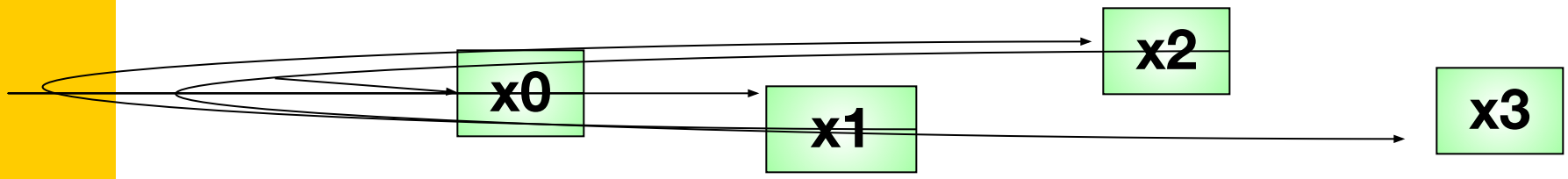
NỘI DUNG

DANH SÁCH LIÊN KẾT ĐƠN (LIST)

Tổ Chức Của DSLK Đơn



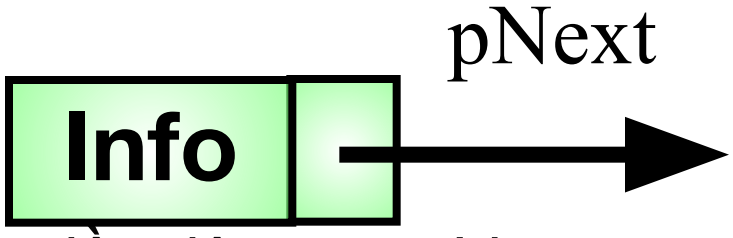
- Mỗi phần tử liên kết với phần tử đứng liền sau trong danh sách
- Mỗi phần tử trong danh sách liên kết đơn là một cấu trúc có hai thành phần
  - ***Thành phần dữ liệu***: Lưu trữ thông tin về bản thân phần tử
  - ***Thành phần liên kết***: Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.

CTDL của DSLK đơn

- Cấu trúc dữ liệu của 1 nút trong List đơn

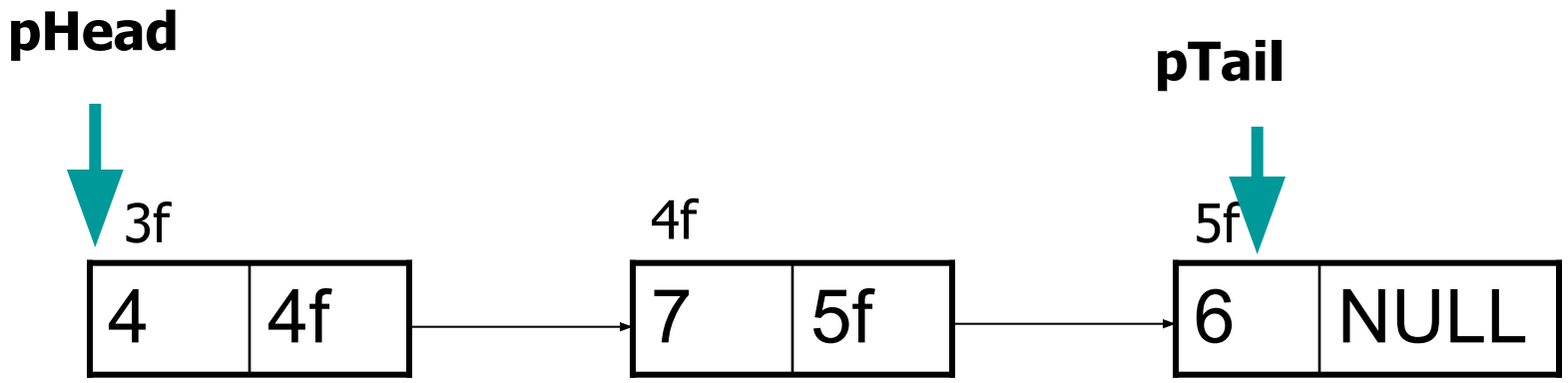
```
typedef  struct  tagNode
{ Data   Info;  // Lưu thông tin bản thân
  struct  tagNode  *pNext; //Lưu địa chỉ của Node đứng sau
}Node;
```

- Cấu trúc dữ liệu của DSLK đơn




```
typedef  struct  tagList
{ Node  *pHead;//Lưu địa chỉ Node đầu tiên trong List
  Node  *pTail; //Lưu địa chỉ của Node cuối cùng trong List
}LIST;        // kiểu danh sách liên kết đơn
```

Ví dụ tổ chức DSLK đơn trong bộ nhớ



Trong ví dụ trên thành phần dữ liệu là 1 số nguyên

Các thao tác cơ bản trên DSLK đơn

- Tạo 1 danh sách liên kết đơn rỗng
- Tạo 1 nút có trường Infor bằng x
- Tìm một phần tử có Info bằng x
- Thêm một phần tử có khóa x vào danh sách
- Hủy một phần tử trong danh sách
- Duyệt danh sách
- Sắp xếp danh sách liên kết đơn

Khởi tạo danh sách liên kết

- Địa chỉ của nút đầu tiên, địa chỉ của nút cuối cùng đều không có

```
void CreateList(List &l)
{
 l.pHead=NULL;
 l.pTail=NULL;
}
```


Tạo 1 phần tử mới

- Hàm trả về địa chỉ phần tử mới tạo

```
Node*    CreateNode(Data x)
{  Node *p;
   p = new Node;//Cấp phát vùng nhớ cho phần tử
   if ( p==NULL)  exit(1);
   p ->Info = x; //gán dữa liệu cho nút
   p->pNext = NULL;
   return p;
}
```


Thêm 1 phần tử vào DSLK

- **Nguyên tắc thêm:** Khi thêm 1 phần tử vào List thì có làm cho pHead, pTail thay đổi?
- **Các vị trí cần thêm 1 phần tử vào List:**
  - Thêm vào đầu List đơn
  - Thêm vào cuối List
  - Thêm vào sau 1 phần tử q trong list

Thuật toán thêm 1 phần tử vào đầu DSLK

- Thêm nút p vào đầu danh sách liên kết đơn

Bắt đầu:

Nếu List rỗng thì

+ pHead = p;

+ pTail = pHead;

Ngược lại

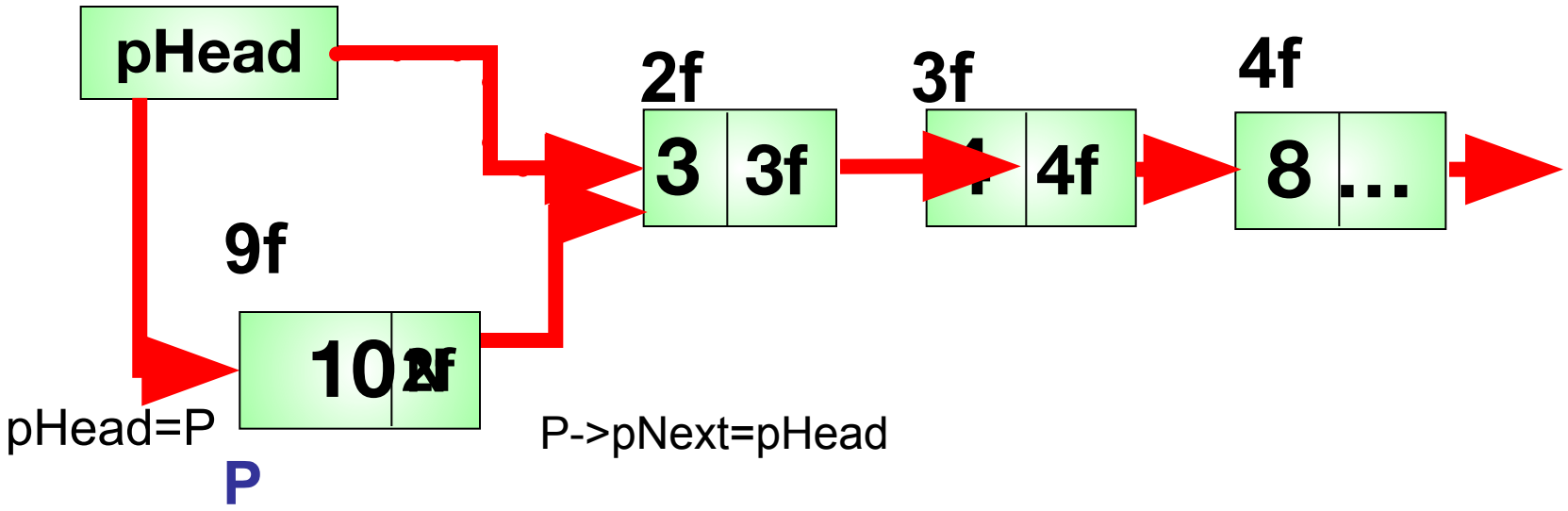
+ p->pNext = pHead;

