Cấu trúc dữ liệu & Giải thuật
(Data Structures and Algorithms)

Các cấu trúc dữ liệu

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@fit.hcmus.edu.vn

Nội dung

1. Các cấu trúc dữ liệu cơ bản
2. Cây nhị phân – Binary Trees
3. Các cấu trúc dữ liệu nâng cao

Các cấu trúc dữ liệu cơ bản

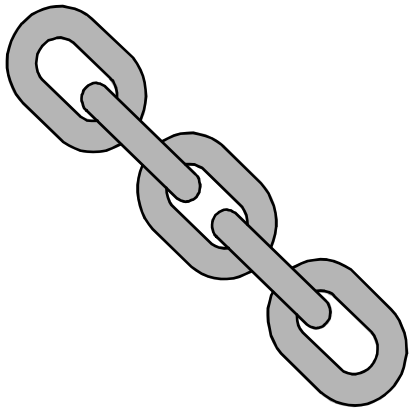
(Fundamental Data Structures)

1.1 Các danh sách liên kết – Linked Lists

1.2 Ngăn xếp – Stack

1.3 Hàng đợi - Queue

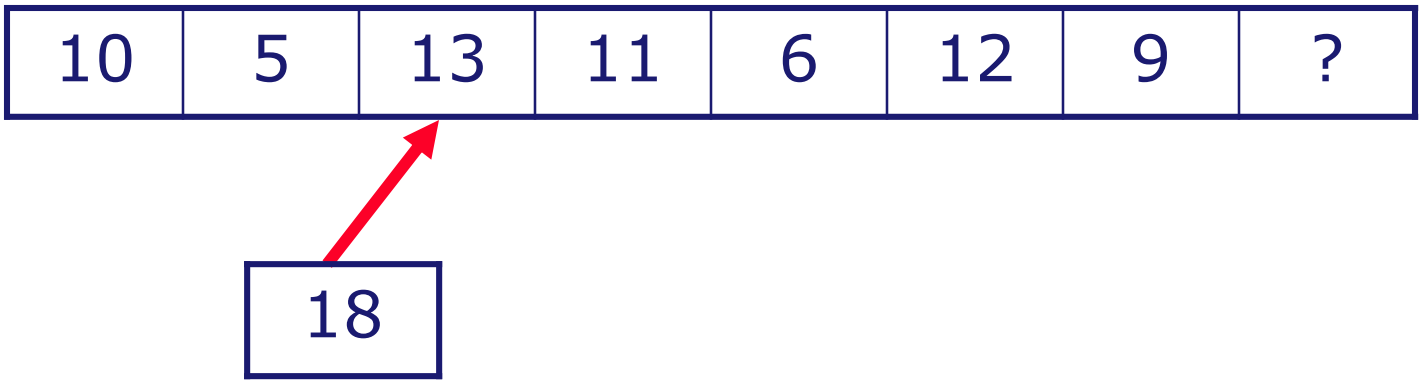
Danh sách liên kết – Linked Lists



- Đặt vấn đề
- Danh sách liên kết là gì ?
- So sánh Mảng và Danh sách liên kết
- Danh sách liên kết đơn (Singly Linked List)
- Danh sách liên kết đôi (Doubly Linked List)

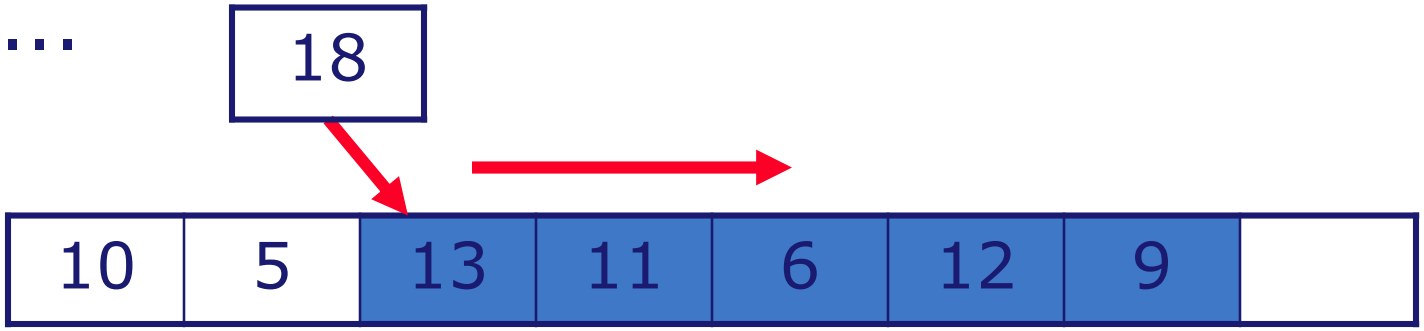
Đặt vấn đề (1)

- Nếu muốn thêm (Insert) 1 phần tử vào mảng, phải làm sao ?



Đặt vấn đề (2)

- Phải di chuyển các phần tử về phía sau 1 vị trí ...



- …rồi chèn phần tử mới vào

- Vậy chi phí là **O(n)**

Đặt vấn đề (3)

- Tương tự, chi phí xóa 1 phần tử trong mảng cũng là **O(n)**

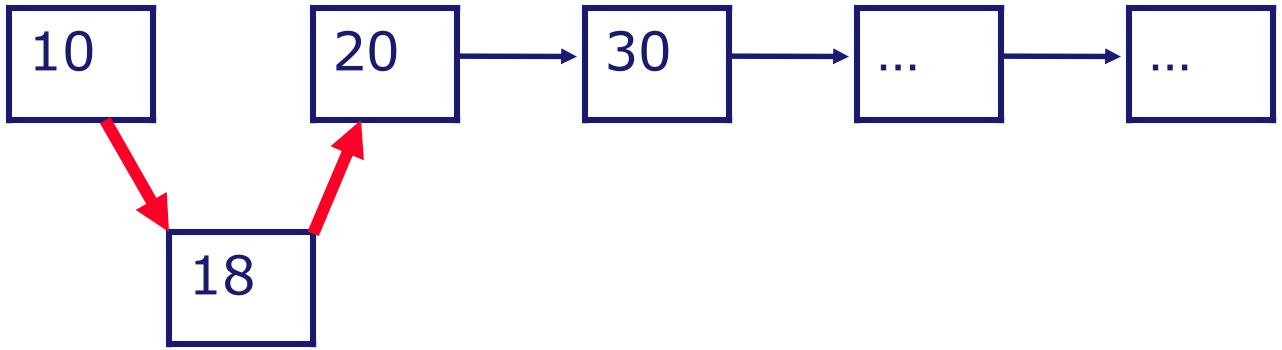
- Làm sao có thể thêm (hay xoá) 1 phần tử mà không phải di chuyển các phần tử khác ?

