c. So sánh hai phương pháp cài đặt

Không thể kết luận phương pháp cài đặt nào hiệu quả hơn, mà nó hoàn toàn tuỳ thuộc vào từng ứng dụng hay tuỳ thuộc vào các phép toán trên danh sách. Tuy nhiên ta có thể tổng kết một số ưu nhược điểm của từng phương pháp làm cơ sở để lựa chọn phương pháp cài đặt thích hợp cho từng ứng dụng:

- Cài đặt bằng mảng đòi hỏi phải xác định số phần tử của mảng, do đó nếu không thể ước lượng được số phần tử trong danh sách thì khó áp dụng cách cài đặt này một cách hiệu quả vì nếu khai báo thiếu chỗ thì mảng thường xuyên bị đầy, không thể làm việc được còn nếu khai báo quá thừa thì lãng phí bộ nhớ.

- Cài đặt bằng con trỏ thích hợp cho sự biến động của danh sách, danh sách có thể rỗng hoặc lớn tuỳ ý chỉ phụ thuộc vào bộ nhớ tối đa của máy. Tuy nhiên ta phải tốn thêm vùng nhớ cho các con trỏ (trường next).

- Cài đặt bằng mảng thì thời gian xen hoặc xoá một phần tử tỉ lệ với số phần tử đi sau vị trí xen/ xóa. Trong khi cài đặt bằng con trỏ các phép toán này mất chỉ một hằng thời gian.

- Phép truy nhập vào một phần tử trong danh sách, chẳng hạn như PREVIOUS, chỉ tốn một hằng thời gian đối với cài đặt bằng mảng, trong khi đối với danh sách cài đặt bằng con trỏ ta phải tìm từ đầu danh sách cho đến vị trí trước vị trí của phần tử hiện hành.Nói chung *danh sách liên kết thích hợp với danh sách có nhiều biến động*, tức là ta thường xuyên thêm, xoá các phần tử.

?

Cho biết ưu khuyết điểm của danh sách đặc và danh sách liên kết?

d. Cài đặt bằng con nháy

Một số ngôn ngữ lập trình không có cung cấp kiểu con trỏ. Trong trường hợp này ta có thể "giả" con trỏ để cài đặt danh sách liên kết. Ý tưởng chính là: dùng mảng để chứa các phần tử của danh sách, các "con trỏ" sẽ là các biến số nguyên (**int**) để giữ chỉ số của phần tử kế tiếp trong mảng. Để phân biệt giữa "con trỏ thật" và "con trỏ giả" ta gọi các con trỏ giả này là con nháy (cursor). Như vậy để cài đặt danh sách bằng con nháy ta cần một mảng mà mỗi phần tử xem như là một ô gồm có hai trường: trường Element như thông lệ giữ giá trị của phần tử trong danh sách (có kiểu Elementtype) trường Next là con nháy để chỉ tới vị trí trong mảng của phần tử kế tiếp. Chẳng hạn hình II.6 biểu diễn cho mảng SPACE đang chứa hai danh sách L_1, L_2. Để quản lí các danh sách ta cũng cần một con nháy chỉ đến phần tử đầu của mỗi danh sách (giống như header trong danh sách liên kết). Phần tử cuối cùng của danh sách ta cho chỉ tới giá trị đặc biệt **Null**, có thể xem **Null** = -1 với một giả thiết là mảng SPACE không có vị trí nào có chỉ số -1.

Trong hình II.6, danh sách L_1 gồm 3 phần tử : f, o ,r. Chỉ điểm đầu của L_1 là con nháy có giá trị 5, tức là nó trỏ vào ô lưu giữ phần tử đầu tiên của L_1, trường next của ô này có giá trị 1 là ô lưu trữ phần tử kế tiếp (tức là o). Trường next tại ô chứa o là 4 là ô lưu trữ phần tử kế tiếp trong danh sách (tức là r). Cuối cùng trường next của ô này chứa **`Null`** nghĩa là danh sách không còn phần tử kế tiếp.

Phân tích tương tự ta có L_2 gồm 4 phần tử : w, i, n, d

Chỉ điểm của danh sách thứ nhất L_1 →

Chỉ điểm của danh sách thứ hai L_2 →

Chỉ số	Elements	Next
0	d	**Null**
1	o	4
2		
3	n	0
4	r	**Null**
5	f	1
6	i	3
7	w	6
8		
9		

