ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT - CO2003

XÂY DỰNG ỨNG DỤNG NGHE NHẠC BotKify
SỬ DỤNG DANH SÁCH

TP. HỒ CHÍ MINH, THÁNG 01/2026

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 1.0

1 Chuẩn đầu ra

Sau khi hoàn thành bài tập lớn này, sinh viên sẽ có khả năng:

- Thành thạo lập trình hướng đối tượng (OOP).
- Phát triển các cấu trúc dữ liệu danh sách.
- Sử dụng các cấu trúc dữ liệu dạng danh sách (LinkedList, Stack, Queue) để hiện thực Playlist cho ứng dụng BotKify.

2 Dẫn nhập

Trong Bài tập lớn 1, sinh viên được yêu cầu hiện thực một ứng dụng nghe nhạc BotKify với các tính năng thêm, xóa, sửa và truy vấn bài hát dựa trên các lượt nghe của từng cá nhân dựa trên nền tảng của danh sách. Thông qua đó, bài tập sẽ giúp sinh viên:

- Củng cố kỹ năng lập trình hướng đối tượng thông qua việc thiết kế các lớp đối tượng.
- Hiểu và hiện thực chi tiết các cấu trúc danh sách (LinkedList, Stack, Queue).
- Áp dụng cấu trúc danh sách(LinkedList, Stack, Queue) vào giải quyết các vấn đề nhằm giảm độ phức tạp khi xử lý.

3 Mô tả

Bài tập lớn này yêu cầu sinh viên hiện thực một ứng dụng nghe nhạc đơn giản mang tên *BotKify*. Trọng tâm của bài tập là việc thiết kế và cài đặt các cấu trúc dữ liệu danh sách, sau đó áp dụng các cấu trúc này để quản lý và xử lý dữ liệu bài hát trong một **Playlist**.

Toàn bộ hệ thống được xây dựng dựa trên ba thành phần chính:

- Cấu trúc danh sách liên kết **BotkifyLinkedList**
- Lớp đối tượng **Song**
- Lớp quản lý danh sách bài hát **Playlist**

Trong bài tập lớn này, danh sách liên kết sẽ thực hiện cấu trúc dữ liệu chính là danh sách do khả năng cấp phát động linh hoạt, không yêu cầu vùng nhớ liên tục và phù hợp với các thao tác thêm, xoá phần tử trong quá trình quản lý playlist.

Danh sách liên kết bao gồm các nút (node), trong đó mỗi nút lưu trữ dữ liệu và con trỏ trỏ tới nút kế tiếp trong danh sách. Nhờ cơ chế liên kết này, danh sách có thể mở rộng hoặc thu hẹp kích thước một cách linh hoạt trong quá trình thực thi chương trình.

3.1 Các cấu trúc dữ liệu dạng danh sách

3.1.1 Danh sách liên kết - BotkifyLinkedList

Danh sách liên kết là cấu trúc dữ liệu nền tảng được sử dụng xuyên suốt trong bài tập lớn này.

Lớp `Node`

Thuộc tính:

- `T data`: Dữ liệu của phần tử được lưu trong nút.
- `Node* next`: Con trỏ trỏ đến nút kế tiếp trong danh sách.

Các thuộc tính của lớp `BotkifyLinkedList`

- `Node* head`: Con trỏ tới nút đầu tiên trong danh sách.
- `Node* tail`: Con trỏ tới nút trong danh sách.
- `int count`: Số lượng phần tử hiện có trong danh sách.

Các phương thức `public` của lớp `BotkifyLinkedList`

- `BotkifyLinkedList()`
  - **Chức năng**: Khởi tạo một danh sách liên kết rỗng.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(1).
- `~BotkifyLinkedList()`
  - **Chức năng**: Giải phóng toàn bộ vùng nhớ các nút đã cấp phát, tránh rò rỉ bộ nhớ.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(n).
- `void add(T e)`

  - **Chức năng**: Thêm phần tử `e` vào cuối danh sách.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(n).

