Danh sách liên kết (Linked Lists)

Bài giảng môn Cấu trúc dữ liệu và giải thuật

Khoa Công nghệ thông tin
Trường Đại học Thủy Lợi

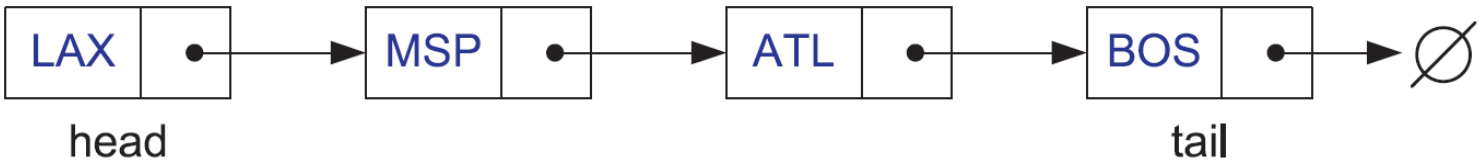
Nội dung

1. Danh sách liên kết
2. Danh sách liên kết đơn
3. Danh sách liên kết đôi
4. Danh sách liên kết vòng tròn

1. Danh sách liên kết

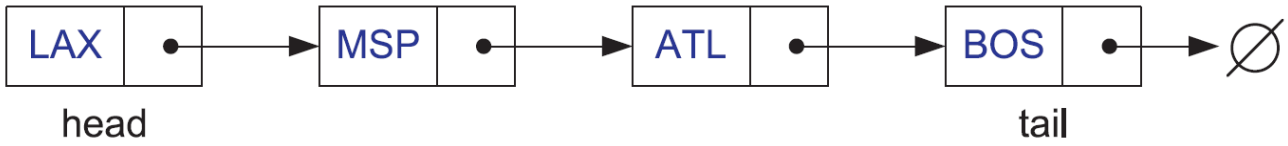
Danh sách liên kết

- Là một tập nút liên kết với nhau theo trật tự tuyến tính (có trước có sau).
- Mỗi nút chứa:
  - một phần tử;
  - một hoặc hai liên kết tới nút lân cận.
- Các nút nằm rải rác trong bộ nhớ máy tính (trong khi các phần tử của mảng và vector nằm liên tục).

