Strutture dati

Le strutture di dati sono collezioni di dati. La loro caratteristica principale è l'organizzazione della collezione piuttosto che il tipo dei dati contenuti. Uno stesso tipo di struttura dati può essere infatti dichiarata con diversi tipi di dati primitivi contenuti in essa. Una struttura dati è caratterizzata da:

- un insieme di operatori che permettono di manipolare la struttura
- un modo sistematico di organizzare l'insieme dei dati

In generale, una definizione del termine struttura dati può essere:

> Una sezione dello spazio di memoria che viene utilizzata per organizzare e memorizzare i dati.

Sommario

Array

Definizione

Un array è una lista di N oggetti dello stesso tipo allocati in posizioni contigue in memoria. La lunghezza dell'array viene definita al momento della dichiarazione e non può essere più modificata. Pertanto, un array di N elementi ha sempre, necessariamente, indici da 0 a N-1 (inclusi).

Operazioni

- Accesso $\rightarrow O(1)$
- Ricerca $\rightarrow O(n)$
- Size $\rightarrow O(1)$

Esempio

```
#include <iostream>
using namespace std;
int main()
{
    const int elems = 5;
    int array[elems];
    for (int i = 0; i < elems; i++)
    {
        cout << "\nInserisci un intero positivo:";
        cin >> array[i];
    }
    cout << "Ecco l'array immesso:" << endl;
    for (i = 0; i < elems; i++)
    {
        cout << "\nElemento" << i << ": " << array[i];
    }
    cout << endl;
    return 1;
}
```

Array dinamico

Definizione

Un array dinamico è simile a un array standard, ma offre funzioni di inserimento e rimozione più efficienti. Funziona allocando un array di dimensione fissa e dividendo questa dimensione in due parti: una contiene gli elementi dell'array, mentre l'altra è inutilizzata. Ciò consente di aggiungere o rimuovere elementi alla fine dell'array in tempo costante, utilizzando lo spazio riservato finché non viene esaurito. Quando lo spazio è esaurito, l'array dinamico raddoppia le sue dimensioni anziché aumentarle di una singola unità. La dimensione effettiva dell'array dinamico è il numero di elementi utilizzati, mentre la capacità dell'array sottostante è lo spazio che occupa in memoria.

Operazioni

- Accesso $\rightarrow O(1)$
- PushBack $\rightarrow O(\bar{1})$
- PopBack $\rightarrow O(1)$
- Ricerca $\rightarrow O(n)$
- Size $\rightarrow O(1)$

Esempio

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> arrayDinamico;
    arrayDinamico.push_back(10);
    arrayDinamico.push_back(20);
    arrayDinamico.push_back(30);
    arrayDinamico[1] = 50;
    arrayDinamico.pop_back();
    for (int num : arrayDinamico)
    {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

Linked List

Definizione

Una linked list è una lista di oggetti composta da nodi. Ogni nodo contiene l'oggetto e un riferimento al successivo nodo nella sequenza. Questa struttura utilizza una disposizione non contigua dei dati in memoria. Per questo motivo, le linked list sono vantaggiose per la gestione dinamica della memoria e l'efficienza nell'inserimento/eliminazione di dati, mentre gli array statici sono più efficienti per l'accesso casuale ai dati e quando le dimensioni e la posizione dei dati sono fisse.

Operazioni

- Accesso $\rightarrow O(n)$
- Inserimento (in qualsiasi posizione) $\rightarrow O(1)$
- Rimozione (in qualsiasi posizione) $\rightarrow O(1)$
- Ricerca $\rightarrow O(n)$
- Size $\rightarrow O(n)$

Esempio

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

struct Node
{
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

int main()
{
    Node* head = new Node(5);
    Node* second = new Node(10);
    Node* third = new Node(15);
    head->next = second;
    second->next = third;
    Node* temp = head;
    cout << "Lista: ";
    while (temp != nullptr)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
    delete second;
    head->next = third;
    temp = head;
    cout << "Lista: ";
```

