class Comparator {

public:

    bool operator()(Node\* left, Node\* right) {

        return left->getFreq() > right->getFreq();

    }

};

class HuffmanTree {

public:

    Node\* XayDungTreeHuffman(map<char, int> freq) {

        priority\_queue<Node\*, vector<Node\*>, Comparator> pq;

……

}

Tính Encapsulation (Đóng gói): Bằng cách đặt hàm so sánh trong một struct, bạn có thể gói nó cùng với dữ liệu và thao tác liên quan đến dữ liệu đó. Điều này giúp giữ cho mã nguồn được tổ chức và dễ dàng hiểu hơn.

\*\*Dễ Dàng Truy Cập: \*\* Trong trường hợp này, bạn cần truy cập vào các thành phần của cặp pair<int, int> để so sánh chúng. Bằng cách định nghĩa hàm so sánh trong struct, bạn có thể truy cập các thành phần của cặp trực tiếp thông qua các tham số của hàm toán tử, mà không cần truyền thêm thông tin hoặc tạo ra một phần tử mới.

Thiết Kế Linh Hoạt: Việc đặt hàm so sánh trong một struct cho phép bạn dễ dàng thay đổi cách thức so sánh hoặc thậm chí sử dụng nhiều hàm so sánh khác nhau cho cùng một kiểu dữ liệu mà không cần phải thay đổi các điểm sử dụng.

Trong trường hợp này, struct CompareSecond chứa hàm so sánh để so sánh phần tử thứ hai của mỗi cặp pair<int, int>. Điều này giúp rõ ràng hơn trong việc sử dụng nó trong các cấu trúc dữ liệu như priority\_queue.

* Lời dẫn thì chúng ta khi sử dụng hàng đợi ưu tiên với pair thì nó sẽ đi kèm với việc sắp xếp theo độ ưu tiên của first trong pair nhưng khi chúng ta muốn nó ưu tiên theo second thì chúng ta phải viết biểu thức operator hoặc biểu thức lambda và từ đó đưa vào thay thế tham số “greater<pair<int,int>> “ thì nó sẽ được theo ý mình.
* Thứ nhất , ta có thể hiểu rằng cái greater<pair<int,int>> thực chất là một function (HÀM) . Điều này có thể được coi như một lớp có một hàm operator() đã được định nghĩa.

* Định nghĩa hàm so sánh tùy chỉnh để sắp xếp theo giá trị của second.

|  |
| --- |
| struct CompareSecond{  bool operator()(const pair<int,int> &left, const pair<int,int> &right) {  return left.second <right.second;  }  };  priority\_queue<pair<int, int>, vector<pair<int, int>>, CompareSecond> p;  auto cmp = [] (const pair<int,int> &left , const pair<int,int> &right) {  return left.second < right.second;  };  priority\_queue<pair<int, int>, vector<pair<int, int>>, decltype(cmp)> p(cmp); |

* Đoạn mã trên cung cấp định nghĩa một struct (Kiểu cấu trúc) có tên là ‘CompareSecond’, trong đó có một hàm toán tử gọi là

operator(). Hàm này so sánh hai phần tử kiểu pair<int,int> dựa trên giá trị thành phần của second.

--------------------------------------------------------------

* Mã mà bạn cung cấp định nghĩa một cấu trúc CompareSecond có một hàm toán tử operator () để so sánh hai phần tử kiểu pair dựa trên giá trị của phần tử thứ hai của mỗi cặp.

--------------------------------------------------------------

* Trong đoạn mã của bạn, bạn đang tạo một comparator để sử dụng trong một hàng đợi ưu tiên. Hãy xem xét phần khai báo của hàng đợi ưu tiên:

priority\_queue<pair<int, int>, vector<pair<int, int>>, CompareSecond> p;

CompareSecond là một Comparator bạn định nghĩa ttrong struct. Comparator bạn được hiểu là một struct hoặc class được sử dụng để so sánh các phần tử trong hàng đợi ưu tiên. Bạn định nghĩa struct CompareSecond để so sánh các phần tử dựa trên giá trị second của pair<int,int>.