- `void add(int index, T e)`
  - **Chức năng**: Thêm phần tử `e` vào vị trí `index`.
  - **Ngoại lệ**: Ném `out_of_range("Index is invalid!")` nếu `index` không hợp lệ
  - **Độ phức tạp**: O(n).

- `T removeAt(int index)`
  - **Chức năng**: Xoá và trả về phần tử tại vị trí `index`.
  - **Ngoại lệ**: Ném `out_of_range("Index is invalid!")` nếu `index` không hợp lệ.
  - **Độ phức tạp**: O(n).

- `bool removeItem(T item)`
  - **Chức năng**: Xoá nút đầu tiên có giá trị bằng `item` trong danh sách. Trả về `true` nếu xoá thành công, `false` nếu không tìm thấy.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(n).

- `bool empty()`
  - **Chức năng**: Kiểm tra danh sách có rỗng hay không.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(1).

- `int size()`
  - **Chức năng**: Trả về số lượng phần tử hiện tại trong danh sách.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(1).

- `void clear()`
  - **Chức năng**: Xoá toàn bộ nút trong danh sách.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(n).

- `T& get(int index)`
  - **Chức năng**: Trả về tham chiếu tới dữ liệu tại vị trí `index` trong danh sách.
  - **Ngoại lệ**: Ném `out_of_range("Index is invalid!")` nếu `index` không hợp lệ.
  - **Độ phức tạp**: O(n).

- `string toString()`
  - **Chức năng**: Trả về chuỗi biểu diễn toàn bộ danh sách. Các phần tử in ra từ đầu đến cuối, cách nhau bởi dấu ‘,‘ và không có khoảng trắng giữa các kí tự.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: O(n).
  - **Định dạng in ra:** `<element 1>,<element 2>,<element 3>...`

Ví dụ 3.1

Nếu mảng bao gồm: 1, 2, 3, 4, 5
Kết quả trả về của toString: `1,2,3,4,5`

3.2 Lớp Song

Lớp **Song** là một đối tượng dùng để lưu trữ và quản lý thông tin của một bài hát trong ứng dụng *BotKify*. Mỗi đối tượng **Song** đại diện cho một bài hát cụ thể và chứa các thuộc tính cơ bản phục vụ cho việc tìm kiếm, phát nhạc và quản lý dữ liệu bài hát trong hệ thống.

Thuộc tính

- **int id**: Mã định danh duy nhất của bài hát.
- **string title**: Tên bài hát.
- **string artist**: Tên ca sĩ hoặc nhóm nhạc thể hiện bài hát.
- **string album**: Tên album chứa bài hát.
- **int duration**: Thời lượng của bài hát (tính bằng giây).
- **int score**: Điểm của một bài hát (nằm trong đoạn [0-1000].
- **string url**: Đường dẫn đến nguồn phát bài hát.

Chức năng

Lớp **Song** cung cấp các phương thức cơ bản để:

- **int getID()**
  - **Chức năng**: Trả về **id** của bài hát.
  - **Ngoại lệ**: Không phát sinh ngoại lệ.
  - **Độ phức tạp**: $O(1)$.

- **int getDuration()**
  - **Chức năng**: Trả về thời lượng - **duration** của bài hát.
  - **Ngoại lệ**: Không phát sinh ngoại lệ.
  - **Độ phức tạp**: $O(1)$.
- **string toString()**
  - **Chức năng**: Dùng để chuyển đổi bài hát sang chuỗi kí tự. Các kí tự nối với nhau bởi dấu gạch ngang (hyphen).
  - **Định dạng**: <title>-<artist>

Ví dụ 3.2

Nếu bài hát có (title, artist) = ("BabyShark","Pinkfong")
Kết quả trả về của toString: `BabyShark-Pinkfong`

3.3 Lớp Playlist

Lớp `Playlist` dùng để quản lý danh sách các bài hát trong ứng dụng BotKify. Danh sách bài hát được cài đặt bằng cấu trúc *BotKifyLinkedList*, trong đó mỗi nút chứa một đối tượng `Song`.