Đặt vấn đề (4)

- Ta tách rời các phần tử của mảng, và kết nối chúng lại với nhau bằng một "móc xích"

Đặt vấn đề (5)

- Thao tác thêm phần tử chỉ cần thay đổi các mối liên kết tại chỗ



- Chi phí **O(1)**

Danh sách liên kết là gì ? (1)

- Hãy viết ra các đặc điểm của DSLK
  - Ít nhất 5 đặc điểm

Danh sách liên kết là gì ? (2)

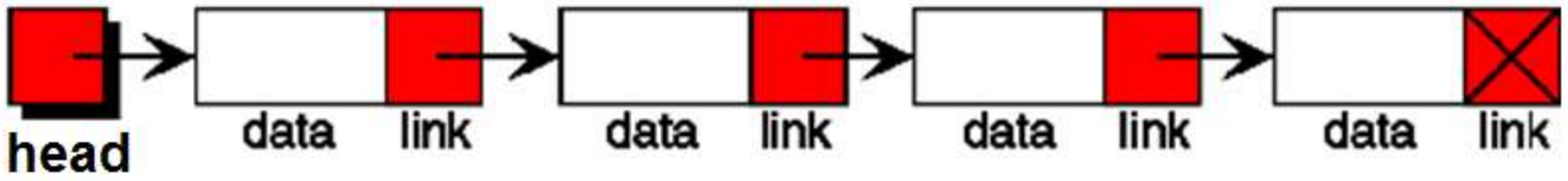
- Đặc điểm của DSLK
  - Sử dụng con trỏ (pointer)
  - Cấp phát bộ nhớ động
  - Dãy tuần tự các node
  - Giữa hai node có 1 hay nhiều con trỏ liên kết
  - Các node không cần phải lưu trữ liên tiếp nhau trong bộ nhớ
  - Có thể mở rộng tuỳ ý (chỉ giới hạn bởi dung lượng bộ nhớ)
  - Thao tác Thêm/Xóa không cần phải dịch chuyển phần tử

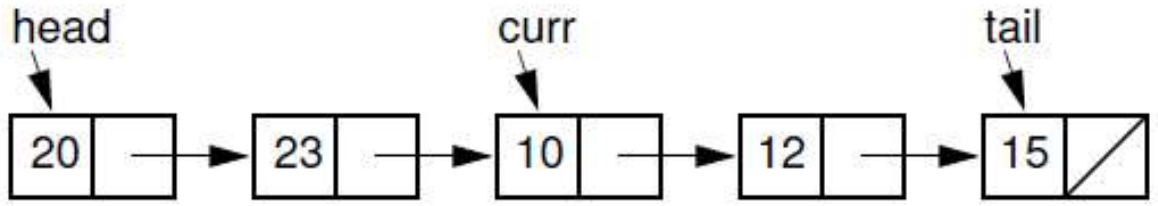
So sánh Mảng và Danh sách liên kết

Mảng	Danh sách liên kết
Kích thước cố định	Số phần tử thay đổi tùy ý → linh hoạt và tiết kiệm bộ nhớ
Các phần tử lưu trữ tuần tự (địa chỉ tăng dần) trong bộ nhớ	Các phần tử lưu trữ rời rạc, liên kết với nhau bằng con trỏ
Phải tịnh tiến các phần tử khi muốn Thêm/Xóa 1 phần tử - chi phí O(n)	Chỉ cần thay đổi con trỏ liên kết khi muốn Thêm/Xóa 1 phần tử - chi phí O(1)
Truy xuất ngẫu nhiên (nhanh)	Truy xuất tuần tự (chậm)
Sử dụng ít bộ nhớ hơn (nếu có cùng số phần tử)	Sử dụng nhiều bộ nhớ hơn (nếu có cùng số phần tử)

Danh sách liên kết đơn (1)

- Đặc điểm:
  - Mỗi node chỉ có 1 con trỏ liên kết (đến node kế tiếp trong danh sách)

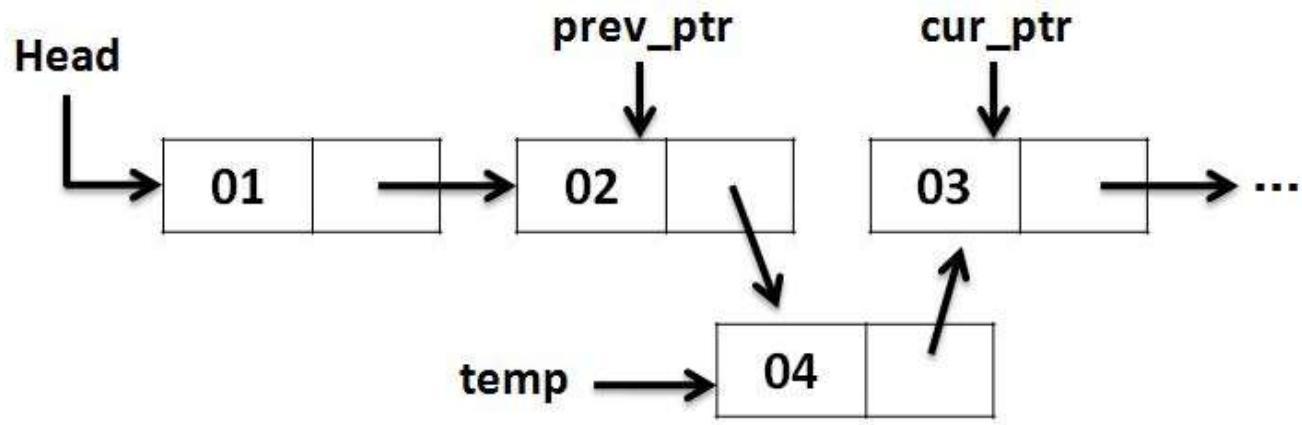




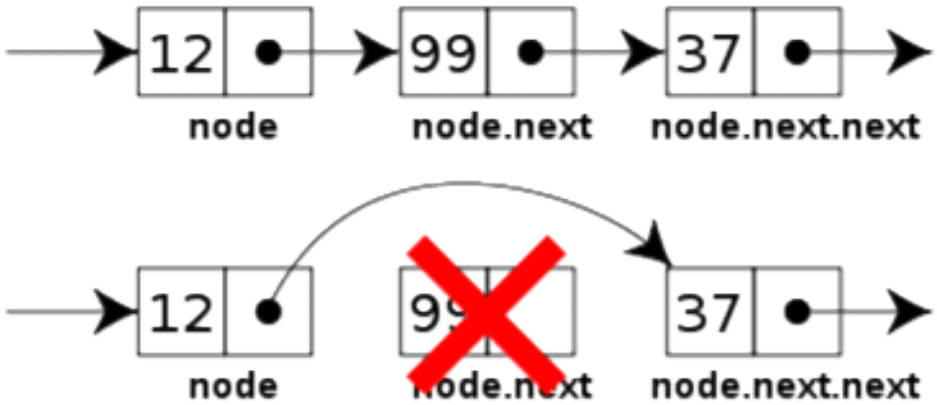
Danh sách liên kết đơn (2)

- Các thao tác cơ bản
  - Khởi tạo danh sách
  - Xóa danh sách
  - Kiểm tra danh sách rỗng
  - Đếm số phần tử trong danh sách
  - Thêm node vào đầu danh sách
  - Xóa node ở đầu danh sách
  - Thêm node ở cuối danh sách
  - Xóa node ở cuối danh sách
  - Tìm một node
  - Lấy thông tin của một node

Danh sách liên kết đơn (3)



Minh họa thao tác thêm node



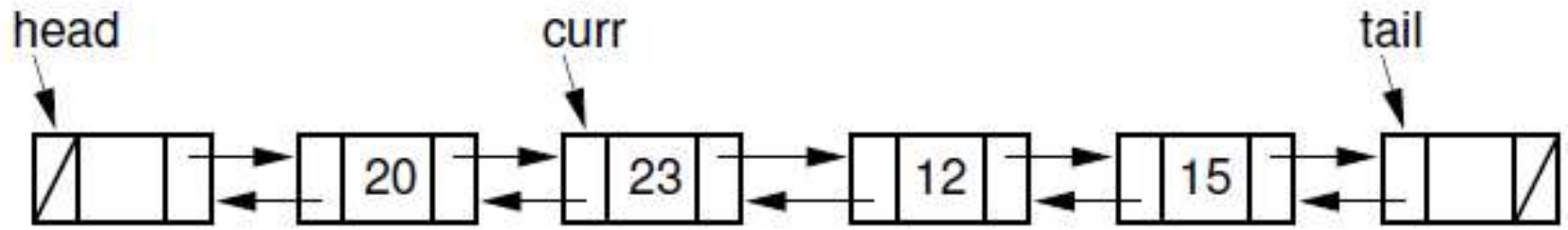
Minh họa thao tác xóa node

Danh sách liên kết đơn (4)

```
template <class T> class LINKED_LIST {
   private:
        struct ListNode  {
                T                data;    // data of node
                ListNode         *next;   // pointer to next node
        };
        int              size;            // number of node in list
        ListNode         *head;           // pointer to 1st node in list
   public:
        LINKED_LIST();                                  // default constructor
        LINKED_LIST(const LINKED_LIST &aList);  // copy constructor
        ~LINKED_LIST();                                 // destructor
        // operations
        bool     isEmpty();
        int      getLength();
        bool     addHead(T newItem);
        bool     removeHead();
        bool     addTail(T newItem);
        bool     removeTail();
        int      findNode(T key);          // return node index or -1
        bool     retrieveNode(int index, T &nodeData);
}; // end class
```