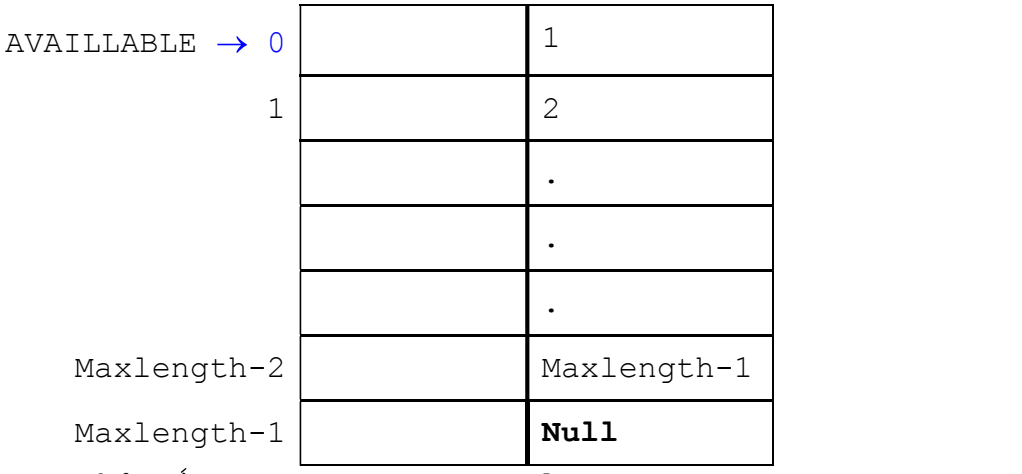
Mảng SPACE

Hình II.6 Mảng đang chứa hai danh sách L_1 và L_2

Khi xen một phần tử vào danh sách ta lấy một ô trống trong mảng để chứa phần tử mới này và nối kết lại các con nháy. Ngược lại, khi xoá một phần tử khỏi danh sách ta nối kết lại các con nháy để loại phần tử này khỏi danh sách, điều này kéo theo số ô trống trong mảng tăng lên 1. Vấn đề là làm thế nào để quản lí các ô trống này để biết ô nào còn trống ô nào đã dùng? một giải pháp là *liên kết tất cả các ô trống vào một danh sách đặc biệt gọi là AVAILABLE*, khi xen một phần tử vào danh sách ta lấy ô trống đầu AVAILABLE để chứa phần tử mới này. Khi xoá một phần tử từ danh sách ta cho ô bị xoá nối vào đầu AVAILABLE. Tất nhiên khi mới khởi đầu việc xây dựng cấu trúc thì mảng chưa chứa phần tử nào của bất kỳ một danh sách nào. Lúc này tất cả các ô của mảng đều là ô trống, và như vậy, tất cả các ô đều được liên kết vào trong AVAILABLE. Việc khởi tạo AVAILABLE ban đầu có thể thực hiện bằng cách cho phần tử thứ i của mảng trỏ tới phần tử i+1.

Các khai báo cần thiết cho danh sách

```
#define MaxLength ... //Chieu dai mang
#define Null -1 //Gia tri Null
typedef ... ElementType; /*kiểu của các phần tử
                           trong danh sách*/
typedef struct{
   ElementType Elements; /*trường chứa phần tử
                           trong danh sách*/
   int Next; //con nháy trỏ đến phần tử kế tiếp
} Node;
Node Space[MaxLength]; //Mang toan cuc
int Available;
```



Mảng **SPACE**

Hình II.7: Khởi tạo Available ban đầu

Khởi tạo cấu trúc – Thiết lập available ban đầu

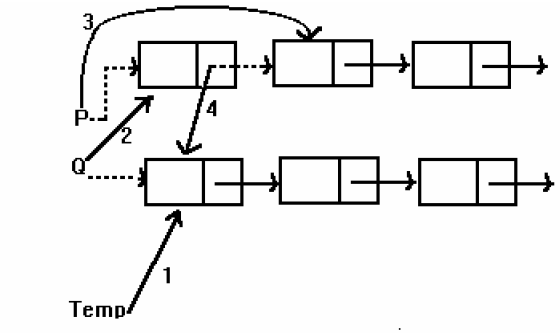
Ta cho phần tử thứ 0 của mảng trỏ đến phần tử thứ 1,..., phần tử cuối cùng trỏ **Null**. Chỉ điểm đầu của AVAILABLE là 0 như trong hình II.7

```
void Initialize(){
   int i;
   for(i=0;i<MaxLength-1;i++)
        Space[i].Next=i+1;
```

```
    Space[MaxLength-1].Next=NULL;
    Available=0;
}
```

Chuyển một ô từ danh sách này sang danh sách khác

Ta thấy thực chất của việc xen hay xoá một phần tử là thực hiện việc chuyển một ô từ danh sách này sang danh sách khác. Chẳng hạn muốn xen thêm một phần tử vào danh sách L_1 trong hình II.6 vào một vị trí p nào đó ta phải chuyển một ô từ AVAILABLE (tức là một ô trống) vào L_1 tại vị trí p; muốn xoá một phần tử tại vị trí p nào đó trong danh sách L_2, chẳng hạn, ta chuyển ô chứa phần tử đó sang AVAILABLE, thao tác này xem như là giải phóng bộ nhớ bị chiếm bởi phần tử này. Do đó tốt nhất ta viết một hàm thực hiện thao tác chuyển một ô từ danh sách này sang danh sách khác và hàm cho kết quả kiểu int tùy theo chuyển thành công hay thất bại (là 0 nếu chuyển không thành công, 1 nếu chuyển thành công). Hàm **Move** sau đây thực hiện chuyển ô được trỏ tới bởi con nháy P vào danh sách khác được trỏ bởi con nháy Q như trong hình II.8. Hình II.8 trình bày các thao tác cơ bản để chuyển một ô (ô được chuyển ta tạm gọi là ô mới):



Hình II.8

Chuyển 1 ô từ danh sách này sang danh sách khác (các liên kết vẽ bằng nét đứt biểu diễn cho các liên kết cũ - trước khi giải thuật bắt đầu)

- Dùng con nháy temp để trỏ ô được trỏ bởi Q.
- Cho Q trỏ tới ô mới.
- Cập nhật lại con nháy P bằng cách cho nó trỏ tới ô kế tiếp.
- Nối con nháy trường next của ô mới (ô mà Q đang trỏ) trỏ vào ô mà temp đang trỏ.

```
int Move(int *p, int *q){
    int temp;
    if (*p==Null)
```