BÀI 15 CẤU TRÚC DỮ LIỆU DANH SÁCH LIÊN KẾT VÀ ỨNG DỤNG

Học xong bài này, em sẽ:

Trình bày được cấu trúc dữ liệu danh sách liên kết và một số ứng dụng của nó.

Hãy cho biết danh sách mảng có nhược điểm gì?

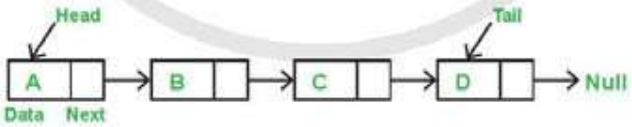
1 Cấu trúc danh sách liên kết

Danh sách liên kết (*linked list*) cũng gọi là danh sách móc nối, gồm các phần tử gọi là nút (*node*). Một nút có hai thành phần: phần *Data* chứa dữ liệu, phần liên kết gọi là *Next*. Phần *Next* chứa địa chỉ của nút liền kề và được thể hiện bằng mũi tên từ *Next* trỏ đến nút đứng kề sau nó (xem *Hình 1*). Trong hình minh hoạ, các chữ cái *A*, *B*, *C*, *D* thể hiện dữ liệu chứa trong nút.

Để tiện trình bày, ta gọi phần *Next* trong một nút là *con trỏ Next*. Kí hiệu mũi tên "→" trong hình minh hoạ thể hiện *Next* đang trỏ vào cái gì. Về bản chất, kí hiệu mũi tên "→" để thể hiện một kiểu dữ liệu kiểu đặc biệt, gọi là kiểu *con trỏ*. Con trỏ cho phép truy cập trực tiếp đến một địa chỉ ô nhớ cụ thể.

Đuôi danh sách là nút cuối cùng trong danh sách, không có nút nào đứng sau, được thể hiện bằng hình vẽ *Next* trỏ đến *Null* và được hiểu rằng "không trỏ đến đâu cả, không đi được tiếp nữa". Con trỏ *Tail* trỏ đến nút đuôi danh sách.

Đầu danh sách được minh hoạ bằng mũi tên *Head* trỏ đến nút đầu tiên trong danh sách.



Hình 1. Minh hoạ một danh sách liên kết có 4 nút

Khi lập trình, cần phân biệt một nút với phần *Data* chứa dữ liệu trong nút đó, phân biệt phần *Data* với chính dữ liệu chứa trong nút đó, thể hiện bằng các chữ cái *A*, *B*, *C*, *D* trong hình minh hoạ. Nhưng để đơn giản trong trình bày các phép toán trong phần tiếp theo, ta gọi là nút *A*, nút *B*, nút *C*, nút *D*. Viết *A.Next* để nói về con trỏ từ nút *A* đến nút đứng sau nó và dễ nhận biết trên hình minh hoạ.

Sự khác nhau giữa danh sách liên kết và mảng

So với mảng, danh sách liên kết có những điểm khác biệt sau:

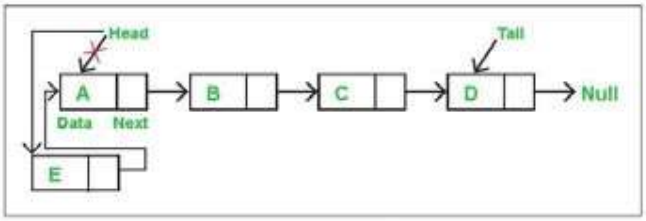
– Các nút danh sách liên kết không được lưu trữ thành một khối liên tục liền kề mà có thể nằm rải rác, tách rời nhau trong bộ nhớ.

– Các nút trong danh sách liên kết không có chỉ số. Phép lặp duyệt tuần tự từng nút của danh sách liên kết sử dụng một con trỏ *curr* (*current*) chỉ vào nút đang xét, thực hiện như sau:

+ *curr* = *Head* bắt đầu từ *Head* để truy cập nút *A*;

+ *curr* = *A.Next* để truy cập nút *B*; *curr* = *B.Next* để truy cập nút *C*;...

