5. Sor adatszerkezet - Queue

5.1 Sor definiálása

Definíció 5.1.1 A sor (queue) egy olyan adatszerkezet, amely az elemeket egy adott sorrendben tárolja és kezeli. A sorban az új elemek az utolsó pozícióra kerülnek és a korábban hozzáadott elemek az első pozícióból lesznek eltávolítva (ezt hívjuk **FIFO**, azaz **"first-in, first-out"** elvnek).

5.2 Sorok osztályozása

A sor osztályozása implementációja alapján

- **Tömbösített sor**: Az adatelemeket egy tömbben tároljuk, és az első és az utolsó mutatók segítségével hivatkozunk arra, hogy hol vannak az aktuális sor elemei.

- **Láncolt sor**: Az adatelemeket egy láncolt listában tároljuk, ahol minden elem tartalmazza a következő elemre mutató címet.

A sor osztályozása működése szerint:

- **Egyszerű sor (Regular queue)**: Csak annyi elemet szúrhatunk be a sorba, amennyi a sor kapacitása. Ha a sor végére értünk, következő elemet nem szúrhatunk be, még akkor sem, ha a sor elejéről eltávolítottunk elemet.

- **Körkörös vagy cirkuláris sor**: lehetővé teszi az elemek folyamatos beszúrását a sorba és eltvolítását a sorból. Az alapvető különbség az Egyszerű sorhoz képest, hogy a Cirkuláris sorban az utolsó elem után a következő elem a sor elején jelenik meg, vagyis a sor körbejárható.

- **Elsőbbségi vagy prioritási sor**: ebben az esetben az elemek prioritásuk alapján kerülnek rendezésre. Az alacsonyabb prioritású elemek általában előbb kerülnek eltávolításra a sorból, míg a magasabb prioritásúak hosszabb ideig maradnak a sorban.

- **Kétvégű sor**: (deque - double-ended queue) egy olyan adatstruktúra, amely lehetővé teszi az elemek hozzáadását és eltávolítását mindkét végéről, vagyis az elemek beszúrása és törlése történhet az elején és a végén egyaránt. A deque támogatja az olyan műveleteket, mint az elejéhez való beszúrás (push_front), a végéhez való beszúrás (push_back), az elejéről való törlés (pop_front), és a végéről való törlés (pop_back).

5.3 Alapvető műveletek

1. **Enqueue** (új elem bevitele): Az új elem hozzáadása a sorhoz a sor végéhez. Az új elem a sor utolsó eleme lesz.
2. **Dequeue** (elem eltávolítása): Az első elem eltávolítása a sor elejéről.
3. **IsEmpty** (Üres-e): A sor ürességének ellenőrzése. Ha a sorban nincsenek elemek, akkor igaz értéket ad vissza, egyébként hamisat.
4. **IsFull** (Tele van-e): A sor tele van-e elemekkel. Ha a sor elemeinek száma eléri a sor méretét, akkor igaz értéket ad vissza, egyébként hamisat.

5.4 Sor alkalmazása

1. Operációs rendszerek: Az operációs rendszerekben a sor adatszerkezetet gyakran használják azokra a folyamatokra, amelyeket az operációs rendszer felügyel. Például, amikor az operációs rendszer kezeli a beérkező kéréseket, vagy amikor az általános célú feladatokat kezeli.
2. Adatbázisok: Az adatbázisokban a sor adatszerkezetet használják a tranzakciók feldolgozására. Amikor egy tranzakció befejeződött, akkor az adatbázis rendszer a sorból veszi ki a következő tranzakciót, amelyet feldolgozni kell.
3. Kommunikáció: Az üzenetküldő alkalmazásokban, például az email szerverekben vagy a chat alkalmazásokban, a sor adatszerkezetet használják a beérkező üzenetek kezelésére. Az üzenetek a sorban várakoznak a feldolgozásra, és az alkalmazás sorrendben dolgozza fel őket.
4. Programozási nyelvek: A sor adatszerkezetet számos programozási nyelvben használják. Például a Python nyelvben a "queue" modul segítségével használható a sor adatszerkezet, míg a C++ nyelvben a "queue" STL (Standard Template Library) osztályt használhatjuk.
5. Algoritmusok: Számos algoritmus használja a sor adatszerkezetet a hatékony feldolgozás érdekében. Például a BFS (Breadth-First Search) algoritmus, amely a gráfokban való keresést teszi lehetővé, a sor adatszerkezetet használja a csúcsok feldolgozásához.