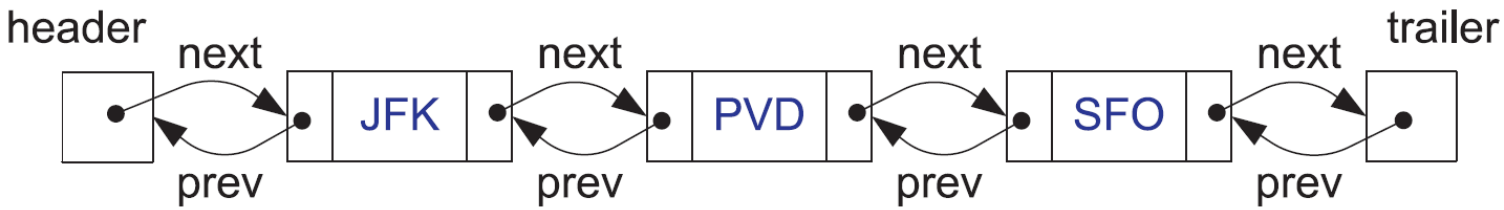


Các kiểu danh sách liên kết

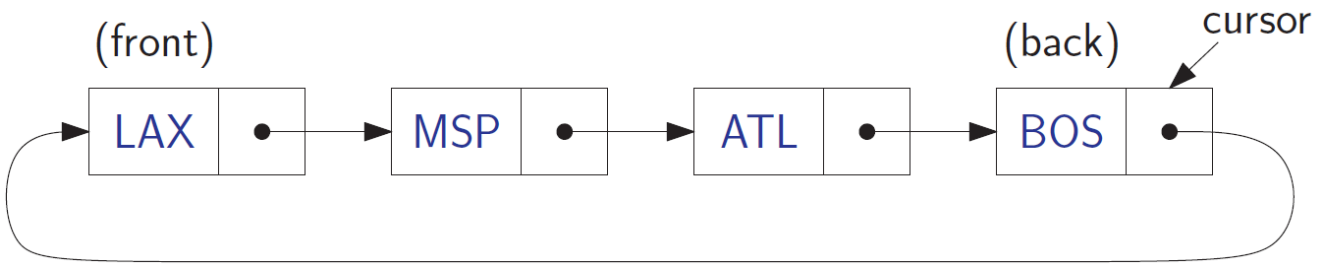
Danh sách liên kết đơn



Danh sách liên kết đôi

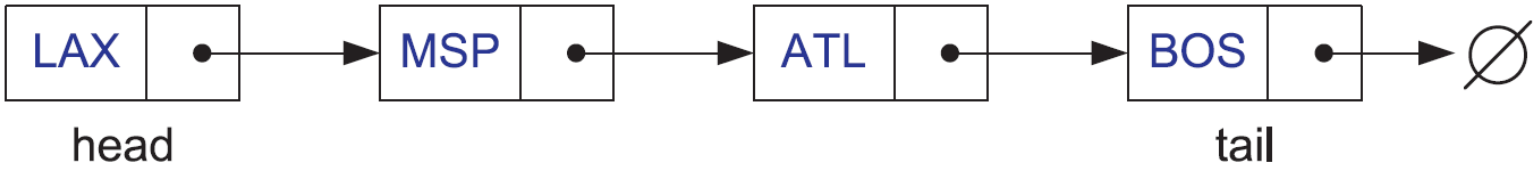


Danh sách liên kết vòng tròn



2. Danh sách liên kết đơn

Danh sách liên kết đơn



- Mỗi nút chỉ có một liên kết trỏ tới nút kế tiếp.
  - Riêng nút cuối cùng không có nút kế tiếp, vì vậy con trỏ của nó bằng `NULL`.
- Các thao tác chính:
  - Chèn phần tử mới vào đầu danh sách;
  - Xóa phần tử đầu danh sách;
  - Lấy phần tử đầu danh sách.

Cài đặt danh sách liên kết đơn

```
// Khai báo kiểu phần tử
typedef int T;

// Định nghĩa kiểu của các nút trong danh sách
struct Node {
