Bài 4. Cấu trúc dữ liệu biểu diễn danh sách

4.1. Khái niệm danh sách

- Danh sách là một tập có thứ tự các phần tử có cùng một kiểu dữ liệu.
- Một số thao tác cơ bản đối với danh sách:
  - Chèn một phần tử vào danh sách,
  - Xoá một phần tử khỏi danh sách,
  - Sắp xếp lại các phần tử trong danh sách theo một trật tự nào đó,
  - Tìm một phần tử trong danh sách.

Khái niệm danh sách

- Ngoài ra còn có các thao tác:
  - Ghép hai hoặc nhiều d/s
  - Tách một d/s thành nhiều d/s
  - Sao chép một d/s
  - Cập nhật d/s
  - …

Khái niệm danh sách

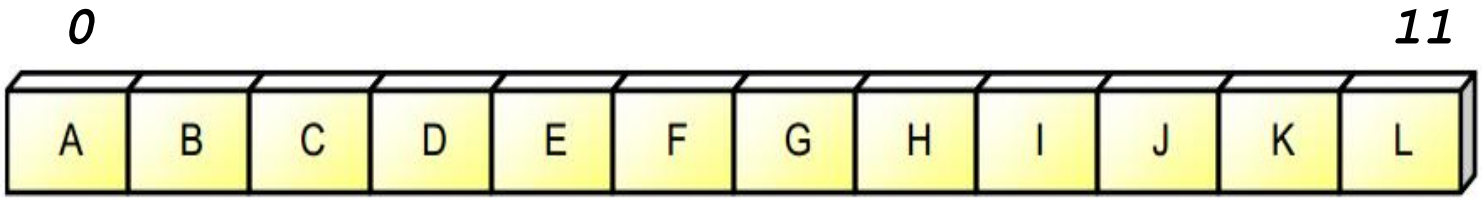
- Tệp (file) cũng là 1 loại d/s có kích thước lớn được lưu trữ ở bộ nhớ ngoài (đĩa):
  - Phần tử của tệp là bản ghi (record) gồm nhiều trường dữ liệu
- Bộ nhớ ngoài có những đặc điểm riêng → xử lý tệp cần có những kỹ thuật riêng.
- Ở đây ta chỉ xét đến biểu diễn d/s trong bộ nhớ trong.

4.2. Biểu diễn danh sách trong máy tính

- Để mô hình hóa cấu trúc dữ liệu danh sách vào các bài toán tin học, ta phải có cách biểu diễn danh sách bằng ngôn ngữ lập trình
- Việc biểu diễn này bao gồm:
  - Tìm cấu trúc dữ liệu cụ thể trong ngôn ngữ lập trình để lưu các phần tử của danh sách
  - Viết các đoạn mã mô tả các thao tác cần thiết với danh sách

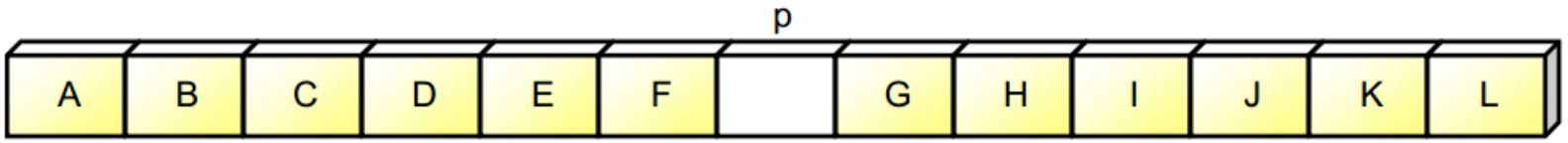
4.2.1. Cài đặt bằng mảng 1 chiều

- Đây là cách đơn giản nhất
- Danh sách có *n* phần tử được lưu trên một mảng, từ ô nhớ có chỉ số ***0*** đến ô nhớ có chỉ số ***n-1***
- Ví dụ: n = 12

