HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ET2100

ONE LOVE. ONE FUTURE.

Các bước xây dựng một CTDL

- Bước 1: Xác định đầy đủ các đặc trưng của CTDL gồm:
  - Các thành phần DL có trong CTDL đó,
  - Các liên kết (quan hệ) về cấu trúc giữa các thành phần DL.

- Bước 2: Xác định các thao tác cơ bản trên CTDL: là các thao tác cơ bản, cần thiết nhất để có thể sử dụng được CTDL này.

- Bước 3: Xác định các giải thuật cần thiết cho các thao tác trên. Khi có nhiều giải thuật cho một thao tác, thì giải thuật có độ phức tạp nhỏ nhất sẽ được chọn.

- Bước 4: Xác định cấu trúc lưu trữ thích hợp để tổ chức lưu trữ CTDL một cách có hiệu quả. Tính hiệu quả thể hiện ở cả hai mặt: kích thước lưu trữ nhỏ nhất và tốc độ thực hiện các thao tác là nhanh nhất.

Đặt vấn đề - Xây dựng một CTDL

- Bước 5: Cài đặt các thao tác cơ bản. Việc cài đặt các thao tác phải theo một số nguyên tắc sau:
  - Thao tác có khả năng sử dụng lại nhiều lần: sử dụng chương trình con để cài đặt
  - Thao tác có tính độc lập về mặt sử dụng và độc lập với các thao tác khác. Để đảm bảo tính chất này thì ta phải chọn các tham số hợp lí cho các thao tác
  - Thao tác phải hiệu quả: chọn giải thuật tốt nhất để cài đặt

Chương II:
Cấu trúc mảng và Danh sách tuyến tính

Giảng viên: TS. Đỗ Thị Ngọc Diệp
Khoa Kỹ thuật Truyền thông – Trường Điện Điện Tử

ONE LOVE. ONE FUTURE.

Các nội dung chính

1. Cấu trúc mảng
   - 1.1. Mô tả
   - 1.2. Cấu trúc lưu trữ tuần tự
   - 1.3. Cài đặt mảng bằng cấu trúc lưu trữ tuần tự
2. Cấu trúc danh sách tuyến tính
   - 2.1. Mô tả
   - 2.2. Cài đặt danh sách bằng cấu trúc lưu trữ tuần tự
   - 2.3. Cài đặt danh sách bằng cấu trúc lưu trữ móc nối
   - 2.4. Một số ứng dụng của ngăn xếp và hàng đợi

1. Cấu trúc mảng

1.1. Mô tả

- *Mảng* (Array) là một tập cố định các phần tử và cùng kiểu dữ liệu.
- Tính chất đặc trưng
  - *Số chiều*: số chiều của mảng tương ứng với số chiều của thông tin cần được biểu diễn.
    - Một mảng bao giờ cũng ít nhất một chiều.
  - *Kích thước mỗi chiều*: giá trị cố định.
  - *Kích thước của mảng*: tích tất cả các kích thước của tất cả các chiều.
  - *Kiểu phần tử mảng*: kiểu dữ liệu cho mỗi phần tử của mảng.

1.1. Mô tả

- Khai báo:

  ARRAY : <*name*>[*dimension, len 1, len 2,..., len n*] **OF** *datatype* ;

- Kích thước của mảng *name*: *LEN(name)*
  - *LEN(name) = len 1 x len 2 x ... =* Π *(len i)* (với i=1,2,..,n)

Ví dụ:

Khai báo mảng 1 chiều:

```
ARRAY: vector [1, N] OF  integer ;
```

Khai báo mảng hai chiều:

```
ARRAY: matran[2, M, N] OF integer;
ARRAY: matran[1, M] OF vector;
```

Khai báo mảng N chiều:

```
ARRAY: a[N, L1, L2,..., LN]  OF  integer;
```

1.1. Mô tả

- Mảng N chiều là mảng 1 chiều của các mảng N-1 chiều.
  - ARRAY: $a_n[N, L_1, L_2,..., L_N]$ OF datatype ;

  ⇔
  - ARRAY: $a_{n-1}[N-1, L_2,..., L_N]$ OF datatype ; AND ARRAY: $a_n[1, L_1]$ OF a_{n-1} ;

  - Khai báo mảng trong C/C++:
    - int vector [N]; // => Chỉ số chạy tương ứng [0..N-1]
    - int matran [M][N]; // => Chỉ số chạy tương ứng [0..M-1][0..N-1]
    - vector matran[M]; // => Chỉ số chạy tương ứng [0..M-1];
    - Lưu ý trong C/C++ quy ước:
      - Kích thước mỗi chiều = chỉ số + 1

1.1. Mô tả

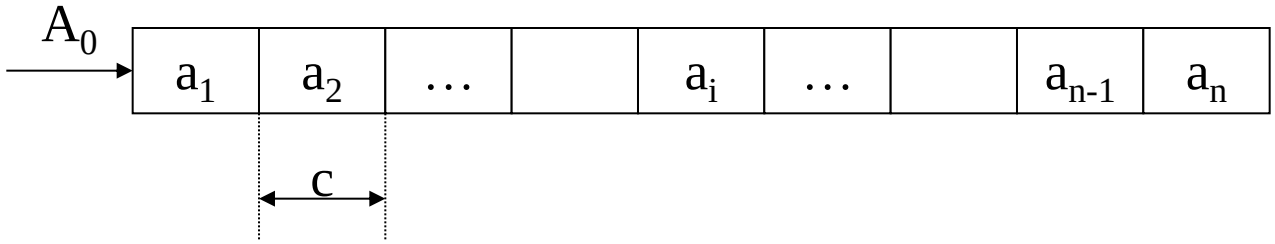
- ***Các thao tác cơ bản:***
- Khai báo:
  - Khai báo mảng, xác định các đặc trưng của mảng
  - Luôn được tiến hành trước tiên
- Truy nhập vào các phần tử của mảng: để sử dụng các phần tử này (lấy giá trị, cập nhật giá trị).
  - Sử dụng *chỉ số (index)* gắn với phần tử đó.
  - Mỗi phần tử của mảng có một chỉ số duy nhất, có vai trò như địa chỉ của phần tử trong mảng.