    T elem;      // Phần tử
    Node * next; // Liên kết tới nút kế tiếp
};

// Định nghĩa cấu trúc danh sách liên kết đơn
struct List {
    Node * head; // Con trỏ tới nút đầu danh sách
};
```

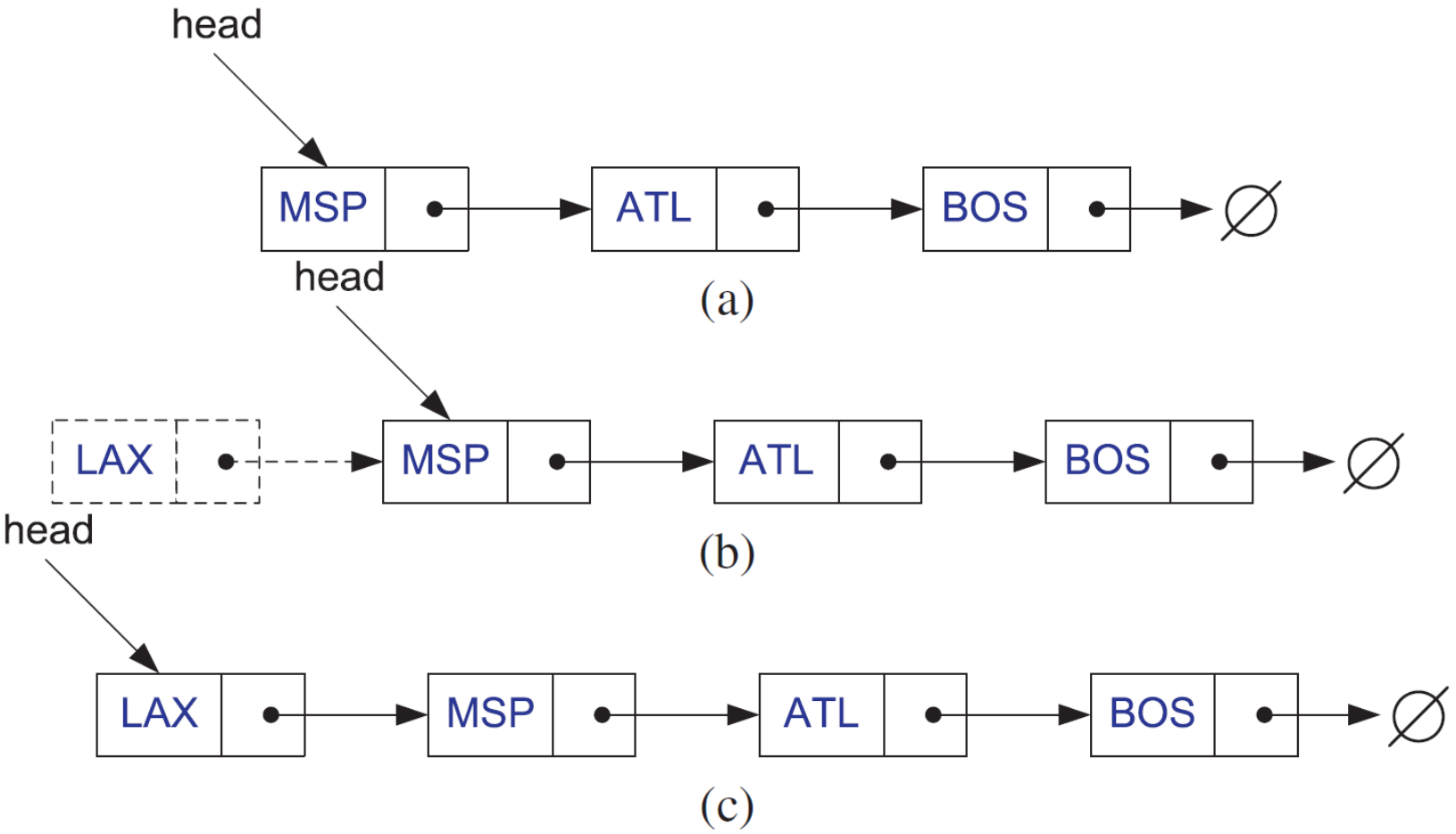
Hàm khởi tạo và hàm hủy

```
void listInit(List & list) {
  list.head = NULL; // Ban đầu danh sách rỗng
}

// Hàm hủy dùng hàm listIsEmpty để kiểm tra danh
// sách rỗng, dùng hàm listPopFront để xóa phần tử
// đầu danh sách (hai hàm đó ta sẽ lập trình sau).
void listDestroy(List & list) {
  while (!listIsEmpty(list))
    listPopFront(list); // Xóa phần tử đầu tiên cho
}                       // đến khi danh sách rỗng
                        // thì thôi.
```