Các thuộc tính của lớp `Playlist`

- **string name**: Tên của playlist.
- **BotkifyLinkedList<Song *> lstSong**: Danh sách liên kết chứa toàn bộ các bài hát trong Playlist.
- **int size**: Số lượng bài hát hiện có trong playlist.

Các phương thức của lớp `Playlist`

- `Playlist(string name)`
  - **Chức năng**: Khởi tạo một Playlist với tên của Playlist.
  - **Ngoại lệ**: Không có.
  - **Độ phức tạp**: O(1).
- `int getSize()`
  - **Chức năng**: Trả về số lượng bài hát hiện có trong Playlist.
  - **Ngoại lệ**: Không có.

  - **Độ phức tạp**: O(1).

- `bool empty()`
  - **Chức năng**: Kiểm tra Playlist có rỗng hay không.
  - **Ngoại lệ**: Không có.
  - **Độ phức tạp**: O(1).
- `void clear()`
  - **Chức năng**: Xoá toàn bộ danh sách bài hát hiện có.
  - **Ngoại lệ**: Không có.
  - **Độ phức tạp**: O(n).
- `void addSong(Song s)`
  - **Chức năng**: Thêm một bài hát mới vào cuối danh sách liên kết của playlist.
  - **Ngoại lệ**: Không có ngoại lệ.
  - **Độ phức tạp**: $O(n)$.
- `void removeSong(int index)`
  - **Chức năng**: Xoá bài hát tại vị trí `index` trong danh sách liên kết.
  - **Ngoại lệ**: Ném `out_of_range("Index is invalid!")` nếu chỉ số không hợp lệ.
  - **Độ phức tạp**: $O(n)$.
- `Song* getSong(int index)`
  - **Chức năng**: Truy xuất bài hát tại vị trí `index` trong playlist.
  - **Ngoại lệ**: Ném `out_of_range("Index is invalid!")` nếu chỉ số không hợp lệ.
  - **Độ phức tạp**: $O(n)$.
- `Song* playNext()`
  - **Chức năng**: Từ bài hát hiện tại, tiếp tục phát bài hát tiếp theo (trả về bài hát ở vị trí tiếp theo). Nếu đang ở vị trí bài hát cuối cùng thì quay lại bài hát đầu tiên. Nếu chỉ có 1 bài thì phát chính bài hát đó.
  - **Ngoại lệ**: Ném `out_of_range("Index is invalid!")` nếu Playlist không có bài hát nào.
  - **Độ phức tạp**: $O(1)$.
- `Song* playPrevious()`
  - **Chức năng**: Từ bài hát hiện tại, phát bài hát liền kề trước đó (trả về bài hát ở vị trí liền kề trước đó). Nếu đang ở vị trí bài hát đầu tiên thì quay lại bài hát cuối cùng. Nếu chỉ có 1 bài thì phát chính bài hát đó.

- **Ngoại lệ**: Ném `out_of_range("Index is invalid!")` nếu Playlist không có bài hát nào.
- **Độ phức tạp**: $O(1)$.
  - Lưu ý: Sinh viên có thể thêm vào con trỏ hỗ trợ cho việc đạt độ phức tạp tối ưu.
  - Lời giải có độ phức tạp lớn hơn $O(1)$ sẽ nhận được 80% số điểm.

- `int getTotalScore()`
  - **Chức năng**: Trả về tổng điểm của Playlist.Điểm số của một Playlist được xác định bởi tổng điểm của tất cả các nhóm bài hát liên tiếp. Trong đó, tổng điểm của một nhóm bài hát liên tiếp được xác định là tích của hai giá trị:
    - Điểm thấp nhất trong nhóm bài hát đó.
    - Tổng điểm của tất cả các bài hát trong nhóm.