Ví dụ

- Truy nhập phần tử thứ i của vector (mảng 1 chiều): *vector[i]*
- Truy nhập phần tử ở hàng i, cột j của ma trận (mảng 2 chiều): *matran[i,j]*.
- Mảng có N chiều $a_n[N, L_1, L_2,..., L_N]$
  - Cấu trúc chỉ số của mỗi phần tử: $[i_1,i_2,...,i_N]$, với i_j (j = 1..N) là các số nguyên thoả mãn: $1 \leq i_j \leq L_j$.

1.2. Cấu trúc lưu trữ tuần tự

- Mô tả
  - SD các ô nhớ liên tiếp, có kích thước bằng nhau để chứa các phần tử, mỗi ô cho một phần tử.



  - A_0 : địa chỉ bắt đầu của cấu trúc lưu trữ, địa chỉ của ô nhớ chứa phần tử đầu tiên.
  - Kích thước mỗi ô nhớ là như nhau, là một hằng số cố định được kí hiệu là c (đơn vị tính thường là byte).
  - Địa chỉ của a_i: Loc (a_i) = A_0 + c* (i-1)

1.2. Cấu trúc lưu trữ tuần tự

- Đặc điểm
  - Đơn giản, dễ sử dụng
  - Kích thước luôn cố định.
    - Cấp phát/giải phóng vùng nhớ cho CTLT này được thực hiện đúng một lần.
  - Truy nhập vào các phần tử nhanh và đồng đều (truy nhập trực tiếp)
    - Địa chỉ mỗi phần tử có thể tính trực tiếp
  - => Là cấu trúc lưu trữ (CTLT) cơ bản nhất được hỗ trợ trong các môi trường lập trình để cài đặt cho cấu trúc mảng

1.3. Cài đặt mảng bằng CTLT tuần tự

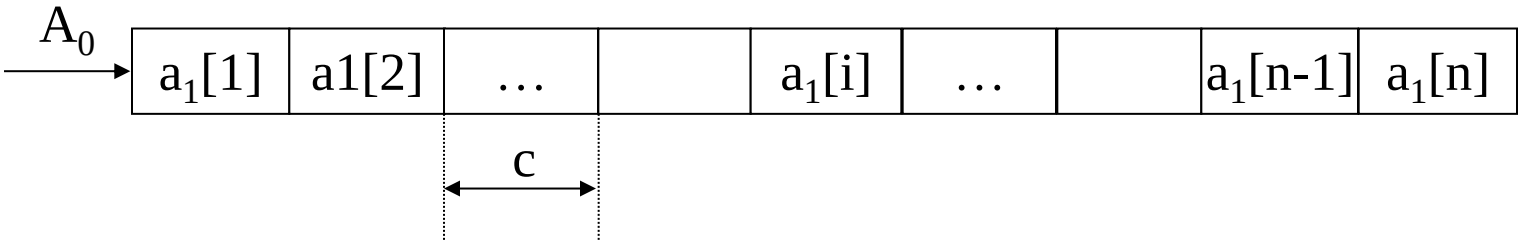
- 2 bước:
  - Xác định các đặc trưng của CTLT:
    - Dựa vào các đặc điểm của cấu trúc mảng cần cài đặt, xác định đủ kích thước của CTLT để có thể lưu trữ đủ toàn bộ các phần tử của mảng.
  - Bố trí hợp lý các phần tử của mảng vào CTLT:
    - Có nhiều cách bố trí
    - Cần chọn ra cách bố trí giúp thao tác truy nhập vào các phần tử là nhanh nhất và dễ dùng nhất cho người dùng.

1.3. Cài đặt mảng bằng CTLT tuần tự

- Cài đặt mảng một chiều:
  - `ARRAY : a`$_1$`[1, N] OF` *`datatype`* `;`
  - Bước 1: xác định các đặc trưng của cấu trúc lưu trữ:
    - Số ô nhớ = N = số phần tử của mảng = kích thước của mảng
    - Kích thước mỗi ô nhớ: *c* = kích thước của kiểu dữ liệu *datatype*
    - Cần dành ra một khối nhớ liên tục có kích thước *c*N*, có địa chỉ đầu tiên là A_0 để lưu trữ cho mảng này.

- Cài đặt mảng một chiều:
  - `ARRAY : a1[1, N] OF datatype ;`
  - Bước 2: Bố trí các phần tử của mảng vào CTLT đã chọn:
    - Bố trí lần lượt các phần tử của mảng vào trong các ô nhớ của CTLT tuần tự
      - Phần tử thứ *i* của mảng sẽ được lưu trữ ở ô nhớ thứ *i* ($1 \le i \le N$, với N là kích thước của mảng).
    - Địa chỉ tuyệt đối của phần tử thứ *i* - $a_1[i]$: $A_i = A_0 + c * (i-1)$

Hàm địa chỉ

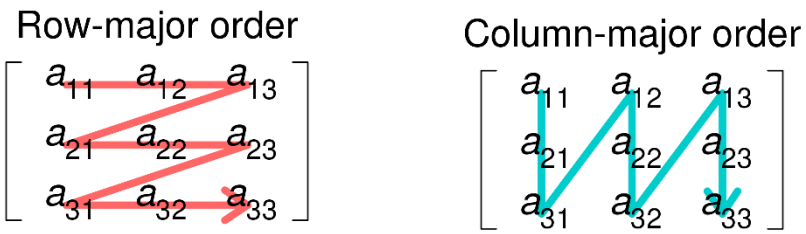


Tránh nhầm lẫn số thứ tự phần tử với chỉ số mảng trong C/C++

1.3. Cài đặt mảng bằng CTLT tuần tự

- Cài đặt mảng hai chiều:
  - `ARRAY : a`$_2$`[2,M,N] OF datatype ;`
  - CTDL mảng là đa chiều vs. CTLT tuần tự chỉ có một chiều.
  - => Chuyển đổi từ cấu trúc đa chiều sang cấu trúc một chiều và ngược lại.
  - Bước 1: xác định các đặc trưng của CTLT tuần tự:
    - Số ô nhớ : M*N, là kích thước của mảng.
    - Kích thước mỗi ô nhớ : là kích thước của *datatype*.
    - Ta cần dành ra một khối nhớ liên tục có kích thước *c*M*N*, có địa chỉ đầu tiên là A_0 để lưu trữ cho mảng này.