5.5 Sorral kapcsolatos függvények

Definiálunk egy Queue struktúrát, amely egy sor (queue) adatszerkezetet reprezentál. A struktúra 4 adattagot tartalmaz:

- **capacity** (kapacitás): Az adatszerkezet maximális méretét határozza meg, vagyis azt, hogy hány elemet tudunk eltárolni a sorban.

- **front**: Az első elem pozícióját jelöli, vagyis azt az indexet, ahol a sor első eleme található.

- **rear**: Az utolsó elem pozícióját jelöli, vagyis azt az indexet, ahol a sor utolsó eleme található.

- **elements**: Egy pointer azokra az elemekre, amelyeket eltárolunk a sorban. A pointer dinamikusan lefoglalt memóriacímet tartalmaz, amely az elemeket mutatja.

Egyszerű sor függvényei

```
typedef struct {
int capacity;
int front;
int rear;
int *elements;
}Queue;
```

5.5.1 Helyfoglalás

```
void createQueue(int capacity, Queue *queue) {
    queue->capacity = capacity;
    queue->front = queue->rear = -1;
    queue->elements = (int *) calloc(queue->capacity, sizeof(int));
    if (!queue->elements) {
        printf(MEMORY_ALLOCATION_ERROR_MESSAGE);
        exit(-1);
    }
}
```

A függvény két bemeneti paramétert használ: a kapacitást és a mutatót az inicializálandó sorra. A függvény először beállítja a kapacitást a bemeneti paraméter alapján. Ezután mindkét mutatót - a front és a rear mutatót - -1-re állítja, ami azt jelzi, hogy a sor üres.

Ezután a függvény dinamikusan lefoglalja a memóriát az adatszerkezet elemeinek tárolására. Az adatszerkezet típusa arra utal, hogy az adatelemek egészek (int) vannak eltárolva. A dinamikus memóriafoglalás int * típusú mutatót ad vissza, amely az egészeket tartalmazza. A függvény ellenőrzi, hogy a memóriafoglalás sikeres-e, és ha nem, akkor hibaüzenetet ír ki, majd kilép a programból.

Összességében a függvény felelős az inicializált sor struktúrájának létrehozásáért és az egész adatelemek memóriájának lefoglalásáért. Ez az adatszerkezet használatra kész, és készen áll arra, hogy adatokat tároljon.

5.5.2 Helyfelszabadítás

```
void destroyQueue(Queue *queue) {
    free(queue->elements);
    queue->front = queue->rear = -1;
    queue->capacity = 0;
    queue = NULL;
}
```

A függvény először felszabadítja az adatszerkezet által lefoglalt memóriát a free() függvénnyel, majd a front és rear mutatókat visszaállítja -1-re, hogy az adatszerkezet újra üres legyen. A függvény aztán nullára állítja az adatszerkezet kapacitását, majd a queue mutatót nullra állítja, hogy biztosítsa a memória felszabadítását.

5.5.3 Üres sor tesztelése

```
bool isEmpty(Queue queue) {
    return queue.front == -1;
}
```

A függvény egy bemeneti paramétert használ, egy Queue típusú adatszerkezetet, amelyet ellenőrizni szeretnénk, hogy üres-e.

