu, să presupunem că un dispozitiv emite date la o rată mai mare decât viteza de procesare a datelor de către dispozitivul receptor. În acest caz, o coadă poate fi utilizată pentru a stoca datele temporar înainte de a fi transferate la buffer și ulterior la dispozitivul receptor. Această abordare asigură faptul că toate datele sunt transmise și procesate în ordinea corectă, evitând pierderea acestora.

APLICATII

Aplicatia 3.1 Implementati pe calculator structura de date stiva, asa cum este descrisa in laborator.

Aplicatia 3.2 Implementati pe calculator structura de date coada, asa cum este descrisa in laborator.

Aplicatia 3.3 Sa se implementeze cu ajutorul unei stive functia undo dintr-un fisier text. Functia undo permite utilizatorului sa se revina la o versiune mai veche a documentului, eliminand ultimile modificari. Pentru a implementa functia undo cu o stiva fiecare modificare va fi retinuta pe stiva, iar cand e necesar sa se revina la o versiune anterioara se va elimina din stiva ultima modificare adaugata. Pe stiva se va retine la fiecare modificare: textul nou scris (sir de maxim 100 de caractere), randul unde a fost scris (int).

Aplicatia 3.4 Sa se implementeze cu ajutorul unei cozi un buffer care retine temporar informatiile primite de la diversi transmitatori (informatia este adaugata in coada) si le transmite mai apoi catre

un receptor (informatia este eliminata din coada). Informatia primita in buffer are urmatoare structura: numar identificare transmitator (int), mesaj (sir de caractere de maxim 256 caractere).