HỌC VIỆN NGÂN HÀNG

KHOA HỆ THỐNG THÔNG TIN QUẢN LÝ

BÀI TẬP LỚN

MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CÁC PHÉP TOÁN THÊM, SỬA, XÓA VỚI DANH SÁCH LIÊN KẾT VÒNG

NHÓM 13

HÀ NỘI – 03/2023

HỌC VIỆN NGÂN HÀNG

KHOA HỆ THỐNG THÔNG TIN QUẢN LÝ

BÀI TẬP LỚN

MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CÁC PHÉP TOÁN THÊM, SỬA, XÓA VỚI DANH SÁCH LIÊN KẾT VÒNG

Giảng viên hướng dẫn: **Nguyễn Thanh Thụy**

Danh sách nhóm:

	Họ tên	**Mã sinh viên**
1.	Nguyễn Tiến Mạnh	24A4040033
2.	Trần Thị Kim Anh	24A4043030
3.	Trần Đức Lộc	24A4041407
4.	Nông Huyền My	24A4041413

HÀ NỘI – 03/2023

MỨC ĐỘ ĐÓNG GÓP CỦA CÁC THÀNH VIÊN

STT	MÃ SV	HỌ TÊN	CÔNG VIỆC	MỨC ĐỘ ĐÓNG GÓP
1	24A4040033	Nguyễn Tiến Mạnh (NT)	Tìm nội dung, tổng hợp word, đóng góp ý kiến, làm slides, thuyết trình	28%
2	24A4043030	Trần Thị Kim Anh	Tìm nội dung, đóng góp ý kiến	24%
3	24A4041407	Trần Đức Lộc	Tìm nội dung, đóng góp ý kiến	24%
4	24A4041413	Nông Huyền My	Tìm nội dung, đóng góp ý kiến	24%

MỤC LỤC

LỜI CẢM ƠN

Được sự phân công của thầy Nguyễn Thanh Thụy trường Học viện Ngân hàng, sau thời gian vừa qua, chúng em đã hoàn thành bài tiểu luận về "Các phép toán thêm, sửa, xóa với danh sách liên kết vòng".

Để hoàn thiện bài tiểu luận này, lời đầu tiên chúng em xin gửi lời cảm ơn sâu sắc nhất đến thầy Nguyễn Thanh Thụy. Thầy đã trực tiếp chỉ bảo và hướng dẫn chúng em trong suốt quá trình nghiên cứu, hoàn thiện bài tiểu luận này.

Do hiểu biết còn nhiều hạn chế, nội dung bài tiểu luận có thể còn nhiều thiếu sót, vì vậy, chúng em rất mông nhận được sự góp ý của thầy để hoàn thiện bài tiểu luận này hơn.

Một lần nữa, chúng em xin chân thành cảm ơn!

LỜI CAM ĐOAN

Từ những kiến thức cũng như những trải nghiệm học tập, nghiên cứu tại Học viện Ngân hàng, chúng em cũng đã tham khảo và tìm hiểu thêm các sách báo, tạp chí hay các tài liệu trên mạng. Từ đó, chúng em đã tập hợp thông tin và chỉnh sửa để có thể hoàn thành bài tiểu luận này.

Chúng em xin cam đoan nội dung bài tiểu luận là kết quả của quá trình nghiên cứu của chính chúng em. Do trình độ còn hạn chế nên bài tiểu luận này không tránh khỏi những sai sót, chúng em rất mong nhận được góp ý từ thầy.

Chúng em xin cam đoan những điều trên là đúng sự thật, nếu phát hiện bất kì sự gian lận nào, chúng em xin hoàn toàn chịu trách nhiệm trước nhà trường và pháp luật.