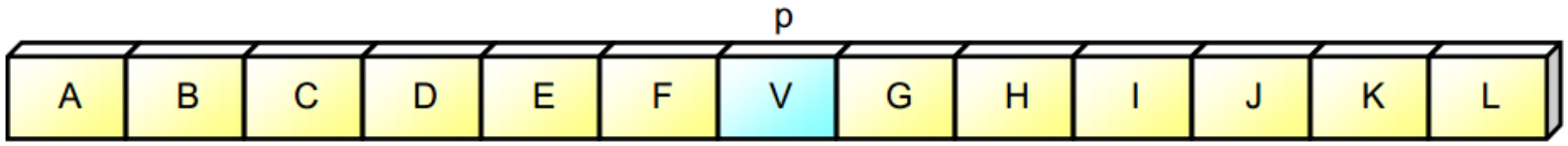


4.2.1. Cài đặt bằng mảng 1 chiều

- **Chèn** một phần tử có giá trị ***V*** vào mảng tại vị trí ***p***:
  - Dồn tất cả các phần tử từ vị trí ***p*** tới tới vị trí ***n*** về sau một vị trí



  - Đặt giá trị ***V*** vào vị trí ***p***



  - Tăng ***n*** lên 1

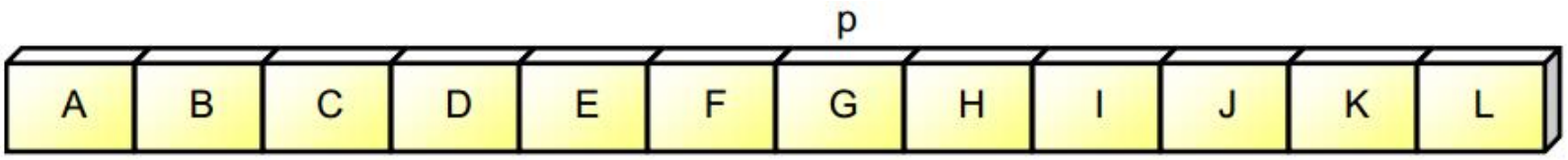
4.2.1. Cài đặt bằng mảng 1 chiều

- Hàm chèn giá trị V vào vị trí p:

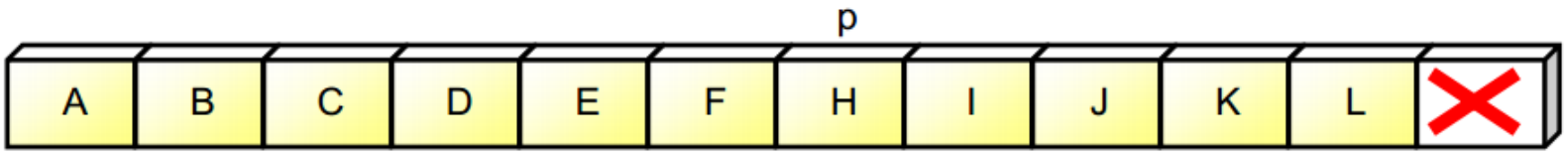
```
int n;
int *list;

void Insert(int p, int V) {
     int i = n;
     while (i  > p) {
          list[i] = list[i-1];
          i --;
     }
     list[p] = V;
     n++;
}
```

4.2.1. Cài đặt bằng mảng 1 chiều

- **Xóa** một phần tại vị trí ***p***:
  - Dồn tất cả các phần tử từ vị trí ***p+1*** tới tới vị trí ***n*** về trước một vị trí



  - Giảm ***n*** đi 1



4.2.1. Cài đặt bằng mảng 1 chiều

- Hàm xóa giá trị tại vị trí p:

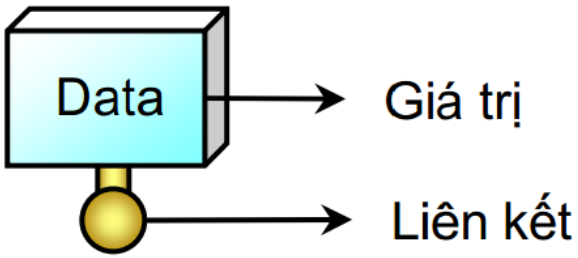
```
int n;
int *list;

void Remove(int p) {
     int i = p;
     while (i < n-1) {
          list[i] = list[i+1];
     }
     n--;
}
```

