Cấu trúc dữ liệu

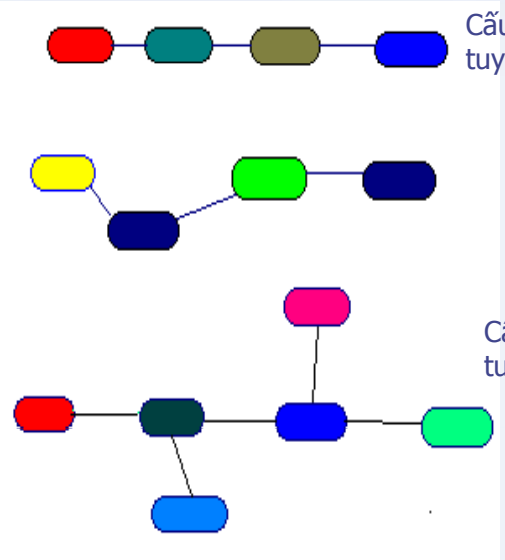
- Vector
- List
- Stack
- Queue
- Tree
- HashTable
- Dictionary

Bài 6

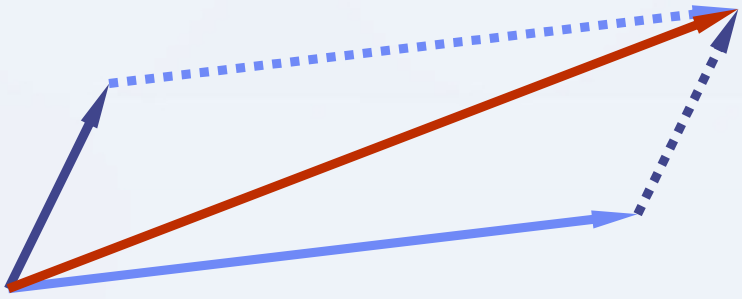
Véc tơ (Vector)

Cấu trúc tuyến tính

- Cấu trúc tuyến tính là một cấu trúc trong đó các phần tử nằm trên một đường không có nhánh, và các phần tử liên tiếp nhau.
- Một số ví dụ:
  - Danh sách (lists)
  - Vector, chuỗi (vectors, sequences)
  - Danh sách kiểu ngăn xếp, danh sách kiểu hàng đợi (stack, queue)

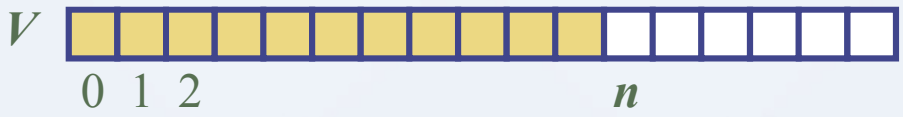


Vector



Kiểu dữ liệu trừu tượng Vector (Vector ADT)

- Kiểu dữ liệu trừu tượng Vector là sự mở rộng của khái niệm mảng. Vector là một mảng lưu trữ một dãy các đối tượng với số lượng tùy ý.



- Một phần tử có thể được truy cập, chèn thêm hoặc loại bỏ đi khi biết chỉ số của nó.

- Khi thực hiện các thao tác trên có thể xảy ra lỗi nếu chỉ số của phần tử không chính xác (Vd, chỉ số âm)

Các thao tác trên Vector

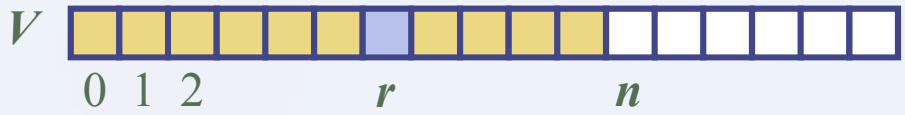
- int getAtRank(int r, object &o): Trả lại phần tử có chỉ số r, nhưng không loại bỏ nó
- int replaceAtRank(int r, object o, object & o1): Thay thế phần tử có chỉ số r bằng phần tử o và trả lại phần tử bị thay thế

- int insertAtRank(int r, object o): Chèn phần tử o vào vị trí r
- int removeAtRank(int r, object &o): loại bỏ phần tử tại vị trí r, và trả lại phần tử bị loại bỏ

- int size() cho biết kích thước của Vector
- int isEmpty() cho biết Vector có rỗng hay không?

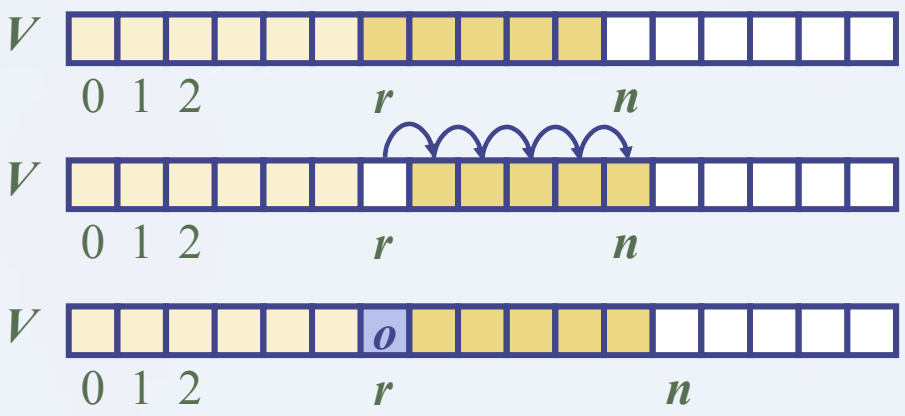
Cài đặt Vector bằng mảng

- Sử dụng mảng V có kích thước N
- Một biến n lưu trữ kích thước của vector (số phần tử được lưu trữ)
- Phép toán *getAtRank*(r,o) được thực hiện trong thời gian $O(1)$ bằng việc trả lại $V[r]$