Các hàm khác

```
// Kiểm tra danh sách có rỗng hay không.
bool listIsEmpty(List & list) {
  return (list.head == NULL);
}

// Lấy phần tử đầu danh sách (có kiểu là T).
T listFront(List & list) {
  return list.head->elem; // head trỏ tới nút đầu;
}                         // trong một nút có elem
                          // là phần tử.
```

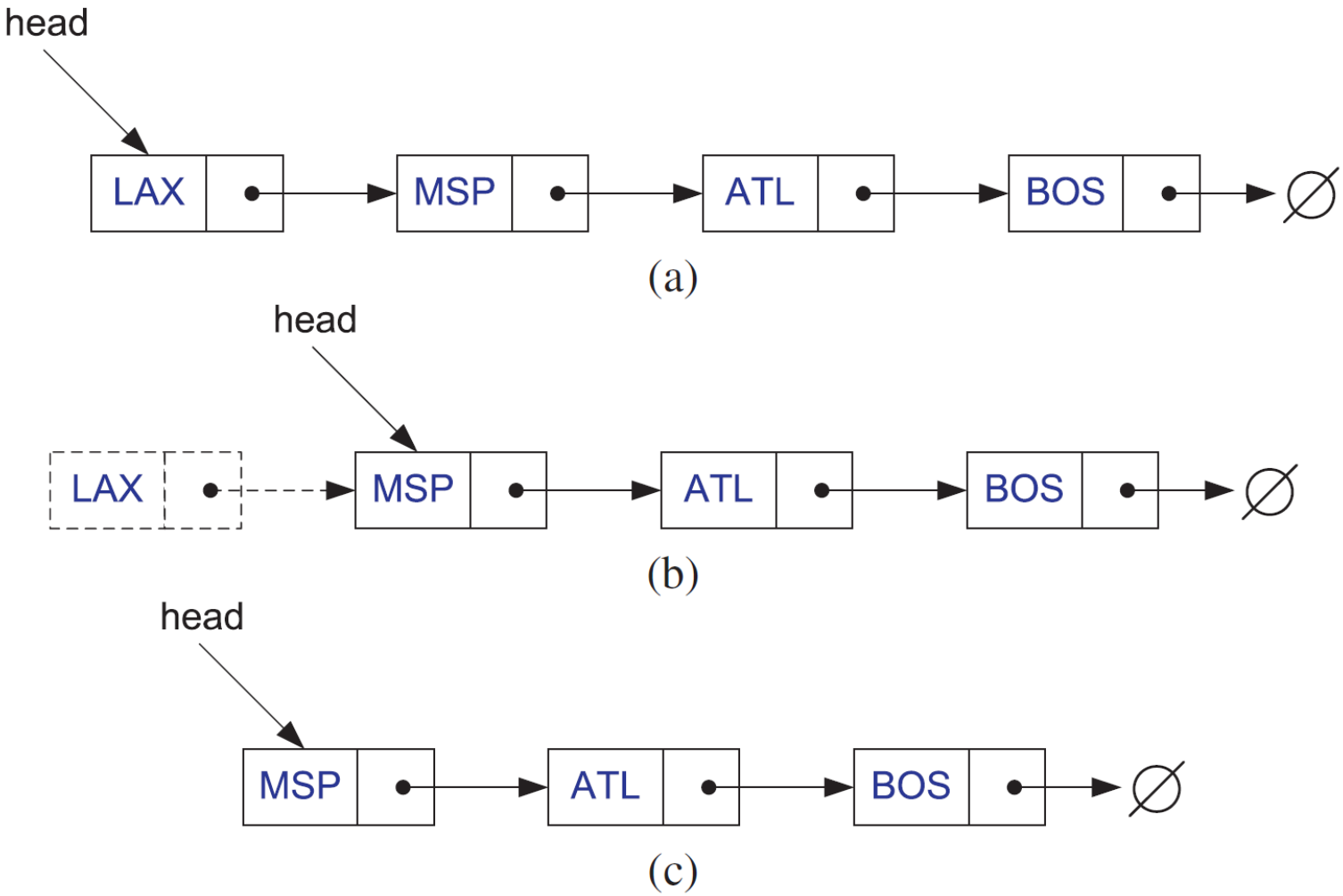
Chèn vào đầu danh sách



Chèn vào đầu danh sách (tiếp)

```
// e (element) là phần tử cần chèn.
void listPushFront(List & list, T e) {
    // Tạo nút mới
    Node * v = new Node;

    // Nút mới chứa phần tử cần chèn
    v->elem = e;

    // Nút mới trỏ tới nút đầu danh sách
    v->next = list.head;

    // Vì nút mới sẽ trở thành nút đầu danh sách,
    // phải cập nhật head cho trỏ tới nút mới.
    list.head = v;
}
```

Xóa phần tử đầu danh sách



Xóa phần tử đầu danh sách (tiếp)

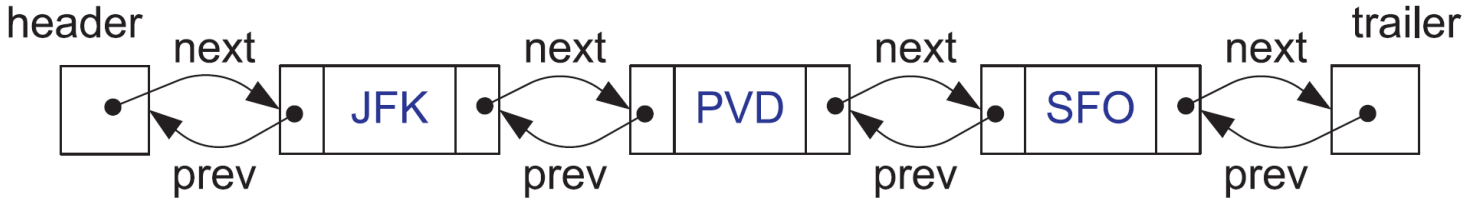
```
void listPopFront(List & list) {
    // Giữ lại địa chỉ của nút đầu danh sách
    // (sẽ cần địa chỉ này khi xóa).
    Node * old = list.head;

    // Vì nút thứ hai sẽ trở thành nút đầu, phải cập nhật
    // head cho trỏ tới nút thứ hai.
    list.head = list.head->next;

    // Xóa nút đầu cũ dùng địa chỉ đã giữ lại bên trên
    delete old;
}
```

Phân tích thời gian chạy

- Hàm khởi tạo: O(1)
- Hàm hủy: O(n) $\rightarrow$ vì phải xóa n phần tử/nút.
- Kiểm tra rỗng: O(1)
- Lấy phần tử đầu danh sách: O(1)
- Chèn/xóa ở đầu danh sách: O(1)

3. Danh sách liên kết đôi

Danh sách liên kết đôi



- Mỗi nút chứa một phần tử và hai liên kết:
  - một liên kết tới nút kế tiếp (next);
  - một liên kết về nút liền trước (previous).
- Các thao tác chính:
  - Chèn/xóa ở đầu, ở cuối hoặc ở vị trí hiện hành;
  - Lấy phần tử ở đầu, ở cuối hoặc ở vị trí hiện hành;
  - Duyệt danh sách tiến hoặc lùi.
- Chú ý: header và trailer là những nút đầu/cuối giả (không chứa phần tử), được dùng để thuận tiện cho việc lập trình.

