III/ Nội dung chính

1/ Tổng quan

* Trong phần báo cáo này, nhóm chúng tôi tập trung triển khai và phân tích Class Priority Queue sử dụng cấu trúc Heap, một thành phần quan trọng trong thiết kế hàng đợi ưu tiên (Priority Queue).
* Lớp Priority Queue được cài đặt theo hướng đối tượng trong ngôn ngữ C++, bao gồm các phương thức cơ bản. Việc tổ chức và cài đặt lớp sẽ được trình bày chi tiết trong các mục sau
* Toàn bộ nội dung được trình bày theo quy ước rằng các phương thức của Class Priority Queue vận hành trên một std::vector.

2/ Tổ chức class

* Class Priority Queue được cài đặt dưới dạng class tổng quát (template), cho phép làm việc với nhiều kiểu dữ liệu khác nhau và có thể tùy chỉnh thứ tự ưu tiên thông qua đối tượng so sánh Compare. Theo mặc định, class sẽ hoạt động như một Min Heap, nơi phần tử nhỏ nhất luôn có độ ưu tiên cao nhất
* Tổ chức của Class Priority Queue gồm 3 thành phần chính:
  + 1. Trường lưu trữ
    - Heap: Một vector để lưu trữ các phần tử của hàng đợi ưu tiên theo dạng cây nhị phân hoàn chỉnh
    - Comp: đối tượng hàm so sánh để xác định thứ tự ưu tiên
  + 2. Phương thức xử lí nội bộ (Private)
    - heapifyUp(int index): dùng sau khi thêm phần tử mới; đảm bảo tính chất heap bằng cách đẩy phần tử lên nếu cần.
    - heapifyDown(int index): dùng sau khi xóa phần từ đầu; khôi phục cấu trúc heap bằng cách đẩy phần tử xuống đúng vị trí
  + 3. Phương thức công khai (public)
    - bool isEmpty() const: Kiểm tra hàng đợi có rỗng không.
    - int size() const: trả về số lượng phần tử trong hàng đợi.
    - void push(const T& value): thêm phần tử mới vào hàng đợi.
    - int pop(): loại bỏ và trả về phần tử có độ ưu tiên cao nhất.
    - const T& top() const: trả về phần tử có độ ưu tiên cao nhất
    - void remove(int value): xóa một phần tử cụ thể nếu tồn tại.
    - void buildHeap(const vector<T>& items): xây dựng heap từ một danh sách phần tử cho trước
    - void clear(): xóa toàn bộ phần tử trong hàng đợi
    - void print() const: in toàn bộ phần tử của heap dưới dạng mảng tuyến tính.
* Thiết kế đảm bảo tính đóng gói, tính tổng quát, tính mở rộng cho phép dễ dàng áp dụng Priority Queue trong nhiều bài toán khác nhau mà không cần thay đổi cấu trúc cốt lõi của lớp.
* A screenshot of a computer screen

  AI-generated content may be incorrect.

3/ Mô tả chi tiết các phương thức

3.1/ heapifyUp(int index)

a/ Mục đích

* Điều chỉnh phần tử tại vị trí index theo hướng lên trên nhằm khôi phục tính chất heap sau khi thêm mới phần tử

b/ Code:

c/ Mô tả chi tiết:

* Bắt đầu từ vị trí index (phần tử mới thêm), so sánh phần tử với nút cha. Nếu phần tử có độ ưu tiên cao hơn (ví dụ: nhỏ hơn trong Min Heap), hoán đổi vị trí giữa hai nút. Tiếp tục quá trình này lặp lại cho đến khi phần tử ở đúng vị trí theo cấu trúc heap hoặc đạt đến nút gốc (index = 0).

d/ Độ phức tạp:

e/ Nhận xét:

- Phương thức này là yếu tố then chốt để duy trì cấu trúc heap sau mỗi lần chèn phần tử mới, đảm bảo hiệu quả và độ ổn định của hàng đợi ưu tiên.

3.2/ heapifyDown(int index)

a/ Mục đích:

* Điều chỉnh phần tử tại vị trí index theo hướng xuống dưới nhằm duy trì tính chất heap sau khi loại bỏ phần tử ưu tiên hoặc thay đổi cấu trúc.

b/ Code

c/ Mô tả thuật toán:

* Bắt đầu từ vị trí index, so sánh phần tử với hai nút con (nếu có). Chọn nút con có độ ưu tiên cao hơn để hoán đổi với phần tử hiện tại nếu cần. Lặp lại quá trình này cho đến khi phần tử ở vị trí đúng hoặc không còn nút con nào thỏa điều kiện hoán đổi.

d/ Độ phức tạp:

e/ Nhận xét:

* Thuật toán này đảm bảo rằng sau khi loại bỏ phần tử ưu tiên hoặc thay đổi phần tử gốc, cấu trúc heap vẫn được duy trì chính xác.