+ pHead = p

Hàm thêm 1 phần tử vào đầu List

```
void AddHead(LIST &l, Node* p)
{
    if (l.pHead==NULL)
    {
        l.pHead = p;
        l.pTail = l.pHead;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```


Minh họa thuật toán thêm vào đầu



Thuật toán thêm vào cuối DSLK

- Ta cần thêm nút p vào cuối list đơn

Bắt đầu:

Nếu List rỗng thì

+ pHead = p;

+ pTail = pHead;

Ngược lại

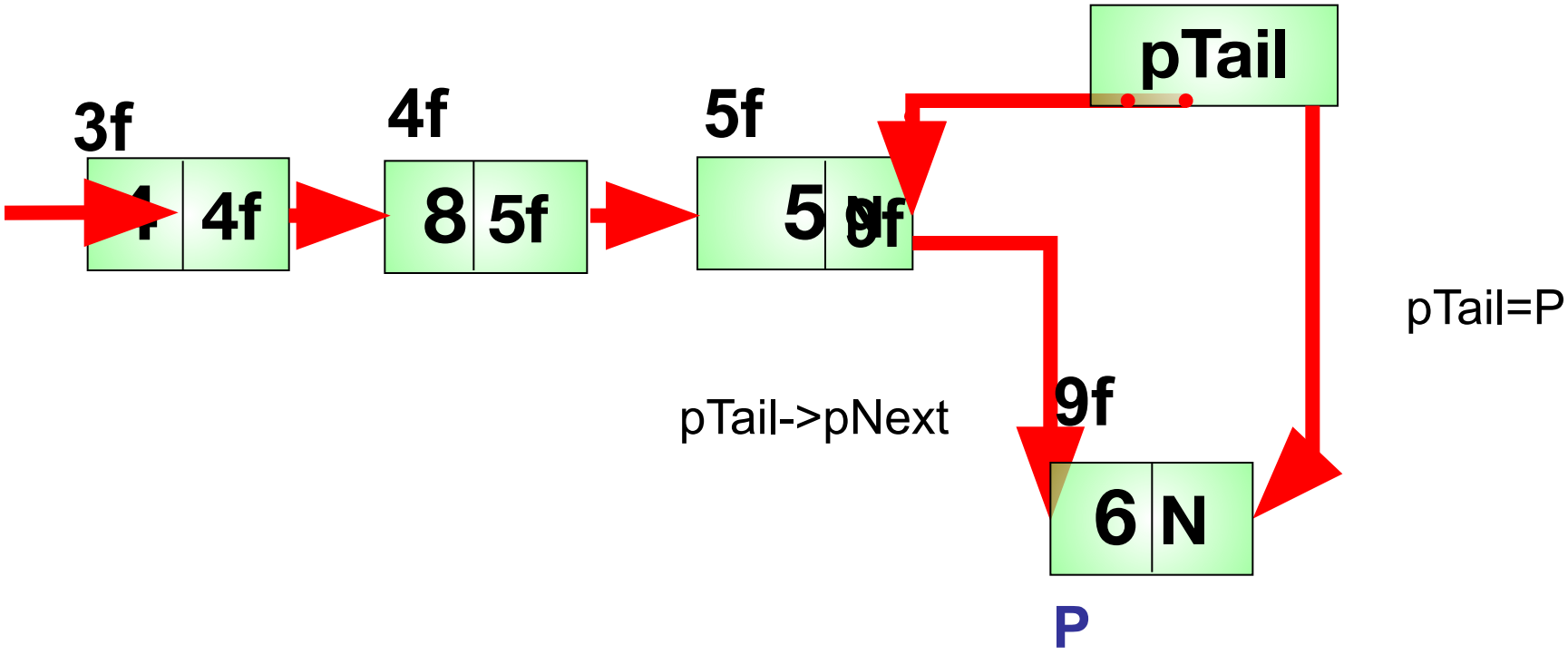
+ pTail->pNext=p;

+ pTail=p

Hàm thêm 1 phần tử vào cuối DSLKD

```
void AddTail(LIST &l, Node *p)
{
 if (l.pHead==NULL)
 {
     l.pHead = p;
     l.pTail = l.pHead;
 }
 else
 {
     l.pTail->Next = p;
     l.pTail = p;
 }
}
```


Minh họa thuật toán thêm vào cuối



Thuật toán phần tử q vào sau phần tử q

- Ta cần thêm nút p vào sau nút q trong list đơn

Bắt đầu:

Nếu (q!=NULL) thì

B1: p->pNext = q->pNext

B2:

+ q->pNext = p

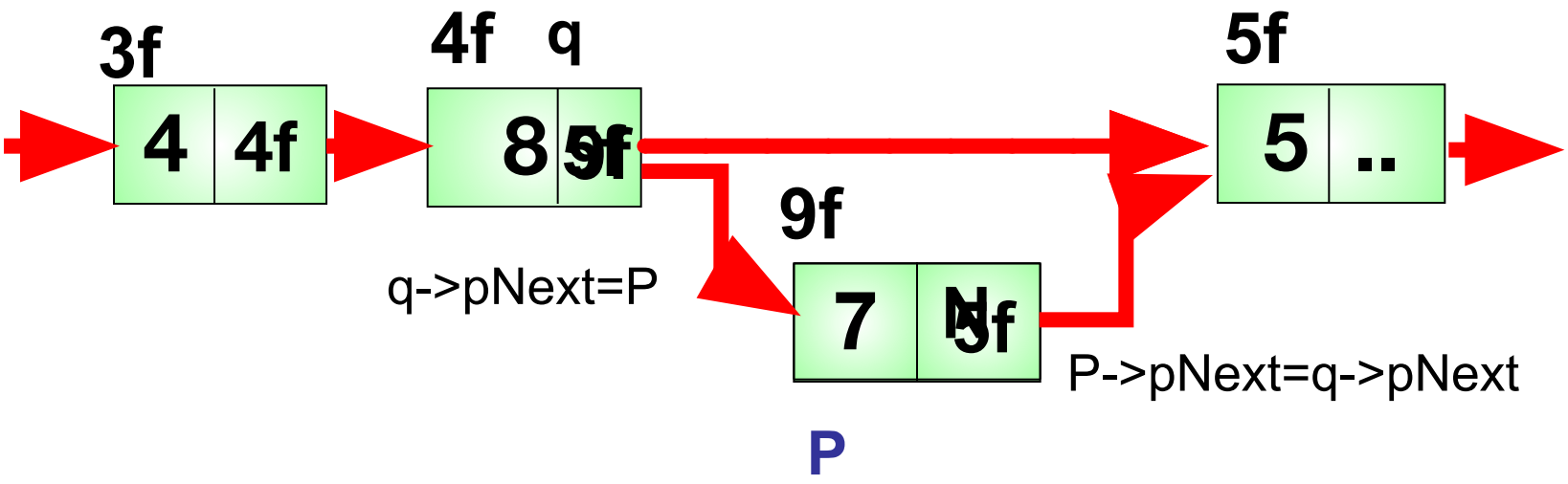
+ nếu q = pTail thì

pTail=p

Cài đặt thuật toán

```
void InsertAfterQ(List &l, Node *p, Node *q)
{
    if(q!=NULL)
    {
   p->pNext=Q->Next;
   q->pNext=p;
   if(l.pTail==q)
        l.Tail=q;
   }
   else
   AddHead(l,q);// thêm q vào đầu list
}
```


Minh họa thuật toán



Hủy phần tử trong DSLK đơn

- **Nguyên tắc**: Phải cô lập phần tử cần hủy trước hủy.
- Các vị trí cần hủy
  - Hủy phần tử đứng đầu List
  - Hủy phần tử có khoá bằng x
  - Huỷ phần tử đứng sau q trong danh sách liên kết đơn
- Ở phần trên, các phần tử trong DSLK đơn được cấp phát vùng nhớ động bằng hàm new, thì sẽ được giải phóng vùng nhớ bằng hàm delete.

Thuật toán hủy phần tử trong DSLK

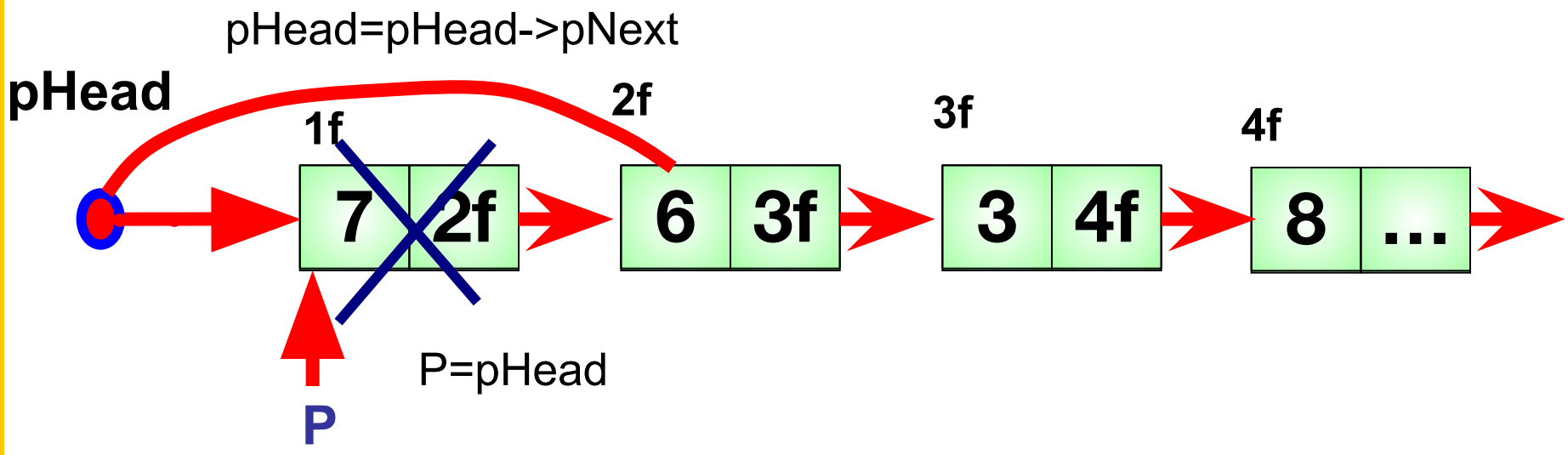
- Bắt đầu:
  - Nếu (pHead!=NULL) thì
  - B1: p=pHead
  - B2:
    + pHead = pHead->pNext
    + delete (p)
  - B3:
    Nếu pHead==NULL thì pTail=NULL

Cài đặt thuật toán

- Hủy được hàm trả về 1, ngược lại hàm trả về 0

