Lab3: Structura de date stiva si coada

Formatori:
Tutor: Stiegelbauer Paul
Tutor: Iovanovici Alexandru
+1

Data de începere a cursului:
19.02.2024
Utilizatori înscriși
Calendar
Note

Lab3: Structura de date stiva si coada

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.

De exemplu, un vector este o structură de date simplă care stochează un număr finit de elemente de același tip. Vectorii sunt utilizați pentru a stoca date care sunt accesate în mod regulat și care trebuie să fie ordonate într-un anumit mod. Pe de altă parte, o listă este o structură de date mai complexă care poate fi utilizată pentru a stoca un număr variabil de elemente și pentru a permite inserarea și ștergerea elementelor în timp real.

În general, alegerea structurii de date potrivite depinde de necesitățile specifice ale aplicației sau algoritmului. Odată ce o structură de date a fost aleasă, aceasta poate fi utilizată pentru a organiza și gestiona datele într-un mod eficient, ceea ce poate îmbunătăți performanța aplicației sau a algoritmului.

Accesul aleator într-un vector se referă la abilitatea de a accesa o valoare dintr-un anumit index sau poziție în interiorul vectorului, fără a fi necesar să parcurgem toate elementele până la acea poziție. De exemplu, dacă avem un vector cu 10 elemente și dorim să accesăm valoarea din poziția a șaptea, accesul aleator ne permite să accesăm acea valoare direct din vector, fără a fi necesar să parcurgem primele șase elemente ale vectorului.

Accesul aleator este un avantaj major al vectorilor în comparație cu alte structuri de date, cum ar fi liste sau arbori, care necesită parcurgerea elementelor în ordine pentru a accesa o anumită valoare. În vectori, fiecare element este stocat într-o poziție de memorie contiguă, ceea ce face accesul aleator foarte rapid și eficient. În plus, în majoritatea limbajelor de programare, accesul aleator la un element dintr-un vector se face printr-o simplă operație de indexare, care este ușor de utilizat și de înțeles.

Cu toate acestea, accesul aleator are și câteva dezavantaje. În primul rând, inserarea și ștergerea elementelor în interiorul unui vector pot fi mai lente și mai complexe decât în alte structuri de date, deoarece trebuie să mutăm toate elementele care se află după poziția inserată sau ștearsă. În plus, vectorii au o dimensiune fixă și nu pot fi extinși sau redimensionați la nevoie, ceea ce poate fi o limitare în anumite situații.

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

Stiva

În programare, o stivă este o structură de date lineară, în care datele sunt stocate în ordine liniară, dar accesul la date se face doar la un capăt, numit vârful stivei. Această structură de date se mai numește și Last-In-First-Out (**LIFO**), adică ultimul element adăugat în stivă este primul element care este scos din stivă.

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.

Coada

În programare, o coadă este o structură de date lineară similară cu o stivă, în care datele sunt stocate în ordine liniară, dar accesul la date se face la două capete diferite, numite capul coada (front) și coada coada (rear). Această structură de date se mai numește și First-In-First-Out (**FIFO**), adică primul element adăugat în coadă este primul element care este scos din coadă.

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.

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.

Aplicatia 1:

Sa se defineasca si implementeze un tip de date **Stack_t** si operatiile specifice unei stive astfel:

```
Stack_t push(Stack_t, Element_t);
Element_t pop(Stack_t);
Element_t peek(Stack_t);
```

a) structura de date va fi implementata in mod static (fara redimensionarea capacitatii); se vor verifica operatiile la depasire de overflow si underflow;

b) structura de date va fi implementata in mod dinamic (cu utilizarea operatiilor de gestiune dinamica a memoriei); in continuare operatiile pot da flag-uri de eroare (var globala)

Aplicatia 2

Sa se defineasca si implementeze un tip de date **Queue_t** si operatiile specifice unei cozi astfel:

```
Queue_t enque(Queue_t, Element_t);
Element_t dequeue(Queue_t);
```

a) structura de date va fi implementata in mod static (fara redimensionarea capacitatii); se vor verifica operatiile la depasire de overflow si underflow;

b) structura de date va fi implementata in mod dinamic (cu utilizarea operatiilor de gestiune dinamica a memoriei); in continuare operatiile pot da flag-uri de eroare (var globala)

Aplicatia 3

Dacă este dat un șir s care conține doar caracterele '(', ')', '{', '}', '[' și ']', să se determine dacă șirul de intrare este valid.

Un șir de intrare este valid dacă:

Parantezele deschise trebuie închise cu aceeași tipologie de paranteze. Parantezele deschise trebuie completate în ordinea corectă. Fiecare paranteză închisă are o paranteză deschisă corespunzătoare de același tip.

Sugestie: Vom itera prin caracterele șirului. Pentru fiecare paranteză deschisă, vom folosi o stivă pentru a o stoca. De asemenea, vom utiliza o hartă care ne indică parantezele închise corespunzătoare pentru fiecare paranteză deschisă.

Pentru fiecare paranteză închisă, verificăm dacă elementul din vârful stivei este paranteza deschisă corespunzătoare. Dacă este, scoatem elementul din vârful stivei și continuăm.

Dacă elementul din vârful stivei nu este paranteza deschisă corespunzătoare sau stiva este goală, vom returna false. Dacă stiva este goală, șirul de intrare este valid. În caz contrar, nu este valid.

Observatii:
Pentru aceasta saptamana NU este necesar sa implementati codul sub forma de "biblioteca". Aceasta va fi abordata in saptamana urmatoare impreuna cu Makefile-urile.

◄ Inregistrare curs S3, anul 2021-2022!!!

Sari la...

Inregistrare curs S4, Vineri, anul 2021-2022!!! ►