Cấu trúc dữ liệu và Giải thuật

Chương III: Mảng và Danh sách

Mảng và Danh sách

Nội dung

- Cấu trúc dữ liệu Mảng
  - Lưu trữ Mảng 1 chiều
  - Lưu trữ Mảng 2 chiều
  - Các phép toán trên cấu trúc Mảng
- Danh sách tuyến tính
  - Lưu trữ kế tiếp
  - Lưu trữ móc nối

Kiểu dữ liệu trừu tượng Mảng

- Đối tượng của Mảng:
  - Một tập các cặp (index, item)
  - Với mỗi giá trị của index sẽ có một giá trị tương ứng của item.
  - Index là một tập có thứ tự có một chiều hoặc nhiều chiều
    - Index 1 chiều : {0, 1, 2, …, n-1}
    - Index 2 chiều : {(0,0) , (0,1), (0,2), …,(0,n), (1,0), (1,1) ….}

Kiểu dữ liệu trừu tượng Mảng

- Các phép toán
  - Create(j, list) : tạo mảng có j chiều, list là một j-bộ với phần tử thứ k của list là kích thước chiều thứ k của mảng.
  - Retrieve(A,i) : Trả ra giá trị của phần tử nhận chỉ số i nếu có
  - Store(A,i,x) : Trả ra một mảng giống như mảng A đã cho ban đầu, chỉ khác là một cặp (i,x) đã được bổ sung vào vị trí đúng

Cấu trúc dữ liệu Mảng

- Mảng là dãy các phần tử được đánh chỉ số
- Khi cài đặt trong máy tính, mảng được lưu trữ trong một dãy các ô nhớ liên tiếp trong bộ nhớ
- Kích thước của mảng được xác định khi khởi tạo và không thay đổi
- Mỗi phần tử trong mảng có một chỉ số xác định
- Truy xuất vào các phần tử của mảng sử dụng chỉ số của phần tử

Mảng trong các ngôn ngữ lập trình

- Tập chỉ số của mảng có thể khác nhau
  - C, Java : chỉ số là số nguyên, liên tục, bắt đầu từ 0
  - Pascal : chỉ số có thể có giá trị rời rạc
  - Perl: cho phép chỉ số không phải là số
- Mảng có thể là thuần nhất hoặc không thuần nhất
- Mảng có thể có thêm các thông tin bổ sung ngoài các phần tử

Mảng 1 chiều

- Khởi tạo
  - Cần chỉ ra số phần tử của mảng
  - Khai báo mảng trong C:
    <kiểu dữ liệu của phần tử ><tên biến>[size]
    - int list[5];
    - char word[25];
- Tham chiếu
  - Các phần tử trong mảng 1 chiều được tham chiếu đến sử dụng địa chỉ được tính
    - int list [5] → list[0] địa chỉ cơ sở = α
      list[1] α + sizeof(int)
      list[2] α + 2*sizeof(int)
      list[3] α + 3*sizeof(int)
      list[4] α + 4*sizeof(int)

Mảng 1 chiều

```
int list[] = {0, 1, 2, 3, 4};
int *ptr; int rows = 5;
int i; ptr = list;
printf("Address Value\n");
for (i=0; i < rows; i++)
  printf("%8u%5d\n", ptr+i, *(ptr+i));
  printf("\n");
```

Address	Value
1228	0
1230	1
1232	2
1234	3
1236	4

Mảng 2 chiều

- Khai báo
  - Cần chỉ ra số hàng, số cột
  - Trong C : <kiểu phần tử> <tên biến> [size1] [size2]
    - int table[4][5];
  - Truy xuất một phần tử
    - table[i][j]
- Lưu trữ mảng 2 chiều trong bộ nhớ máy tính
  - Theo thứ tự ưu tiên hàng
  - Theo thứ tự ưu tiên cột

Mảng 2 chiều

- Lưu trữ mảng 2 chiều theo thứ tự ưu tiên hàng

a_{00}	a_{01}	a_{02}
a_{10}	a_{11}	a_{12}
a_{20}	a_{21}	a_{22}
a_{30}	a_{31}	a_{32}