```
int   RemoveHead(List &l, int &x)
{     Node *p;
 if(l.pHead!=NULL)
 {    p=l.pHead;
      x=p->Info; //lưu Data của nút cần hủy
      l.pHead=l.pHead->pNext;
      delete p;
      if(l.pHead==NULL)
          l.pTail=NULL;
      return 1;
 }
 return 0;
}
```


Minh hoạ thuật toán



Hủy phần tử sau phần tử q trong List

- Bắt đầu

Nếu (q!=NULL) thì //q tồn tại trong List

- B1: p=q->pNext;// p là phần tử cần hủy
- B2: Nếu (p!=NULL) thì // q không phải là phần tử cuối

+ q->pNext=p->pNext;// tách p ra khỏi xâu

+ nếu (p== pTail) // nút cần hủy là nút cuối

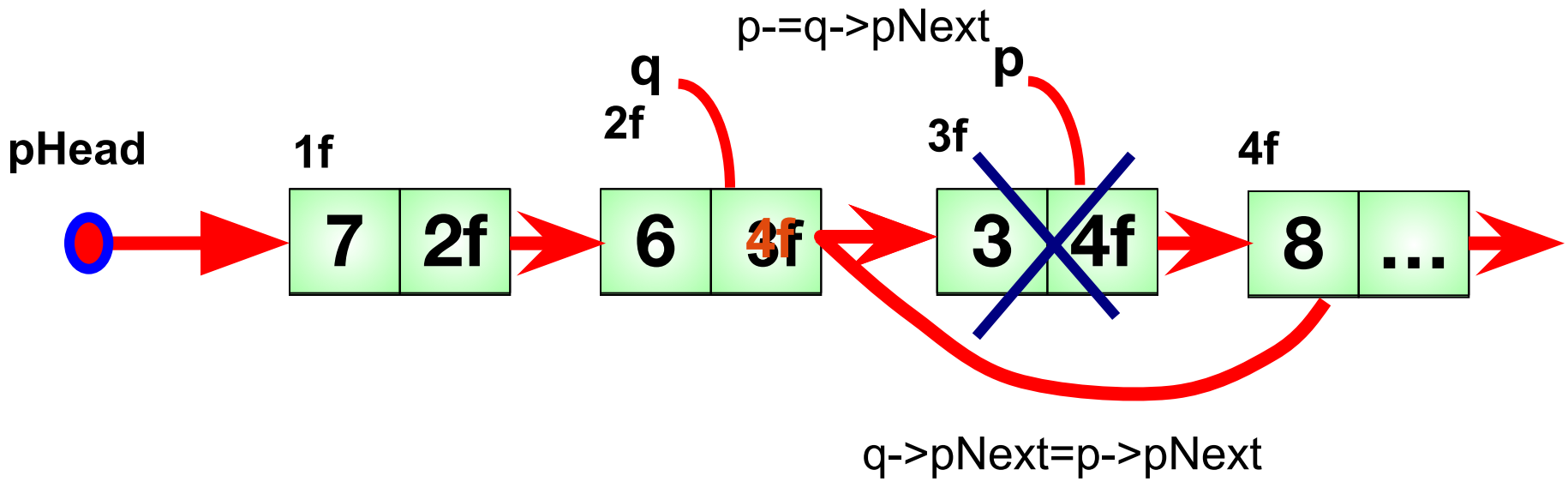
pTail=q;

+ delete p;// hủy p

Cài đặt thuật toán

```
int RemoveAfterQ(List &l,Node *q, int &x)
{    Node *p;
 if(q!=NULL)
 {   p=q->pNext; //p là nút cần xoá
     if(p!=NULL) // q không phải là nút cuối
     {   if(p==l.pTail) //nút cần xoá là nút cuối cùng
             l.pTail=q;// cập nhật lạ pTail
         q->pNext=p->pNext; x=p->Info;
         delete p;
     }
     return 1;
 }
 else
      return 0;}
```


Minh họa thuật toán



Thuật toán hủy phần tử có khoá x

Bước 1:

Tìm phần tử p có khoá bằng x, và q đứng trước p

Bước 2:

Nếu (p!=NULL) thì //tìm thấy phần tử có khoá bằng x

Hủy p ra khỏi List bằng cách hủy phần tử đứng sau q

Ngược lại

Báo không tìm thấy phần tử có khoá

Cài đặt thuật toán

```
int RemoveX(List &l, int x)
{  Node *p,*q = NULL; p=l.Head;
   while((p!=NULL)&&(p->Info!=x)) //tìm
   {    q=p;
    p=p->Next;
   }
   if(p==NULL) //không tìm thấy phần tử có khoá bằng x
    return 0;
   if(q!=NULL)//tìm thấy phần tử có khoá bằng x
    DeleteAfterQ(l,q,x);
   else //phần tử cần xoá nằm đầu List
    RemoveHead(l,x);
   return 1;
}
```


Tìm 1 phần tử trong DSLK đơn

- Tìm tuần tự (hàm trả về), các bước của thuật toán tìm nút có Info bằng x trong list đơn

Bước 1: p=pHead;// địa chỉ của phần tử đầu trong list đơn

Bước 2:

Trong khi p!=NULL và p->Info!=x

p=p->pNext;// xét phần tử kế

Bước 3:

+ Nếu p!=NULL thì p lưu địa chỉ của nút có Info = x

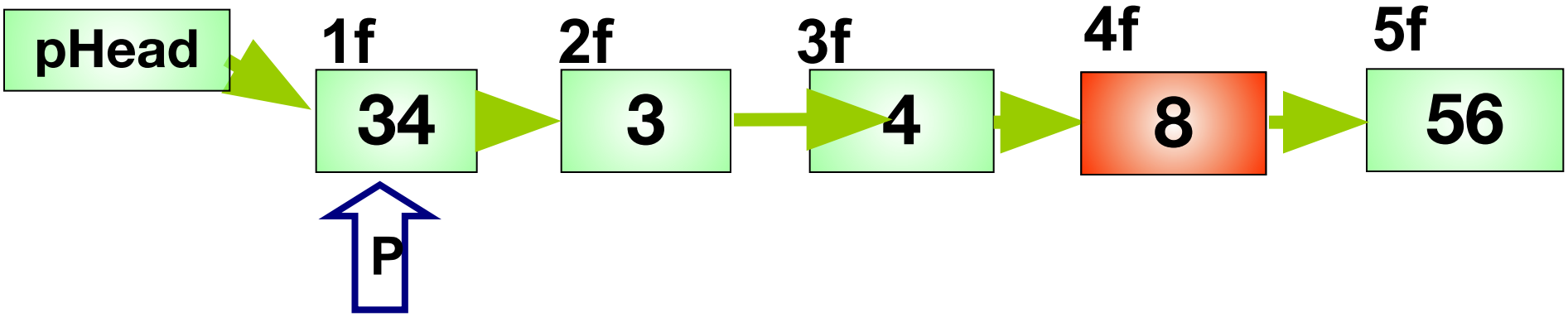
+ Ngược lại : Không có phần tử cần tìm

Hàm tìm 1 phần tử trong DSLK đơn

- Hàm tìm phần tử có Info = x, hàm trả về địa chỉ của nút có Info = x, ngược lại hàm trả về NULL

```
Node *Search(LIST l, Data  x)
{
    Node     *p;
    p = l.pHead;
    while((p!= NULL)&&(p->Info != x))
        p = p->pNext;
   return p;
}
```


Minh họa thuật toán tìm phần tử trong DSLK



X = 8

Tìm thấy, hàm trả về địa chỉ của nút tìm thấy là 4f

Duyệt danh sách

- Duyệt danh sách là thao tác thường được thực hiện khi có nhu cầu cần xử lý các phần tử trong danh sách như:
  - Đếm các phần tử trong danh sách
  - Tìm tất cả các phần tử trong danh sách thảo điều kiện
  - Hủy toàn bộ danh sách

Thuật toán duyệt danh sách

- Bước 1:

  p = pHead;// p lưu địa chỉ của phần tử đầu trong List

- Bước 2:

  Trong khi (danh sách chưa hết) thực hiện

  + xử lý phần tử p

  + p=p->pNext;// qua phần tử kế

Cài đặt in các phần tử trong List