    Ví dụ 3.3

    Ví dụ danh sách Playlist có các bài với điểm số như sau: 4,1,3,5
    Cách tính tổng điểm là: - Nhóm có độ dài 1: 4*4 + 1*1 + 3*3 + 5*5 = 51
    - Nhóm có độ dài 2: 1*(1+4) + 1*(1+3) + 3*(3+5) = 33
    - Nhóm có độ dài 3: 1*(4+1+3) + 1*(1+3+5) = 17
    - Nhóm có độ dài 4: 1*(4+1+3+5) = 13
    Vậy tổng điểm của Playlist là: 51+33+17+13 = 114
  - **Ngoại lệ**: Không có
  - **Độ phức tạp**: $O(N)$. Lời giải có độ phức tạp lớn hơn $O(N)$ sẽ nhận được 80% số điểm.

- `bool compareTo(Playlist p, int numSong)`
  - **Chức năng**: Ngoài việc xác định điểm của toàn bộ playlist, ứng dụng BotKify hỗ trợ người dùng sắp xếp playlist dựa trên định nghĩa về điểm số và số lượng bài dự định nghe (numSong) liên tiếp. Để thực hiện việc sắp xếp này, sinh viên cần hiện thực hàm **compareTo** với các tác vụ sau:
    - Với mỗi playlist, dựa trên các nhóm gồm numSong bài hát liên tiếp để tìm ra bài có điểm số cao nhất.
    - Tính trung bình số điểm của tất cả các nhóm của từng playlist và so sánh. Trả về *true* nếu playlist hiện tại có điểm trung bình lớn hơn hoặc bằng **playlist p**, ngược lại, trả về *false*.

Ví dụ 3.4

Ví dụ danh sách Playlist A có các bài với điểm số là 4,1,3,5, Playlist B có các bài hát với điểm số là 1,2,3. Xác định so sánh Playlist với số lượng bài là 2 (numSong=2).

Tính điểm dựa trên Playlist A:

- (max(4,1) + max(1,3)+ max(3,5))/3 = 4

Tính điểm dựa trên Playlist B:

- (max(1,2) + max(2,3))/2 = 2.5

Lời gọi hàm **A.compareTo(B, 2)** trả về *true*

- **Ngoại lệ**: Không có
- **Độ phức tạp**: $O(N)$. Lời giải có độ phức tạp lớn hơn $O(N)$ sẽ nhận được 80% số điểm.

- `void playRandom(int index)`
  - **Chức năng**: Thực hiện phát bài nhạc ngẫu nhiên bằng cách bắt đầu ở một bài hát có vị trí **index** trong danh sách bài hát. Bài hát tiếp theo được phát thỏa mãn tính chất là bài gần nhất có **duration** lớn hơn bài mới vừa phát. Tiếp tục phát như vậy đến khi không thể phát được nữa.
  - **Ngoại lệ**: Không có
  - **Độ phức tạp**: $O(N)$. Lời giải có độ phức tạp lớn hơn $O(N)$ sẽ nhận được 80% số điểm.
  - **Định dạng**: In ra các bài hát theo thứ tự phát với định dạng: **<title1>-<artist1>,<title2>-<artist2>,...**

- `int playApproximate(int step)`
  - **Chức năng**: Ứng dụng tối ưu thời gian nghe nhạc với một thời lượng gần như nhau để tránh việc trong một lượt nghe playlist có các bài liên tiếp phát chênh lệch nhau về thời lượng quá nhiều. Tuy nhiên, do vấn đề kết nối mạng nên số bài có thể skip qua tối đa là step bài. Do đó, sinh viên thực hiện phát các bài nhạc từ đầu tới bài nhạc cuối cùng (bắt buộc phải có bài đầu và bài cuối) với ràng buộc tổng thời gian chênh lệch của tất cả các bài hát được phát kề nhau là ít nhất. Trả về tổng thời lượng chênh lệch nhỏ nhất.