Danh sách liên kết đôi (1)

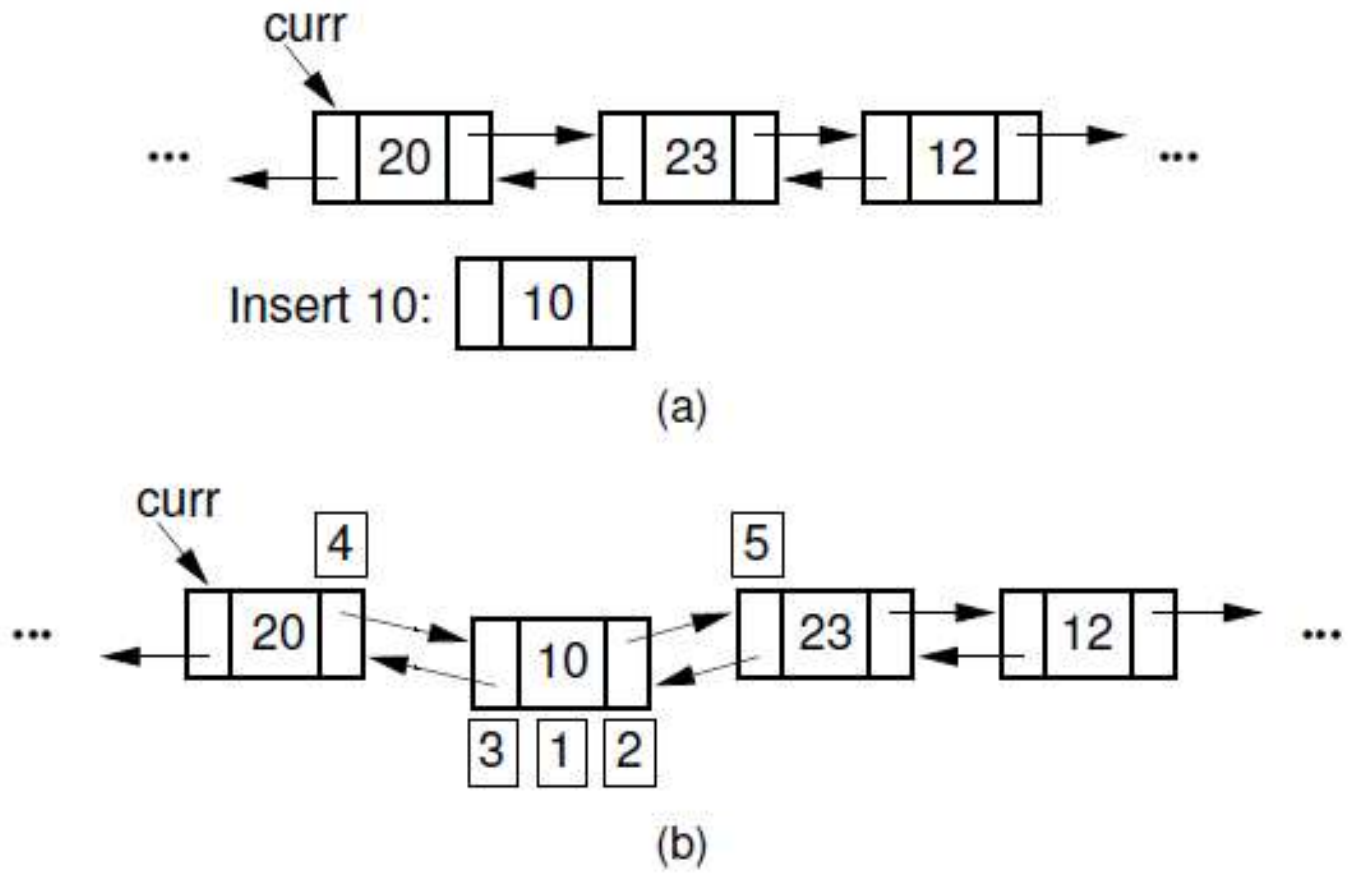
- Đặc điểm:
  - Mỗi node có 2 con trỏ liên kết đến node kế tiếp (next) và node phía trước (prev) trong danh sách



Danh sách liên kết đôi (2)

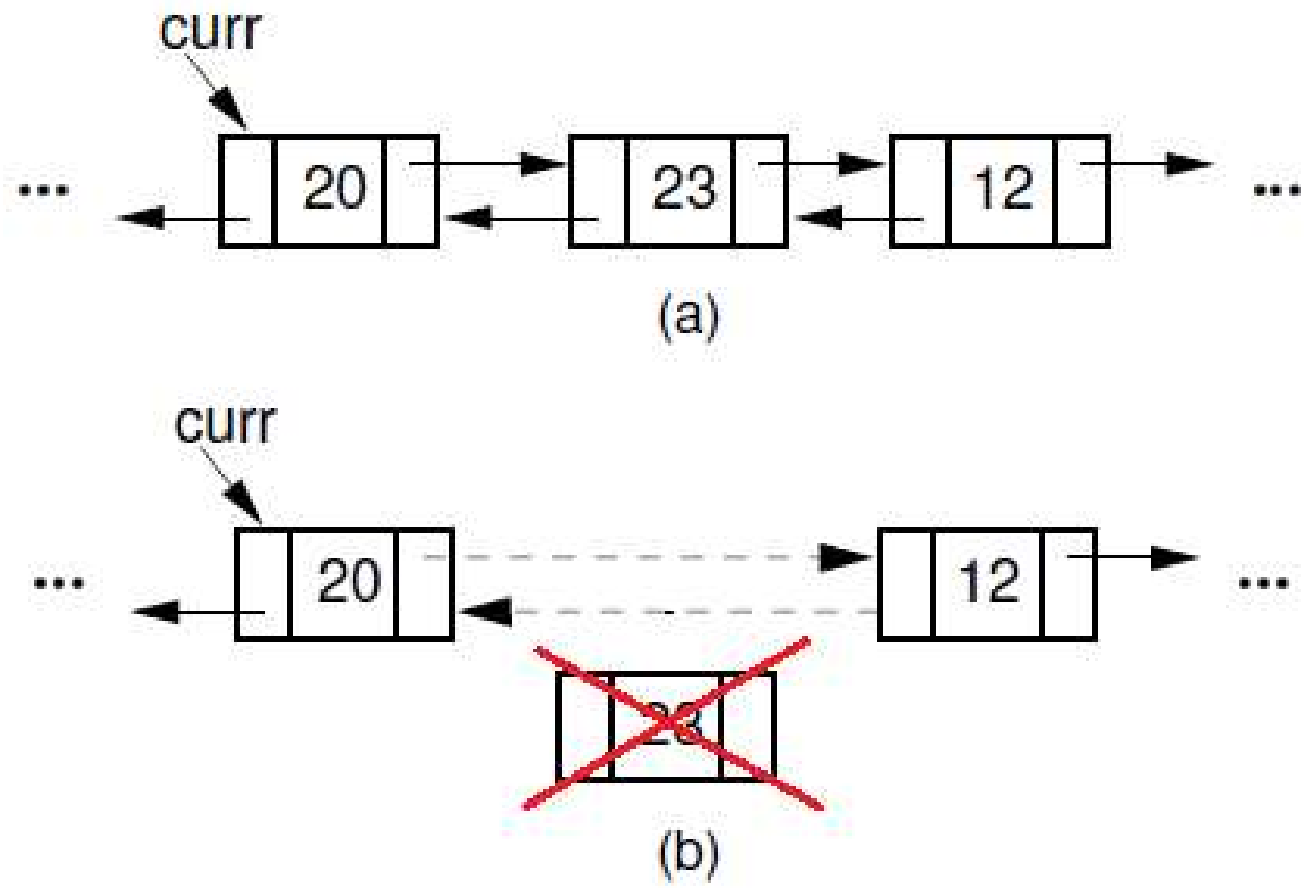
- Các thao tác cơ bản
  - Khởi tạo danh sách
  - Xóa danh sách
  - Kiểm tra danh sách rỗng
  - Đếm số phần tử trong danh sách
  - Thêm node vào đầu danh sách
  - Xóa node ở đầu danh sách
  - Thêm node ở cuối danh sách
  - Xóa node ở cuối danh sách
  - Tìm một node
  - Lấy thông tin của một node