LỜI MỞ ĐẦU

Hầu như tất cả chúng ta đều đã từng chơi qua một trò chơi rất nổi tiếng có tên là "Chinese Whispers". Đó là một trò chơi mà một người sẽ thì thầm một thông điệp tới người tiếp theo cho đến khi kết thúc hàng. Trò chơi này được chơi phổ biến để nhằm mục đích phá vỡ không khí ngại ngùng giữa nhiều người trong một nhóm.

Trò chơi này cũng có thể được chơi theo một vòng tròn. Khi được chơi trong một vòng tròn, người cuối cùng sẽ gửi thông điệp mà người ấy nghe được tới người đầu tiên một lần nữa. Theo cách này, một liên kết giữa người cuối cùng và người đầu tiên được hình thành.

Trong danh sách liên kết vòng đơn cũng tương tự như vậy, thay vì lưu trữ NULL trong phần địa chỉ của nút cuối cùng, địa chỉ của nút đầu được lưu để tạo một danh sách vòng.

Có thể thấy, danh sách liên kết vòng đơn rất phổ biến trong thục tế; vậy trong lập trình danh sách liên kết đơn sẽ như thế nào?

Vậy chúng ta sẽ cùng đi tìm hiểu về "Các phép toán thêm, sửa, xóa với danh sách liên kết vòng".

NỘI DUNG

1. Giới thiệu về thuật toán danh sách liên kết vòng

Danh sách liên kết vòng (Circularly linked list) là một biến thể của Danh sách liên kết (Linked list), trong đó phần tử đầu tiên trỏ tới phần tử cuối cùng và phần tử cuối cùng trỏ tới phần tử đầu tiên.

Trong danh sách liên kết đơn, điểm trỏ tới kế tiếp của nút cuối sẽ trỏ tới nút đầu tiên, thay vì sẽ trỏ tới NULL.

Trong danh sách liên kết đôi, điểm trỏ tới kế tiếp của nút cuối trỏ tới nút đầu tiên và điểm trỏ tới phía trước của nút trước sẽ trỏ tới nút cuối cùng. Quá trình này sẽ tạo vòng ở cả hai hướng.

Cả hai loại Danh sách liên kết đơn (Singly Linked List) và Danh sách liên kết đôi (Doubly Linked List) đều có thể được tạo thành dạng Danh sách liên kết vòng.

2. Ý tưởng và các bước tiến hành

2.1. Ý tưởng

Ta có thể khai báo danh sách liên kết vòng như khai báo danh sách liên kết đơn. Trên danh sách liên kết vòng không có phần tử đầu danh sách rõ rệt, nhưng ta có thể đánh dấu một phần tử bất kỳ trên danh sách xem như phần tử đầu danh sách để kiểm tra việc duyệt đã qua hết các phần tử của danh sách hay chưa.

Việc cải tiến hơn của danh sách liên kết vòng so với đanh sách liên kết đơn là điểm trỏ tới kế tiếp của nút cuối sẽ trỏ tới nút đầu tiên, thay vì sẽ trỏ tới NULL. Điều này làm cho việc truy nhập vào các nút trong danh sach được linh hoạt hơn. Chúng ta chỉ cần biết con trỏ trỏ tới nút muốn loại bỏ ta vẫn thực hiện được vì vẫn tìm được đến nút đứng trước nó, với phép ghép và phép tách cũng có những thuận lợi nhất định.

Tuy nhiên, danh sách nối vòng có một nhược điểm rất rõ là trong xử lý, nếu không cẩn thận sẽ dẫn tới một chu trình không kết thúc, sở dĩ là vì không biết được chỗ kết thúc của danh sách. Điều này có thể khắc phục được bằng cách đưa thêm vào một nút đặc biệt gọi là “nút đầu danh sách”. Trường INFO của nút này không chứa dữ liệu của phần tử nào và con trỏ HEAD bây giờ trỏ tới nút đầu danh sách này, cho phép ta truy cập danh sách. Việc thêm nút đầu danh sách đã khiến cho danh sách về mặt hình thức không bao giờ rỗng.