```
void PrintList(List l)
{
 Node *p;
 p=l.pHead;
 while(p!=NULL)
 {   printf("%d     ", p->Info);
     p=p->pNext;
 }
}
```


Hủy danh sách liên kết đơn

- Bước 1:

  Trong khi (danh sách chưa hết) thực hiện

  - B11:

    p = pHead;

    pHead = pHead->pNext;// cập nhật pHead

  - B12:

    Hủy p

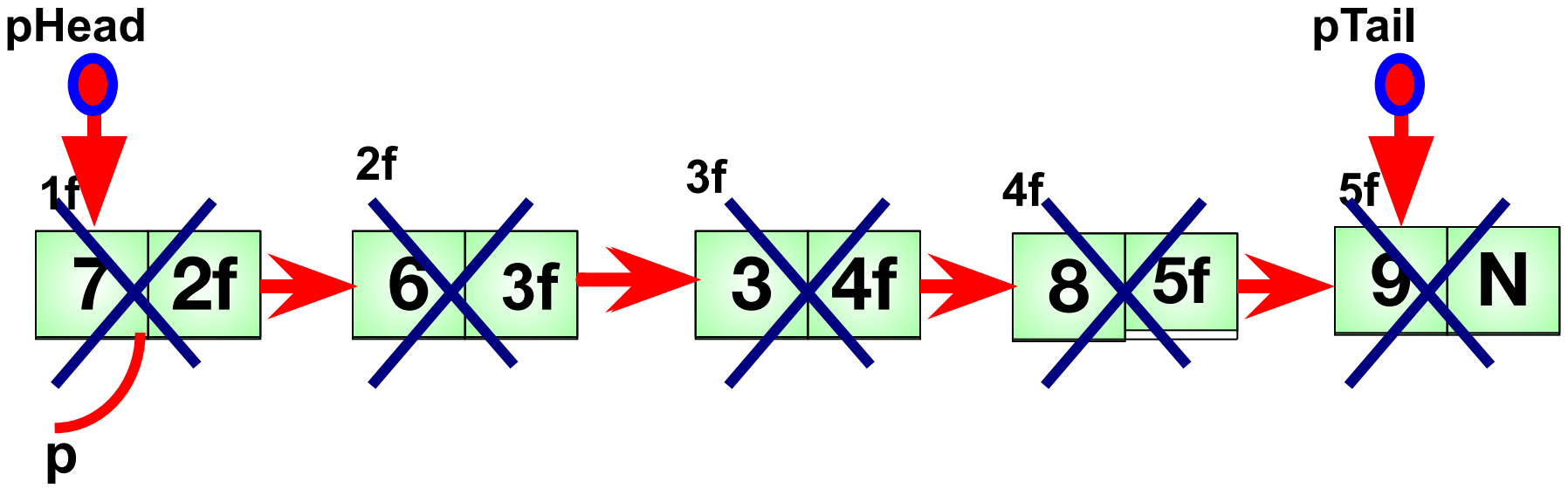
- Bước 2:

  pTail = NULL;// bảo toàn tính nhất quán khi xâu rỗng

Cài đặt thuật toán

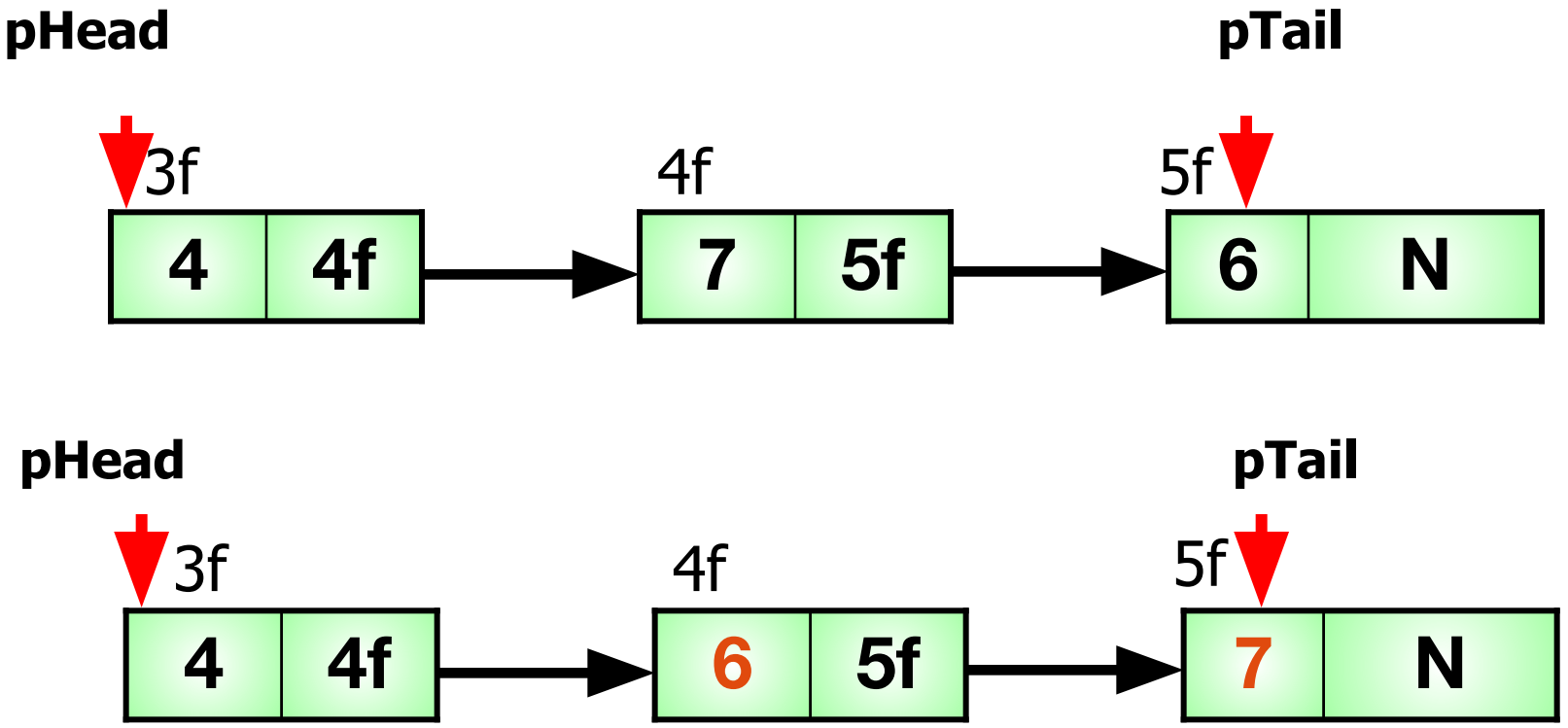
```
void RemoveList(List &l)
{
 Node *p;
 while(l.pHead!=NULL)//còn phần tử trong List
 {
      p = l.pHead;
      l.pHead = p->pNext;
      delete p;
 }
}
```


Minh họa thuật toán



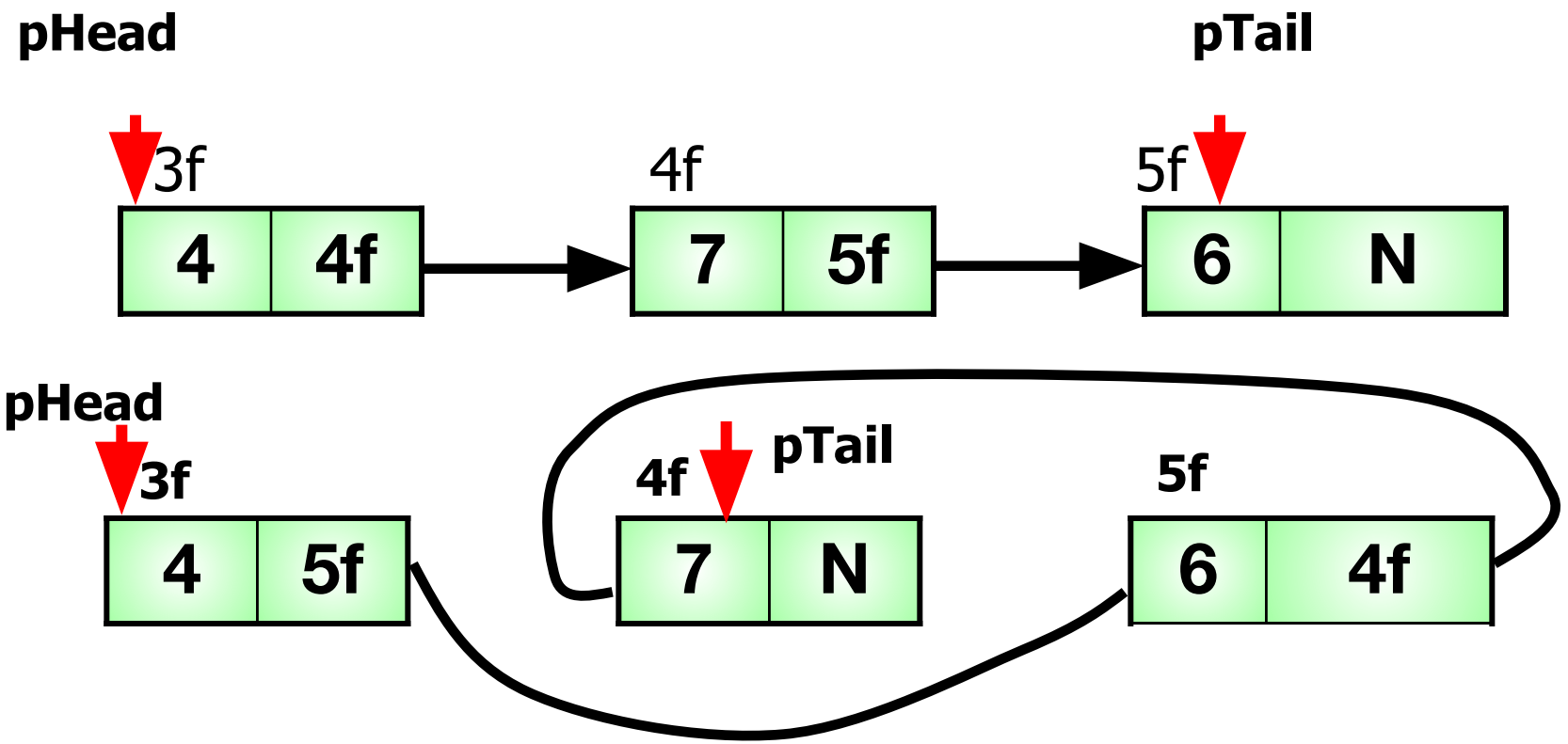
Sắp xếp danh sách

- Có hai cách tiếp cận
- **Cách 1**: Thay đổi thành phần Info



Sắp xếp danh sách

- **Cách 2**: Thay đổi thành phần pNext (thay đổi trình tự móc nối của các phần tử sao cho tạo lập nên được thứ tự mong muốn)



Ưu, nhược điểm của 2 cách tiếp cận

- Thay đổi thành phần Info (dữ liệu)
  - Ưu: Cài đặt đơn giản, tương tự như sắp xếp mảng
  - Nhược:
    - Đòi hỏi thêm vùng nhớ khi hoán vị nội dung của 2 phần tử -> chỉ phù hợp với những xâu có kích thước Info nhỏ
    - Khi kích thước Info (dữ liệu) lớn chi phí cho việc hoán vị thành phần Info lớn
  - ✔ Làm cho thao tác sắp xếp chậm
- Thay đổi thành phần pNext
  - Ưu:
    - Kích thước của trường này không thay đổi, do đó không phụ thuộc vào kích thước bản chất dữ liệu lưu tại mỗi nút.
  - ✔ Thao tác sắp xếp nhanh
  - Nhược: Cài đặt phức tạp

Dùng thuật toán SX SelectionSort để SX List