3.3/ push(const T& value)

a/ Mục đích: Thêm một phần tử vào hàng đợi ưu tiên và đảm bảo tính chất heap được duy trì.

b/ Code:

c/ Mô tả thuật toán:

* Thêm phần từ và cuối heap.
* Gọi heapifyUp() để đưa phần tử về đúng vị trí trong heap.
* Quá trình sẽ kết thúc khi phần tử ở đúng vị trí hoặc đạt đến gốc

d/ Độ phức tạp

e/ Nhận xét:

Phương thức đảm bảo sự chính xác và hiệu quả khi thêm phần tử, sử dụng thao tác nâng đơn giản để duy trì heap.

3.4/ pop()

a/ Mục đích: Thêm một phần tử vào hàng đợi ưu tiên và đảm bảo tính chất heap được duy trì, rồi trả về giá trị của phần tử đó (nếu không có phần tử sẽ báo lỗi)

b/ Code:

c/ Mô tả chi tiết:

* Ghi nhận giá trị tại gốc (ưu tiên cao nhất)
* Thay thế gốc bằng phần tử cuối và loại bỏ phần tử cuối
* Gọi heapifyDown() để điều chỉnh heap từ gốc xuống

d/ Độ phức tạp

e/ Nhận xét

* Đây là thao tác quan trọng nhất của hàng đợi ưu tiên. Việc thay gốc bằng phần tử cuối giúp giữ cấu trúc cây nhị phân hoàn chỉnh.

3.5/ top()

a/ Mục đích: Trả về phần tử ưu tiên nhất mà không loại bỏ khỏi hàng đợi

b/ Code:

c/ Mô tả thuật toán: Trả về phần tử tại chỉ số 0 trong mảng heap.

d/ Độ phức tạp:

e/ Nhận xét: Thao tác đọc đỉnh heap rất nhanh, phù hợp với yêu cầu xử lí thời gian thực

3.6/ remove(int value)

a/ Mục đích: Xóa một phần tủ cụ thể khỏi hàng đợi (nếu tồn tại), sau đó khôi phục tính chất của heap

b/ Code:

c/ Mô tả chi tiết:

* Tìm vị trí phần tử cần xóa
* Hóa đổi vị trí với phần tử cuối và loại bỏ
* Gọi heapifyDown() để đảm bảo cấu trúc heap

d/ Độ phức tạp

e/ Nhận xét: Thao tác xóa phần tử cụ thể không tối ưu với cấu trúc heap nhưng vẫn đảm bảo tính chất của heap (phải tìm tuyến tính)

3.7/ isEmpty()

a/ Mục đích: Kiểm tra hàng đợi ưu tiên có rỗng không.

b/ Code:

c/ Mô tả thuật toán: Trả về kết quả xem vector heap có phần từ nào không

d/ Độ phức tạp

e/ Nhận xét: Đơn giản, cần thiết cho việc kiểm soát luồng chương trình khi tháo tác với hàng đợi (tránh lỗi)

3.8/ size()

a/ Mục đích: Trả về số lượng phần tử có trong hàng đợi

b/ Code:

c/ Mô tả thuật toán: Trả về số lượng phần tử hiện tại của vector heap

d/ Độ phức tạp:

e/ Nhận xét: Hữu ích cho các thao tác kiểm tra hoặc thống kê

3.9/ buildHeap(const vector<T>& items)

a/ Mục đích: Xây dựng heap ban đầu từ một tập hợp phần tử cho trước

b/ Code:

c/ Mô tả thuật toán

* Gán các phần tử vào vector heap;
* Từ các phần từ n / 2 – 1 🡪 0 sẽ gọi heapifyDown để tạo thành cấu trúc heap (vì các phần từ n / 2 🡪 n đã tạo thành heap tự nhiên)

d/ Đọ phức tạp

e/ Nhận xét: Hiệu quả hơn việc thêm từng phần tử bằng push(). Thuật toán trên tối ưu cho khởi tạo mảng ban đầu

3.10/ clear()

a/ Mục đích: Xóa toàn bộ phần tử trong hàng đợi ưu tiên

b/ Code :

c/ Mô tả thuật toán : Thực hiện toàn bộ phần tử trong vector heap bằng phương thức clear

d/Độ phức tạp

e/ Nhận xét : Hữu ích khi muốn tái sử dụng hàng đợi mà không cần tạo đối tượng mới

3.11/ print() const

a/ Mục đích : In ra các phần tử hiện có trong hàng đợi dưới dạng mảng tuyến tính theo thứ tự trong vector heap

b/ Code :

c/ Mô tả thuật toán : Duyệt qua toàn bộ phần vector heap và in từng phần tử ra màn hình

d/ Độ phức tạp

e/ Nhận xét : Phục vụ mục đích hiển thị, kiểm tra hoặc minh họa trạng thái hiện tại của hàng đợi ưu tiên