2.2. Các bước thực hiện

Xác định cấu trúc nút: Tạo cấu trúc nút chứa trường dữ liệu và con trỏ tới nút tiếp theo. Đối với danh sách vòng tròn, nút cuối cùng sẽ trỏ lại nút đầu tiên.

Khởi tạo danh sách: Tạo một nút mới sẽ là đầu danh sách. Đặt con trỏ tiếp theo của phần đầu thành chính nó để tạo danh sách vòng.

Thực hiện chèn: Để chèn một nút mới, ta tạo một cấu trúc nút mới và gán giá trị cho trường dữ liệu của nó. Sau đó, đặt con trỏ tiếp theo của nút mới để trỏ đến nút tiếp theo sau nút hiện tại. Sau đó, đặt con trỏ tiếp theo của nút hiện tại để trỏ đến nút mới.

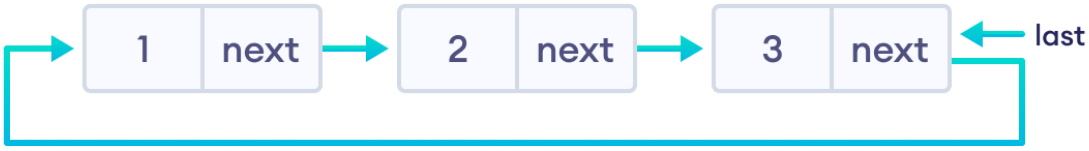
Thực hiện xóa: Để xóa một nút, duyệt qua danh sách để tìm nút cần xóa. Sau đó, cập nhật con trỏ tiếp theo của nút trước để trỏ đến nút tiếp theo sau nút đã xóa.

Triển khai duyệt: Sử dụng một vòng lặp để duyệt qua danh sách liên kết vòng, bắt đầu từ phần đầu và đi cho đến khi đạt đến nút cuối cùng (nghĩa là con trỏ tiếp theo của phần đầu bằng với phần đầu).

Cập nhật phần đầu: Nếu di chuyển phần đầu đến một vị trí khác trong danh sách, đặt con trỏ phần đầu thành nút mới và cập nhật con trỏ tiếp theo của nút cuối cùng để trỏ đến phần đầu mới.

2.3. Ví dụ minh họa

Dưới đây là các hình ảnh minh họa biểu diễn danh sách liên kết vòng. Giả sử chúng ta có danh sách liên kết vòng sau:



Nút 1:

- `Next` lưu trữ địa chỉ của 2 (không có nút nào trước nó)

Nút 2:

- `Next` lưu địa chỉ của 3

Nút 3:

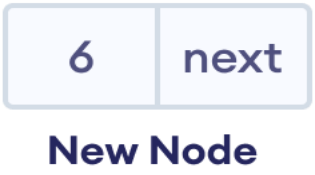
- `Next` lưu lại NULL (không có nút nào sau nó)
- `Next` trỏ đến nút 1

- Chèn trong danh sách liên kết vòng

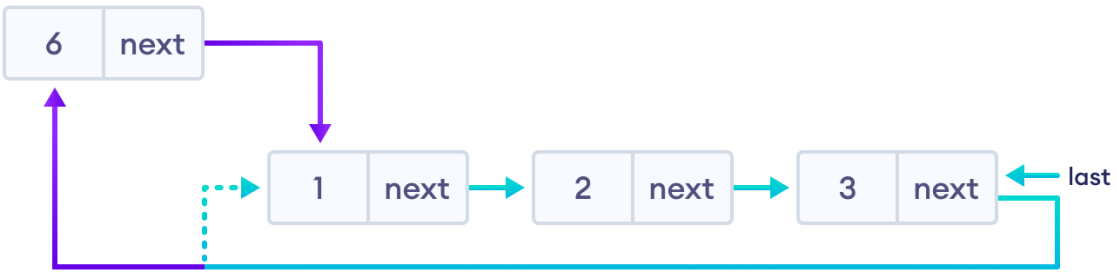
Ta có thể chèn ở 3 vị trí khác nhau trong danh sách liên kết vòng