* Vì vậy, tham số thứ ba trong khai báo của bạn là một comparator . Bạn viết nó trong struct để tùy chỉnh cách so sánh các phần tử trong hàng đợi ưu tiên . Khi bạn thêm mới một phần tử trong hàng đợi ưu tiên, comparetor sẽ quyết định xem phần tử đó có nên đứng trước hay sau hàng đợi dựa trên quy tắc bạn đã định nghĩa.

-------------------------------------------------------------

* Comparator function trong C++ là một hàm hoặc một toán tử được sử dụng để so sánh hai phần tử hoặc các đối tượng của một kiểu dữ liệu nhất định.
* Comparator function thường được sử dụng khi cần sắp xếp các phần tử trong một cấu trúc dữ liệu như vector, set, map, priority\_queue, và nhiều cấu trúc dữ liệu khác.

Cách thức hoạt động của Comparator Function:

Comparator function cung cấp một cách để xác định thứ tự của hai phần tử bằng cách so sánh chúng và trả về một giá trị dựa trên kết quả của phép so sánh. Cụ thể, comparator function trả về true nếu phần tử đầu tiên được xem là nhỏ hơn phần tử thứ hai, và false nếu ngược lại.

Cách áp dụng của Comparator Function:

Sắp xếp mảng hoặc vector: Comparator function thường được sử dụng khi sắp xếp các phần tử trong một mảng hoặc vector theo một tiêu chí nhất định.

Sắp xếp các cấu trúc dữ liệu khác: Ngoài việc sắp xếp mảng hoặc vector, comparator function cũng được sử dụng trong các cấu trúc dữ liệu khác như set, map, priority\_queue, để quyết định thứ tự các phần tử được lưu trữ hoặc truy xuất.

Customized sorting: Comparator function cho phép bạn tạo ra các loại sắp xếp tùy chỉnh dựa trên nhu cầu cụ thể của vấn đề.

Ví dụ về Comparator Function:

Dưới đây là một ví dụ về việc sử dụng comparator function để sắp xếp một vector các số nguyên theo thứ tự tăng dần:

#include <iostream>

#include <vector>

#include <algorithm>

bool compare(int a, int b) {

return a < b;

}

int main() {

vector<int> numbers = {3, 1, 4, 1, 5, 9, 2, 6};

sort(numbers.begin(), numbers.end(), compare);

cout<< "Sorted numbers: ";

for (int num : numbers) {

cout << num << " ";

}

cout << endl;

return 0;

}

Cấu trúc của một hàm comparator trong C++ thường là một hàm hoặc một toán tử được sử dụng để so sánh hai phần tử của một kiểu dữ liệu nhất định.

Cấu trúc cụ thể của một hàm comparator phụ thuộc vào cách bạn muốn so sánh các phần tử và kiểu dữ liệu của chúng. Dưới đây là các yếu tố cơ bản thường xuất hiện trong cấu trúc của một hàm comparator:

Tham số đầu vào: Hàm comparator thường nhận vào các tham số để so sánh. Số lượng và kiểu dữ liệu của các tham số phụ thuộc vào loại so sánh bạn định nghĩa.

Kiểu trả về: Hàm comparator thường trả về một giá trị kiểu bool, thể hiện kết quả của phép so sánh. Giá trị true thường chỉ ra rằng phần tử thứ nhất nhỏ hơn phần tử thứ hai, trong khi giá trị false chỉ ra ngược lại.

Cơ chế so sánh: Hàm comparator thường thực hiện so sánh trên một hoặc nhiều thuộc tính của các phần tử để xác định thứ tự của chúng. Cơ chế so sánh này có thể là bất kỳ loại so sánh nào phù hợp với nhu cầu cụ thể của vấn đề, như so sánh theo giá trị, theo chuỗi ký tự, theo thứ tự từ điển, v.v.

Dưới đây là một ví dụ về cấu trúc của một hàm comparator đơn giản trong C++:

bool compare(int a, int b) { return a < b; }

Trong ví dụ này, hàm compare nhận hai số nguyên a và b làm đối số và trả về true nếu a nhỏ hơn b, ngược lại trả về false. Đây là một cấu trúc đơn giản nhưng rất phổ biến cho một hàm comparator

------------------------------------------------------------

* Khi bạn khai báo priority\_queue với CompartSecond làm tham số cuối cùng, bạn đang chỉ định rằng priority\_queue nên sử dụng hàm so sánh đã được định nghĩa trong CompareSecond để sắp xếp các phần tử.
* Câu lệnh mà bạn đưa ra là một ví dụ về việc sử dụng hàng đợi ưu tiên trong trong C++.