Danh sách liên kết đôi (3)



Minh họa thao tác thêm node

Danh sách liên kết đôi (4)



Minh họa thao tác xóa node

Danh sách liên kết đôi (5)

```
template <class T> class DLINKED_LIST {
   private:
        struct ListNode  {
                T               data;           // data of node
                ListNode        *prev, *next;
        };
        int             size;           // number of node in list
        ListNode        *head;          // pointer to 1st node in list
   public:
        DLINKED_LIST();                         // default constructor
        DLINKED_LIST(const DLINKED_LIST &aList);// copy constructor
        ~DLINKED_LIST();                        // destructor
        // operations
        bool    isEmpty();
        int     getLength();
        bool    addHead(T newItem);
        bool    removeHead();
        bool    addTail(T newItem);
        bool    removeTail();
        int     findNode(T key);        // return node index or -1
        bool    retrieveNode(int index, T &nodeData);
}; // end class
```

Các cấu trúc dữ liệu cơ bản

(Fundamental Data Structures)

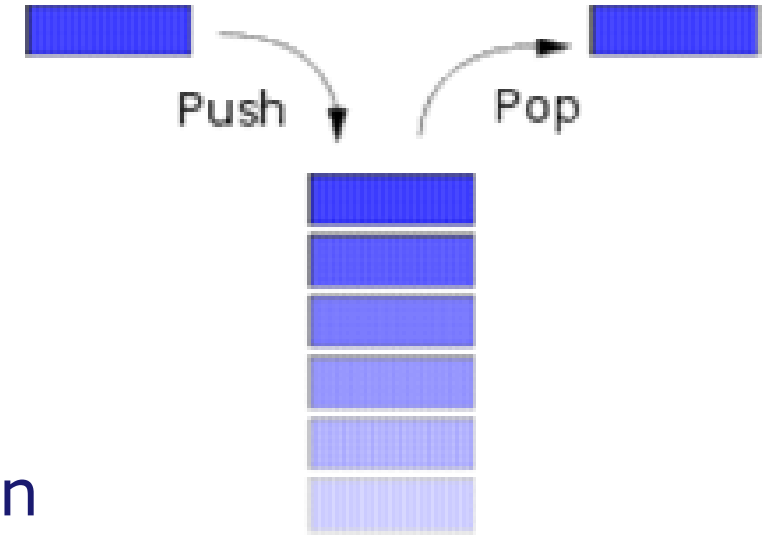
1.1 Các danh sách liên kết – Linked Lists

1.2 Ngăn xếp – Stack

1.3 Hàng đợi - Queue

Ngăn xếp - Stack

- Định nghĩa
- Các thao tác cơ bản
- Cài đặt Stack bằng mảng
- Cài đặt Stack bằng DSLK đơn
- Ứng dụng Stack



Định nghĩa

- Stack là một cấu trúc dữ liệu:
  - Dùng để lưu trữ nhiều phần tử dữ liệu
  - Hoạt động theo cơ chế "Vào sau – Ra trước" (Last In/First Out – LIFO)

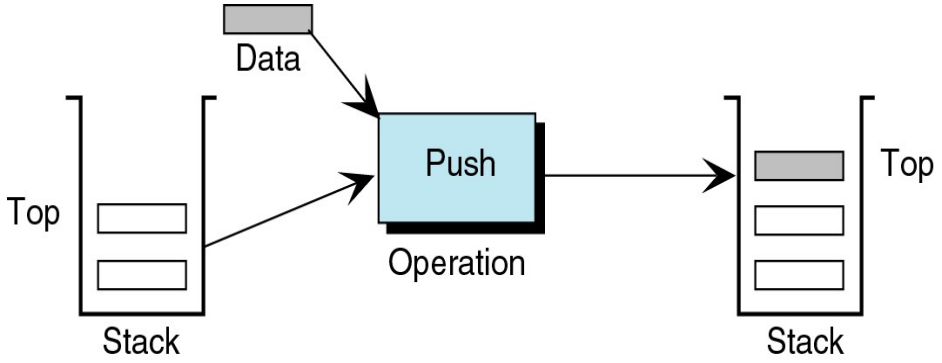
*** Cấu trúc Stack được phát minh năm 1955, được đăng ký bản quyền năm 1957, bởi tác giả Friedrich L. Bauer (người Đức)*

Các thao tác cơ bản (1)

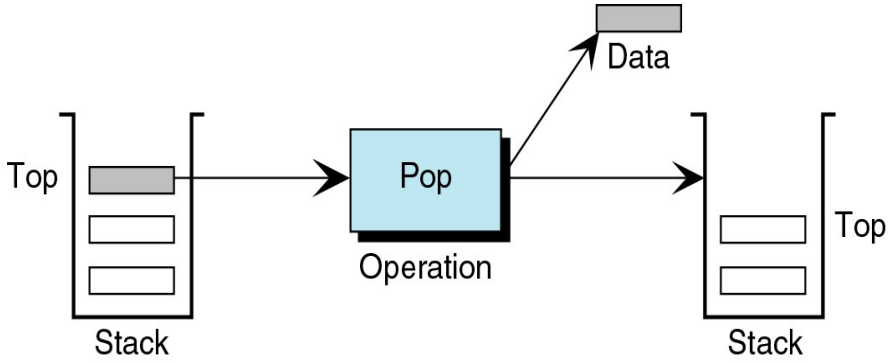
- Khởi tạo Stack rỗng
- Xóa Stack
- Kiểm tra Stack rỗng
- Thêm một phần tử vào đỉnh Stack (Push)
- Xóa một phần tử ở đỉnh Stack (Pop)
- Lấy phần tử ở đỉnh Stack mà không xóa nó

Các thao tác cơ bản (2)