A függvény egyszerűen ellenőrzi, hogy az adatszerkezet front mutatója -1-e. Ha az, akkor azt jelenti, hogy az adatszerkezet üres, és a függvény true értékkel tér vissza, máskülönben false értékkel tér vissza, mert az adatszerkezet nem üres.

5.5.4 Telített sor tesztelése

```
bool isFull(Queue queue) {
    return queue.rear == queue.capacity-1;
}
```

A függvény egy bemeneti paramétert használ, egy Queue típusú adatszerkezetet, amelyet ellenőrizni szeretnénk, hogy tele van-e. A függvény egyszerűen ellenőrzi, hogy az adatszerkezet rear mutatója egyenlő-e a kapacitással minusz egyel. Ha igen, akkor azt jelenti, hogy az adatszerkezet tele van, és a függvény true értékkel tér vissza, máskülönben a függvény false értékkel tér vissza.

5.5.5 Elem beszúrása - Enqueue

```
void enqueue(Queue *queue, int item) {
    if(isFull(*queue)) {
        printf(FULL_MESSAGE);
        return;
    }
    if(isEmpty(*queue)) {
        queue->front = 0;
    }
    queue->elements[++queue->rear]=item;
}
```

A függvény két bemeneti paramétert használ: a queue egy Queue típusú adatszerkezet, amelyhez az elemet hozzá akarjuk adni, és az item egy int típusú érték, amelyet hozzá akarunk adni az adatszerkezethez.

A függvény először ellenőrzi, hogy az adatszerkezet tele van-e vagy sem a isFull() függvénnyel. Ha az adatszerkezet tele van, a függvény visszatér, és nem ad hozzá új elemet. Ha az adatszerkezet nincs tele, akkor ellenőrzi, hogy az adatszerkezet üres-e vagy sem a isEmpty() függvénnyel. Ha az üres, akkor a front mutatót az első elemre állítja, a rear mutatót növeli, majd az elemet az adatszerkezet végére helyezi.

5.5.6 Elem eltávolítása - Dequeue

```
int dequeue(Queue *queue) {
    if(isEmpty(*queue)) {
        printf(EMPTY_MESSAGE);
        return NULL;
    }
    int pos = queue->front;
    if(queue->front == queue->rear) {
        queue->front = queue->rear = -1;
    }
    else {
        queue->front++;
    }
    return queue->elements[pos];
}
```

A függvény egy bemeneti paramétert használ: a queue egy Queue típusú adatszerkezet, amelyből az első elemet el szeretnénk távolítani.

A függvény először ellenőrzi, hogy az adatszerkezet üres-e vagy sem a isEmpty() függvénnyel. Ha az adatszerkezet üres, a függvény kiír egy üzenetet és visszatér NULL értékkel.

Ha az adatszerkezet nem üres, akkor az első elem pozícióját menti el a pos változóban, majd ellenőrzi, hogy az adatszerkezetben van-e még elem a front és a rear mutatók segítségével. Ha az adatszerkezetben csak egy elem van, akkor mind a front, mind a rear mutatót visszaállítja -1-re, mert az adatszerkezet most üres lesz. Ha az adatszerkezetben több mint egy elem van, akkor csak a front mutatót növeli eggyel, hogy a következő elem lesz az első.

Végül a függvény visszaadja az eltávolított elemet, amely az adatszerkezetben a pos változóban lévő pozíción található.

Összességében a dequeue() függvény megfelelően eltávolítja az első elemet az adatszerkezetből, ha az adatszerkezet nem üres. Ha az adatszerkezet üres, a függvény visszatér NULL értékkel és egy üzenettel.

5.5.7 Sor elemeinek kiíratása - Display

```
void display(Queue queue) {
    if (isEmpty(queue)) {
        printf("The queue is ");
        printf(EMPTY_MESSAGE);
        return;
    }
    for (int i = queue.front; i <= queue.rear; ++i) {
        printf("%s\n", queue.elements[i]);
    }
}
```

Ha az adatszerkezet nem üres, akkor a függvény kiírja az összes elemet a sorba. A for ciklussal a front és a rear mutatók közötti összes elemet bejárja, majd minden elemet kiír a printf() függvénnyel.