1. struct CompareSecond

+ Là một cấu trúc struct được định nghĩa bởi bạn.

+ Cấu trúc này có một hàm thành viên là operator() , được sử dụng để so sánh hai phần tử kiểu pair<int,int>.

+ Hàm operator() này nhận vào hai tham số kiểu pair<int,int> là left và right

+ Nó lấy giá trị so sánh second của hai cái này để so sánh.

+ Nếu cái bên left nhỏ hơn cái bên right thì hàm trả về true và ngược lại hàm trả về false.

1. priority\_queue định nghĩa một hàng đợi ưu tiên với tham số p. Loại dữ liệu lưu trữ trong hàng đợi là pair<int,int>. Bộ sắp xếp sử dụng là ‘CompareSecond’, tức là sử dụng cách so sánh đã được định nghĩa trong CompareSecond, tức là sử dụng cách so sánh đã được định nghĩa trong CompareSecond.

-----------------------------------------------------------

* Các lớp so sánh được sử dụng để so sánh các đối tượng của các lớp do người dùng định nghĩa. Để phát triển một [hàm chung](https://www.geeksforgeeks.org/generics-in-c/) , hãy sử dụng [mẫu](https://www.geeksforgeeks.org/templates-cpp/) và để làm cho hàm trở nên chung chung hơn, hãy sử dụng [các thùng chứa](https://www.geeksforgeeks.org/containers-cpp-stl/) để có thể thực hiện so sánh giữa các dữ liệu.

|  |
| --- |
| class comparator\_class {  public:      // Comparator function      bool operator()(object o1, object o2)      {            // There can be any condition          // implemented as per the need          // of the problem statement          return (o1.data\_member == o2.data\_member);      }  } |

Giải thích: Lớp toán tử () của hàm so sánh ở trên nhận hai cặp đối tượng cùng một lúc và trả về true nếu các thành viên dữ liệu của hai toán tử giống nhau. Có thể có bất kỳ điều kiện nào tùy theo nhu cầu của bài toán trong hàm so sánh. Trong ví dụ trên, hàm trả về true nếu các thành viên dữ liệu giống nhau.

Ví dụ 1:

Để triển khai [Tìm kiếm tuyến tính](https://www.geeksforgeeks.org/linear-search/) trên các phần [tử mảng](https://www.geeksforgeeks.org/introduction-to-arrays/) , [việc tìm kiếm một số nguyên trong một mảng nhất định](https://www.geeksforgeeks.org/recursive-c-program-linearly-search-element-given-array/) có thể được thực hiện dễ dàng. Nhưng việc tìm kiếm bất kỳ phần tử nào trên [kiểu dữ liệu do người dùng xác định](https://www.geeksforgeeks.org/user-defined-data-types-in-c/) không thể được thực hiện dễ dàng như trong trường hợp mảng. Trong trường hợp này, lớp so sánh được sử dụng để triển khai nó. Dưới đây là chương trình tương tự:

C++

|  |
| --- |
| // C++ program for the Comparator Class  // for implementing linear search  #include <bits/stdc++.h>  using namespace std;    // Forward Declaration of classes  class student;  class studentcompare;    // Generic function to search for object  template <class ForwardIterator, class T>  ForwardIterator search(ForwardIterator start,                         ForwardIterator end, T key,                         studentcompare& cmp)  {      // Iterate until start equals to end      while (start != end) {            // If the value with given key is          // found the return that index          if (cmp(\*start, key)) {              return start;          }          start++;      }      return end;  }    // Student Class  class student {  public:      // To store Name and Roll Number      string name;      int rollnum;        // Overloaded Constructor      student(string name, int rollnum)      {          this->name = name;          this->rollnum = rollnum;      }  };    // Comparator Class to compare 2 objects  class studentcompare {  public:      // Comparator function      bool operator()(student a, student b) const      {          // If values are the same then          // return true          if (a.name == b.name) {              return true;          }          return false;      }  };    // Driver Code  int main()  {      // Object of class student      student s1("Raj", 23);      student s2("Prerna", 24);        // List of students      list<student> s;      s.push\_back(s1);      s.push\_back(s2);        // Search student("Prerna", 24)      student searchstudent("Prerna", 24);        studentcompare cmp;        // Print if element is found using custom search      // function      if (search(s.begin(), s.end(), searchstudent, cmp)          != s.end()) {          cout << "Student found!";      }      else {          cout << "Not found";      }        return 0;  } |

đầu ra

Sinh viên được tìm thấy!