```
    while (temp != nullptr)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
    delete head;
    delete third;
    return 0;
}
```

Stack

Definizione

Lo stack è una lista di oggetti gestita in modalità Last In First Out (LIFO), cioè l'ultimo oggetto inserito è il primo ad essere rimosso. Le due operazioni principali sono quindi push (aggiunge un elemento in cima allo stack) e pop (rimuove un elemento dalla cima dello stack).

Operazioni

- Top $\rightarrow O(1)$
- PushBack $\rightarrow O(1)$
- PopBack $\rightarrow O(1)$
- Size $\rightarrow O(1)$

Esempio

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main()
{
    stack<int> myStack;
    myStack.push(10);
    myStack.push(20);
    myStack.push(30);
    if (myStack.empty())
    {
        cout << "Lo stack è vuoto!" << endl;
    }
    else
    {
        cout << "Lo stack non è vuoto." << endl;
    }
    cout << "Valore in cima allo stack: " << myStack.top() << endl;
    myStack.pop();
    cout << "Valore in cima allo stack: " << myStack.top() << endl;
    cout << "Dimensione dello stack: " << myStack.size() << endl;
    return 0;
}
```

Queue

Definizione

Una queue è una raccolta di oggetti tenuti in una sequenza che può essere modificata aggiungendo oggetti a un estremo e rimuovendoli dall'altro estremo della sequenza. Per questo motivo è una struttura di tipo FIFO (first in first out).

Operazioni

- Front $\rightarrow O(1)$
- PushBack $\rightarrow O(1)$
- PopFront $\rightarrow O(1)$
- Size $\rightarrow O(1)$

Esempio

```
#include <iostream>
#include <queue>
using namespace std;
int main()
{
    queue<int> myQueue;
    myQueue.push(5);
    myQueue.push(10);
    myQueue.push(15);
    cout << "Elementi nella coda:" << endl;
    while (!myQueue.empty())
    {
        cout << myQueue.front() << " ";
        myQueue.pop();
    }
    cout << endl;
    return 0;
}
```

Set

Definizione

Il set è una collezione di valori disposti in ordine casuale e senza valori ripetuti. I set vengono utilizzati per eseguire operazioni come l'aggiunta e la rimozione di elementi e la verifica della presenza di un elemento all'interno della collezione.

Operazioni

- Inserimento $\rightarrow O(\log n)$
- Eliminazione $\rightarrow O(\log n)$
- Ricerca $\rightarrow O(\log n)$
- Size $\rightarrow O(1)$

Esempio

```cpp
#include <iostream>
#include <set>
using namespace std;
int main()
{
    set<int> MySet;
    MySet.insert(5);
    MySet.insert(2);
    MySet.insert(8);
    cout << "Numeri nell'insieme: ";
    for (const auto& numero : MySet)
    {
        cout << numero << " ";
    }
    cout << endl;
    int numeroDaCercare = 5;
    if (MySet.find(numeroDaCercare) != MySet.end()) {
        cout << numeroDaCercare << " è presente nell'insieme." << endl;
    }
    else
    {
        cout << numeroDaCercare << " non è presente nell'insieme." << endl;
    }
    MySet.erase(8);
    cout << "Numeri nell'insieme dopo la rimozione di 8: ";
    for (const auto& numero : MySet)
    {
        cout << numero << " ";
    }
    cout << endl;
    return 0;
}
```

Mappa

Definizione

La mappa è una struttura dati che associa coppie chiave-valore. Ogni elemento all'interno di una mappa consiste in una coppia ordinata, in cui una chiave univoca è associata a un valore corrispondente. La chiave funge da identificatore unico e viene utilizzata per recuperare il valore associato ad essa. Le mappe consentono l'accesso efficiente e rapido ai valori tramite la chiave corrispondente.

Operazioni

- Accesso $\rightarrow O(\log n)$
- Inserimento $\rightarrow O(\log n)$
- Eliminazione $\rightarrow O(\log n)$
- Ricerca $\rightarrow O(\log n)$
- Size $\rightarrow O(1)$

Esempio

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
    map<string, int> MyMap;
    MyMap["Uno"] = 1;
    MyMap["Due"] = 2;
    MyMap["Tre"] = 3;
    cout << "Il valore associato a 'Uno' è: " << MyMap["Uno"] << endl;
    cout << "Il valore associato a 'Due' è: " << MyMap["Due"] << endl;
    cout << "Il valore associato a 'Tre' è: " << MyMap["Tre"] << endl;
    string chiave = "Quattro";
    if (MyMap.find(chiave) != MyMap.end())
    {
        cout << "Il valore associato a '" << chiave << "' è: " << MyMap[chiave] <<
endl;
    }
    else
    {
        cout << "'" << chiave << "' non è presente nella mappa." << endl;
    }
    return 0;
}
```