4.2.1. Nhận xét

- Với danh sách biểu diễn bằng mảng 1 chiều:
  - Toàn bộ mảng phải được cấp phát một khối bộ nhớ liên tục một lần cho tất cả phần tử
  - Việc lấy phần tử thứ ***i*** của danh sách (list[i]) là trực tiếp (truy cập ngẫu nhiên) - **Θ(1)**.
  - Đối với thao tác chèn và xóa phần tử:
    - Tốt nhất: khi phần tử nằm cuối danh sách - Θ(1)
    - Tồi nhất: khi phần tử nằm đầu danh sách - Θ(n)
    - Trung bình: khi phần tử nằm vị trí bất kỳ - **Θ(n)**

4.2.2. Cài đặt bằng danh sách nối đơn

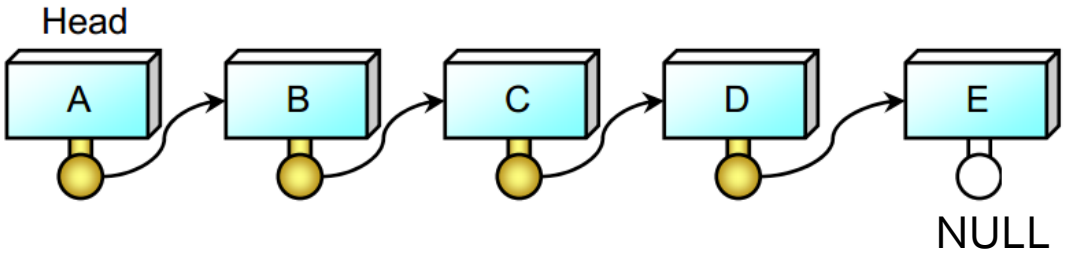
- Danh sách nối đơn gồm các nút được nối với nhau theo một chiều. Mỗi nút là một bản ghi (record) gồm hai phần:
  - Phần ***data*** - chứa giá trị của nút đó
  - Phần ***next*** chứa liên kết (con trỏ) tới nút kế tiếp, đối với nút cuối cùng, trường next này được gán một giá trị đặc biệt (NULL)




```
typedef struct node
{
        int data;
        TNode *next;
} TNode;
```

4.2.2. Cài đặt bằng danh sách nối đơn

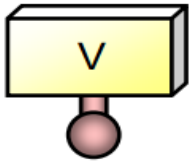
- Nút đầu tiên trong danh sách được gọi là **chốt** của danh sách nối đơn (**Head**).
- Để duyệt danh sách nối đơn, ta bắt đầu từ **chốt**, dựa vào trường liên kết để đi sang nút kế tiếp, đến khi gặp giá trị đặc biệt (duyệt qua nút cuối) thì dừng lại



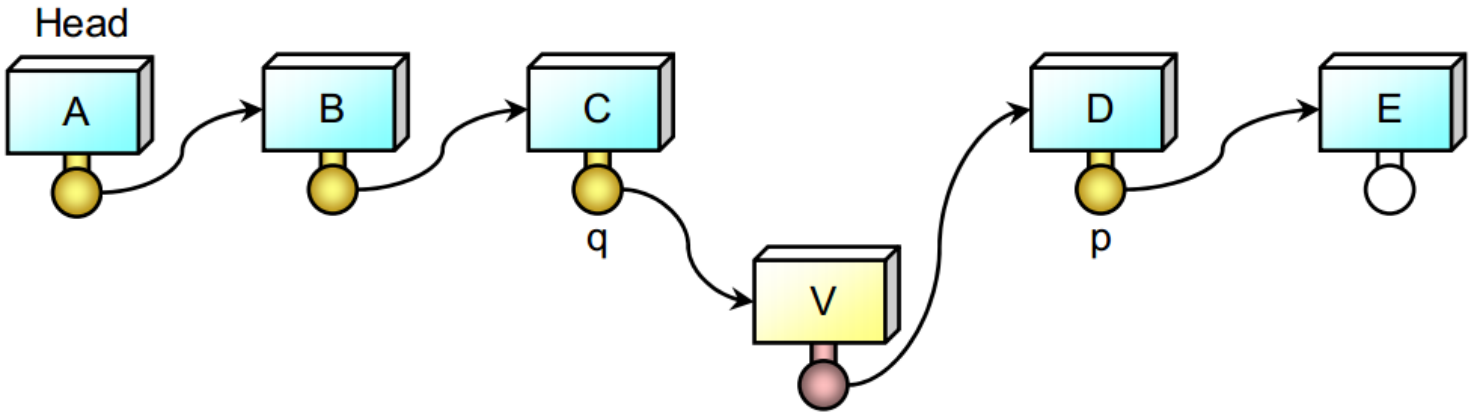

```
TNode * tmp = head;
while(tmp != NULL)
{
   print(tmp->data);
   tmp = tmp->next;
}
```

