Байланысқан тізімдер

Кіріспе

Алгоритмдер мен деректер құрылымдары — ақпаратты өңдеудің негізі. Алгоритмдер — проблемаларды шешу жолдары, ал деректер құрылымдары — ақпаратты ұйымдастыру тәсілдері. Дұрыс таңдалған деректер құрылымы бағдарламаның тиімділігін арттырады.

Байланысқан тізімдер — элементтерді сілтемелер арқылы байланыстыратын деректер құрылымы. Әр түйін өз деректерін және келесі түйінге сілтемені сақтайды. Бұл құрылым жаңа элементтерді қосу мен жоюды оңайлатады және динамикалық мөлшерді қолдайды. Байланысқан тізімдер көптеген алгоритмдер мен бағдарламаларда, мысалы, стектер мен кезектерде қолданылады.

Тізімдер

Бір жақты байланысқан тізім:

Бұл тізімдер тек бір бағытта байланысты. Әр түйін келесі түйіннің сілтемесін ғана ұстайды. Мысалы, 1 → 2 → 3 → NULL. Мұнда NULL — тізімнің соңын білдіреді.

Екі жақты байланысқан тізім:

Әр түйінде алдыңғы және кейінгі түйіндерге сілтеме болады. Бұл құрылым элементтер арасында екі жақты байланыс орнатады, яғни элементтерді екі бағытта да Traversal (аралау) жасауға мүмкіндік береді. Мысалы, NULL ← 1 ↔ 2 ↔ 3 → NULL.

Циклдік байланысқан тізім:

Соңғы түйіні бірінші түйінге сілтеме жасайды, яғни тізімде цикл пайда болады. Мұндай тізімдерде элементтерді Traversal жасау шексіз циклге ұрынбау үшін арнайы шарттар қолданылуы керек. Мысалы, 1 → 2 → 3 → 1.

Артықшылықтары мен кемшіліктері

Артықшылықтары:

Динамикалық мөлшерлеме: Жаңа элементтерді қосу мен жою оңай, бұл байланысқан тізімдерді динамикалық деректермен жұмыс істеуге ыңғайлы етеді.

Элементтерді оңай орналастыру: Тізімнің кез келген жеріне элементтерді қосу немесе жою кезінде басқа элементтерді жылжыту қажеттілігі жоқ.

Кемшіліктері:

Іздеу жылдамдығы: Тұрақты тізімдермен (массивтермен) салыстырғанда, элементтерді іздеу жылдамдығы төмен. Себебі, элементтерді табу үшін тізімді басынан бастап қарау қажет, бұл уақытты талап етеді.

Операциялар

Элементтерді қосу:

Байланысқан тізімге жаңа элемент қосу бірнеше тәсілмен жүзеге асады:

- Алдыңғы позицияға: Жаңа түйінді тізімнің басына қосу. Мысалы, жаңа элемент бірінші түйін ретінде енгізіледі.

- Кейінгі позицияға: Жаңа түйінді тізімнің соңына немесе кез келген басқа позицияға қосу. Бұл жағдайда, жаңа түйіннің сілтемесі қажетті түйіннің сілтемесіне тағайындалады.

Элементтерді жою:

Байланысқан тізімнен түйінді жою операциясы орындалады. Бұл операция жойылатын түйіннің сілтемесін түзету арқылы жүзеге асырылады, яғни жойылған түйіннің алдындағы және кейінгі түйіндердің сілтемелерін жаңарту қажет.

Іздеу:

Байланысқан тізімде белгілі бір мәнді табу үшін элементтерді тізімді басынан бастап қарау қажет. Егер элемент табылса, оның мәні қайтарылады; әйтпесе, элементтің тізімде жоқтығы көрсетіледі.

main.cpp

Share | Run

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;      // Түйін дерегі
    Node* next;    // Келесі түйінге сілтеме

    Node(int val) : data(val), next(nullptr) {}  // Конструктор
};

class LinkedList {
public:
    Node* head;    // Тізімнің басы

    LinkedList() : head(nullptr) {}  // Конструктор

    void append(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;  // Тізім бос болса, жаңа түйінді бос ретінде тағайындаймыз
            return;
        }
        Node* lastNode = head;
        while (lastNode->next) {  // Соңғы түйінді табу
            lastNode = lastNode->next;
        }
        lastNode->next = newNode;  // Соңына жаңа түйінді қосамыз
    }

    void display() {
        Node* current = head;
        while (current) {  // Тізімді басынан соңына дейін шегеру
            cout << current->data << " -> ";
            current = current->next;
        }
```

Output

Clear

```
/tmp/HPqRLXPu05.o
1 -> 2 -> 3 -> NULL


=== Code Execution Successful ===
```

main.cpp

Share | Run | Output | Clear

```
public:
    Node* head;    // Тізімнің басы

    LinkedList() : head(nullptr) {}  // Конструктор

    void append(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;  // Тізім бос болса, жаңа түйінді бас ретінде тағайындаймыз
            return;
        }
        Node* lastNode = head;
        while (lastNode->next) {  // Соңғы түйінді табу
            lastNode = lastNode->next;
        }
        lastNode->next = newNode;  // Соңына жаңа түйінді қосамыз
    }

    void display() {
        Node* current = head;
        while (current) {  // Тізімді басынан соңына дейін жүреміз
            cout << current->data << " -> ";
            current = current->next;
        }
        cout << "NULL" << endl;  // Тізімнің соңы
    }
};

int main() {
    LinkedList ll;
    ll.append(1);
    ll.append(2);
    ll.append(3);
    ll.display();  // Шығарады: 1 -> 2 -> 3 -> NULL
    return 0;
}
```

```
/tmp/HPqRLXPu05.o
1 -> 2 -> 3 -> NULL


=== Code Execution Successful ===
```

Қорытынды

Байланысқан тізімдер — алгоритмдер мен деректер құрылымдары саласындағы маңызды құралдардың бірі. Олардың динамикалық мөлшерлемесі мен элементтерді басқару жеңілдігі бағдарламалау кезінде көптеген артықшылықтар ұсынады.

Байланысқан тізімдер деректерді тиімді ұйымдастыруға мүмкіндік береді, бұл әсіресе деректер көлемі өзгеретін жағдайларда пайдалы. Олар **операциялық жүйелерде**, **деректер базаларында** және түрлі алгоритмдер мен құрылымдарды жүзеге асыруда кеңінен пайдаланылады.

Осылайша, байланысқан тізімдер деректерді ұйымдастырудың икемді әрі тиімді тәсілі ретінде көптеген бағдарламалар мен жүйелерде маңызды рөл атқарады, олардың алгоритмдердің тиімділігін арттыруға әсері зор.

Рахмет!