Cài đặt danh sách liên kết đôi

Chú ý: Lớp `list` *trong thư viện chuẩn C++ thực thi danh sách liên kết đôi.*

```
// Khai báo kiểu phần tử
typedef int T;

// Định nghĩa kiểu của các nút trong danh sách
struct DNode {
    T elem;         // Phần tử
    DNode * next; // Liên kết về phía sau
    DNode * prev; // Liên kết về phía trước
};

// Định nghĩa cấu trúc danh sách liên kết đôi
struct DList {
    DNode * header;       // Con trỏ tới đầu danh sách (nút giả)
    DNode * trailer;      // Con trỏ tới cuối danh sách (nút giả)
    DNode * currentPos;   // Con trỏ tới nút hiện hành
};
```

Khai báo các hàm/thao tác

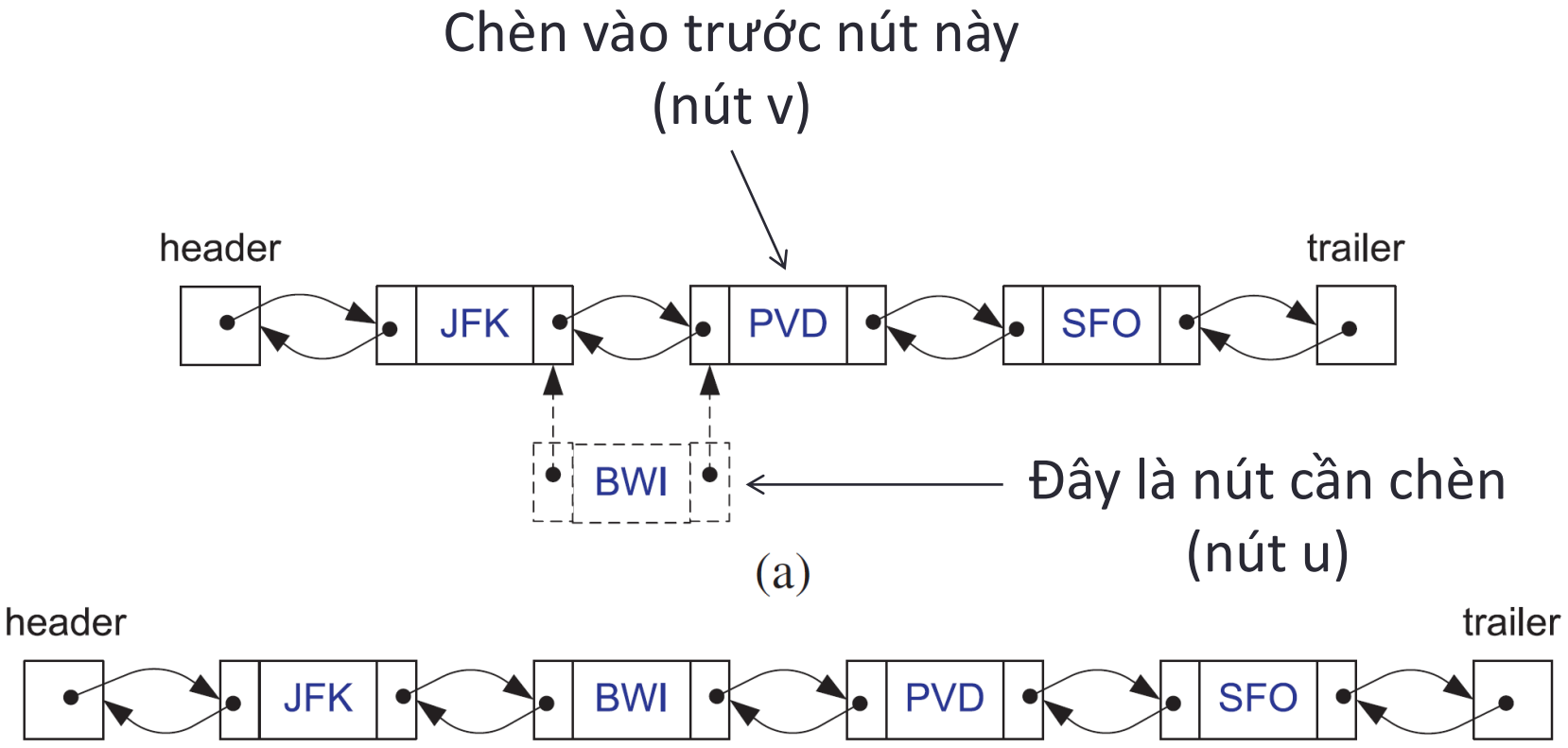
```
void dlistInit(DList & list);       // Hàm khởi tạo
void dlistDestroy(DList & list);    // Hàm hủy
bool dlistIsEmpty(DList & list);    // Kiểm tra rỗng
T dlistFront(DList & list);         // Lấy phần tử đầu danh sách
T dlistBack(DList & list);          // Lấy phần tử cuối danh sách
T dlistCurrent(DList & list);       // Lấy phần tử hiện hành
void dlistMoveNext(DList & list);   // Chuyển sang nút kế tiếp
void dlistMovePrev(DList & list);   // Chuyển về nút liền trước
void dlistMoveFront(DList & list); // Chuyển về đầu danh sách
void dlistMoveBack(DList & list);   // Chuyển về cuối danh sách
```