- Cài đặt mảng hai chiều:
  - `ARRAY : a2[2,M,N] OF datatype ;`
  - Bước hai: bố trí các phần tử:
    - Có hai phương pháp bố trí:
    - *Theo thứ tự ưu tiên hàng*: các phần tử của mảng sẽ được bố trí lần lượt theo từng hàng, hết hàng này đến hàng kia theo thứ tự các hàng từ trên xuống dưới.
    - *Theo thứ tự ưu tiên cột*: các phần tử của mảng được bố trí theo từng cột, hết cột này đến cột kia theo thứ tự các cột từ trái sang phải.



2 cách bố trí

mt[1,1]	mt[1,2]
mt[2,1]	mt[2,2]
mt[3,1]	mt[3,2]
...	...

Theo thứ tự ưu tiên hàng

mt[1,1]	mt[1,2]
mt[2,1]	mt[2,2]
mt[3,1]	mt[3,2]
...	...

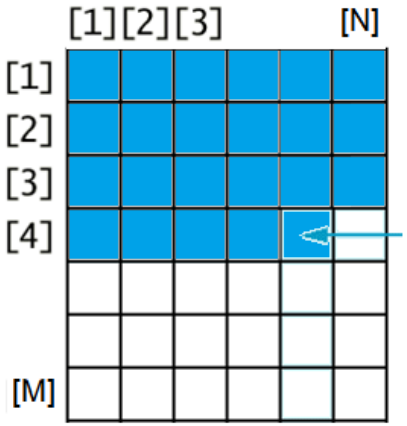
Theo thứ tự ưu tiên cột

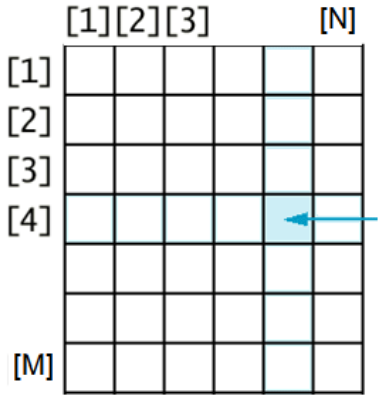
2 cách bố trí

`ARRAY : a2[2,M,N] OF datatype ;`

- *Theo thứ tự ưu tiên hàng*
  - `a2[i,j]` được lưu trữ ở ô nhớ thứ `z`? `z = N*(i-1) + j`
  - Ngược lại, ở ô nhớ thứ z sẽ lưu trữ phần tử `a2[i,j]` ?

`i = (z - 1) DIV N + 1` và `j = (z-1) MOD N +1`



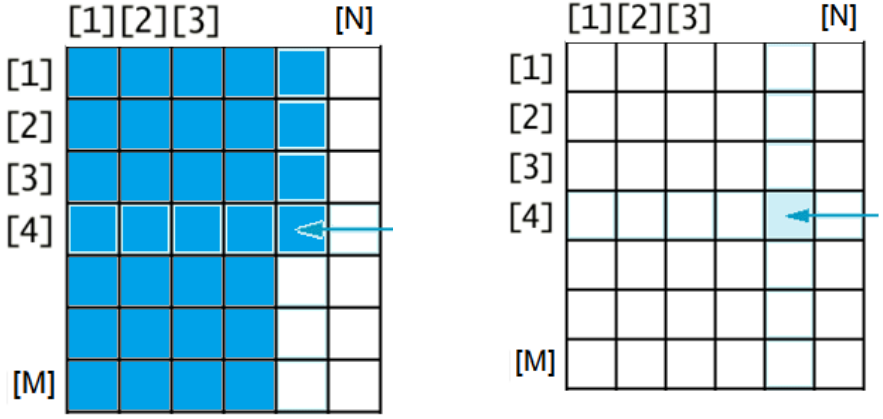


2 cách bố trí

ARRAY : a_2[2,M,N] OF *datatype* ;

- *Theo thứ tự ưu tiên cột*
  - a_2[i,j] được lưu trữ ở ô nhớ thứ z? z = M*(j-1) + i
  - Ngược lại, ở ô nhớ thứ z sẽ lưu trữ phần tử a_2[i,j] với?

    i = (z -1) MOD M +1 và j = (z-1) DIV M +1



Cài đặt mảng nhiều hơn hai chiều ?

1.4. Hàm địa chỉ

- Phương pháp để chuyển đổi từ cấu trúc đa chiều của mảng sang cấu trúc lưu trữ một chiều:
  - Xác định một ánh xạ 1-1 từ các phần tử của mảng lên các ô nhớ của cấu trúc lưu trữ.
- Khái niệm hàm địa chỉ
  - Hàm địa chỉ có dạng:
    - $f : X \rightarrow Y$
  - Với:
    - X là tập các phần tử của một mảng
    - Y là địa chỉ các ô nhớ của cấu trúc lưu trữ tuần tự

1.4. Hàm địa chỉ

- Hàm địa chỉ cho mảng một chiều:
  - `ARRAY: vector[1,N] OF datatype ;`
  - Mảng này được lưu trữ bằng cấu trúc lưu trữ tuần tự.
- Địa chỉ phần tử thứ i: $A_i = A_0 + c(i-1)$;
- Gọi hàm địa chỉ là f_1 thì nó sẽ có dạng:
  - $f_1(i) = A_0 + c*(i-1)$
- Để đơn giản, giả sử $A_0 = 0$;
  - $f_1(i) = c*(i-1)$