- Push: thêm 1 phần tử vào đỉnh Stack



- Pop: lấy ra 1 phần tử ở đỉnh Stack



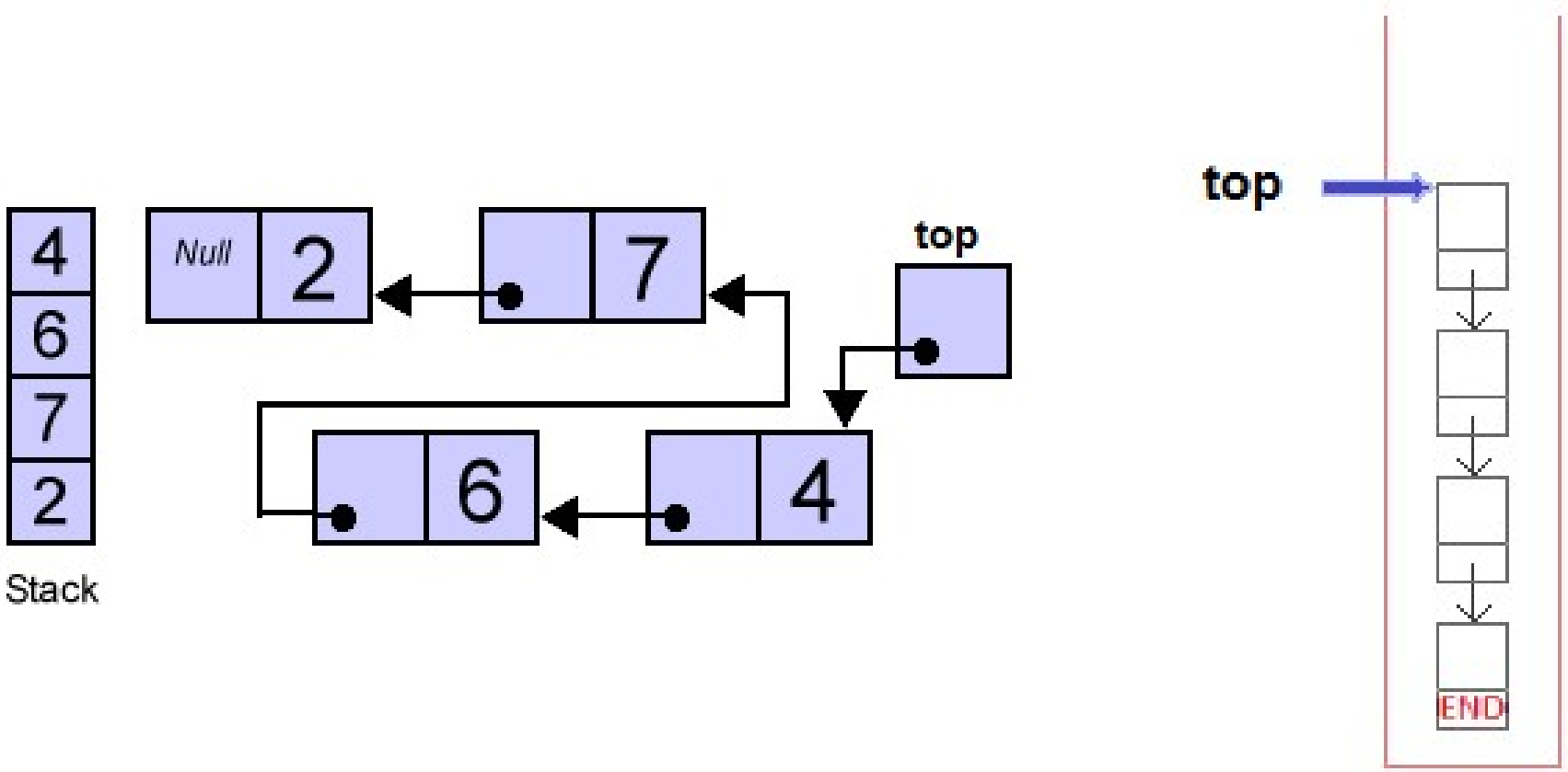
Cài đặt Stack bằng mảng

```
template <class T> class STACK {
   private:
        T               *items;        // array of stack items
        int             top;           // index to top of stack
        int             maxSize;       // maximum size of stack

   public:
        STACK(int size);               // create stack with
                                       // 'size' items
        STACK(const STACK &aStack);// copy constructor
        ~STACK();                      // destructor

        // operations
        bool    isEmpty();
        bool    push(T newItem);
        bool    pop(T &item);
        bool    topValue(T &item);
}; // end class
```

Áp dụng

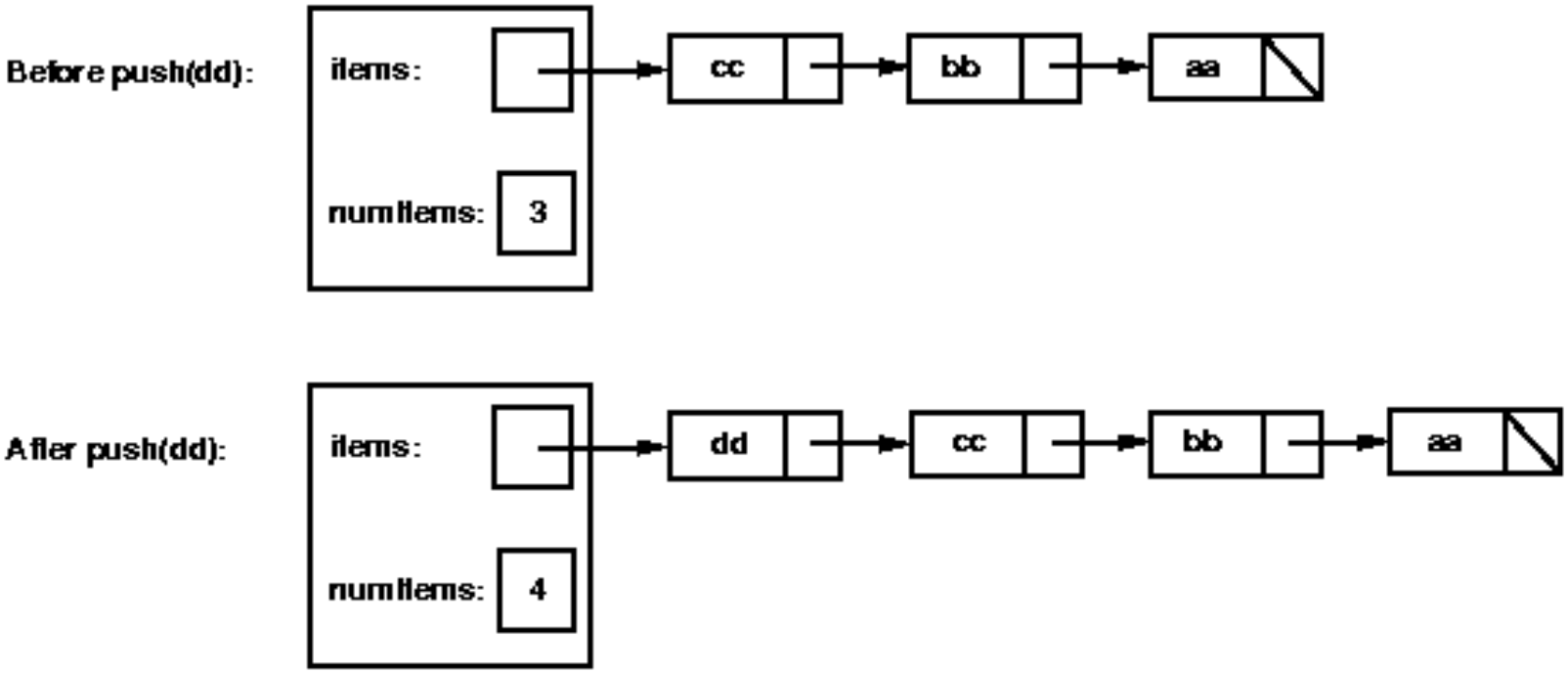
- Viết lệnh để thực hiện các yêu cầu sau đây:
  - Khai báo biến stack S, và khởi tạo S có N phần tử
  - Đưa các giá trị sau vào S: 15, 8, 6, 21
  - Lấy 21 ra khỏi S
  - Lấy 8 ra khỏi S
  - Gán các giá trị 1->99 vào S
  - Lần lượt lấy các phần tử trong S ra và in lên màn hình
  - Cho mảng a chứa dãy số nguyên từ 1->N, hãy đảo ngược các phần tử của mảng a

Cài đặt Stack bằng DSLK đơn (1)



Hình minh họa cấu trúc Stack sử dụng DSLK đơn

Cài đặt Stack bằng DSLK đơn (2)



Push(): chính là thêm node vào đầu DSLK

Cài đặt Stack bằng DSLK đơn (3)

```
template <class T> class STACK {
   private:
        struct StackNode  {
               T               data;   // data of item on the stack
               StackNode       *next;  // pointer to next node
        };
        StackNode       *top;           // pointer to top of stack

   public:
        STACK();                        // default constructor
        STACK(const STACK &aStack);     // copy constructor
        ~STACK();                       // destructor

        // operations
        bool    isEmpty();
        bool    push(T newItem);
        bool    pop(T &item);
        bool    topValue(T &item);
}; // end class
```