Ví dụ muốn thêm một nút có giá trị 6 ở vị trí khác nhau của danh sách liên kết vòng ta có ở trên. Đầu tiên ta tạo một nút mới:

- Cấp bộ nhớ cho nút mới
- Gán dữ liệu tới nút mới

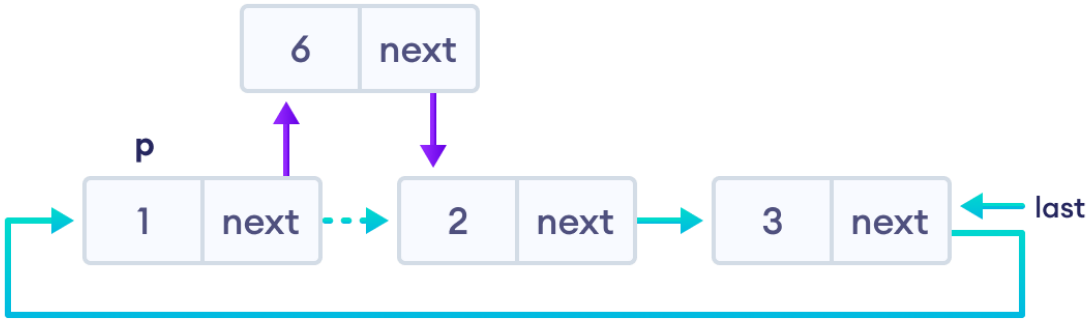


- Chèn đầu
  - Lưu địa chỉ của nút đầu hiện tại vào nút mới (trỏ nút mới tới nút đầu)
  - Trỏ nút cuối tới nút mới (tạo nút mới làm nút đầu)



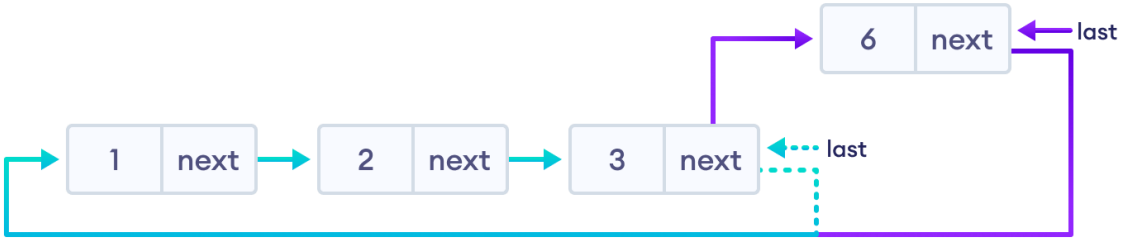
- Chèn giữa 2 nút

  Để chèn nút mới vào sau nút đầu
  - Di chuyển đến nút đã cho (giả sử nút này là p)
  - Trỏ next của nút mới tới nút bên cạnh p
  - Lưu địa chỉ của nút mới tại next của p



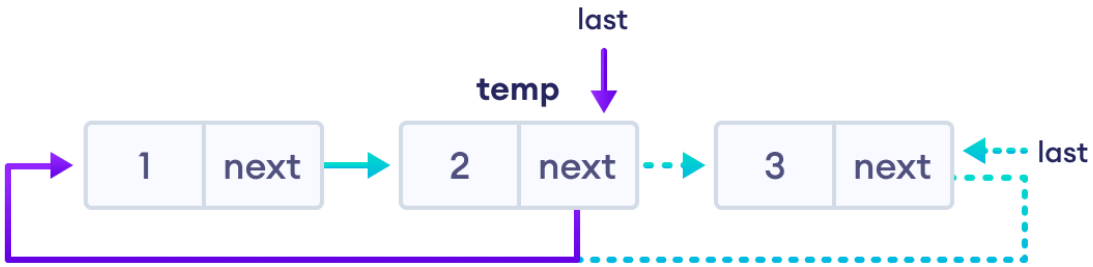
- Chèn cuối
  - Lưu địa chỉ của nút đầu tới next của nút mới (làm nút mới là nút cuối)
  - Trỏ nút cuối hiện tại tới nút mới