Từ mảng 2 chiều lưu trữ sang bộ nhớ kế tiếp sử dụng thứ tự ưu tiên hàng

a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}	a_{20}	a_{21}	a_{22}	a_{30}	a_{31}	a_{32}

Mảng 2 chiều

- Lưu trữ mảng 2 chiều theo thứ tự ưu tiên cột

a_{00}	a_{01}	a_{02}
a_{10}	a_{11}	a_{12}
a_{20}	a_{21}	a_{22}
a_{30}	a_{31}	a_{32}

Từ mảng 2 chiều lưu trữ sang bộ nhớ kế tiếp sử dụng thứ tự ưu tiên cột

a_{00}	a_{10}	a_{20}	a_{30}	a_{01}	a_{11}	a_{21}	a_{31}	a_{02}	a_{12}	a_{22}	a_{32}

Danh sách tuyến tính

- Danh sách là một tập hợp có thứ tự gồm một số biến động các phần tử cùng kiểu $\{a_1, a_2, ..., a_{n-1}, a_n\}$
- a_i là phần tử ở vị trí i trong danh sách
- a_1 là phần tử đầu tiên, a_n là phần tử cuối cùng của danh sách
- n là độ dài của danh sách tại 1 thời điểm
- Trường hợp n =0 ta có danh sách rỗng
- Trong danh sách tuyến tính, thứ tự trước sau của các phần tử được xác định rõ ràng.

Các cách cài đặt danh sách tuyến tính

- Dùng Mảng:
  - Lưu trữ các phần tử của danh sách trong một vector lưu trữ bao gồm các ô nhớ liên tiếp
- Dùng Con trỏ:
  - Các phần tử được lưu trữ trong các ô nhớ ở các vị trí tùy ý trong bộ nhớ
  - Các phần tử liên kết với nhau bằng con trỏ
- Dùng địa chỉ gián tiếp
  - Các phần tử được lưu trữ trong các ô nhớ ở các vị trí tùy ý trong bộ nhớ
  - Có một mảng địa chỉ trong đó phần tử thứ i của mảng chứa địa chỉ của phần tử thứ i trong danh sách

Lưu trữ kế tiếp đối với danh sách

- Danh sách lưu trữ trong một phần bộ nhớ bao gồm các ô nhớ liên tiếp
  - Các phần tử liền kề nhau được lưu trữ trong những ô nhớ liền kề nhau
  - Mỗi phần tử của danh sách cũng được gán một chỉ số chỉ thứ tự được lưu trữ trong vector
  - Có một chỉ số last dùng để xác định chỉ số của phần tử cuối cùng trong danh sách

Lưu trữ kế tiếp đối với danh sách

- Khai báo danh sách sử dụng lưu trữ kế tiếp trong C

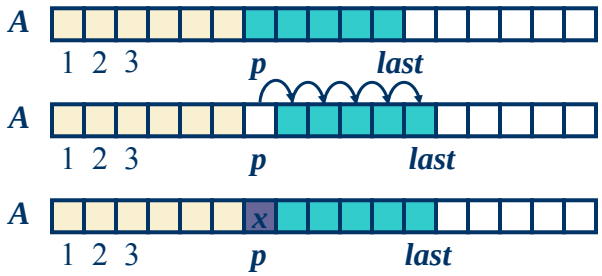
```
#define max 100
typedef etype integer
typedef struct LIST{
        etype elements[max];
        int last;
} LISTTYPE
```

Lưu trữ kế tiếp đối với danh sách

- Ưu điểm của cách lưu trữ kế tiếp
  - Tốc độ truy cập vào các phần tử của danh sách nhanh
- Nhược điểm của cách lưu trữ kế tiếp
  - Cần phải biết trước kích thước tối đa của danh sách
    - Tại sao?
  - Thực hiện các phép toán bổ sung các phần tử mới và loại bỏ các phần tử cũ khá tốn kém
    - Tại sao?