+ Kết thúc khi gặp *curr* = *Null* tức là ở tình huống *curr* = *D.Next*.



Hình 2a. Thêm nút vào đầu danh sách

Thêm nút và gỡ bỏ nút

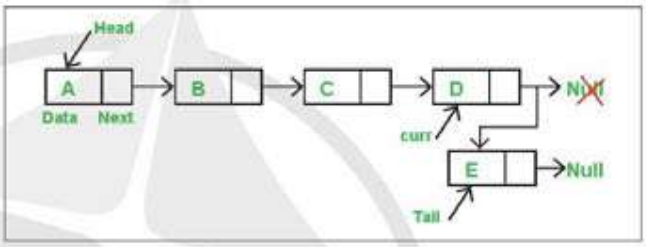
• Thêm nút có ba trường hợp:

a) Thêm nút vào đầu danh sách

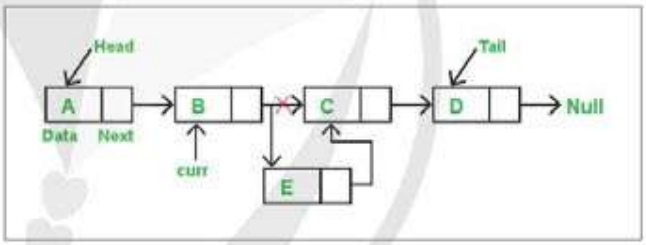
Nút mới thêm thành nút đầu tiên. Gọi nút mới là *E*. Thao tác theo hai bước sau (*Hình 2a*):

– Cho *E.Next* trỏ đến nút *A*: gán *E.Next* = *Head*.

– Cho *Head* trỏ đến nút *E*: *Head* → *E*.



Hình 2b. Thêm nút vào cuối danh sách



Hình 2c. Thêm nút vào giữa danh sách

Thời gian thực hiện phép thêm nút vào đầu danh sách là $O(1)$, không phụ thuộc độ dài danh sách.

b) Thêm nút vào cuối danh sách

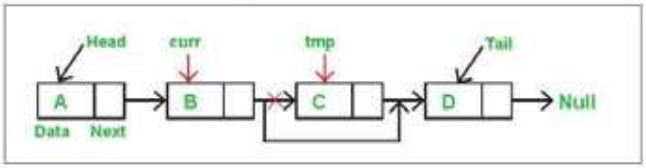
Nối thêm nút mới vào cuối danh sách, nó trở thành nút cuối cùng. Con trỏ *Tail* trỏ đến nút cuối cùng của danh sách (*Hình 1*). Thao tác như minh hoạ trong *Hình 2b*. Chú ý rằng phải sửa lại, cho *Tail* trỏ vào *E*. Thời gian thực hiện phép thêm nút vào cuối danh sách là $O(1)$, không phụ thuộc độ dài danh sách.

c) Thêm nút vào giữa danh sách

Tình huống minh hoạ: curr → *B*. Thêm nút vào sau nút *B*. Thao tác như minh hoạ trong *Hình 2c*. Thời gian thực hiện phép thêm nút vào giữa danh sách là $O(1)$, không phụ thuộc độ dài danh sách.

• Gỡ bỏ nút:

Tình huống minh hoạ: curr → *B*. Gỡ bỏ nút sau nút *B* (*Hình 3*).



Hình 3. Gỡ bỏ nút sau nút B

– Ghi lưu con trỏ để truy cập nút C: $tmp = B.Next$, tức là $tmp \rightarrow C$.

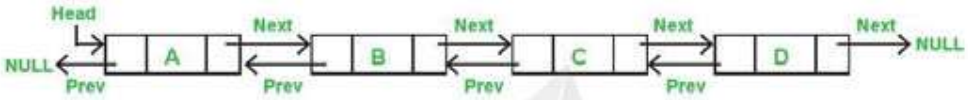
– Cho $B.Next$ trỏ đến nút đứng sau C (là nút D): $B.Next = C.Next$.