1.4. Hàm địa chỉ

- Hàm địa chỉ cho mảng hai chiều
  - `ARRAY : mt[2,M,N] OF datatype ;`
  - Xét phương pháp lưu trữ theo thứ tự ưu tiên hàng
    - f_2`(i,j) = c *(N * (i-1) + j - 1)`
  - Xét phương pháp lưu trữ theo thứ tự ưu tiên cột
    - f_2`(i,j) = c *(M * (j-1) + i - 1)`

1.4. Hàm địa chỉ

- Hàm địa chỉ cho mảng nhiều hơn hai chiều
  - ARRAY: a_n[N, L_1, L_2,..., L_n] OF datatype ;

$\Leftrightarrow$

  - ARRAY: a_n[1, L_1] OF a_{n-1} ; AND ARRAY: a_{n-1}[N-1, L_2,..., L_n] OF datatype;

- Để tính hàm địa chỉ của a_n, ta dựa vào hàm địa chỉ của a_{n-1}

$$f_n(i_1, i_2, ..., i_n) = c(\prod_{i=2}^{n} L_i)(i_1 - 1) + f_{n-1}(i_2, i_3, ..., i_n)$$

$$\rightarrow \quad f_n(i_1, i_2, ..., i_n) = c\left[(\prod_{i=2}^{n} L_i)(i_1 - 1) + (\prod_{i=3}^{n} L_i)(i_2 - 1) + ... + (i_n - 1)\right]$$

Nhận xét

- Lợi ích của sử dụng cấu trúc lưu trữ tuần tự để cài đặt mảng :
  - Cách cài đặt tương đối đơn giản, dễ mở rộng cho mảng có số chiều khác nhau.
  - Tính được hàm địa chỉ duy nhất cho tất cả các phần tử trong mảng, thao tác truy nhập vào các phần tử của mảng đều nhanh chóng và đồng đều như nhau.
- Hạn chế:
  - Chỉ phù hợp khi mảng có kích thước cố định và đồng nhất
  - Đòi hỏi vùng nhớ liên tục trong bộ nhớ => khó khăn, vì trong thực tế vùng nhớ của máy tính thường bị phân mảnh
  - Thêm, xóa phần tử trong mảng cần định vị lại các khối nhớ mới, thao tác phức tạp trên bộ nhớ, chậm, v.v.

2. Danh sách tuyến tính

Danh sách tuyến tính

- 2.1. Mô tả
  - Mô tả cấu trúc
  - Các thao tác trên danh sách
  - Các cấu trúc danh sách thông dụng
  - Các phương pháp cài đặt danh sách
- 2.2. Cài đặt danh sách bằng cấu trúc lưu trữ tuần tự
- 2.3. Cài đặt danh sách bằng cấu trúc lưu trữ móc nối

- Danh sách tuyến tính: CTDL gồm một hay nhiều phần tử cùng kiểu dữ liệu và tồn tại một trật tự tuyến tính giữa các phần tử.
- Kí hiệu: $L = <x_1, x_2, ..., x_n>$
  - $n \geq 1$ và $x_1, x_2, ..., x_n$ là các phần tử của danh sách
  - x_1 được gọi là phần tử đầu tiên (đầu) của danh sách
  - x_n được gọi là phần tử cuối cùng (đuôi) của danh sách
- Trật tự tuyến tính: trật tự *trước-sau* giữa các phần tử
  - Mọi cặp phần tử $<x_i, x_j>$ ($1 \leq i,j \leq n$ và $i \neq j$) luôn có duy nhất một trật tự trước sau.
- Quy ước: danh sách rỗng, kí hiệu ∅ (L = ∅).

- Kích thước hay độ dài danh sách: số phần tử của danh sách.
  - Kích thước của danh sách rỗng bằng 0.
  - Kích thước danh sách không cố định, không biết trước mà biến đổi trong quá trình xử lý, thao tác
- Kiểu dữ liệu của các phần tử:
  - Có một kiểu dữ liệu duy nhất luôn luôn cố định
- Trật tự tuyến tính trong danh sách:
  - Một danh sách luôn có hai phía: đầu, đuôi của danh sách
  - Trật tự trước-sau là trật tự từ đầu đến cuối

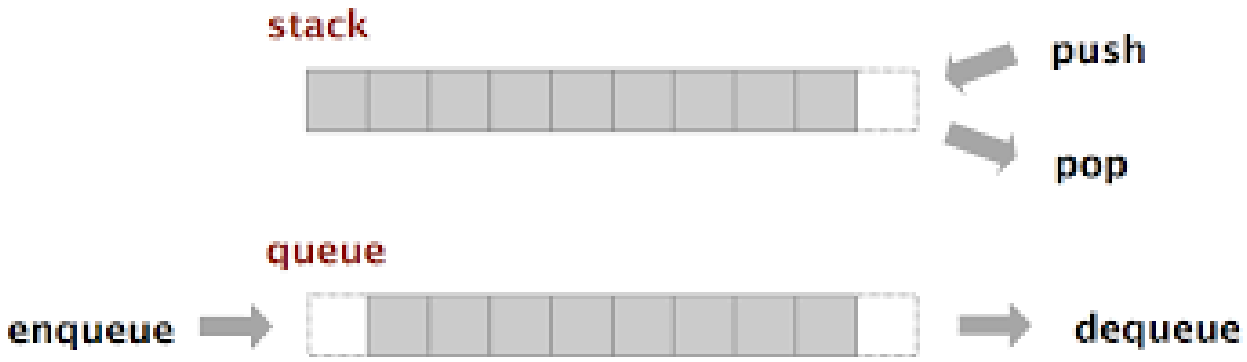
- Khởi tạo danh sách
  - Định ra cấu trúc danh sách, xác định các đặc trưng của danh sách.
  - Sau thao tác khởi tạo, thu được một danh sách rỗng (danh sách chưa có nội dung, mà mới chỉ có phần khung).
  - Trong các NNLT = khai báo cấu trúc lưu trữ thích hợp để biểu diễn cho cấu trúc danh sách yêu cầu.