Cirkuláris sor függvényei

Az egyszerű sortól eltérő függvények a következőek:

5.5.8 Telített sor tesztelése - isFull

```
bool isFull(Queue queue) {
    return queue.front == 0 && queue.rear == queue.capacity-1 || queue.rear ==
        queue.front-1;
}
```

Ehhez két feltételt vizsgál:

Ha a sor elejétől (front) kezdve a sor végéig (rear) minden hely foglalt (a sorban nincs szabad hely), akkor az adatszerkezet tele van. Ha a sor végén (rear) az utolsó elem után van még szabad hely, azaz az adatszerkezet végét elérte, de az elején (front) is van legalább egy szabad hely, akkor az adatszerkezet nincs tele. A második feltételnek egy rövidebb megfogalmazása a következő: front<rear.

5.5.9 Elem beszúrása - Enqueue

```
void enqueue(Queue *queue, int item) {
    if(isFull(*queue)) {
        printf(FULL_MESSAGE);
        return;
    }
    if(isEmpty(*queue)) {
        queue->front = 0;
    }
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->elements[queue->rear] = item;
}
```

A függvény ellenőrzi, hogy a sor tele van-e. Ha igen, kiír egy kegfelelő üzenetet. Ha a sor üres, akkor az első elemet a 0. indexre állítja. Az új rear mutató pozícióját körkörös módon számolja ki az előző rear és a capacity osztási maradéka alapján. Az új rear mutató által mutatott helyre beteszi az új elemet (item).

5.5.10 Elem eltávolítása - Dequeue

```
int dequeue(Queue *queue) {
    if(isEmpty(*queue)) {
        printf(EMPTY_MESSAGE);
        return INT_MIN;
    }
    int save = queue->elements[queue->front];
    if(queue->front == queue->rear)
    {
        queue->front = queue->rear = -1;
    }
    else {
        queue->front = (queue->front + 1) % queue->capacity;
    }
    return save;
}
```

A függvény először ellenőrzi, hogy a sor üres-e, ha igen, akkor kiír egy megfelelő üzenetet, és visszatér az INT MIN értékével. Ha nem üres a sor, akkor eltárolja az első elem értékét a save változóban. Ha a sor csak egy elemet tartalmaz, akkor a front és rear mutatókat visszaállítjuk az üres sor állapotára, különben a front mutatót az egyel növelt értékére állítjuk be, figyelembe véve, hogy a Queue egy cirkuláris tömbben van tárolva. Végül a függvény visszaadja a save változóban eltárolt értéket, ami a sorból eltávolított elem értéke.

5.5.11 Sor elemeinek kiíratása - Display

```
void display(Queue queue) {
    if(!(isEmpty(queue)))
    {int i=queue.front;
        do{
```

```
            printf("%s ",queue.items[i]);
            i=(i+1)%queue.capacity;
        }
        while (i!=queue.rear);
        printf("%s ",queue.items[i]);
        printf("\n");
    }
}
```

A függvény a sor elemeit egymás után jeleníti meg, figyelembe véve a kör ciklikus sajátosságait, vagyis a "visszakanyarodását" a végéről az elejére, ha szükséges.