- Tạo nút mới làm nút cuối



- Xóa trong danh sách liên kết vòng

- Nếu nút bị xóa là nút đầu tiên
  - Chuyển nút đầu sang nút tiếp theo và giải phóng bộ nhớ của nút đầu tiên
- Nếu nút cuối là nút bị xóa
  - Tìm nút cuối thứ 2 (giả sử nó là temp)
  - Lưu địa chỉ của nút bên cạnh nút cuối vào temp
  - Giải phòng bộ nhớ của nút cuối
  - Tạo temp làm nút cuối



- Nút bị xóa là nút bất kỳ
  - Di chuyển đến nút bị xóa (ở đây nút ta xóa là nút 2)
  - Giả sử nút trước nút 2 là temp
  - Lưu địa chỉ của nút cạnh nút 2 vào temp
  - Giải phóng bộ nhớ của 2

3. Cài đặt thuật toán

```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
  int data;
  struct Node* next;
};

struct Node* addToEmpty(struct Node* last, int data) {
  if (last != NULL) return last;

  // cấp phát bộ nhớ cho nút mới
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

  // gán dữ liệu cho nút mới
  newNode->data = data;

  // gán cuối cho nút mới
  last = newNode;

  // tạo link tới chính nó
  last->next = last;

  return last;
}

// thêm nút vào đầu
struct Node* addFront(struct Node* last, int data) {
  // kiểm tra danh sách có rỗng hay không
  if (last == NULL) return addToEmpty(last, data);

  // cấp phát bộ nhớ cho nút mới
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

  // thêm dữ liệu vào nút
```

```
  newNode->data = data;

  // lưu địa chỉ của nút đầu hiện tại vào nút mới
  newNode->next = last->next;

  // tạo nút mới làm nút đầu
  last->next = newNode;

  return last;
}

// Chèn nút vào cuối
struct Node* addEnd(struct Node* last, int data) {
  // kiểm tra xem nút có rống không
  if (last == NULL) return addToEmpty(last, data);

  // cấp phát bộ nhớ cho nút mới
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

  // thêm dữ liệu vào nút
  newNode->data = data;

  // lưu địa chỉ của nút đầu tới next của nút mới
  newNode->next = last->next;

  // trỏ nút cuối hiện tại tới nút mới
  last->next = newNode;

  // tạo nút mới làm nút cuối
  last = newNode;

  return last;
}

// chèn nút sau một nút cụ thể
struct Node* addAfter(struct Node* last, int data, int item) {
  // kiểm tra danh sách có rỗng không
  if (last == NULL) return NULL;

  struct Node *newNode, *p;

  p = last->next;
  do {
  // nếu p được tìm thấy, đặt nút mới sau nó
  if (p->data == item) {
    // cấp phát bộ nhớ tới nút mới
    newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    // thêm dữ liệu vào nút
    newNode->data = data;

    // đặt nút tiếp theo của nút hiện tại làm nút tiếp theo của newNode
    newNode->next = p->next;

    // đặt nút mới tới nút tiếp theo của p
    p->next = newNode;

    // nếu p là nút cuối cùng, tạo nút mới làm nút cuối
    if (p == last) last = newNode;
    return last;
  }

  p = p->next;
  } while (p != last->next);

  printf("\nThe given node is not present in the list");
  return last;
}

// xóa một nút
void deleteNode(struct Node** last, int key) {
  // nếu danh sách liên kết rỗng
  if (*last == NULL) return;

  // nếu danh sách liên kết chỉ chứa một nút đơn
  if ((*last)->data == key && (*last)->next == *last) {
  free(*last);
  *last = NULL;
  return;
  }

  struct Node *temp = *last, *d;

  // nếu nút cuối bị xóa
  if ((*last)->data == key) {
  // tìm nút đứng trước nút cuối
  while (temp->next != *last) temp = temp->next;

  // trỏ nút temp tới next của nút cuối (ở dây là nút đầu)
  temp->next = (*last)->next;
  free(*last);
  *last = temp->next;
  }
```

```
  // di chuyển tới nút bị xóa
  while (temp->next != *last && temp->next->data != key) {
  temp = temp->next;
  }

  // nếu nút bị xóa được tìm thấy
  if (temp->next->data == key) {
  d = temp->next;
  temp->next = d->next;
  free(d);
  }
}
```