```
void SelectionSort(LIST &l)
{
    Node *p,*q,*min;
    p=l.pHead;
    while(p!=l.pTail)
   {
     min=p;
     q=p->Next;
     while(q!=NULL)
     {
     if(q->Info<p->Info)
          min=q;
     q=q->Next;
     }
     HV(min->Info,p->Info);
     p=p->Next;
     }
}
```


Các thuật toán sắp xếp hiệu quả trên List

- Các thuật toán sắp xếp xâu (List) bằng các thay đổi thành phần pNext (thành phần liên kết) có hiệu quả cao như:
  - Thuật toán sắp xếp Quick Sort
  - Thuật toán sắp xếp Merge Sort
  - Thuật toán sắp xếp Radix Sort

Thuật toán sắp xếp Quick Sort

- Bước 1:

  Chọn X là phần tử đầu xâu L làm phần tử cầm canh

  Loại X ra khỏi L

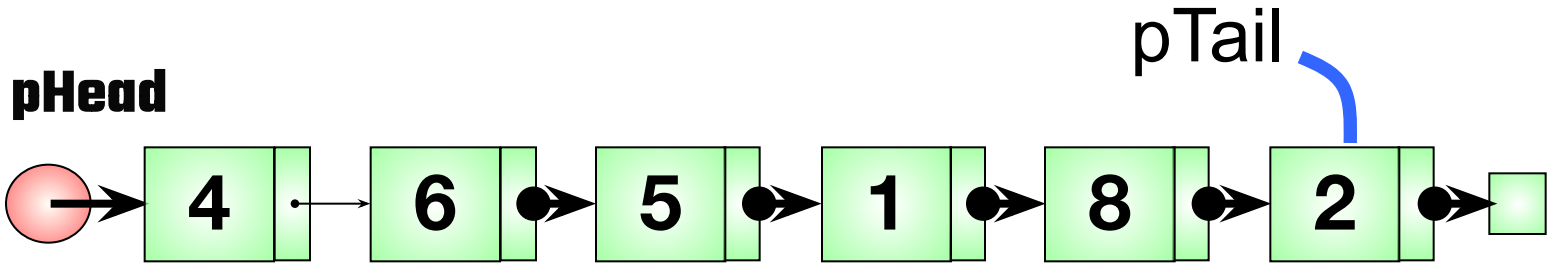
- Bước 2:

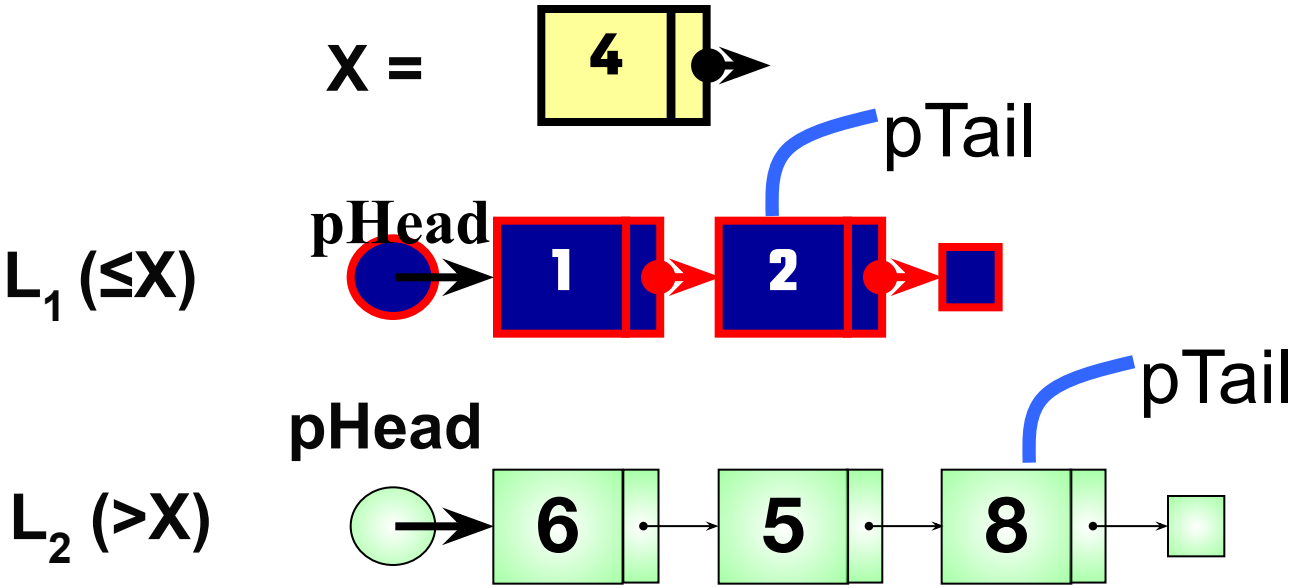
  Tách xâu L ra làm 2 xâu L_1(gồm các phần tử nhỏ hơn hoặc bằng x) và L_2 (gồm các phần tử lớn hơn X)

- Bước 3: Nếu (L_1 !=NULL) thì QuickSort(L_1)
- Bước 4: Nếu (L_2!=NULL) thì QuickSort(L_2)
- Bước 5: Nối L_1, X, L_2 lại theo thứ tự ta có xâu L đã được sắp xếp

Minh họa thuật toán

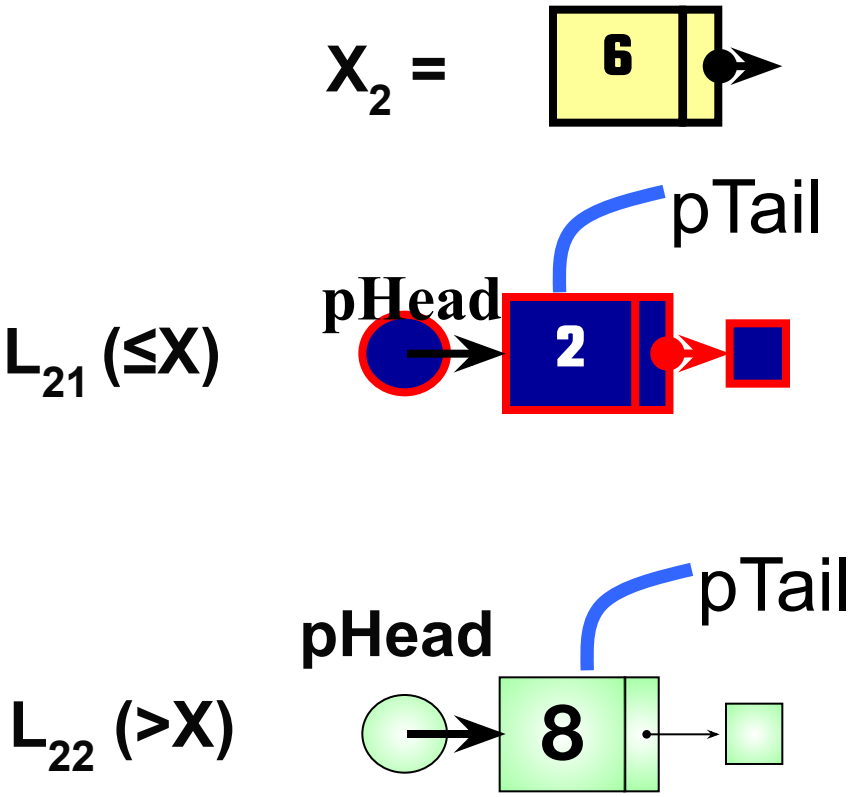
- Cho danh sách liên kết gồm các phần tử sau:





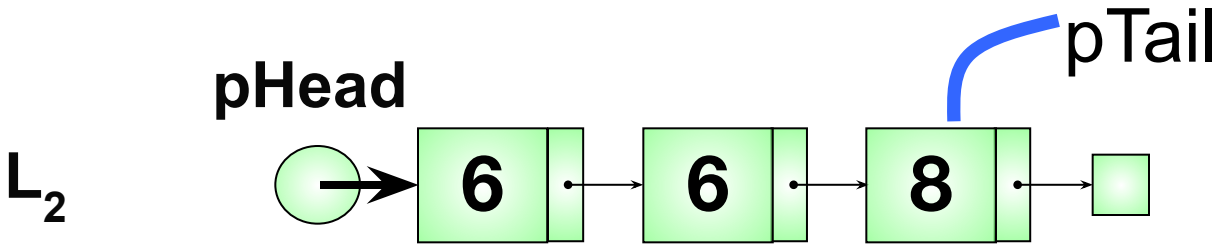
Minh họa thuật toán (tt)

- Sắp xếp L_1
- Sắp xếp L_2
  - Chọn x=6 cầm canh, và tách L_2 thành L_{21} và L_{22}

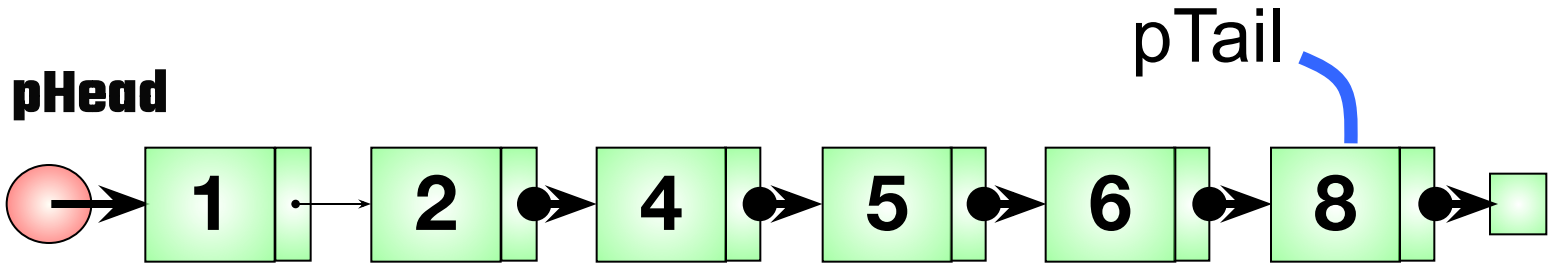


Minh họa thuật toán (tt)

- Nối L_{21}, X_2, L_{22} thành L_2



- Nối L1, X, L2 thành L



Cài đặt thuật toán