– Sử dụng tmp để giải phóng phần bộ nhớ dành cho C.

Thao tác gỡ bỏ nút đầu danh sách hay cuối danh sách chỉ khác chút ít. Thời gian thực hiện gỡ bỏ là $O(1)$, không phụ thuộc vào độ dài danh sách.

Danh sách liên kết kép

Cấu trúc danh sách liên kết trình bày trên chỉ có một con trỏ *Next* trỏ đến nút đứng kề ngay sau. Nếu mỗi nút có thêm một con trỏ nữa là *Prev* trỏ đến nút đứng kề ngay trước thì ta sẽ có danh sách nối kép (*Hình 4*).



Hình 4. Mô hình một danh sách nối kép có 4 nút

Thời gian thực hiện các phép toán của danh sách liên kết

– *Phép tìm kiếm*: Tìm nút chứa dữ liệu X ($Data = X$) để xử lí. Phải thực hiện tìm kiếm tuần tự từ đầu danh sách. Độ phức tạp của phép tìm kiếm là $O(n)$ với n là số nút của danh sách.

– Các thao tác thêm nút, gỡ bỏ nút của danh sách liên kết dù ở bất cứ vị trí nào thì thời gian thực hiện đều là $O(1)$. Đây là điểm ưu việt hơn danh sách mảng.

– Danh sách liên kết tốn thêm chỗ để lưu trữ thành phần *Next*. Đây là nhược điểm so với danh sách mảng.

2 Một số kiểu danh sách đặc biệt và ứng dụng của danh sách liên kết

Dịch vụ cung cấp bài hát trực tuyến thường đưa ra một danh sách N bài hát đứng đầu một tuần, một tháng,... Sau đó, danh sách này được sử dụng theo nhiều cách khác nhau: phát lại theo trình tự ngẫu nhiên, phát lại từ bài $N - 1$ đến 0 hay ngược lại,... Em hãy cho biết:

1) Kiểu danh sách này có những đặc điểm gì?

2) Có nên dùng cấu trúc danh sách liên kết để thực hiện kiểu danh sách này hay không?

Sử dụng cấu trúc móc nối để liên kết các nút thành một dãy tuần tự tạo ra kiểu danh sách rất linh hoạt. Danh sách liên kết phát huy ưu điểm trong những trường hợp thường xuyên phải:

– Thêm phần tử, gỡ bỏ phần tử ở bất cứ vị trí nào trong danh sách;

– Độ dài danh sách thay đổi nhanh và nhiều trong quá trình sử dụng.

Một số ví dụ ứng dụng danh sách liên kết

Danh sách nhóm đứng đầu *top N* được cập nhật bằng các thao tác: gỡ bỏ một số phần tử tại các vị trí bất kì nào đó; chèn thêm phần tử vào vị trí bất kì nào đó. Độ dài *N* của danh sách cũng có các lựa chọn khác nhau, ví dụ là 5, 10, 20,...

Một số bài toán thực tế cần mô hình hoá một mạng lưới (điện, đường giao thông,...) hay một cấu trúc phân cấp hình cây (tổ chức hành chính, cây gia phả,...) thì không thể dùng một danh sách. Việc thể hiện mối liên kết giữa các nút (điểm giao cắt) phải cho phép cập nhật các thay đổi một cách linh hoạt. Danh sách liên kết sẽ được sử dụng trong các trường hợp này.

Dựa trên hình minh hoạ, mô tả các bước thực hiện các phép toán sau của danh sách liên kết để minh hoạ chúng đều có thời gian là $O(1)$.

a) Thêm nút vào cuối danh sánh, thêm nút vào giữa danh sách.

b) Gỡ bỏ nút ở cuối danh sánh, ở đầu danh sách.