Các thao tác trên danh sách kế tiếp

– Bổ sung một phần tử vào vị trí p trong danh sách

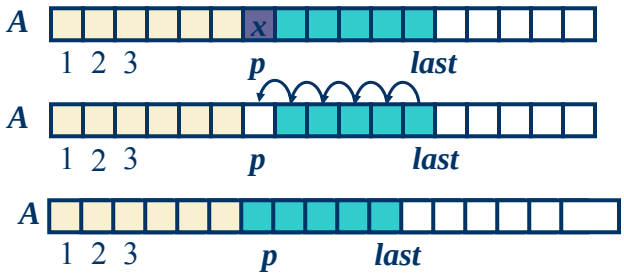


Các thao tác trên danh sách kế tiếp

```
Procedure INSERT-LIST(L, x, p)
Begin
{ L là danh sách được lưu trữ dưới dạng mảng, x là giá trị phần tử mới, p là vị trí
phần tử mới, L có số tối đa là max phần tử , last là chỉ số phần tử cuối cùng trong
danh sách }
1.  {Danh sách đã đầy}  if (last > max) then ERROR;
2.  {Kiểm tra giải trị p}  else  if (p > last ) OR (p < 1) then ERROR;
3.  else
        begin {Dịch chuyển các phần tử, tạo ô trống để bổ sung}
                for i = last down to p do L[i+1] = L[i];
                {Lưu giá trị mới vào vị trí p} L[p] = x;
                last = last+1; {Số lượng phần tử trong danh sách tăng thêm 1}
        end.
End
```

Các thao tác trên danh sách kế tiếp

– Loại bỏ một phần tử trong danh sách



Các thao tác trên danh sách kế tiếp

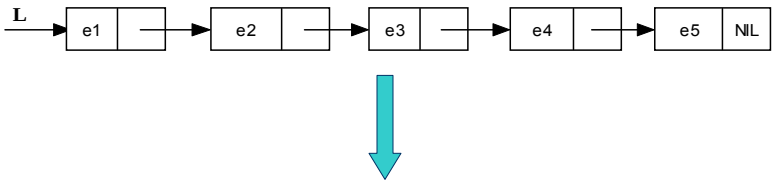
```
Procedure DELETE-LIST(L, p)
Begin
{ Loại bỏ phần tử ở vị trí p trong danh sách kế tiếp L.
 L có tối đa max phần tử , hiện tại phần tử cuối cùng ở vị trí last}
1.  {Kiểm tra p} if (p > last ) OR (p <1) then ERROR;
2.  {Dồn các phần tử ở đuôi danh sách lên trên 1 vị trí}
    for i:= p to last-1 do
            S[i] := S[i+1];
    last:= last-1;
3.  End.
```

Lưu trữ móc nối đối với danh sách

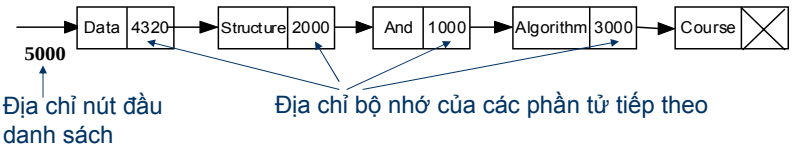
- Danh sách móc nối đơn (Singly Linked-List)
  - Một phần tử trong danh sách = một nút
  - Một nút có hai thành phần
    - INFO: chứa thông tin (nội dung, giá trị) ứng với phần tử
    - NEXT: chứa địa chỉ của nút tiếp theo
  - Để thao tác được trên danh sách, cần nắm được địa chỉ của nút đầu tiên trong danh sách ⇔ biết được con trỏ L trỏ tới đầu danh sách

Danh sách móc nối đơn

Hình ảnh danh sách móc nối đơn



Ví dụ danh sách móc nối đơn



Danh sách móc nối đơn

- Danh sách rỗng là danh sách không có chứa nút nào, lúc đó L = NULL
- Tham chiếu đến các thành phần của một nút có địa chỉ p (trỏ bởi con trỏ p)
  - INFO(p): Tham chiếu vào giá trị
    - INFO(p) = 234 ←→ giá trị dữ liệu lưu trữ tại nút trỏ bởi p là 234;
  - NEXT(p)
    - NEXT(p) = 234 ←→ Ô nhớ chứa phần tử sau nút trỏ bởi p có địa chỉ là 234
- Cấp phát một nút trống sẽ được trỏ bởi p