So sánh 2 cách cài đặt Stack

- So sánh
  - Cài đặt Stack bằng mảng (array-based stack)
  - Cài đặt Stack bằng Danh sách liên kết đơn (pointer-based stack)

Ứng dụng của Stack

- Tính giá trị biểu thức toán học (thuật toán Balan ngược – Reverse Polish notation)
- Bài toán tìm đường đi trong mê cung, bài toán mã đi tuần, bài toán 8 quân hậu,…
- Khử đệ qui
- …

Thuật toán Balan ngược

- Cho 1 biểu thức ở dạng chuỗi:
  - S = “5 + ((1 + 2) * 4) – 3”
  - Biểu thức gồm các toán tử +,-,*,/ và dấu ngoặc ()
- Tính giá trị biểu thức trên

Các cấu trúc dữ liệu cơ bản

(Fundamental Data Structures)

1.1 Các danh sách liên kết – Linked Lists

1.2 Ngăn xếp – Stack

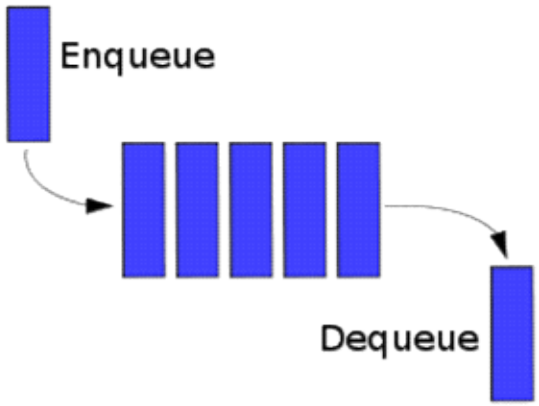
1.3 Hàng đợi - Queue

Hàng đợi - Queue

- Định nghĩa
- Các thao tác cơ bản
- Cài đặt Queue bằng mảng
- Cài đặt Queue bằng DSLK đơn
- Ứng dụng Queue

Định nghĩa

- Queue là một cấu trúc dữ liệu:
  - Dùng để lưu trữ nhiều phần tử dữ liệu
  - Hoạt động theo cơ chế "Vào trước – Ra trước" (First In/First Out – FIFO)

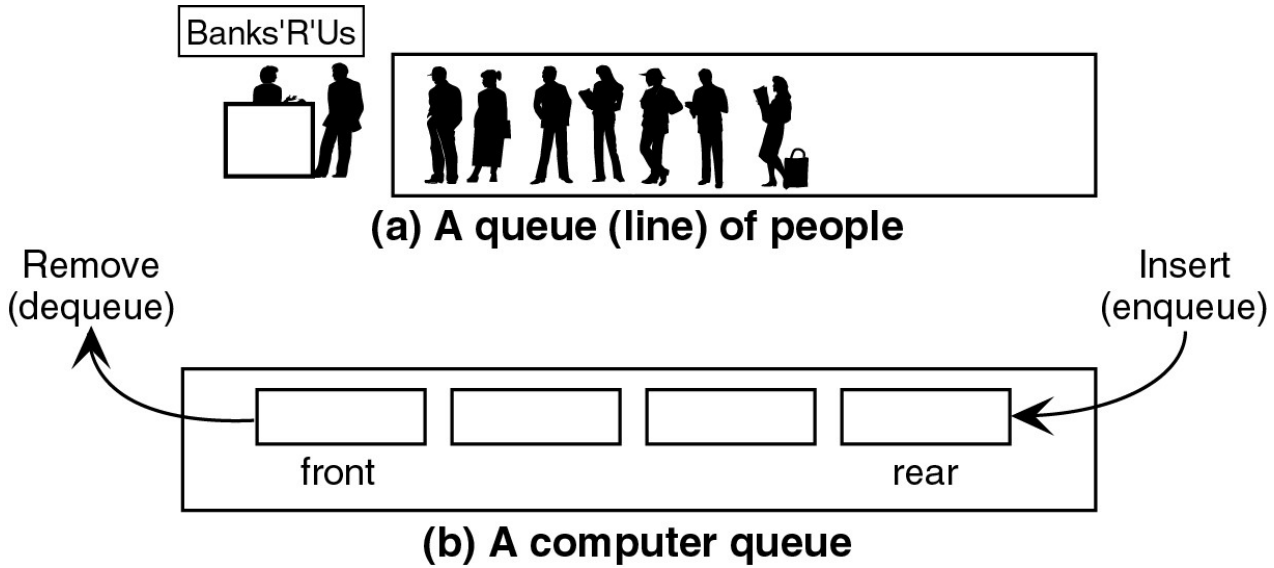


Các thao tác cơ bản (1)

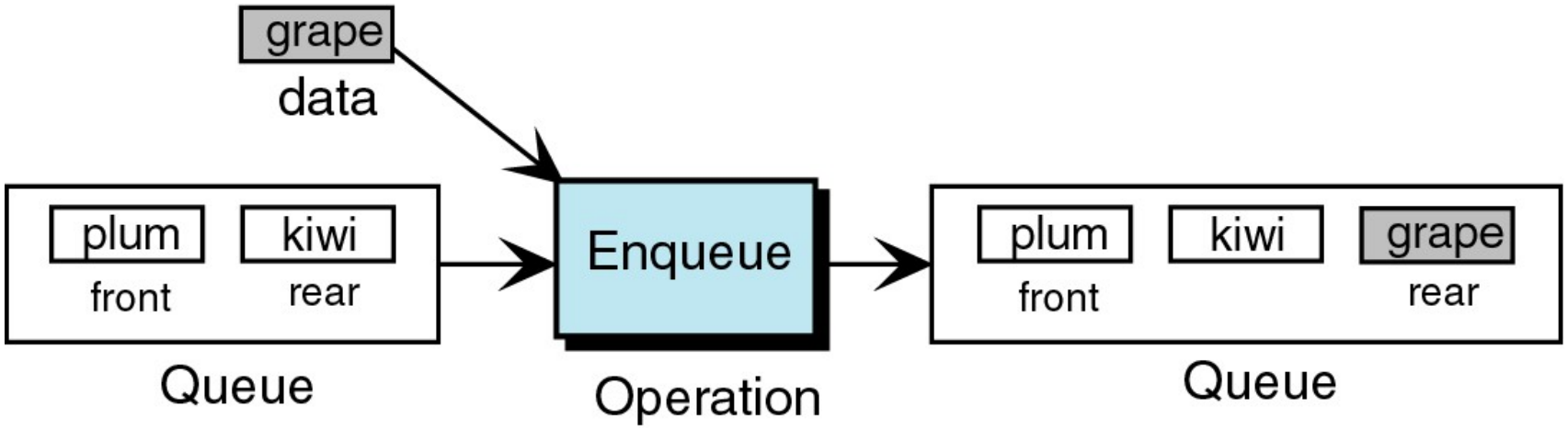
- Khởi tạo Queue rỗng
- Xóa Queue
- Kiểm tra Queue rỗng ?
- Thêm 1 phần tử vào cuối Queue (EnQueue)
- Lấy ra 1 phần tử ở đầu Queue (DeQueue)
- Lấy phần tử ở đầu Queue mà không xóa nó

Các thao tác cơ bản (2)