Khai báo các hàm/thao tác (tiếp)

```
void dlistPushFront(DList & list, T e); // Chèn vào đầu danh sách
void dlistPushBack(DList & list, T e);  // Chèn vào cuối danh sách
void dlistPopFront(DList & list); // Xóa phần tử đầu danh sách
void dlistPopBack(DList & list);  // Xóa phần tử cuối danh sách

// Chèn vào (ngay trước) vị trí hiện hành.
void dlistInsert(DList & list, T e);

// Xóa phần tử ở vị trí hiện hành.
void dlistRemove(DList & list);
```

Vì thời gian trên lớp có hạn, sau đây chúng ta chỉ tập trung cài đặt hai hàm này.

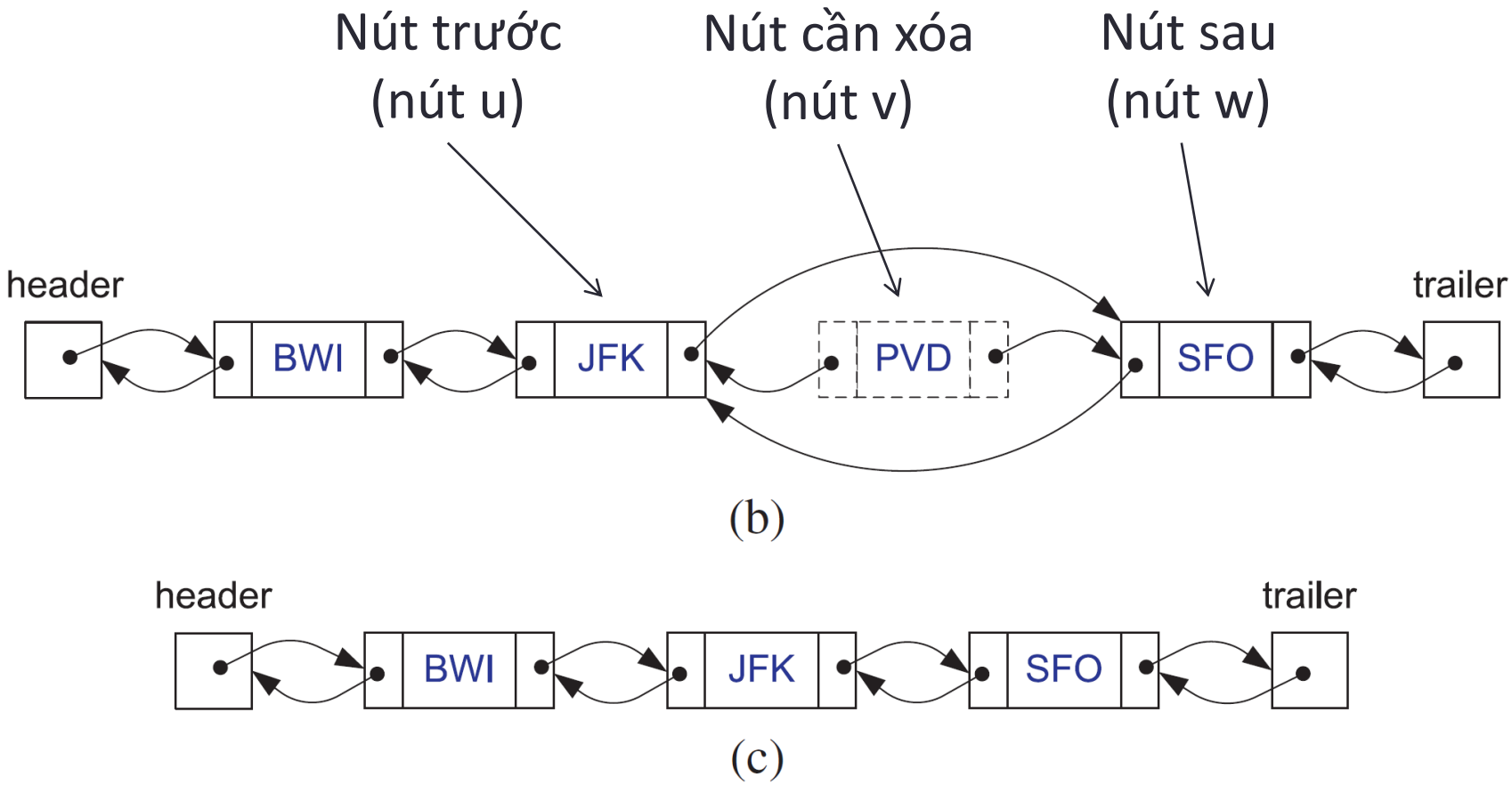
Chèn vào trước vị trí hiện hành



Chèn vào trước vị trí hiện hành (tiếp)

```
// Trước khi chèn: nút trước (v->prev) ↔ nút sau (v)
// (nút sau là nút hiện hành);
// e là phần tử mới cần chèn.
void dlistInsert(DList & list, T e) {
    DNode * v = list.currentPos; // Lấy nút hiện hành