  Câu lệnh trong giả ngôn ngữ : *call New(p)*
- Thu hồi một nút trỏ bởi p

  Câu lệnh trong giả ngôn ngữ: *call Dispose(p)*

Danh sách móc nối đơn

- Khai báo trong ngôn ngữ C

```
typedef <kiểu dữ liệu của phần tử> element_type;
struct node{
        element_type info;
        struct node * next;
} ;
typedef struct node LISTNODE;
typedef LISTNODE *LISTNODEPTR;
```

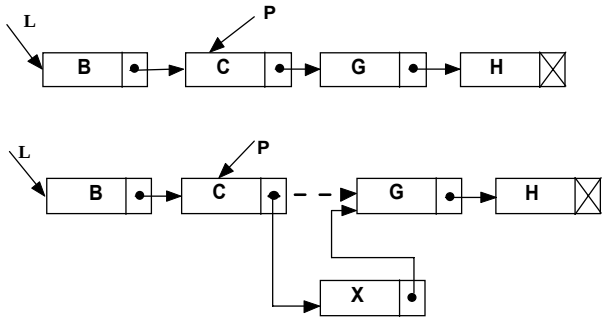
Các thao tác trên danh sách nối đơn

- Duyệt danh sách nối đơn:

```
Procedure TRAVERSE(L)
{Đầu vào của giải thuật là một LISTNODEPTR L}
Begin
    p:= L;
    while p <> NULL do
    begin
        writeln(INFO(p));
        p:= NEXT(p);
    end;
End
```

Các thao tác trên danh sách nối đơn

- Bổ sung một phần tử mới vào danh sách
  - Hãy bổ sung thêm một nút mới có thông tin là X vào sau một nút trong danh sách được trỏ tới bởi con trỏ P



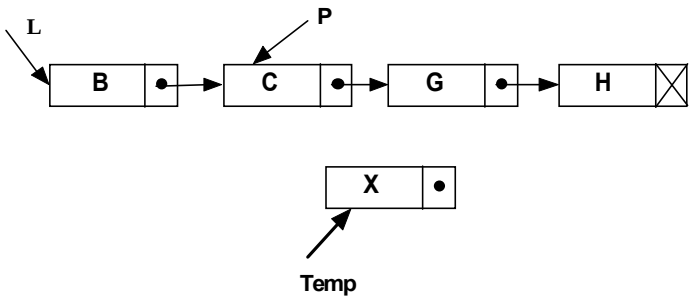
Các thao tác trên danh sách nối đơn

– Bổ sung một phần tử mới vào danh sách

```
Procedure INSERT(L, X, P)
Begin
    1. { Tạo nút mới chứa giá trị X, được trỏ đến bởi con trỏ Temp}
          Call New(Temp) ;
          INFO(Temp) = X;
    2. { Gắn nút mới vào vị trí cần chèn}
          NEXT(Temp) = NEXT(P);
          NEXT(P) = Temp;
End
```

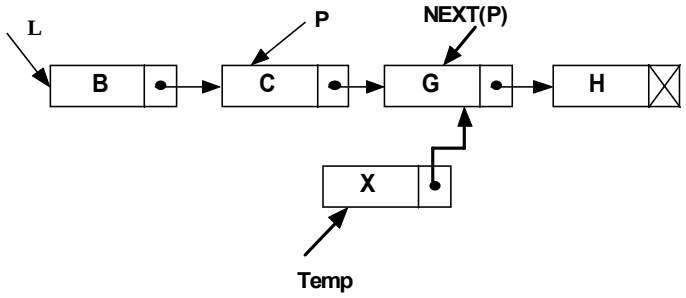
Các thao tác trên danh sách nối đơn

Sau khi khởi tạo nút mới và gán giá trị cho phần tử mới



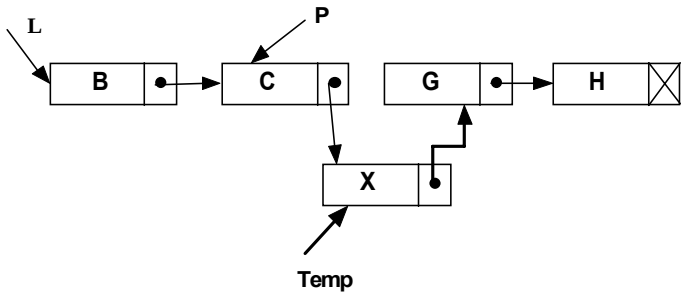
Các thao tác trên danh sách nối đơn

Sau khi thực hiện NEXT(Temp) = NEXT(P);