//duyệt danh sách liên kết vòng

```
void traverse(struct Node* last) {
  struct Node* p;

  if (last == NULL) {
  printf("The list is empty");
  return;
  }

  p = last->next;

  do {
  printf("%d ", p->data);
  p = p->next;

  } while (p != last->next);
}

int main() {
  struct Node* last = NULL;

  last = addToEmpty(last, 6);
  last = addEnd(last, 8);
  last = addFront(last, 2);

  last = addAfter(last, 10, 2);

  traverse(last);

  deleteNode(&last, 8);

  printf("\n");

  traverse(last);
```

```
  return 0;
}
```

4. Độ phức tạp

Thêm/xóa 1 phần tử mới vào đầu/cuối: O(1)

Truy cập 1 phần tử ở vị trí bất kỳ: O(N)

5. Ưu, nhược điểm của thuật toán

5.1. Ưu điểm

Cả hai loại danh sách liên kết đơn (Circular Linked List) và danh sách liên kết đôi (Doubly Linked List) đều giá rẻ, có thể tối ưu được tạo thành dạng Danh sách liên kết vòng bản quyền.

Bất kỳ node nào cũng có thể là điểm bắt đầu (starting point). Chúng ta có thể duyệt toàn bộ danh sách bằng cách bắt đầu tại bất kỳ điểm nào, tùy ý. Chúng ta chỉ cần dừng lại khi node đầu tiên được duyệt, lại được duyệt tới một lần nữa.

Danh sách liên kết vòng rất hữu dụng trong việc triển khai queue – hàng đợi. Khi sử dụng danh sách liên kết vòng để cài đặt queue, chúng ta sẽ không cần phải duy trì hai điểm front (điểm đầu) và rear (điểm cuối) của queue. Chúng ta chỉ cần duy trì một con trỏ dành cho note mới nhất được chèn vào (nằm ở cuối danh sách), lấy node cuối cùng này làm mốc, ta có thể dễ dàng truy cập tới các nodes nằm ở phần đầu danh sách.

Danh sách liên kết dạng vòng còn hữu dụng trong các ứng dụng phần mềm, để lặp đi lặp lại danh sách. Ví dụ, khi nhiều ứng dụng đang chạy trên một Máy tính, thông thường hệ điều hành sẽ đặt các ứng dụng đang chạy vào một danh sách, và sau đó duyệt qua chúng, mỗi lần duyệt tới ứng dụng nào thì lại cho ứng dụng đó một khoảng thời gian để thực thi, và sau đó lại đưa chúng về trạng thái đợi trong khi CPU được cấp cho một ứng dụng khác. Vì thế, sẽ rất lý tưởng khi hệ điều hành sử dụng một danh sách liên kết vòng để khi duyệt đến cuối danh sách, nó có thể quay vòng lên phía trước danh sách/phần đầu danh sách.

Danh sách liên kết vòng kép (Circular Doubly Linked List) được sử dụng để cài đặt các cấu trúc dữ liệu nâng cao, ví dụ như Fibonacci Heap.