- EnQueue: thêm 1 phần tử vào cuối Queue
- DeQueue: lấy ra 1 phần tử ở đầu Queue

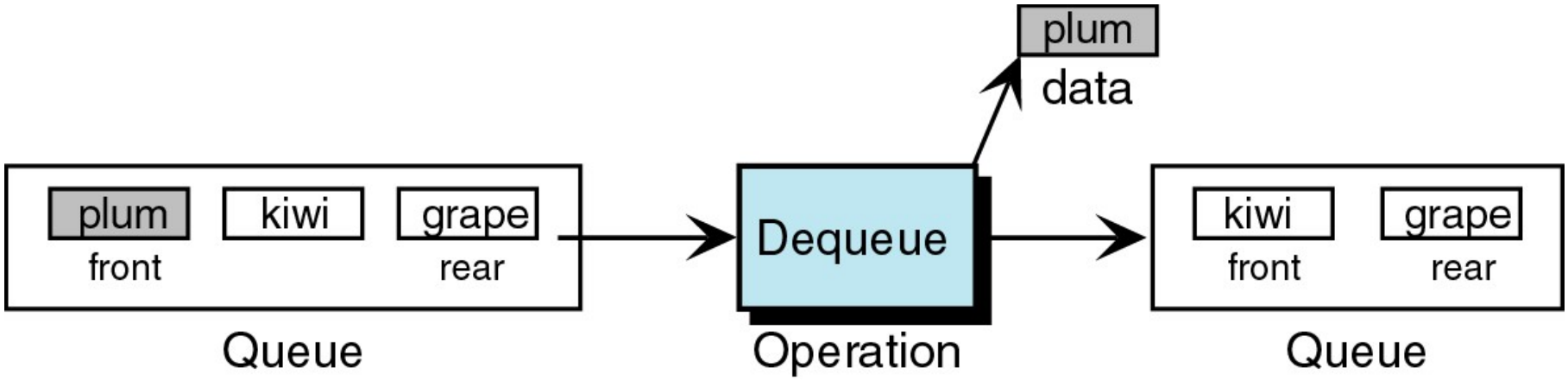


Các thao tác cơ bản (3)



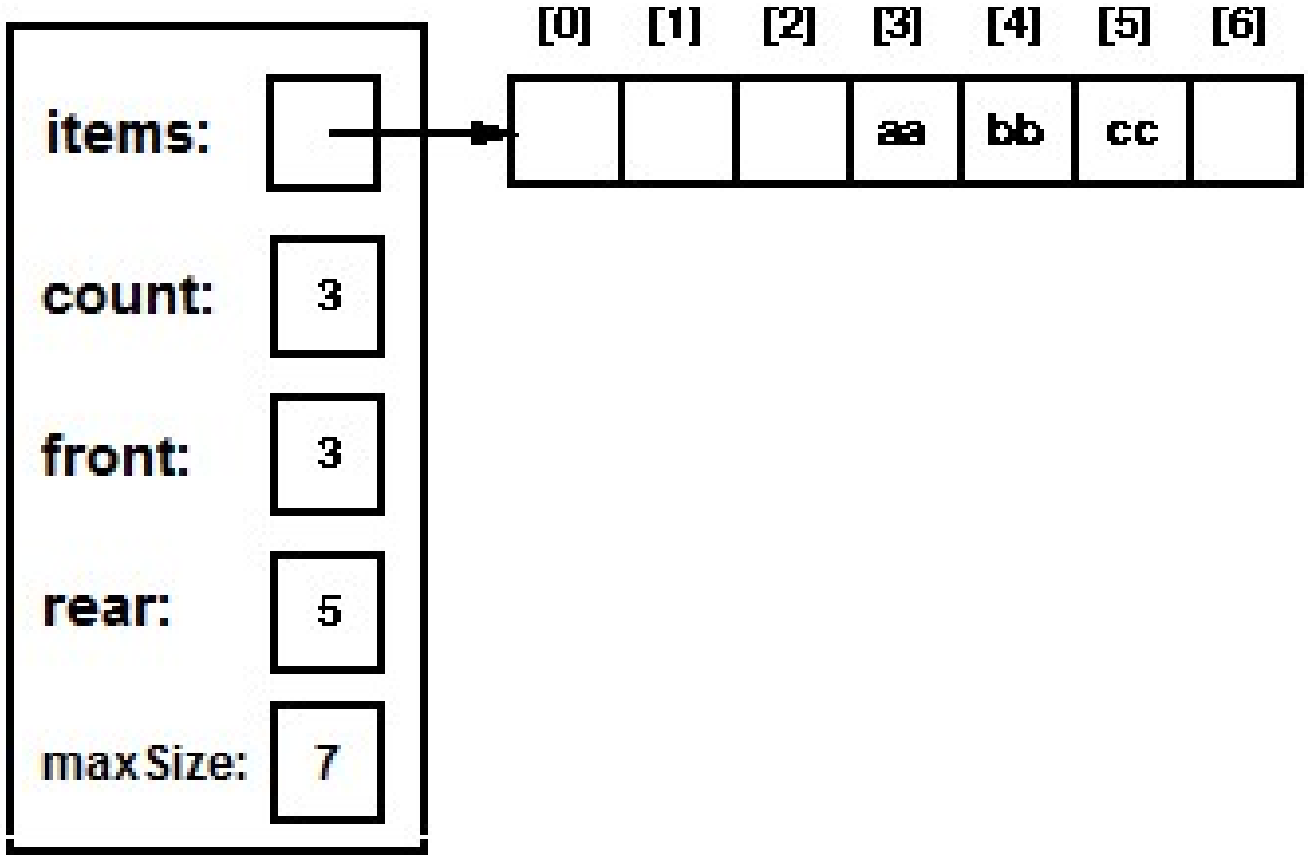
Minh họa thao tác EnQueue

Các thao tác cơ bản (4)



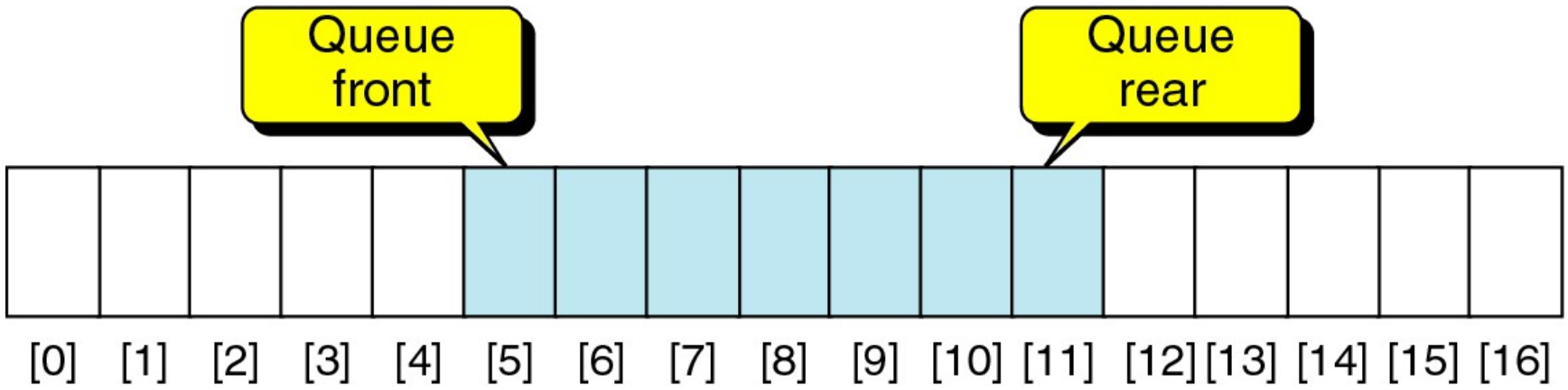
Minh họa thao tác DeQueue

Cài đặt Queue dùng mảng (1)