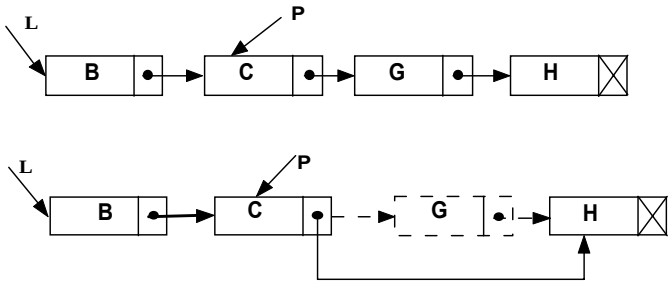
Các thao tác trên danh sách nối đơn

Sau khi thực hiện NEXT(P) = Temp;



Các thao tác trên danh sách nối đơn

- Loại bỏ nút xác định trước:
  - Hãy loại bỏ nút đằng sau nút trỏ bởi con trỏ P cho trước



Các thao tác trên danh sách nối đơn

```
Procedure DELETE(L, p)
Begin
{Trường hợp tổng quát}
    1. Temp = NEXT(p) ;
    2. Next(p) = Next(Temp);
    3. call Dispose(Temp);
End.
```

Minh họa thao tác trong NNLT C

- Cho một danh sách chứa các số nguyên, được sắp xếp theo chiều tăng dần
  - Viết đoạn chương trình C thực hiện bổ sung một nút mới có giá trị x cho trước vào danh sách
  - Viết đoạn chương trình C thực hiện việc loại bỏ một nút có giá trị biết trước

Minh họa thao tác trong NNLT C

- Khai báo danh sách

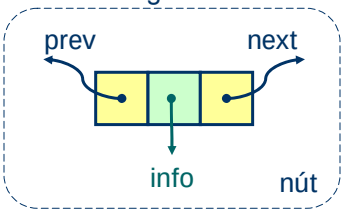
```
struct node{
        int  info;
        struct node * next;
} ;
typedef struct node LISTNODE;
typedef LISTNODE *LISTNODEPTR;

void insert(LISTNODEPTR *, int );
int  delete(LISTNODEPTR *, int);
```

```
void INSERT_ORDER( LISTNODEPTR *startPtr, int value){
/* Chương trình bổ sung một nút vào danh sách có sắp xếp theo chiều tăng dần
của giá trị các phần tử */
    LISTNODEPTR temp, current, previous ;
    temp = malloc(sizeof(LISTNODE));
    if (temp!= NULL) {
        1.      temp->info = value; temp->next = NULL;
                previous = NULL; current = *startPtr;
        2.      while (current != NULL && value >current->info) {
                        previous = current; current = current->next;
                }
        3.      if (previous = NULL) {
                        temp->next = *startPtr;
                        *startPtr = temp;
                }
        4.      else { previous->next = temp; temp->next = current; }
    }
}
```

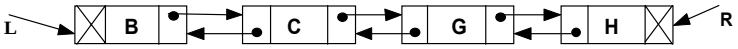
```
int DELETE_ORDER( LISTNODEPTR *startPtr, int value){
/* Chương trình bổ sung một nút vào danh sách có sắp xếp theo chiều tăng dần
của giá trị các phần tử */
    LISTNODEPTR temp, current, previous ;
    if (value == (* startPtr) -> info ) {
        temp = *startPtr; *startPtr = (* startPtr) -> next; free(temp);
        return value;
    }else {
        previous = *startPtr; current = (*startPtr) -> next;
        while(current != NULL && current->info != value){
                previous = current; current = current->next;
        }
        if (current != NULL) { temp = current; previous->next = current->next;
                free(temp) ; return value;
        }
    }
    return '\0';
}
```