5.2. Nhược điểm

Thời gian truy cập tuyến tính (và khó thực thi ống dẫn). Việc truy cập nhanh hơn, ví dụ như truy cập ngẫu nhiên là không khả thi. Mảng có vùng đệm (cache locality) tốt hơn so với danh sách liên kết.

Chính vì lý do trên mà một số phép tính như tìm phần tử cuối cùng, xóa phần tử ngẫu nhiên hay chèn thêm, tìm kiếm có thể phải duyệt tất cả các phần tử.

6. Ứng dụng của thuật toán

Được sử dụng trong các ứng dụng trong đó toàn bộ danh sách được truy cập từng cái một trong một vòng lặp. Ví dụ: Hệ điều hành.

Được sử dụng trong các trò chơi nhập vai nhiều để xoay tua lượt chơi giữa các người chơi.

KẾT LUẬN

Tóm lại, thuật toán danh sách liên kết vòng là một cấu trúc dữ liệu mạnh mang lại mức độ linh hoạt cao, dễ thực hiện và sử dụng bộ nhớ hiệu quả. Thuật toán này lý tưởng cho các ứng dụng yêu cầu cấu trúc dữ liệu vòng tròn, chẳng hạn như lập lịch quay vòng, đệm âm thanh và video, theo dõi chuyển động vòng tròn, v.v. Tuy nhiên, mặc dù có nhiều lợi ích, điều quan trọng cần lưu ý là thuật toán danh sách liên kết vòng có thể phức tạp và đòi hỏi sự chú ý cẩn thận đến từng chi tiết để triển khai hiệu quả. Bằng cách hiểu các nguyên tắc cơ bản đằng sau thuật toán cấu trúc dữ liệu mạnh mẽ này, các nhà phát triển có thể tạo các ứng dụng mạnh mẽ và đáng tin cậy, có thể xử lý các cấu trúc dữ liệu phức tạp một cách dễ dàng.

TÀI LIỆU THAM KHẢO

[1] Đ. X. Lôi, Cấu trúc dữ liệu và giải thuật, In lần thứ chín, có chỉnh sửa ed., Hà Nội: Nhà xuất bản Đại học Quốc gia Hà Nội, 2006.

[2] D. Xuân, "cafedev," 07 10 2020. [Online]. Available: https://cafedev.vn/ctdl-gioi-thieu-circular-linked-list-danh-sach-lien-ket-vong/?fbclid=IwAR01UuHHlvIyuqjlxhYrV6xuB2wk6YirQXaegcNcE1YsvOLr1O4iGfmNpec. [Accessed 27 03 2023].

[3] K. C. Đ. K. H. Yên, "Cấu trúc dữ liệu và giải thuật," in *Cấu trúc dữ liệu và giải thuật*, K. C. Đ. K. H. Yên, Ed., Hưng Yên, pp. 106 - 110.

[4] PREPBYTES, "PrepBytes Blog," 03 08 2021. [Online]. Available: https://www.prepbytes.com/blog/linked-list/circular-linked-list-introduction-and-applications/. [Accessed 27 03 2023].

[5] iamvtn, "Cộng đồng lập trình Việt Nam," 01 2007. [Online]. Available: http://diendan.congdongcviet.com/threads/t5507::danh-sach-lien-ket-vong-circular-linked-list.cpp. [Accessed 27 03 2023].

[6] C. M. Tân, N. L. T. Hiếu, N. Đ. Duy and K. T. Đạt, "Prezi," 24 05 2019. [Online]. Available: https://prezi.com/p/-nn3hgzmvtf-/danh-sach-lien-ket-on-vong/?fallback=1. [Accessed 27 03 2023].

[7] Simplilearn, "Simplilearn - Online Certification Training Course Provider," 18 09 2021. [Online]. Available: https://www.programiz.com/dsa/circular-linked-list. [Accessed 27 03 2023].