Ví dụ 3.5

Ví dụ playlist A gồm các bài hát có độ dài (duration) như sau: 50, 60, 30, 90, 100. Số bài tối đa được skip qua là 1 bài (step=1)
Các bài nên phát là: bài 0, bài 1, bài 3, bài 4
Tổng thời gian trả về là: 10 + 30 + 10 = 50

- **Ngoại lệ**: Không có
- **Độ phức tạp**: $O(N * step)$. Lời giải có độ phức tạp lớn hơn $O(N * step)$ sẽ nhận được 80% số điểm.

4 Yêu cầu và chấm điểm

4.1 Yêu cầu

Để hoàn thành bài tập này, sinh viên cần:

1. Đọc toàn bộ file mô tả này.
2. Tải file **initial.zip** và giải nén. Sau khi giải nén, sinh viên sẽ nhận được các file bao gồm: `utils.h`, `main.cpp`, `main.h`, `Playlist.h`, `Playlist.cpp` và `BotkifyLinkedList.h` . Sinh viên chỉ nộp 3 file, đó là `BotkifyLinkedList.h`, `Playlist.h` và `Playlist.cpp`. Do đó, không được phép chỉnh sửa file `main.h` khi kiểm tra chương trình.
3. Sinh viên sử dụng lệnh sau để biên dịch:
   `g++ -o main main.cpp Playlist.cpp -I . -std=c++17`
   Lệnh trên được sử dụng trong Command Prompt/Terminal để biên dịch chương trình. Nếu sinh viên sử dụng IDE để chạy chương trình, cần lưu ý: thêm tất cả các file vào project/workspace của IDE; chỉnh lại lệnh biên dịch trong IDE cho phù hợp. IDE thường cung cấp nút Build (biên dịch) và Run (chạy). Khi bấm Build, IDE sẽ chạy câu lệnh biên dịch tương ứng, thông thường chỉ biên dịch file `main.cpp`. Sinh viên cần tìm cách cấu hình để thay đổi câu lệnh biên dịch, cụ thể: thêm file `Playlist.cpp`, thêm tùy chọn `-std=c++17`, và `-I .`
4. Chương trình sẽ được chấm trên nền tảng Unix. Môi trường của sinh viên và trình biên dịch có thể khác với môi trường chấm thực tế. Khu vực nộp bài trên LMS được thiết lập tương tự môi trường chấm thực tế. Sinh viên bắt buộc phải kiểm tra chương trình trên trang nộp bài, đồng thời sửa tất cả lỗi phát sinh trên hệ thống LMS để đảm bảo kết quả chính xác khi chấm cuối cùng.

5. Chỉnh sửa file `BotkifyLinkedList.h`, `Playlist.h` và `Playlist.cpp` để hoàn thành bài tập, đồng thời đảm bảo hai yêu cầu sau:
   - Tất cả các phương thức được mô tả trong file hướng dẫn phải được cài đặt để chương trình có thể biên dịch thành công. Nếu sinh viên chưa hiện thực một phương thức nào đó, cần cung cấp phần hiện thực rỗng cho phương thức đó. Mỗi testcase sẽ gọi một số phương thức để kiểm tra kết quả trả về.
   - Trong file `Playlist.h` chỉ được phép có dòng `#include "main.h"` và `#include "BotkifyLinkedList.h"`, và trong file `Playlist.cpp` chỉ được phép có một dòng `#include "Playlist.h"`. Ngoài hai dòng này, không được phép thêm bất kỳ lệnh `#include` nào khác trong các file này.
6. Khuyến khích sinh viên được phép viết thêm các lớp, phương thức và thuộc tính phụ trợ trong các lớp yêu cầu hiện thực. Nhưng sinh viên phải đảm bảo các lớp, phương thức này không làm thay đổi yêu cầu của các phương thức được mô tả trong đề bài.
7. Sinh viên phải thiết kế và sử dụng các cấu trúc dữ liệu đã học.
8. Sinh viên bắt buộc phải giải phóng toàn bộ vùng nhớ cấp phát động khi chương trình kết thúc.