4.2.2. Cài đặt bằng danh sách nối đơn

- **Chèn** một phần tử có giá trị ***V*** vào danh sách nối đơn tại vị trí của nút ***p***:
  - Tạo ra nút mới ***newNode*** chứa giá trị ***V***



  - Tìm nút ***q*** là nút đứng trước nút ***p*** trong danh sách
  - Chỉnh lại liên kết: q trỏ tới newNode, newNode trỏ tới p



4.2.2. Cài đặt bằng danh sách nối đơn

- Chèn giá trị V danh sách:
  - Tạo nút mới chứa giá trị V

```
TNode *newNode =
(TNode*)malloc(sizeof(TNode));
newNode->data = V;
```

  - Chèn vào đầu danh sách:

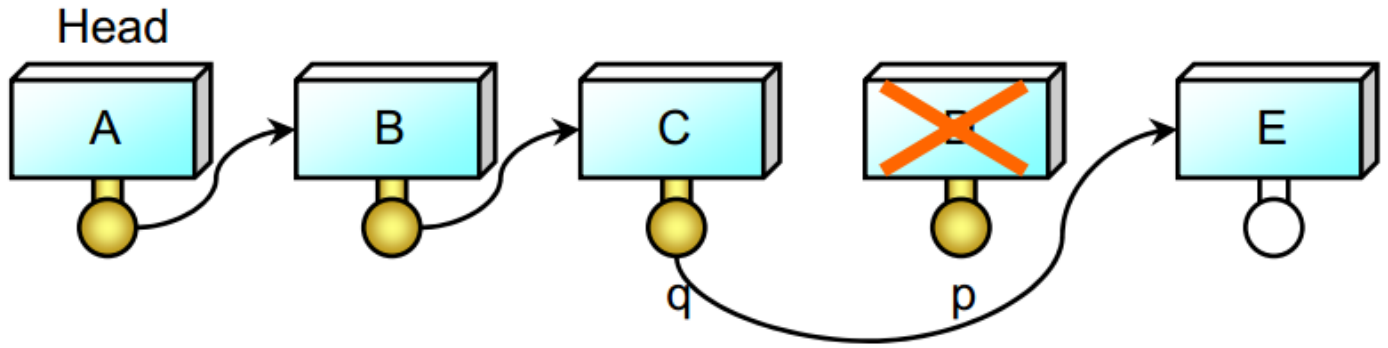
```
newNode->next=head;
head = newNode;
```

  - Chèn vào vị trí của node p

```
TNode * q = head;
while(q->next!=p) q = q->next;
newNode->next = p;
q->next = newNode;
```

4.2.2. Cài đặt bằng danh sách nối đơn

- **Xóa** một phần tử của danh sách nối đơn tại vị trí của nút ***p***:
  - Nếu p == head: đặt lại Head bằng nút đứng kế tiếp nó, thể giải phóng bộ nhớ cấp cho nút p (Head cũ)
  - Nếu p != head: tìm nút q đứng trước p, chỉnh lại liên kết: cho q trỏ đến nút sau của nút p và giải



4.2.2. Cài đặt bằng danh sách nối đơn

- Xóa nút p khỏi danh sách:
  - Xóa phần tử đầu danh sách:

```
TNode * tmp = head;
if (head!=NULL) {
   head = head->next;
   free(tmp);
}
```