Danh sách nối kép

- Qui cách của nút trong danh sách nối kép



  - Trường PREV của nút đầu tiên và trường NEXT của nút cuối cùng đều có giá trị NULL
  - Cần nắm được hai con trỏ, con trỏ L trỏ tới nút cực trái, con trỏ R trỏ tới nút cực phải của danh sách
  - Với danh sách rỗng , L = R = NULL



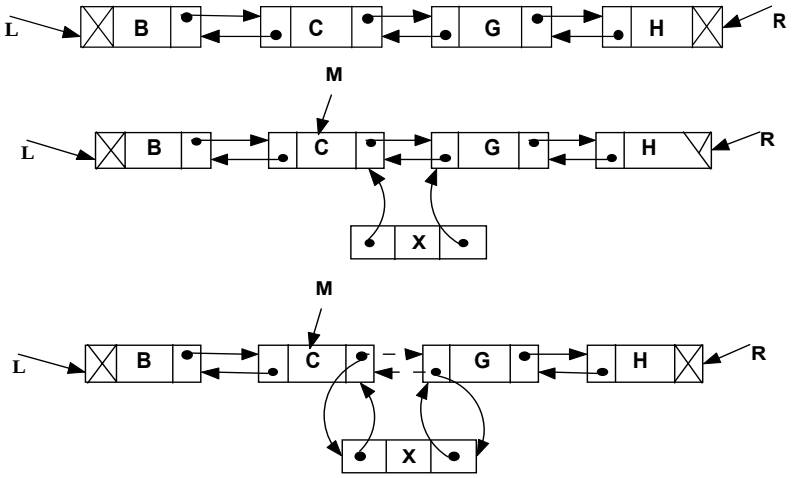
Danh sách nối kép

- Khai báo danh sách nối kép trong C

```
struct dlnode{
        int  info;
        struct dlnode *next;
        struct dlnode *prev;
} ;
typedef struct dlnode DLNODE;
typedef DLNODE *DLNODEPTR;
DLNODEPTR left, right;
```

Các thao tác trên danh sách nối kép

Bổ sung một phần tử vào sau một nút được trỏ bởi con trỏ M biết trước



Các thao tác trên danh sách nối kép

- Giải thuật bổ sung một phần tử mới vào danh sách nối kép

```
Procedure INSERT-DOUBLE (L, R, M, X)
{Bổ sung một phần tử chứa dữ liệu X vào sau phần tử trỏ bởi M}
1.  {Tạo lập nút mới}
    call New(p) ; {xin cấp phát một nút mới có địa chỉ là p}
    INFO(p) := X;
2.  {Danh sách rỗng}
    if L = R= NULL then begin
                        PREV(p):= NEXT(p) := NULL;
                        L:= R:=p;
                        return;
                    end;
(Còn tiếp)
```

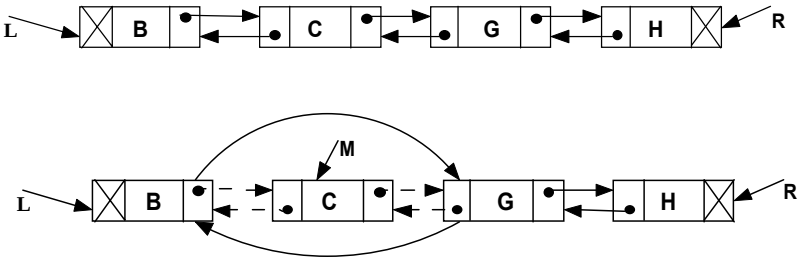
Các thao tác trên danh sách nối kép

- Bổ sung vào danh sách nối kép (tiếp)

```
3. {Trường hợp M là nút cực phải}
if M = R then begin
    NEXT(p) := NULL; PREV(p) := M; NEXT(M) := p;
    R:= p;
end;
4. { Bổ sung vào giữa}
PREV(p) := M; NEXT(p) := NEXT(M);
PREV(NEXT(M)) := p;
NEXT(M) := p;
5. return.
```