- Bổ sung phần tử mới vào danh sách
  - Xác định vị trí trong danh sách mà phần tử mới sẽ được đưa vào: đầu, cuối, giữa danh sách.
  - Mỗi thao tác bổ sung này làm tăng kích thước danh sách lên 1 hoặc nhiều.
  - Chú ý, điều kiện để thực hiện được thao tác bổ sung là danh sách chưa "*đầy*" hay chưa bão hoà.

- Loại bỏ phần tử khỏi danh sách
  - Xác định vị trí loại bỏ: phần tử đầu, cuối, giữa, toàn bộ.
  - Giảm kích thước của danh sách đi 1 hoặc nhiều.
  - Điều kiện để loại bỏ một phần tử là danh sách phải không rỗng, tức là nó phải còn ít nhất một phần tử.

- Tìm kiếm một phần tử trong danh sách:
  - Tìm kiếm theo một điều kiện tìm kiếm, nếu xuất hiện thì ở vị trí nào trong danh sách.
  - Thao tác tìm kiếm thường đi đôi với các thao tác bổ sung và loại bỏ
- Sắp xếp danh sách:
  - Sắp xếp các phần tử của danh sách theo một trật tự nhất định
- Nối hai hay nhiều danh sách
- Tách danh sách thành hai hay nhiều danh sách con

- Cấu trúc vào sau ra trước (Last In, First Out - LIFO)
  - Cấu trúc ngăn xếp – Stack
- Cấu trúc vào trước ra trước (First In, First Out - FIFO)
  - Cấu trúc *hàng đợi* – Queue



- Cấu trúc *hàng đợi có ưu tiên* - Priority queue
  - Ở đây mỗi phần tử có thêm một thuộc tính gọi là độ ưu tiên.
  - Với các phần tử có độ ưu tiên như nhau thì danh sách hoạt động giống như nguyên tắc hàng đợi.
  - Khi có hai phần tử bất kỳ có độ ưu tiên khác nhau thì danh sách luôn ưu tiên phần tử có độ ưu tiên cao hơn
    - Phần tử có độ ưu tiên cao hơn được bổ sung vào sau phần tử có độ ưu tiên thấp hơn thì nó vẫn được loại bỏ ra trước.
  - Thông thường giá trị của phần tử được dùng là độ ưu tiên của phần tử
  - Cài đặt:
    - Không sắp xếp trước
    - Sắp xếp trước

- Cài đặt bằng cấu trúc lưu trữ tuần tự (hay mảng một chiều) (2.2):
  - Đơn giản và nhanh nhất.
  - Hạn chế : do sự khác nhau cơ bản giữa cấu trúc danh sách và cấu trúc mảng, giữa một bên là cấu trúc động, một bên là cấu trúc tĩnh.
- Cài đặt bằng cấu trúc lưu trữ móc nối (2.3):
  - Cấu trúc lưu trữ động với kích thước và tổ chức lưu trữ có thể biến đổi linh hoạt theo yêu cầu của cấu trúc dữ liệu
  - Thích hợp để tổ chức và lưu trữ các cấu trúc dữ liệu động.

- Nguyên tắc chung:
  - Tính đầy đủ: CTLT phải có các đặc trưng thích hợp để biểu diễn đầy đủ các đặc tính và khả năng của CTDL.
    - **Khả năng lưu trữ:** là kích thước, miền giá trị của cấu trúc dữ liệu mà nó cài đặt.
    - **Khả năng xử lý:** là tập các thao tác được cài đặt trên CTLT đã chọn.
  - Tính hiệu quả: cài đặt phải đảm bảo các thao tác xử lý chạy nhanh nhất và cấu trúc lưu trữ sử dụng bộ nhớ tiết kiệm nhất.
  - Tính mở rộng, và che giấu

=> Khó có thể đồng thời đạt được cả hai tiêu chí này, trong thực tế phải có các biện pháp dung hoà, ưu tiên tiêu chí này hay tiêu chí kia trong từng ứng dụng cụ thể.

2.2. Cài đặt danh sách bằng CTLT tuần tự

- Nội dung
  - Mô tả
    - Cài đặt danh sách bằng cấu trúc mảng
    - Cấu trúc ngăn xếp (stack/LIFO)
    - *Cài đặt Stack bằng CTLT tuần tự*
  - Khai báo cấu trúc
  - Cài đặt các thao tác cơ bản

2.2. Cài đặt danh sách bằng CTLT tuần tự
Mô tả

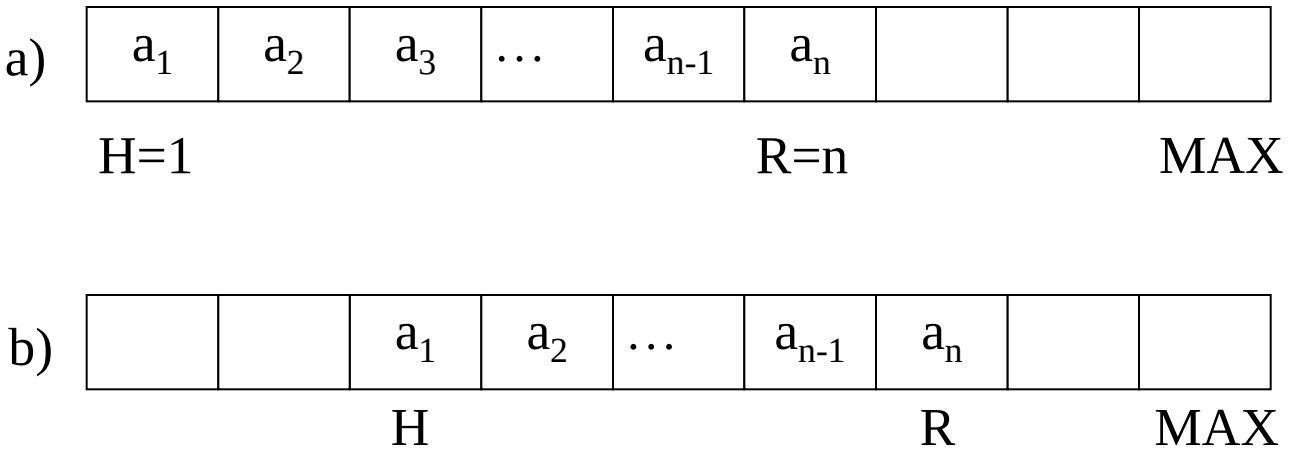
- CTLT tuần tự: cấu trúc mảng
- Cài đặt danh sách bằng cấu trúc mảng
- Tương như như khi cài đặt mảng, quá trình cài đặt một danh sách sử dụng cấu trúc lưu trữ tuần tự cũng gồm hai bước:
  - Bước 1: Chọn cấu trúc lưu trữ phù hợp.
  - Bước 2: Bố trí hợp lý các phần tử của danh sách trong cấu trúc lưu trữ.
  - => Cài đặt chi tiết bằng cách sử dụng một ngôn ngữ lập trình cụ thể