  - Xóa phần tử p != head

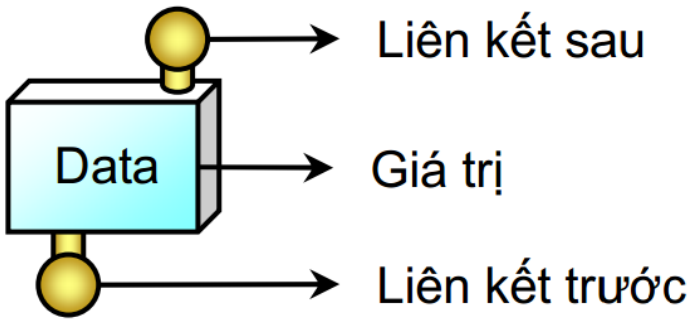
```
TNode * q = head;
while(q->next!=p) q = q->next;
q->next = p->next;
free(p);
```

4.2.2. Nhận xét

- Với danh sách biểu diễn bằng danh sách nối đơn:
  - Mỗi phần tử của mảng được cấp phát riêng biệt, có thể nằm trên những vùng nhớ khác nhau
  - Để đến phần tử thứ ***i*** của danh sách, buộc phải duyệt từ đầu danh sách đến vị trí *i*. Tốt nhất - Θ(1)**,** tồi nhất và trung bình - **Θ(n)**.
  - Cài đặt bằng danh sách liên kết đơn tốn bộ nhớ hơn so với trường hợp của mảng do cần thêm ô nhớ để chứa liên kết.
  - Danh sách liên kết đơn chỉ có thể duyệt theo một chiều từ đầu đến cuối dãy
  - Đối với thao tác chèn và xóa phần tử, độ phức tạp tính toán là **Θ(1)**. Tuy nhiên, trước khi chèn vào vị trí *p*, thì ta cần tìm nút q đứng trước nó, việc tìm nút q này có độ phức tạp - **Θ(n)**

4.2.3. Cài đặt bằng danh sách nối kép

- Danh sách nối kép gồm các nút được nối với nhau theo hai chiều. Mỗi nút là một bản ghi (record) gồm ba trường:
  - Trường ***data*** - chứa giá trị của nút đó
  - Trường ***next*** chứa liên kết tới nút kế tiếp, đối với nút cuối cùng, trường next = NULL
  - Trường ***prev*** chứa liên kết tới nút liền trước, đối với nút đầu tiên, trường prev = NULL

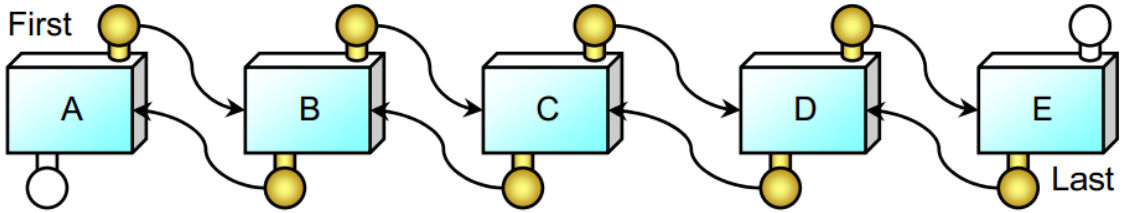



```
typedef struct node
{
        int data;
        TNode *next;
        TNode *prev;
} TNode;
```

4.2.3. Cài đặt bằng danh sách nối kép

- Danh sách nối kép có hai chốt: Nút đầu tiên (**first**) và nút cuối cùng (**last**).
- 2 cách duyệt danh sách nối kép:
  - Hoặc bắt đầu từ **first**, dựa vào liên kết **next** để đi sang nút kế tiếp, đến nút cuối (next == NULL) thì dừng
  - Hoặc bắt đầu từ **last**, dựa vào liên kết **prev** để đi sang nút liền trước, đến nút đầu tiên (prev == NULL) thì dừng

4.2.3. Cài đặt bằng danh sách nối kép




```
TNode * tmp = last;
while(tmp != NULL)
{
   printf(tmp->data);
   tmp = tmp->prev;
}
```