Các thao tác trên danh sách nối kép

- Loại bỏ một phần tử



Các thao tác trên danh sách nối kép

- Giải thuật loại bỏ một phần tử khỏi danh sách nối kép

```
Procedure DELETE-DOUBLE (L, R, M)
{Loại bỏ phần tử trỏ bởi M }
1.   {Danh sách rỗng}
     if L= R= NULL  then return;
2.  {Loại bỏ}
     if L= R and L = M then L:=R:= NULL;
     else if M = L then begin L:= NEXT(L); PREV(L) := NULL; end;
     else if M = R then begin R:= PREV(R); NEXT(R) := NULL; end;
               else begin NEXT(PREV(M)) :=NEXT(M); PREV(NEXT(M)) := PREV(M);
               end;
     call Dispose(M);
3. return.
```

Biểu diễn đa thức sử dụng danh sách

– Bài toán cộng hai đa thức

- Dạng tổng quát của một đa thức

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

$$A(x) = 2x^8 - 5x^7 + 3x^2 + 4x - 7$$

$$B(x) = 6x^8 + 5x^7 - 2x^6 + x^4 - 8x^2$$

- Viết giải thuật tìm tổng 2 đa thức trên

Cách tiếp cận sử dụng danh sách kế tiếp

- Biểu diễn đa thức sử dụng danh sách lưu trữ kế tiếp
  - Mỗi số hạng của đa thức ứng với một phần tử của vector lưu trữ
  - Một vector có kích thước n có các phần tử đánh số từ 1 đến n thì lưu trữ được một đa thức có số mũ tối đa là n-1
  - Phần hệ số a_i của một số hạng được lưu trong chính phần tử của vector lưu trữ
  - Phần số mũ i của một số hạng thì ẩn trong thứ tự của phần tử lưu trữ
  - Phần tử thứ i trong vector lưu trữ lưu thông tin về số hạng $a_{i-1}x^{i-1}$
    - Phần tử thứ 1 lưu trữ thông tin a_0
    - Phần tử thứ 2 lưu trữ thông tin về a_1
    - …

Cách tiếp cận sử dụng lưu trữ kế tiếp

- Ví dụ:

$$A(x) = 2x^8 - 5x^7 + 3x^2 + 4x - 7$$
$$B(x) = 6x^8 + 5x^7 - 2x^6 + x^4 - 8x^2$$

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
-7	4	3	0	0	0	0	-5	2

B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]
0	0	-8	0	1	0	-2	5	6

Cách tiếp cận sử dụng lưu trữ kế tiếp

- Giải thuật cộng hai đa thức lưu trữ trên vector

```
Procedure ADD-POLY1(A,m, B, n, C)
Begin
{A, B là hai vector lưu trữ hai đa thức đã cho;
m,n lần lượt là kích thước của A,B, giả sử m <= n ;
C là vector lưu trữ kết quả}
for i:= 1 to n do begin
        if i<= m then
            C[i] := A[i] + B[i] ;
        else
            C[i] := B[i] ;
    end.
End
```

Cách tiếp cận sử dụng lưu trữ móc nối

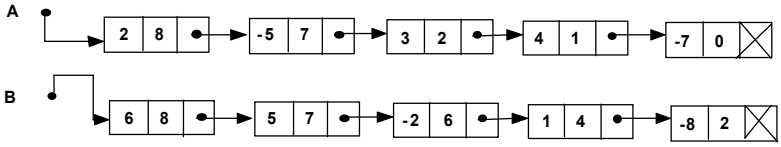
- Biểu diễn đa thức sử dụng lưu trữ móc nối
  - Một đa thức được biểu diễn dưới dạng danh sách nối đơn
  - Quy cách của 1 nút