- Giả sử có danh sách: L=<a_1,a_2,...,a_n>
  - Chọn cấu trúc lưu trữ phù hợp
    - Xác định kích thước tối đa MAX của danh sách
    - Xác định kiểu dữ liệu D cho mỗi phần tử của danh sách
    - Chọn CTLT là mảng một chiều kích thước và kiểu dữ liệu như đã chọn
  - Bố trí hợp lý các phần tử của danh sách trong CTLT đã chọn
    - Bố trí mỗi phần tử trong một ngăn nhớ
    - Bố trí các phần tử lần lượt theo trật tự tuyến tính của chúng
    - Bố trí các phần tử ở các vị trí kề nhau để đảm bảo số thông tin cần quản lý danh sách là ít nhất.
    - Chọn cách bố trí thích hợp để các thao tác cơ bản thực hiện trên danh sách là hiệu quả nhất.

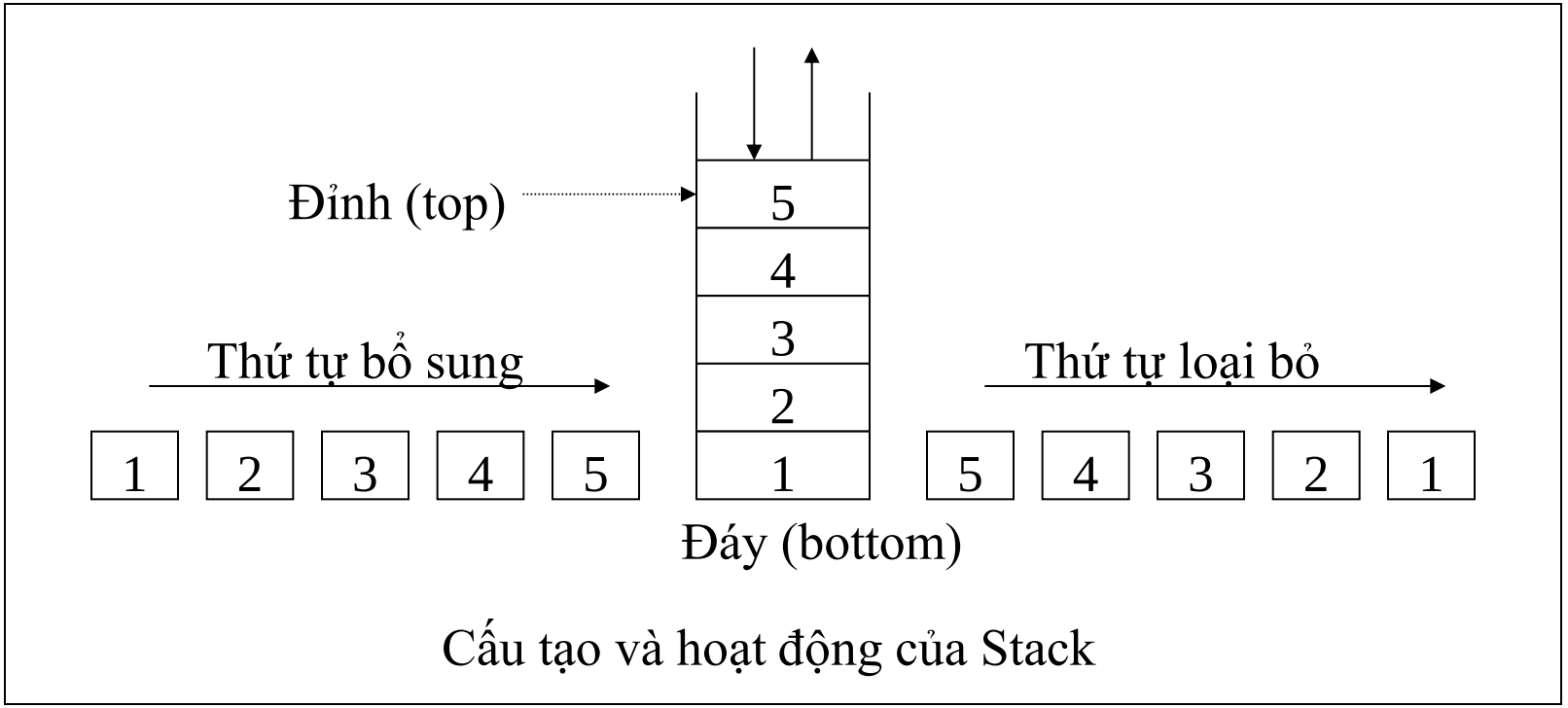
- Một số cách bố trí các phần tử của danh sách trong mảng

a) Bố trí liền từ đầu mảng: *H=1*, *R=n* => cần thông tin *n*

b) Bố trí dãy liên tục, cần hai trong ba thông tin *H*, *R*, *n*.