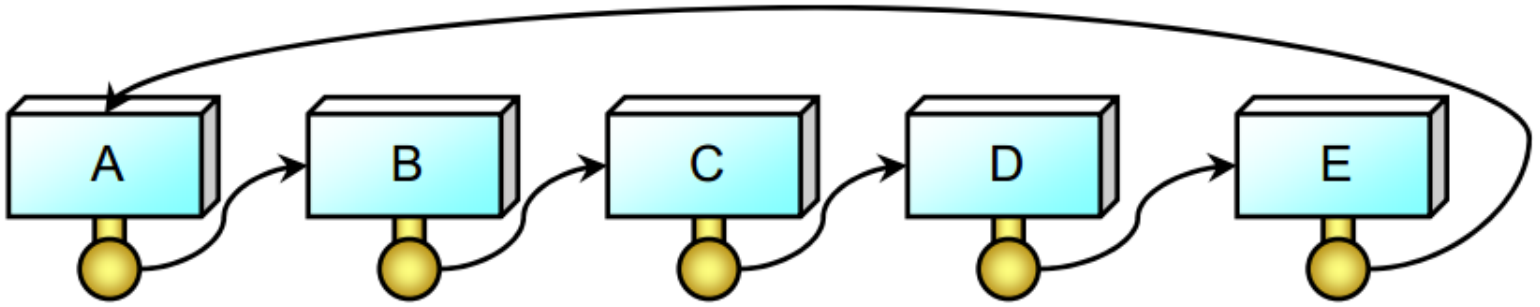
- Việc chèn / xoá vào danh sách nối kép cũng đơn giản chỉ là kỹ thuật chỉnh lại các mối liên kết giữa các nút cho hợp lý, ta coi như bài tập.

4.2.3. Danh sách nối kép – Nhận xét

- So với việc cài đặt bằng mảng, danh sách nối kép có ưu nhược điểm giống với danh sách nối đơn:
  - Truy cập tuần tự
  - Lấy phần tử theo chỉ số chậm - **Θ(n)**
  - Chèn/xóa nhanh - **Θ(1)**
- So với danh sách nối đơn:
  - Danh sách nối kép tốn bộ nhớ hơn (cho 2 liên kết trên mỗi nút)
  - Các thao tác cơ bản cần nhiều bước chỉnh liên kết hơn
  - Danh sách nối kép có thể duyệt 2 chiều → việc thêm bớt phần tử có thể lấy ngày phần tử trước/sau nó mà không cần duyệt từ đầu/cuối

4.2.4. Cài đặt bằng danh sách nối vòng một hướng

- Trong danh sách nối đơn nếu ta cho trường liên kết của phần tử cuối cùng trỏ thẳng về phần tử đầu tiên (thay vì chứa giá trị NULL) của danh sách thì ta sẽ được một danh sách nối vòng một hướng
- Với danh sách có một phần tử, trường liên kết trỏ đến chính nó