```
void QuickSort(List &l)
{   Node *p,*X;//X lưu địa chỉ của phần tử cầm canh
    List l1,l2;
    if(l.pHead==l.pTail) return;//đã có thứ tự
    CreateList(l1);
    CreateList(l2);
    X=l.pHead;
    l.pHead=X->pNext;
    while(l.pHead!=NULL)//tách L = L1 va L2
    {p=l.pHead;
     l.pHead=p->pNext;
     p->pNext=NULL;
     if(p->Info<=X->Info)
          AddHead(l1,p);
     else
          AddHead(l2,p);
    }
```


Cài đặt thuật toán (tt)

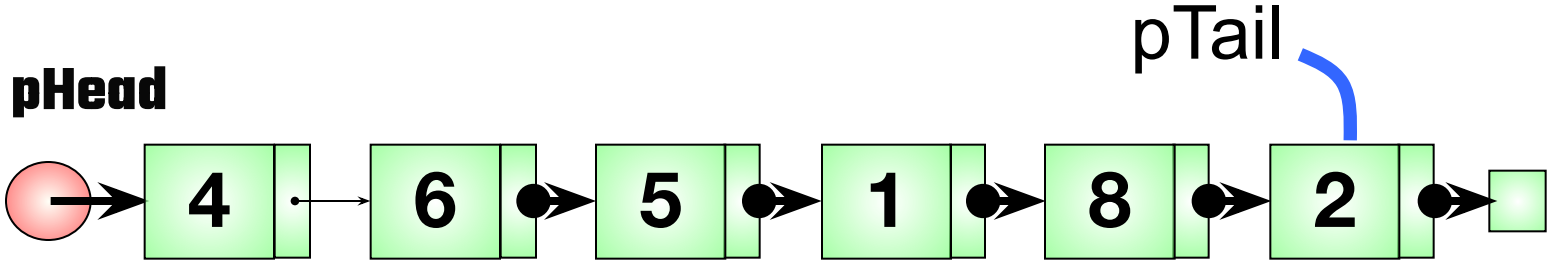
```
    QuickSort(l1);//Gọi đệ quy sắp xếp L1
    QuickSort(l2);//Gọi đệ quy sắp xếp L2
    if(l1.pHead!=NULL)//nối l1, l2 va X vao l
    {
     l.pHead=l1.pHead;
     l1.pTail->pNext=X;//nối X vào
    }
    else
     l.pHead=X;
    X->pNext=l2.pHead;
    if(l2.pHead!=NULL) //l2 có trên một phần tử
     l.pTail=l2.pTail;
    else  //l2 không có phần tử nào
     l.pTail=X;
}
```


Thuật tốn sắp xếp Merge Sort

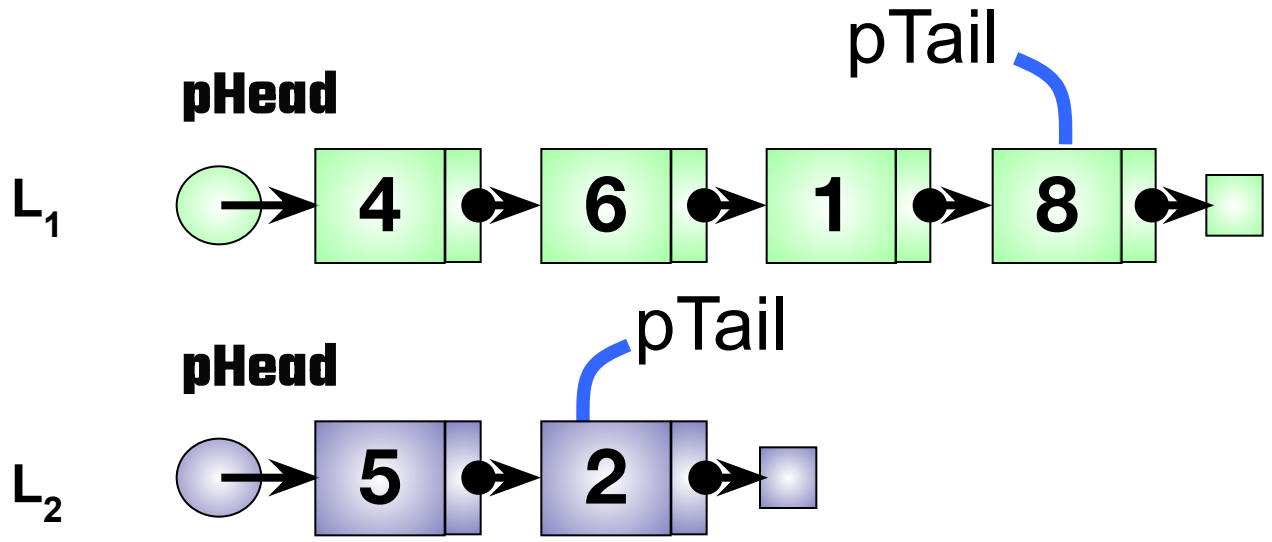
- Bước 1: Phân phối luân phiên từng đường chạy của xâu L vào 2 xâu con L_1 và L_2.
- Bước 2: Nếu L_1 != NULL thì Merge Sort (L_1).
- Bước 3: Nếu L_2 != NULL thì Merge Sort (L_2).
- Bước 4: Trộn L_1 và L_2 đã sắp xếp lại ta có xâu L đã được sắp xếp.
- Không tốn thêm không gian lưu trữ cho các dãy phụ

Minh họa thuật toán

- Cho danh sách liên kết gồm các phần tử sau:

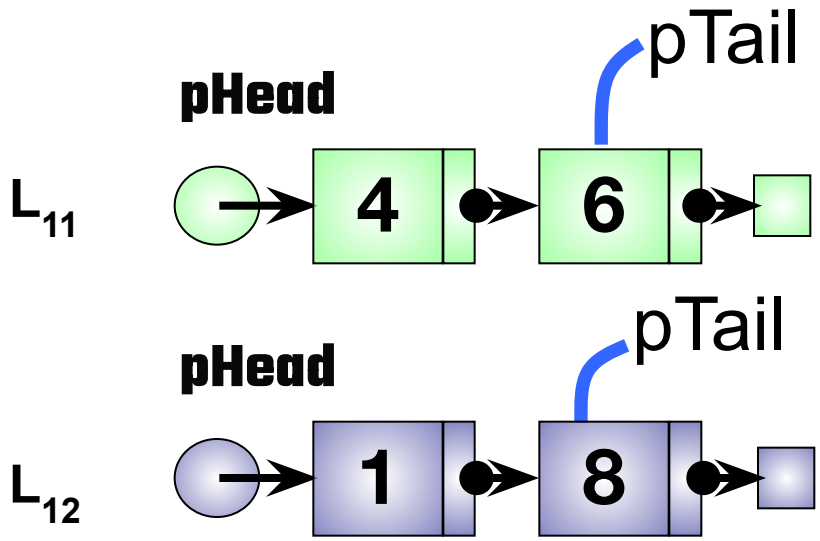


- Phân phối các đường chạy của L_1 vào L_1, L_2

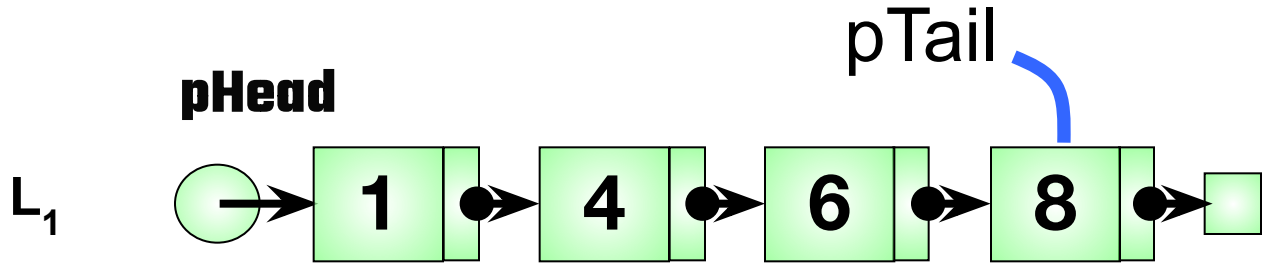


Minh họa thuật toán (tt)

- Sắp xếp L_1
  - Phân phối các đường chạy L_1 vào L_{11}, L_{12}

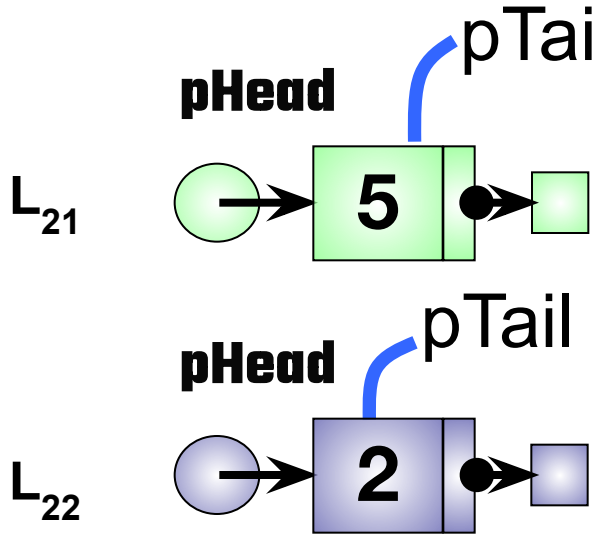


  - Trộn L_{11} và L_{12} vào L_1

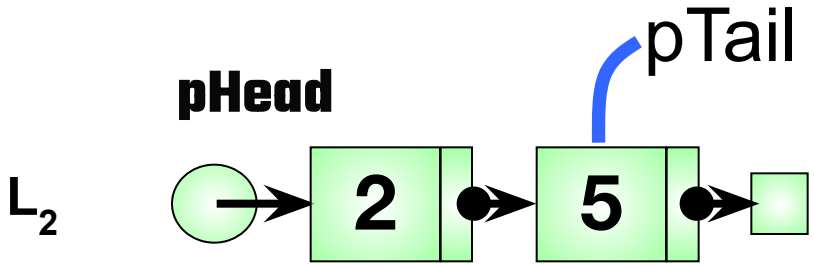


Minh họa thuật toán(tt)

- Sắp xếp L_2
  - Phân phối các đường chạy của L_2 vào L_{21}, L_{22}

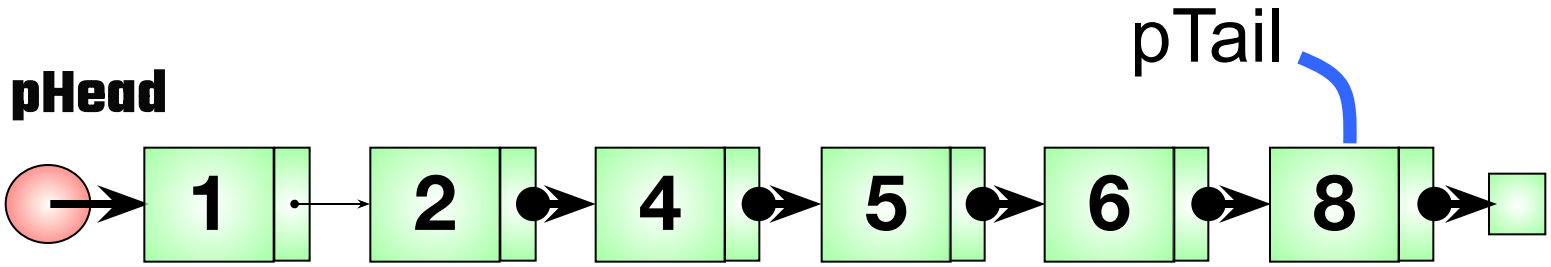


- Trộn L_{21}, L_{22} thành L_2



Minh họa thuật toán (tt)

- Trộn L_1, L_2 thành L



Cài đặt hàm main()

- Yêu cầu: Viết chương trình thành lập 1 xâu đơn, trong đó thành phần dữ liệu của mỗi nút là 1 số nguyên dương.

1. Liệt kê tất thành phần dữ liệu của tất cả các nút trong xâu
2. Tìm 1 phần tử có khoá bằng x trong xâu.
3. Xoá 1 phần tử đầu xâu
4. Xoá 1 phần tử có khoá bằng x trong xâu
5. Sắp xếp xâu tăng dần theo thành phần dữ liệu (Info)
6. Chèn 1 phần tử vào xâu, sao cho sau khi chèn xâu vẫn tăng dần theo trường dữ liệu

..vv

Cài đặt hàm main() (tt)