    DNode * u = new DNode;       // Tạo nút mới

    u->elem = e;         // Nút mới chứa phần tử mới,
    u->next = v;         // trỏ tới nút sau và
    u->prev = v->prev; // trỏ về nút trước.

    v->prev->next = u; // Nút trước trỏ tới nút mới
    v->prev = u;         // Nút sau trỏ về nút mới
}
```

Xóa phần tử ở vị trí hiện hành

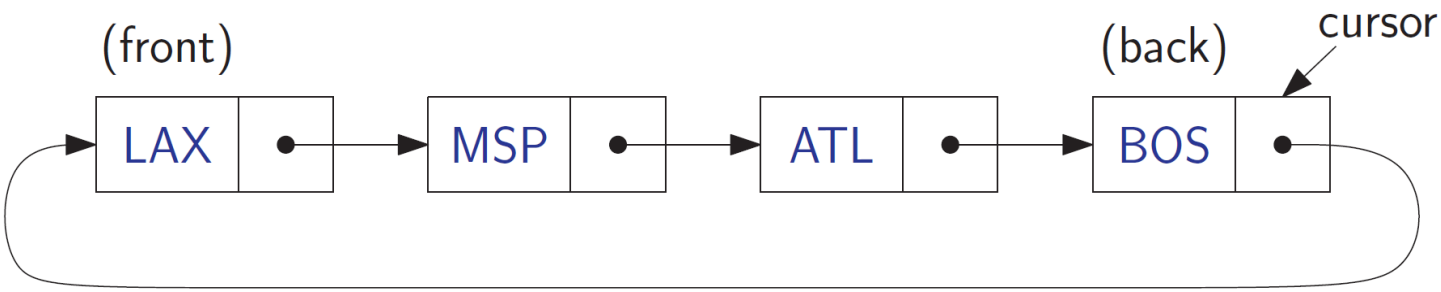


Xóa phần tử ở vị trí hiện hành (tiếp)

```
// nút trước (u) ↔ nút cần xóa (v) ↔ nút sau (w)
void dlistRemove(DList & list) {
   DNode * v = list.currentPos; // Nút hiện hành cần xóa
   DNode * u = v->prev;         // Nút trước
   DNode * w = v->next;         // Nút sau

   u->next = w; // Nút trước trỏ tới nút sau
   w->prev = u; // Nút sau trỏ về nút trước

   delete v;    // Xóa nút hiện hành cũ

   list.currentPos = w; // Sau khi xóa, nút sau trở
                        // thành nút hiện hành mới.
}
```