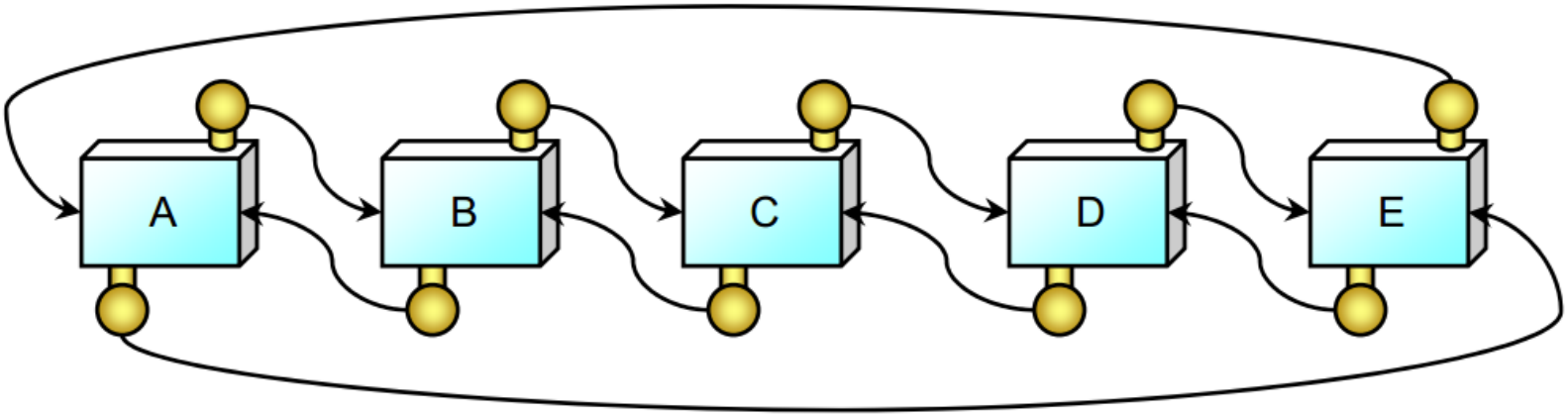


4.2.4. Cài đặt bằng danh sách nối vòng một hướng

- Ta chỉ cần biết một **nút bất kỳ** của danh sách là ta có thể duyệt được hết các nút trong danh sách bằng cách đi theo hướng của các liên kết.
  - Quá trình duyệt dừng lại khi ta gặp phải đúng nút bắt đầu!
- Khi chèn xoá vào danh sách nối vòng, ta không phải xử lý các trường hợp riêng khi chèn xoá tại vị trí của chốt
- Danh sách liên kết vòng thích hợp hơn cho các danh sách có tính chất xoay vòng (không có điểm đầu và điểm cuối)
  - Danh sách các đỉnh của một đa giác,
  - Mô tả cấu trúc hàng đợi (FIFO - first in first out)
- Ưu nhược điểm của danh sách nối vòng so với mảng cũng giống như trường hợp của danh sách nối đơn

4.2.5. Cài đặt bằng danh sách nối vòng hai hướng

- Danh sách nối vòng hai hướng có thể tạo thành từ danh sách nối kép nếu cho:
  - trường **prev** của nút *first* trỏ tới nút *last*
  - trường **next** của nút *last* thì trỏ về nút *first*



4.2.5. Cài đặt bằng danh sách nối vòng hai hướng

- Ta chỉ cần biết một **nút bất kỳ** của danh sách là ta có thể duyệt được hết các nút trong danh sách
- Với danh sách nối vòng hai hướng thì ta có thể duyệt các nút của danh sách cả hai chiều
- Khi chèn xoá vào danh sách nối vòng, ta không phải xử lý các trường hợp riêng khi chèn xoá tại vị trí của chốt
- Các thao tác trên danh sách cũng chỉ là việc chỉnh lại liên kết, coi như vài tập
- Ưu nhược điểm của danh sách nối vòng 2 hướng so với mảng cũng giống như trường hợp của danh sách nối đơn
- Ưu nhược điểm của danh sách nối vòng 2 hướng so với danh sách nối đơn cũng giống như trường hợp của danh sách nối kép

Kết luận

- Cấu trúc dữ liệu danh sách là cấu trúc đơn giản và được sử dụng rất rộng rãi trong các bài toán tin học cũng như trong các giải thuật khác
- Cần nắm vững các cách biểu diễn danh sách, ưu nhược điểm của từng phương pháp để lựa chọn cấu trúc phù hợp nhất với bài toán đặt ra
- Trong bài học tiếp theo ta sẽ xem xét hai cấu trúc dữ liệu quan trọng khác trong các bài toán tin học là ngăn xếp và hàng đợi (cũng được cài đặt dựa trên cấu trúc danh sách)

Trò chơi đếm và loại

- Trò chơi đếm và loại (**counting-out game**) hay bài toán **Josephus** (Josephus problem):
  - Có n người chơi ngồi quanh một vòng tròn, cùng chơi trò chơi
  - Một người nào đó đếm 1, người kế tiếp, theo chiều kim đồng hồ đếm 2… cứ như vậy cho tới người đếm đến số **S** cho trước thì phải ra khỏi vòng tròn, người kế tiếp lại đếm bắt đầu từ 1,
  - Cứ như vậy đến khi chỉ còn lại 1 người là người thắng cuộc.
  - Bài toán đặt ra là, với **n** và **S** cho trước, tìm vị trí bắt đầu đếm để người thứ **k** luôn thắng cuộc.
- Bài toán này có thể cài đặt một cách rất tự nhiên và hiệu quả bằng danh sách liên kết vòng