```
void main()
{    LIST  l1; Node *p; int x;
 CreateList(l1);
 do{
     printf("nhap x="); scanf("%d",&x);
     if(x>0)
     {   p = CreateNode(x);
         AddHead(l1,x);
     }
 }while(x>0);
 printf("Danh sách mới thành lập là\n");
 PrintList(l1);
 printf("nhập x cần tìm x="); scanf("%d",&x);
```


Cài đặt hàm main() (tt)

```
p = Search(l1,x);
if(p==NULL) printf(“không tìm thấy”);
else printf(“tìm thấy”);
RemoveHead(l1,x);
printf(“danh sách sau khi xóa\n”);
PrintList(l1);
printf(“nhập khoá cần xoá\n”);
scanf(“%d”,&x);
RemoveX(l1,x);
```


Cài đặt hàm main() (tt)

```
    printf("danh sách sau khi xoá");
    PrintfList(l1);
    SelectionSort(l1);
    printf("Danh sách sau khi sắp xếp");
    PrintfList(l1);
    RemoveList(l1);
}
```


Vài ứng dụng danh sách liên kết đơn

- Dùng xâu đơn để lưu trữ danh sách các học viên trong lớp học
- Dùng xâu đơn để quản lý danh sách nhân viên trong một công ty, trong cơ quan
- Dùng xâu đơn để quản lý danh sách các cuốn sách trong thư viện
- Dùng xâu đơn để quản lý các băng đĩa trong tiệm cho thuê đĩa.
- ..vv

Dùng xâu đơn để quản lý lớp học

- Yêu cầu: Thông tin của một sinh viên gồm, mã số sinh viên, tên sinh viên, điểm trung bình.

1. Hãy khai báo cấu trúc dữ liệu dạng danh sách liên kết để lưu danh sách sinh viên nói trên.
2. Nhập danh sách các sinh viên, và thêm từng sinh viên vào đầu danh sách (việc nhập kết thúc khi tên của một sinh viên bằng rỗng)
3. Tìm một sinh viên có trong lớp học hay không
4. Xoá một sinh viên có mã số bằng x (x nhập từ bàn phím)
5. Liệt kê thông tin của các sinh viên có điểm trung bình lớn hơn hay bằng 5.

Dùng xâu đơn để quản lý lớp học

6. Xếp loại và in ra thông tin của từng sinh viên, biết rằng cách xếp loại như sau:

   ĐTB <=3.6 : Loại yếu

   ĐTB>=50 và ĐTB<6.5 : Loại trung bình

   ĐTB>=6.5 và ĐTB < 7.0: Loại trung bình khá

   ĐTB>=7.0 và ĐTB <8.0: Loại khá

   ĐTB>=8.0 và ĐTB < 9.0: Loại giỏi.

   ĐTB>=9.0 : Loại xuất sắc

7. Sắp xếp và in ra danh sách sinh viên tăng theo điểm trung bình.
8. Chèn một sinh viên vào danh sách sinh viên tăng theo điểm trung bình nói trên, sao cho sau khi chèn danh sách sinh viên vẫn tăng theo điểm trung bình

..vv

Cấu trúc dữ liệu cho bài toán

- Cấu trúc dữ liệu của một sinh viên

```
typedef struct
{    char tên[40];
 char Maso[40];
 float ĐTB;
}SV
```

- Cấu trúc dữ liệu của 1 nút trong xâu