COEF	EXP	LINK

  - Ví dụ:

$$A(x) = 2x^8 - 5x^7 + 3x^2 + 4x - 7$$

$$B(x) = 6x^8 + 5x^7 - 2x^6 + x^4 - 8x^2$$



Cách tiếp cận sử dụng lưu trữ móc nối

```
Procedure ADD-POLY2(A, B,  C)
Begin
1.  p:= A; q:=B;
2.  call New(C) ; d:= C; {d trỏ vào nút cuối cùng của C}
3. while p <> NULL and q <> NULL do
    case
        EXP(p) = EXP(q):  x := COEF(p) + COEF(q) ;
                          if x<>0 then call ATTACH(x, EXP(p), d) ;
                          p:= LINK(p) ; q:= LINK(q);
        EXP(p) > EXP(q):  call  ATTACH(COEF(p), EXP(p),d);
                          p:= LINK(p);
        EXP(p) < EXP(q):  call  ATTACH(COEF(q), EXP(q),d);
                          q:= LINK(q);
    end case;  {Còn tiếp}
```

Cách tiếp cận sử dụng lưu trữ móc nối

```
4.  {Trường hợp A kết thúc trước, A ngắn hơn}
    while q <> NULL do begin
        call ATTACH(COEF(q), EXP(q),d); q:= LINK(q);
    end ;
5.  {Trường hợp B kết thúc trước}
    while p <> NULL do begin
        call ATTACH(COEF(p), EXP(p), d) ; p := LINK(p);
    end ;
6. {Kết thúc danh sách tổng} LINK(d) := NULL;
7. {Cho con trỏ C trỏ tới danh sách tổng}
    t:= C; C:= LINK(t); call dispose(t);
8.  return.
```

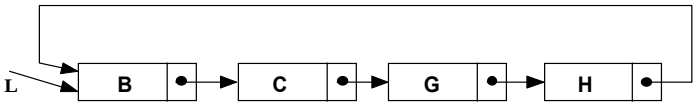
Cách tiếp cận sử dụng lưu trữ móc nối

- Giải thuật gắn một nút mới vào đuôi một danh sách

```
Procedure ATTACH(c, e, d)
Begin
{c, e lần lượt là hệ số và số mũ của nút mới;
d là con trỏ trỏ tới nút đuôi của danh sách }
1.  {Khởi tạo nút mới}
    call New(p); COEF(p) := c; EXP(p) := e;
2.  {Gắn vào danh sách tổng, biến nó thành nút đuôi mới }
    LINK(d) := p;
    d:=p;
End
```

Các dạng danh sách móc nối khác

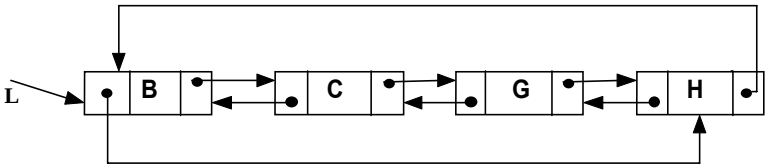
- Danh sách nối vòng (Circularly Linked-List)
  - Trường LINK của nút cuối cùng của danh sách chứa địa chỉ của nút đầu tiên trong danh sách




```
struct clnode{
        int  info;
        struct clnode *next;
} ;
```

Các dạng danh sách móc nối khác

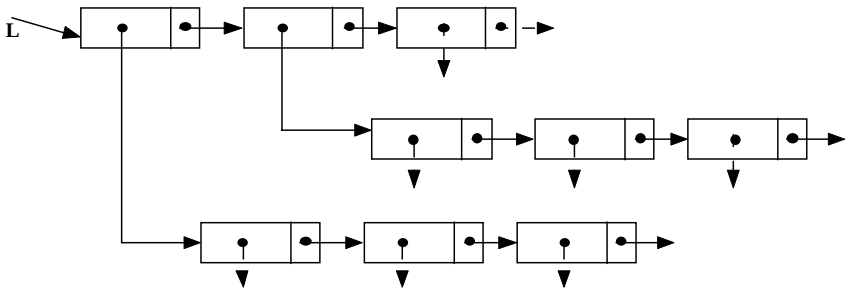
- Danh sách nối vòng kép




```
struct cdlnode{
        int  info;
        struct cdlnode *next;
        struct cdlnode *prev;
} ;
```

Các dạng danh sách móc nối khác

- Danh sách của danh sách




```
struct node{
        struct node*  info;
        struct node *next;
} ;
```