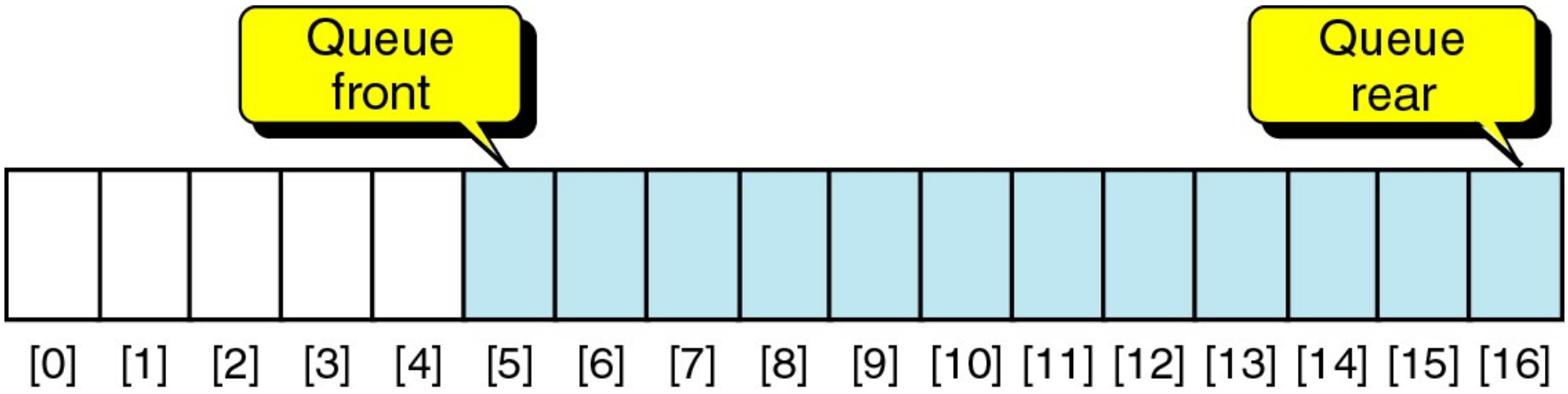
Cấu tạo của Queue

Cài đặt Queue dùng mảng (2)



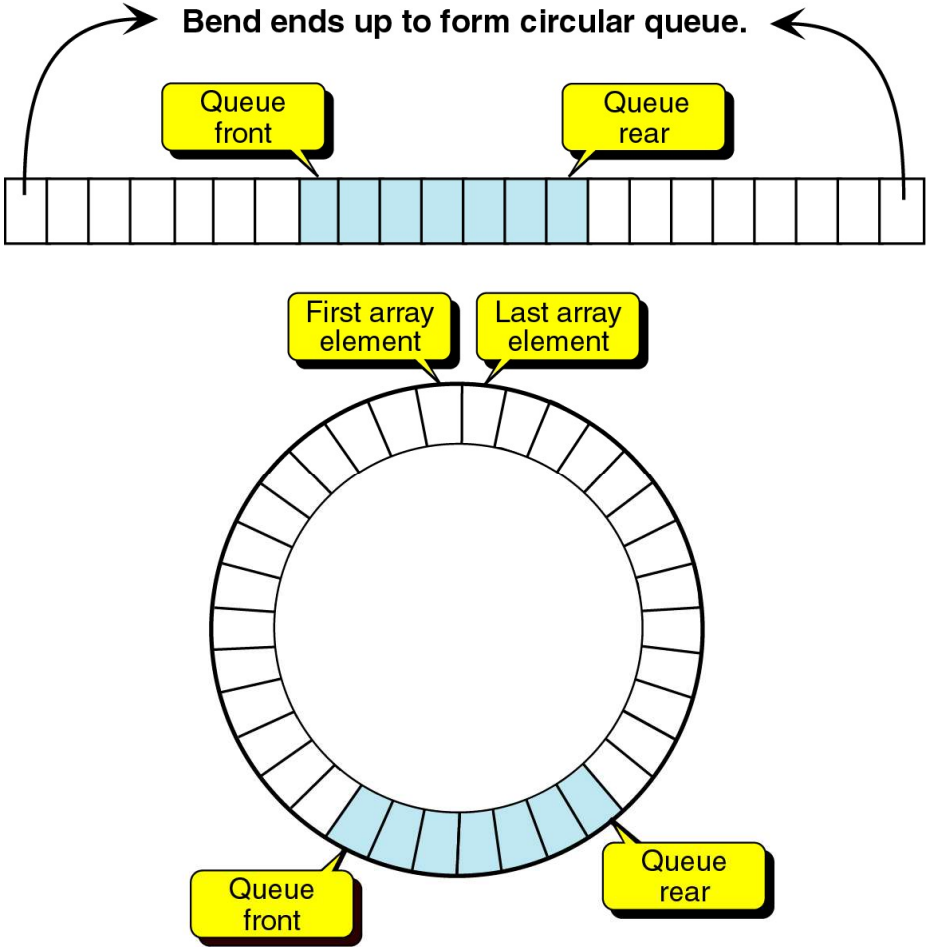
Minh họa hình ảnh các phần tử đang chứa trong Queue

Cài đặt Queue dùng mảng (3)



Khi thêm nhiều phần tử, sẽ làm "tràn" mảng → "Tràn giả"

Cài đặt Queue dùng mảng (4)

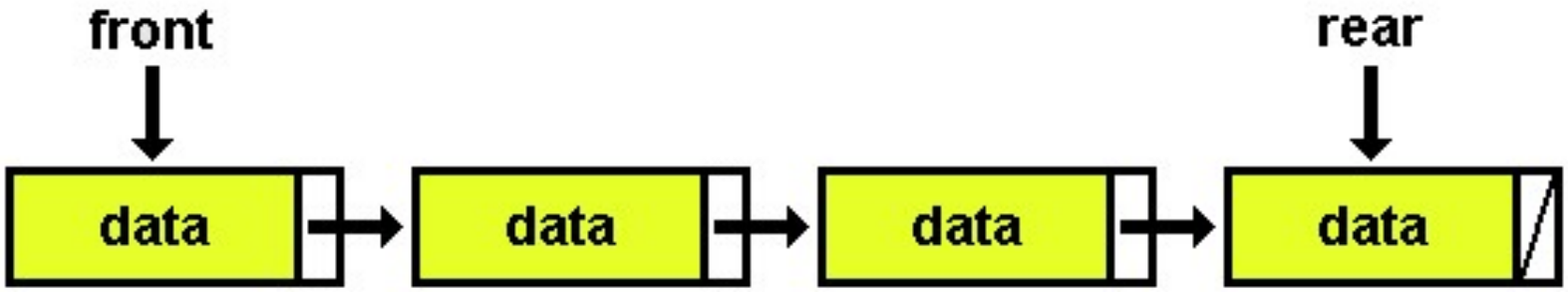


Giải pháp cho tình huống "tràn giả": xử lý mảng như là 1 danh sách vòng

Cài đặt Queue dùng mảng (5)

```
template <class T> class QUEUE {
   private:
        T              *items;        // array of queue items
        int            front;
        int            rear;
        int            count;
        int            maxSize;       // maximum size of queue
  public:
        QUEUE(int size);              // create queue with
                                      // 'size' items
        QUEUE(const QUEUE &aQueue); // copy constructor
        ~QUEUE();                     // destructor

        // operations
        bool   isEmpty();
        bool   enqueue(T newItem);
        bool   dequeue(T &item);
        bool   frontValue(T &item);
}; // end class
```

Cài đặt Queue dùng DSLK đơn (1)



- Enqueue: thêm node vào cuối DSLK đơn
- Dequeue: xóa node ở đầu DSLK đơn

Cài đặt Queue dùng DSLK đơn (2)

```
template <class T> class QUEUE {
   private:
        struct QueueNode  {
               T                data;   // data of item on the queue
               QueueNode        *next;  // pointer to next node
        };
        QueueNode       *front;
        QueueNode       *rear;

  public:
        QUEUE();                          // default constructor
        QUEUE(const QUEUE &aStack);       // copy constructor
        ~QUEUE();                         // destructor

        // operations
        bool    isEmpty();
        bool    enqueue(T newItem);
        bool    dequeue(T &item);
        bool    frontValue(T &item);
}; // end class
```

Ứng dụng của Queue

- Quản lý xếp hàng (theo số thứ tự).
  VD. Tại các ngân hàng, bệnh viện,…

- Quản lý phục vụ in ấn (máy in)