4. Danh sách liên kết vòng tròn

Danh sách liên kết vòng tròn



- Cấu trúc tương tự như danh sách liên kết đơn.
- Có thêm con trỏ đặc biệt **`cursor`** trỏ đến cuối danh sách (back); liên kết `next` của nút cuối trỏ vòng về đầu danh sách (front).
- Các thao tác chính:
  - Chèn và xóa ở sau `cursor` (đồng nghĩa với ở đầu danh sách);
  - Lấy phần tử ở đầu và cuối danh sách;
  - Dịch chuyển `cursor` sang vị trí tiếp theo.

Cài đặt danh sách liên kết vòng tròn

```
// Khai báo kiểu phần tử
typedef int T;

// Định nghĩa kiểu của các nút trong danh sách
struct CNode {
    T elem;        // Phần tử
    CNode * next; // Liên kết tới nút kế tiếp
};

// Định nghĩa cấu trúc danh sách liên kết vòng tròn
struct CList {
    CNode * cursor; // Trỏ tới nút cuối danh sách
};
```

Khai báo các hàm/thao tác

```
void clistInit(CList & list);    // Hàm khởi tạo
void clistDestroy(CList & list); // Hàm hủy
bool clistIsEmpty(CList & list); // Kiểm tra rỗng
T clistFront(CList & list); // Lấy phần tử đầu danh sách
T clistBack(CList & list);  // Lấy phần tử cuối danh sách
void clistMoveNext(CList & list); // Dịch chuyển cursor

// Chèn vào sau cursor
void clistInsert(CList & list, T e);

// Xóa nút sau cursor
void clistRemove(CList & list);
```

Vì thời gian trên lớp có hạn, sau đây chúng ta chỉ tập trung cài đặt hai hàm này.

Chèn vào sau cursor

```
// Chèn phần tử mới (e) vào sau cursor (đầu danh sách).
void clistInsert(CList & list, T e) {
  CNode * v = new CNode; // Tạo nút mới
  v->elem = e;           // Nút mới chứa phần tử mới

  if (list.cursor == NULL) { // Nếu danh sách rỗng,
    v->next = v;       // nút mới trỏ tới chính nó và
    list.cursor = v;   // cursor trỏ tới nút mới (vì
  }                    // nút mới cũng là nút cuối).
  else {                       // Nếu danh sách có phần tử,
    v->next = list.cursor->next; // nút mới trỏ tới nút đầu,
    list.cursor->next = v;       // rồi trở thành nút đầu.
  } // Chú ý: list.cursor->next giữ địa chỉ nút đầu.
}
```

Xóa nút sau cursor

```
void clistRemove(CList & list) {
    // Giữ lại địa chỉ của nút cần xóa (nút đầu)
    CNode * old = list.cursor->next;

    if (old == cursor) // Nếu danh sách chỉ có một nút
                       // (nút đầu và cuối trùng nhau),
        cursor = NULL; // danh sách sẽ rỗng sau khi xóa.
    else               // Nếu danh sách có nhiều nút,
        cursor->next = old->next; // nút thứ hai sẽ trở
                                  // thành nút đầu mới.

    delete old; // Xóa nút đầu cũ.
}
```

Bài tập

1. Vì sao hàm hủy của danh sách liên kết đơn tốn thời gian chạy O(n), trong khi của vector chỉ là O(1)?
2. Hãy đề xuất các thuật toán chèn/xóa ở cuối danh sách liên kết đơn? So sánh thời gian chạy với các thao tác chèn/xóa ở đầu danh sách? Trong trường hợp thuật toán đề xuất chạy chậm hơn, có cách nào làm cho nó chạy nhanh hơn không?
3. Hãy đề xuất một thuật toán truy nhập phần tử của danh sách liên kết đơn thông qua chỉ số. So sánh thời gian chạy của thuật toán này với thuật toán tương ứng của vector.

Bài tập

4. Vì sao chèn/xóa ở giữa danh sách liên kết (đơn, đôi hoặc vòng tròn) chạy nhanh hơn chèn/xóa ở giữa vector?
5. Nêu một ứng dụng của danh sách liên kết vòng tròn (*Gợi ý: chương trình nghe nhạc*). Phải điều chỉnh ứng dụng đó như thế nào nếu dùng danh sách liên kết đơn thay cho danh sách liên kết vòng tròn?