4.2 Thời hạn nộp bài

Thời hạn **theo thông báo trên LMS**. Sinh viên vui lòng nộp bài lên hệ thống trước thời hạn thông báo. Sinh viên tự chịu trách nhiệm nếu xuất hiện các lỗi trên hệ thống do nộp gần sát giờ quy định.

4.3 Chấm điểm

Toàn bộ mã nguồn của sinh viên sẽ được chấm trên bộ testcases ẩn, điểm được tính theo từng yêu cầu, cụ thể:

- Hiện thực danh sách liên kết cho Botkify: **4 điểm**.
- Hiện thực Playlist: **6 điểm**.

5 Harmony cho Bài tập lớn

Bài kiểm tra cuối kì của môn học sẽ có một số câu hỏi Harmony với nội dung của BTL.

Sinh viên phải giải quyết BTL bằng khả năng của chính mình. Nếu sinh viên gian lận trong BTL, sinh viên sẽ không thể trả lời câu hỏi Harmony và nhận điểm 0 cho BTL.

Sinh viên **phải** chú ý làm câu hỏi Harmony trong bài kiểm tra cuối kỳ. Các trường hợp không làm sẽ tính là 0 điểm cho BTL, và bị không đạt cho môn học. **Không chấp nhận giải thích và không có ngoại lệ.**

6 Quy định và xử lý gian lận

Bài tập lớn phải được sinh viên TỰ LÀM. Sinh viên sẽ bị coi là gian lận nếu:

- Có sự giống nhau bất thường giữa mã nguồn của các bài nộp. Trong trường hợp này, TẤT CẢ các bài nộp đều bị coi là gian lận. Do vậy sinh viên phải bảo vệ mã nguồn bài tập lớn của mình.
- Sinh viên không hiểu mã nguồn do chính mình viết, trừ những phần mã được cung cấp sẵn trong chương trình khởi tạo. Sinh viên có thể tham khảo từ bất kỳ nguồn tài liệu nào, tuy nhiên phải đảm bảo rằng mình hiểu rõ ý nghĩa của tất cả những dòng lệnh mà mình viết. Trong trường hợp không hiểu rõ mã nguồn của nơi mình tham khảo, sinh viên được đặc biệt cảnh báo là KHÔNG ĐƯỢC sử dụng mã nguồn này; thay vào đó nên sử dụng những gì đã được học để viết chương trình.
- Nộp nhầm bài của sinh viên khác trên tài khoản cá nhân của mình.
- Sinh viên sử dụng các công cụ AI trong quá trình làm bài tập lớn dẫn đến các mã nguồn giống nhau.

Trong trường hợp bị kết luận là gian lận, sinh viên sẽ bị điểm 0 cho toàn bộ môn học (không chỉ bài tập lớn).

KHÔNG CHẤP NHẬN BẤT KỲ GIẢI THÍCH NÀO VÀ KHÔNG CÓ BẤT KỲ NGOẠI LỆ NÀO!

Sau mỗi bài tập lớn được nộp, sẽ có một số sinh viên được gọi phỏng vấn ngẫu nhiên để chứng minh rằng bài tập lớn vừa được nộp là do chính mình làm.

Một số quy định khác:

- Mọi quyết định của giảng viên phụ trách bài tập lớn là quyết định cuối cùng.
- Sinh viên không được cung cấp testcase sau khi chấm bài.

- Nội dung Bài tập lớn sẽ được Harmony với các câu hỏi trong bài kiểm tra cuối kì với nội dung tương tự.

——————————HẾT——————————