Giải trình:

Trong chương trình trên, một danh sách các đối tượng sinh viên được tạo ra, trong đó ban đầu có 2 đối tượng sinh viên được chèn vào.

Bây giờ để tìm kiếm một học sinh cụ thể, một hàm tìm kiếm tuyến tính mẫu được viết bằng cách sử dụng lớp so sánh.

Lớp so sánh so sánh sinh viên cần tìm kiếm từ danh sách sinh viên dựa trên thuộc tính tên của họ.

Nếu thuộc tính tên của đối tượng cần tìm kiếm bằng bất kỳ thuộc tính tên nào của đối tượng trong danh sách thì nó sẽ trả về true, nếu không, nó sẽ trả về false.

Ví dụ 2:

Chúng ta hãy lấy một ví dụ khác về việc sử dụng lớp Comparator để [sắp xếp](https://www.geeksforgeeks.org/sorting-algorithms/) , giả sử nhiệm vụ là sắp xếp một [mảng các đối tượng](https://www.geeksforgeeks.org/how-to-initialize-array-of-objects-with-parameterized-constructors-in-c/) dựa trên giá trị thuộc tính của nó, sau đó ý tưởng là tạo một lớp so sánh tùy chỉnh trong đó chức năng sắp xếp phải được thực hiện có thể được đề cập. Sau đó, nó có thể được truyền dưới dạng đối số trong hàm [Sort()](https://www.geeksforgeeks.org/arrays-sort-in-java-with-examples/) .

C++

|  |
| --- |
| // C++ program for the Comparator Class  // implementing sorting    #include <bits/stdc++.h>  using namespace std;    // Student Class  class student {  public:      // To store Name and Roll Number      string name;      int rollnum;        // Overloaded Constructor      student(string name, int rollnum)      {          this->name = name;          this->rollnum = rollnum;      }  };    // Comparator Class to compare 2 objects  class studentcompare {  public:      // Comparator function      bool operator()(const student& a, const student& b)      {          // Compare on basis of roll number          if (a.rollnum < b.rollnum) {              return true;          }          return false;      }  };    // Driver Code  int main()  {      // Object of class student      student s1("Raj", 23);      student s2("Prerna", 24);      student s3("Harshit", 21);        // List of students      list<student> s;      s.push\_back(s1);      s.push\_back(s2);      s.push\_back(s3);        // Creating object of      // comparator class      studentcompare cmp;        // Passing the object of      // comparator class to sort()      s.sort(cmp);        // Printing the list after sorting      for (auto stu : s) {          cout << stu.name << " ";      }        return 0;  } |

đầu ra

Raj Prerna khắc nghiệt

Giải trình:

Trong chương trình trên, một danh sách các đối tượng học sinh được tạo ra, trong đó ban đầu có 3 đối tượng học sinh được chèn vào.

Bây giờ, nếu danh sách học sinh phải được sắp xếp trên cơ sở thuộc tính rollno của học sinh thì một lớp so sánh tùy chỉnh phải được chuyển làm đối số cho hàm [sắp xếp()](https://www.geeksforgeeks.org/sort-c-stl/) .

Trong lớp so sánh, một logic vị ngữ phải được đề cập để trả về đúng hoặc sai trên cơ sở danh sách được sắp xếp.

Bạn cảm thấy lạc lõng trong thế giới của những chủ đề DSA ngẫu nhiên, lãng phí thời gian mà không tiến bộ? Đã đến lúc phải thay đổi! Hãy tham gia khóa học DSA của chúng tôi, nơi chúng tôi sẽ hướng dẫn bạn một hành trình thú vị để thành thạo DSA một cách hiệu quả và đúng tiến độ.  
Sẵn sàng để đi sâu vào? Khám phá Nội dung demo miễn phí của chúng tôi và tham gia khóa học DSA của chúng tôi, được hơn 100.000 chuyên viên máy tính tin cậy!