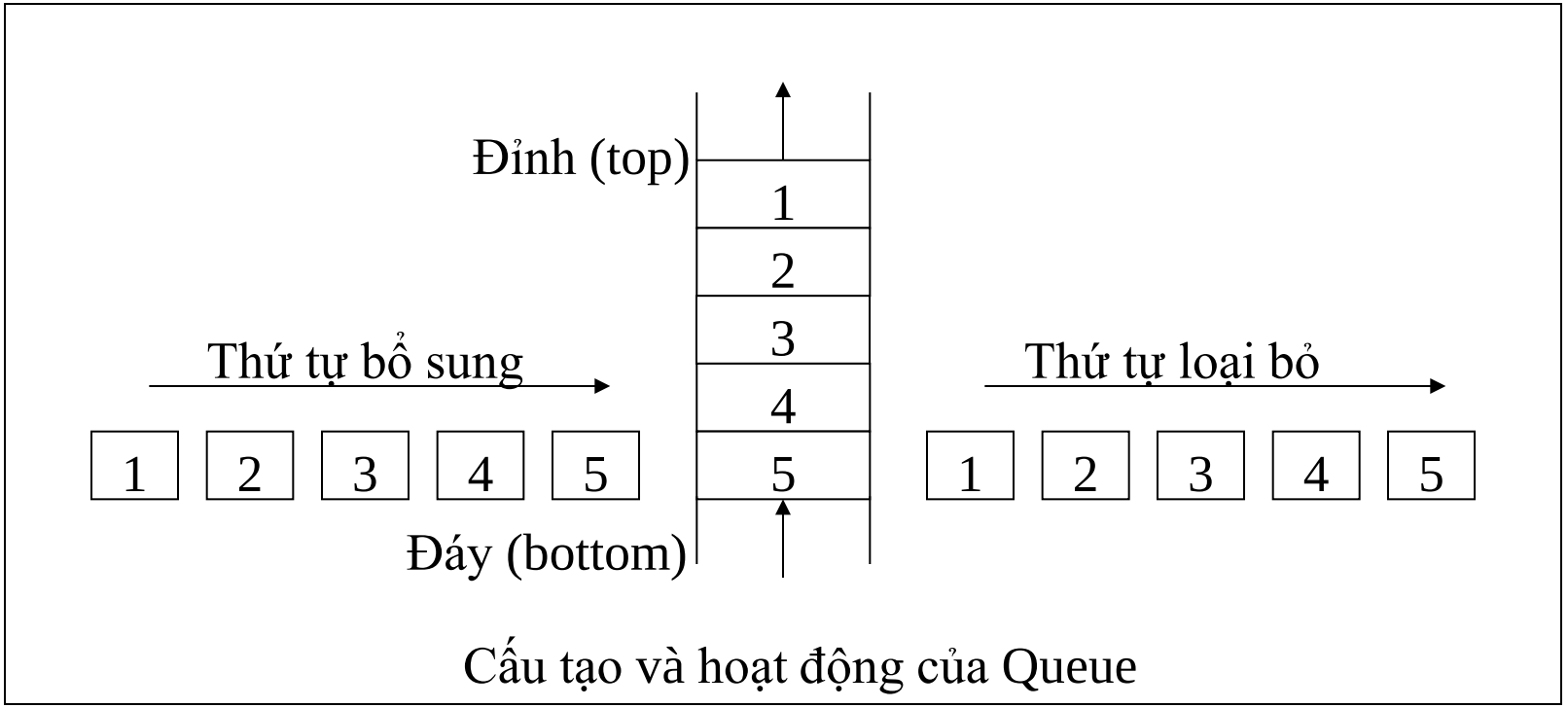


- Một trong các loại danh sách thông dụng:
  - Cấu trúc ngăn xếp (stack): cấu trúc vào sau ra trước (Last In, First Out - LIFO)



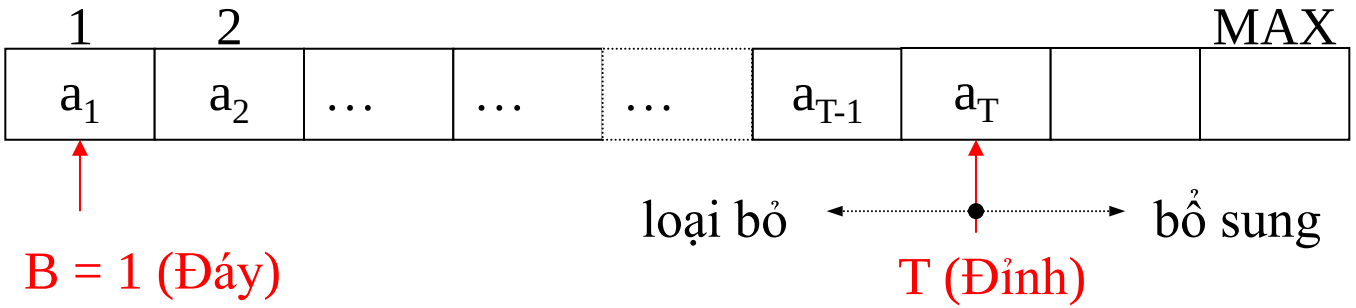
Cấu tạo và hoạt động của Stack

- Một trong các loại danh sách thông dụng:
  - Cấu trúc hàng đợi (queue): cấu trúc vào trước ra trước (First In, First Out - FIFO)



Cấu tạo và hoạt động của Queue

- Nguyên tắc cài đặt
  - Chọn cấu trúc lưu trữ phù hợp
    - Xác định kích thước tối đa MAX của danh sách
    - Xác định kiểu dữ liệu D cho mỗi phần tử của danh sách
    - Chọn CTLT là mảng một chiều kích thước và kiểu dữ liệu như đã chọn
    - Bổ sung thêm hai con trỏ để trỏ vào hai ngăn nhớ, đáy và đỉnh của danh sách => Hai con trỏ di chuyển một cách thích hợp mỗi khi có sự cập nhật nội dung của danh sách