5.6 Javasolt kérdések

1. Adott egy sor, amelybe az 1 és 2 elemeket tettük, ebben a sorrendben: 1 2. Jelöljük enqueue(x)-el azt a műveletet, amely beteszi az x értékét a sorba, és a dequeue az a művelet, amely töröl egy elemet a sorból. Hány eleme lesz a sornak a következő műveletek elvégzése után: engueue(4);dequeue;dequeue;enqueue(5);dequeue;enqueue(3)?
2. Tekintsünk egy vermet, amelybe a 1, 2 és 3 értékeket tettük be ebben a sorrendben. Jelölje push(x) azt a műveletet, amely során a x értéket betesszük a verembe (a tetejére), valamint pop azt a műveletet, amely a verem tetejéről eltávolít egy elemet. A verem esetén a következő műveleteket hajtjuk végre: push(4);pop;push(5);pop;push(6);pop;pop Mennyi lesz a verem tetején levő elem értéke a fenti műveletsor végrehajtása után? Mennyi lesz a fenti műveletsor végrehajtása után a veremben maradt elemek értékeinek összege?
3. Adott egy verem, amelybe a következő elemeket tettük, ebben a sorrendben: 1, 2 és 3. Jelöljük push(x)-el azt a műveletet, amely beszúrja az x elemet a verem tetejére és az pop az a művelet, amely törli a verem tetején levő elemet. Rajzold le, hogy mi lesz a veremben a következő műveletek végrehajtása után: push(4);pop;pop;push(5);pop?
4. Az S verem és a Q sor egész számokat tárol. Mindkettőbe betesszük az 1, 2, 3, 4 számokat, ebben a sorrendben. Jelöljük S->Q vel azt a műveletet, amely kivesz egy elemet az S veremből és beteszi a Q sorba, és Q->S -sel azt a műveletet, amely kivesz egy elemet a Q sorból és beteszi az S verembe. Mennyi lesz az S verembe utolsónak betett elem értéke, és mennyi lesz a Q sor végére betett utolsó elem értéke, ha a következő műveletsort hajtjuk végre: Q->S; Q->S; S->Q; Q->S; S->Q; S->Q; Q->S;
5. Egy egész számokat tároló verembe, betesszük a megadott sorrendben a következő számokat: 1, 2,3, 4, 5, 6, 7. Hány elemet kell töröljünk a veremből ahhoz, hogy a verem tetején az 5-ös legyen?
6. Egy sor típusú adatszerkezetbe a 3, 10, 2, 8 és 6 értékeket helyezték, ebben a sorrendben. Melyik volt az utoljára kiemelt elem, ha a következő műveleteket végezték: egy elem kiemelése a 100 értékű elem hozzáadása három elem kiemelése. A műveleteket a megadott sorrendben hajtották végre.
7. Tekintsünk egy sort, amelybe kezdetben a 2 és 1 értékeket helyeztük, ebben a sorrendben. Ha az enqueue x művelet a sorhoz ad egy x értéket, az dequeue művelet pedig kivesz egy értéket, mi lesz a sor tartalma a következő műveletsor elvégzése után: enqueue 10; enqueue 15; dequeue; enqueue 4; dequeue; enqueue 20; dequeue?
8. Egy kezdetben üres veremben elhelyezzük rendre az 1, 2, 3 értékeket, egy kezdetben üres sorban pedig rendre a 6, 5, 4 értékeket. Mennyi lesz a verem tetején levő elem értéke, ha a

sorból minden elemet kivesszük és rendre, a kivétel sorrendjében, az adott verembe tesszük?

9. Egy kezdetben üres S1 veremben elhelyezzük a 10, 12, 3 számokat, ebben a sorrendben, egy másik, kezdetben üres S2 veremben pedig elhelyezzük a 6, 5, 4, 3 számokat, ebben a sorrendben. Melyik szám lesz az S1 verem tetején levő elem értéke, és melyik szám lesz az S2 verem tetején levő elem értéke, ha az S2 veremből kivesszük az elemek felét, és a kivétel sorrendjében az S1 verembe rakjuk őket?
10. Egy sorban elhelyeztük az első három páratlan természetes számot: 1, 3 és 5. Jelölje enqueue x azt a műveletet, amely elhelyezi az x értéket a sorban, dequeue azt a műveletet, amely eltávolít egy elemet a sorból. A soron elvégezzük, pontosan ebben a sorrendben, a dequeue; enqueue 4; enqueue 6 műveleteket. Ábrázolja a sor tartalmát minden művelet elvégzése után.