Phân tích yêu cầu ứng dụng của một danh sách nhóm đứng đầu *top N* và cho biết, nếu dùng kiểu danh sách của Python để thực hiện thì:

a) Những thao tác cần làm với danh sách *top N* sẽ thực hiện qua các phép toán danh sách Python như thế nào?

b) Kể tên một vài phép toán danh sách của Python không cần dùng đến cho trường hợp này.

Câu 1. Hãy nêu các phép toán danh sách liên kết có thời gian thực hiện $O(1)$.

Câu 2. Hãy nêu các phép toán danh sách liên kết có thời gian thực hiện $O(n)$.

Câu 3. Nếu muốn truy cập nút chứa dữ liệu X thì phải làm gì? Ước lượng thời gian thực hiện.

Tóm tắt bài học

- Danh sách liên kết là một chuỗi nhiều nút, không đánh chỉ số tuần tự các nút, lưu trữ rải rác không liền kề trong bộ nhớ; các nút móc nối với nhau để có thể duyệt theo chiều tiến hoặc cả hai chiều tiến và lùi.
- Danh sách liên kết khắc phục nhược điểm của danh sách mảng: không bị giới hạn độ dài; thời gian thực hiện các phép thêm nút, gỡ bỏ nút là O(1).
- Danh sách liên kết được dùng để mô hình hoá một mạng lưới (đồ thị) hay một cây phân cấp.

BÀI TÌM HIỂU THÊM

DANH SÁCH LIÊN KẾT TRONG PYTHON

Có thể lập trình Python thực hiện cấu trúc danh sách liên kết theo mô hình chung đã trình bày trên. Tuy nhiên, việc này đòi hỏi kiến thức cơ bản phương pháp hướng đối tượng và lập trình hướng đối tượng, một chủ đề rất quan trọng của Tin học nhưng chưa có trong nội dung chương trình bậc phổ thông.

Kiểu danh sách của Python rất tổng quát, có đủ các phép toán danh sách, phép thêm phần tử, gỡ bỏ phần tử có thể thực hiện tại bất cứ vị trí nào. Kiểu danh sách của Python đủ đáp ứng các yêu cầu ứng dụng nếu như không xem xét đến hiệu quả thời gian.

Python có một số kiểu dữ liệu làm sẵn có thể dùng để thực hiện những kiểu danh sách đặc thù, phù hợp hơn dùng kiểu danh sách tổng quát.

1) Thư viện **`collections`** có định nghĩa sẵn kiểu dữ liệu hàng đợi hai đầu **`deque`** (Doubly Ended Queue) là một kiểu danh sách được tối ưu hoá cho trường hợp thường xuyên phải lấy ra hay thêm vào một phần tử ở vị trí cuối hay ở đầu danh sách. Tên gọi phép thêm vào cuối là **`append`**, phép thêm vào đầu là **`appendleft`**, các phép lấy ra là **`pop`** và **`popleft`**. Các phép thêm vào và lấy ra đều có độ phức tạp thời gian $O(1)$ trong khi với kiểu **`list`** thì độ phức tạp là $O(n)$.

Để biểu diễn một hàng đợi thông thường (**`queue`**) trong thực tế, chỉ cần dùng **`append`** và **`popleft`**.

2) Trong một số ngôn ngữ lập trình khác, để biểu diễn dữ liệu thực tế không phải là dãy tuần tự, ví dụ như: cây gia phả, bản đồ giao thông,... người lập trình sẽ phải tự định nghĩa một kiểu dữ liệu dựa theo cấu trúc danh sách liên kết và sử dụng kiểu dữ liệu này trong văn bản chương trình.

Trong Python, có thể dễ dàng thể hiện một cây phân cấp hay đồ thị bằng cách sử dụng kiểu từ điển mà Python đã định nghĩa sẵn.

3) Có các gói thư viện bên ngoài cho Python đã làm sẵn danh sách liên kết, người lập trình chỉ cần tìm hiểu để sử dụng khi cần thiết.