- Khai báo cấu trúc ngăn xếp
  - CTLT tuần tự được thể hiện qua mảng *info*,
  - *D* là kiểu dữ liệu của danh sách
  - *n* là số phần tử của ngăn xếp.
  - Đỉnh của ngăn xếp sẽ ở vị trí *n-1*

```
typedef  struct  {
   D  info [MAX];
   unsigned int n;
}  Stack;
```

// Chú ý phân biệt số lượng phần tử *n* và chỉ số của phần tử trong mảng

// Triển khai trên C

```c
// A structure to represent a stack
typedef  struct  {
    int top;
    unsigned capacity;
    int* array;
} Stack;
```

- Trạng thái của ngăn xếp ở 1 thời điểm:
  - Rỗng (*empty*): n=0
    - Không thể loại bỏ phần tử
  - Đầy (*full*): n=MAX.
    - Không thể bổ sung phần tử
  - Bình thường (*normal*): trạng thái không rỗng mà cũng không đầy.
    - Có thể thực hiện thao tác bổ sung và loại bỏ

- Các thao tác cơ bản:

```
typedef  struct  {
   D  info [MAX];
   unsigned int n;
} Stack;
```

```
void Initialize (Stack *S){
      S→n = 0 ;
}
```

// Triển khai trên C

```
// A structure to represent a stack
typedef  struct  {
    int top;
    unsigned capacity;
    int* array;
} Stack;
```

```
// function to create a stack of given capacity. It initializes size of
// stack as 0
Stack* createStack(unsigned capacity)
{




}
```

- Các thao tác cơ bản:

```
typedef  struct  {
   D  info [MAX];
   unsigned int n;
}  Stack;
```

```
bool IsEmpty (Stack *S){
      if (S→n == 0) return true;
      else return false;
}
```

```
bool IsFull (Stack *S){
      if (S→n == MAX) return true;
      else return false;
}
```

- Các thao tác cơ bản:

// Triển khai trên C

```c
// A structure to represent a stack
typedef  struct  {
    int top;
    unsigned capacity;
    int* array;
} Stack;
```

```c
// Stack is full when top is equal to the last index
int isFull(Stack* stack)
{

}

// Stack is empty when top is equal to -1
int isEmpty(Stack* stack)
{

}
```

- Các thao tác cơ bản:

```
typedef  struct  {
  D  info [MAX];
  unsigned int n;
}  Stack;
```

```
void Push (D K, Stack *S){
      if (IsFull(S)) return;
      S→n++;
      S→info[S→n] = K;
}
```

```
D Pop (Stack *S){
      if (IsEmpty(S)) return NULL;
      return S→info[S→n--];
}
```

```
//Trả về phần tử ở đỉnh của ngăn xếp (không lấy phần tử đó ra)
D Top (Stack *S){
      if (IsEmpty(S)) return NULL;
      return S→info[S→n];

}
```

- Các thao tác cơ bản:

// Triển khai trên C

```c
// A structure to represent a stack
typedef  struct  {
    int top;
    unsigned capacity;
    int* array;
} Stack;
```

```c
// Function to add an item to stack.  It
void push(Stack* stack, int item)
{



}
// Function to remove an item from stack.
int pop(Stack* stack)
{



}
```

```c
// Function to return the top from stack without removing it
int peek(Stack* stack)
{



}
```

Bài Tập Chương 2

1. Viết hàm chèn 1 số vào 1 vị trí trong mảng
2. Viết hàm xóa 1 số ở 1 vị trí trong mảng
3. Viết hàm gộp 2 mảng đã sắp xếp tăng dần thành 1 mảng được sắp xếp tăng dần

4. Viết chương trình khai báo và nhập giá trị cho 1 mảng 2 chiều chỉ sử dụng con trỏ. Số phần tử mỗi chiều được nhập vào bởi người dùng.
5. Viết chương trình đảo ngược 1 dãy số do người dùng nhập vào sử dụng stack lưu trữ tuần tự
6. Viết chương trình triển khai hàng đợi sử dụng mảng (và các hàm khởi tạo, thêm phần tử, xóa phần tử, lấy ra phần tử)
7. Viết chương trình triển khai hàng đợi ưu tiên sử dụng mảng (và các hàm khởi tạo, thêm phần tử, xóa phần tử, lấy ra phần tử) với cấu trúc mỗi phần tử là *struct item*.

```
struct item {
    int value;
    int priority;
};
```

8. Tìm hiểu các thư viện làm việc với *stack*, *queue*, *priority queue* của C/C++

Priority Queue

```
struct item {
    int value;
    int priority;
};
```

```
item pr[100000];
int size = -1;
```

```
void enqueue(int value, int priority)
{
    size++;
    pr[size].value = value;
    pr[size].priority = priority;
}
```

```
void dequeue()
{
    int ind = peek();

    for (int i = ind; i < size; i++) {
        pr[i] = pr[i + 1];
    }
    size--;
}
```

Priority Queue

```
int peek()
{
   int highestPriority = INT_MIN;
   int ind = -1;

  for (int i = 0; i <= size; i++) {
    if (highestPriority == pr[i].priority && ind > -1
        && pr[ind].value < pr[i].value) {
        highestPriority = pr[i].priority;
        ind = i;
      }
      else if (highestPriority < pr[i].priority) {
        highestPriority = pr[i].priority;
        ind = i;
      }
   }
    return ind;
}
```

Bài tập mở rộng

Stack:

- In giá trị số nguyên x không âm dưới dạng nhị phân
- Kiểm tra tính hợp lệ về các cặp {}, [] và () trong một biểu thức
- Mô phỏng thao tác back/forward của các trình duyệt web

Queue:

- Khi lập trình cho window, không thể xóa thư mục nếu trong đó còn các file và các thư mục con. Hãy tạo cơ chế để đảm bảo có thể xóa được một thư mục.
- Bệnh nhân trong bệnh viện đa khoa thường được đánh mã màu: xanh, vàng và đỏ. Đưa ra giải pháp xếp hàng làm xét nghiệm của các bệnh nhân với độ ưu tiên từ đậm đến nhạt.