```
typedef struct tagNode
{    SV Info;
 struct tagNode *pNext;
}Node;
```


Câu hỏi và Bài tập

1. Nêu các bước để thêm một nút vào đầu, giữa và cuối danh sách liên kết đơn.
2. Nêu các bước để xóa một nút ở đầu, giữa và cuối danh sách liên kết đơn.
3. Viết thủ tục để in ra tất cả các phần tử của 1 danh sách liên kết đơn.
4. Viết chương trình thực hiện việc sắp xếp 1 danh sách liên kết đơn bao gồm các phần tử là số nguyên.
5. Viết chương trình cộng 2 đa thức được biểu diễn thông qua danh sách liên kết đơn.

Các cấu trúc đặc biệt của danh sách đơn

- Stack (ngăn xếp): Là 1 vật chứa các đối tượng làm việc theo cơ chế LIFO (Last In First Out), tức việc thêm 1 đối tượng vào Stack hoặc lấy 1 đối tượng ra khỏi Stack được thực hiện theo cơ chế “vào sau ra trước”
- Queue (hàng đợi): Là 1 vật chứa các đối tượng làm việc theo cơ chế FIFO (First In First Out), tức việc thêm 1 đối tượng vào hàng đợi hay lấy 1 đối tượng ra khỏi hàng đợi thực hiện theo cơ chế “vào trước ra trước”.

Ứng dụng Stack và Queue

- Stack:
  - Trình biên dịch
  - Khử đệ qui đuôi
  - Lưu vết các quá trình quay lui, vét cạn
- Queue:
  - Tổ chức lưu vết các quá trình tìm kiếm theo chiều rộng, và quay lui vét cạn,
  - Tổ chức quản lý và phân phối tiến trình trong các hệ điều hành,
  - Tổ chức bộ đệm bàn phím, …

Các thao tác trên Stack

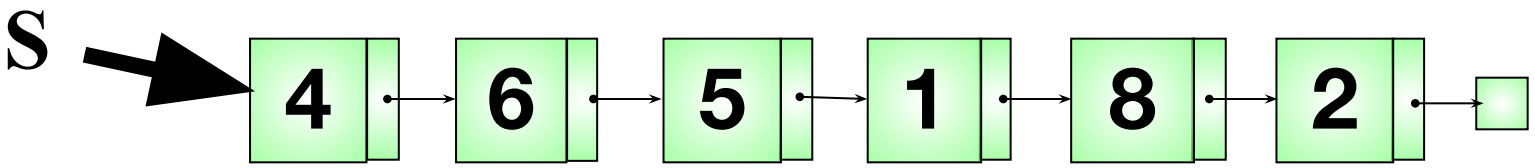
- Push(o): Thêm đối tượng o vào Stack
- Pop(): Lấy đối tượng từ Stack
- isEmpty(): Kiểm tra Stack có rỗng hay không
- Top(): Trả về giá trị của phần tử nằm đầu Stack mà không hủy nó khỏi Stack.

Cài đặt Stack

- Dùng mảng 1 chiều

Data S [N];
int t;

- Dùng danh sách liên kết đơn



List S

- **Thêm và hủy cùng phía**

Cài Stack bằng mảng 1 chiều

- *Cấu trúc dữ liệu của Stack*

```
typedef struct tagStack
{
   int a[max];
   int t;
}Stack;
```

- *Khởi tạo Stack:*

```
void CreateStack(Stack &s)
{
   s.t=-1;
}
```


Kiểm tra tính rỗng và đầy của Stack

```
int IsEmpty(Stack s)//Stack có rỗng hay không
{
    if(s.t==-1)
     return 1;
    else
     return 0;
}
int IsFull(Stack s) //Kiểm tra Stack có đầy hay không
{
    if(s.t>=max)
     return 1;
    else
     return 0;
}
```


Thêm 1 phần tử vào Stack

```
int Push(Stack &s, int x)
{
   if(IsFull(s)==0)
   {
    s.t++;
    s.a[s.t]=x;
    return 1;
   }
   else
    return 0;
}
```


Lấy 1 phần tử từ Stack

```
int Pop(Stack &s, int &x)
{
   if(IsEmpty(s)==0)
   {
    x=s.a[s.t];
    s.t--;
    return 1;
   }
   else
    return 0;
}
```


Cài Stack bằng danh sách liên kết

- Kiểm tra tính rỗng của Stack

```
int IsEmpty(List &s)
{
 if(s.pHead==NULL)//Stack rong
      return 1;
 else
       return 0;
}
```


Thêm 1 phần tử vào Stack

```
void Push(List &s,Node *Tam)
{
   if(s.pHead==NULL)
   {
    s.pHead=Tam;
    s.pTail=Tam;
   }
   else
   {
    Tam->pNext=s.pHead;
    s.pHead=Tam;
   }
}
```


Lấy 1 phần tử từ Stack

```
int Pop(List &s,int &trave)
{   Node *p;
    if(IsEmpty(s)!=1)
    {
     if(s.pHead!=NULL)
     {    p=s.pHead;
          trave=p->Info;
          s.pHead=s.pHead->Next;
          if(s.pHead==NULL)
               s.Tail=NULL;
          return 1;
          delete p;
     }
    }
    return 0;
}
```


Các thao tác trên Queue

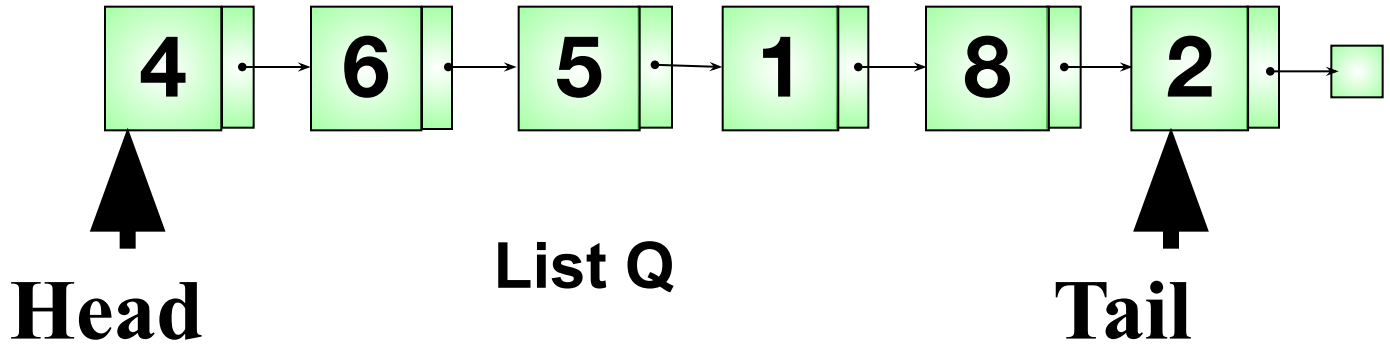
- EnQueue(O): Thêm đối tượng O vào cuối hàng đợi.
- DeQueue(): Lấy đối tượng ở đầu hàng đợi
- isEmpty(): Kiểm tra xem hàng đợi có rỗng hay không?
- Front(): Trả về giá trị của phần tử nằm đầu hàng đợi mà không hủy nó.

Cài đặt Queue

- Dùng mảng 1 chiều

Data S [N];
int f,r;

- Dùng danh sách liên kết đơn



* **Thêm và hủy Khác phía**

Cài đặt Queue bằng mảng 1 chiều

- *Cấu trúc dữ liệu:*

```
typedef struct tagQueue
{
    int a[100];
    int Front; //chỉ số của phần tử đầu trong Queue
    int Rear; //chỉ số của phầ tử cuối trong Queue
}Queue;
```

- *Khởi tạo Queue rỗng*

```
void CreateQueue(Queue &q)
{  q.Front=-1;
   q.Rear=-1;
}
```


Lấy 1 phần tử từ Queue

```
int DeQueue(Queue &q,int &x)
{
    if(q.Front!=-1)     //queue khong rong
    {
     x=q.a[q.Front];
     q.Front++;
     if(q.Front>q.Rear)//truong hop co mot phan tu
     {
          q.Front=-1;
          q.Rear=-1;
     }
     return 1;
    }
    else //queue trong
    {
     printf("Queue rong");
     return 0;
    }
}
```


Thêm 1 phần tử vào Queue

```
void EnQueue(Queue &q,int x)
{
	int i;
	int f,r;
	if(q.Rear-q.Front+1==N)//queue bi day khong the them vao duoc nua
	  printf("queue day roi khong the them vao duoc nua");
	else
	{
	  if(q.Front==-1)
	  {
		q.Front=0;
		q.Rear=-1;
	  }
	  if(q.Rear==N-1)//Queue đầy ảo
	  {
		f=q.Front;
		r=q.Rear;
		for(i=f;i<=r;i++)
			q.a[i-f]=q.a[i];
		q.Front=0;
		q.Rear=r-f;
	  }
	  q.Rear++;
	  q.a[q.Rear]=x;
	}
}
```


Cài đặt Queue bằng List

- Kiểm tra Queue có rỗng?

```
int IsEmpty(List &Q)
{
   if(Q.pHead==NULL)//Queue rỗng
      return 1;
   else
      return 0;
}
```


Thêm 1 phần tử vào Queue

```
void EnQueue(List &Q, Node *Tam)
{
   if(Q.pHead==NULL)
   {
    Q.pHead=Tam;
    Q.pTail=Tam;
   }
   else
   {
    Q.pTail->Next=tam;
    Q.pTail=tam;
   }
}
```


Lấy 1 phần tử từ Queue

```
int DeQueue(List &Q,int &trave)
{   Node *p;
    if(IsEmpty(Q)!=1)
    {
     if(Q.pHead!=NULL)
     {    p=Q.pHead;
          trave=p->Info;
          Q.pHead=Q.pHead->Next;
          if(Q.pHead==NULL)
               Q.pTail=NULL;
          return 1;
          delete p;
     }
    }
    return 0;
}
```