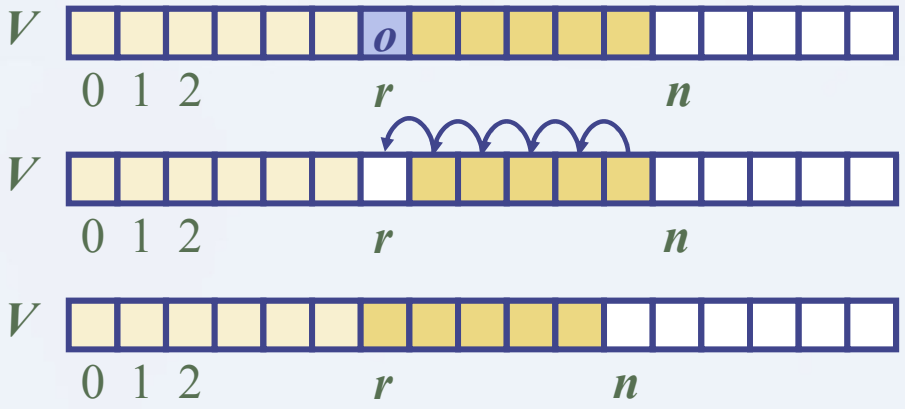
Chèn thêm phần tử

- Phép toán *insertAtRank*(r, o), Chúng ta cần tạo một ô mới có chỉ số r bằng cách đẩy n-r phần tử từ $V[r], \ldots, V[n-1]$ về sau 1 vị trí
- Trong trường hợp xấu nhất ($r = 0$), phép toán thực hiện trong thời gian $O(n)$



Loại bỏ phần tử

- Phép toán *removeAtRank*(*r*,*o*), chúng ta cần đẩy n – *r* – 1 phần tử từ *V*[*r* + 1], …, *V*[*n* – 1] về trước một vị trí
- Trong trường hợp xấu nhất (*r* = 0), phép toán thực hiện trong thời gian *O*(*n*)



Các ứng dụng của Vector

- Ứng dụng trực tiếp
  - Lưu trữ tập hợp các đối tượng (cơ sở dữ liệu đơn giản)
- Ứng dụng gián tiếp
  - Cấu trúc dữ liệu bổ trợ cho các thuật toán
  - Thành phần của các cấu trúc dữ liệu khác

Tóm lại

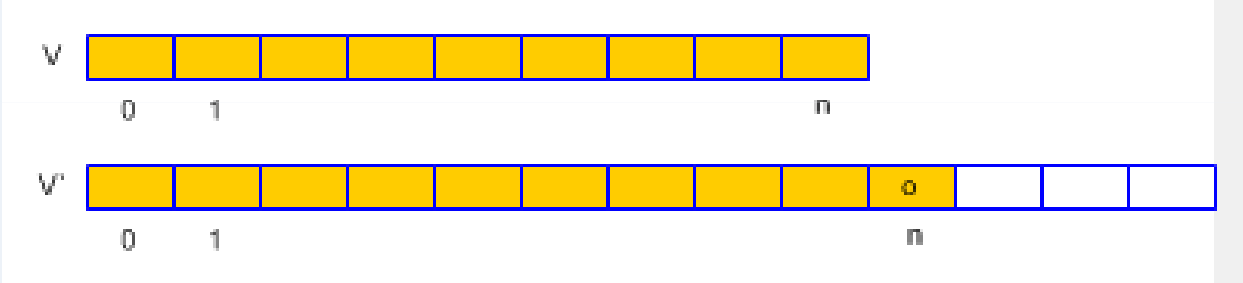
- Cài đặt Vector bằng mảng:
  - Không gian sử cho cấu trúc dữ liệu là $O(n)$
  - *Các phép toán* ***size***, ***isEmpty***, ***getAtRank*** và ***replaceAtRank*** chạy trong thời gian $O(1)$
  - ***insertAtRank*** và ***removeAtRank*** chạy trong thời gian $O(n)$
- Nếu chúng ta sử dụng một mảng quay vòng thì phép toán, ***insertAtRank***(0) và ***removeAtRank***(0) chạy trong thời gian là $O(n)$
- Với phép toán ***insertAtRank***, khi mảng đầy sẽ dẫn đến ngoại lệ, để tránh trường hợp này chúng ta thay mảng hiện tại bằng mảng lớn hơn

Phát triển mảng

- Khi thực hiện phép toán. Nếu mảng đầy ➔ lỗi. Để có thể thêm phần tử đó vào ta phải mở rộng mảng.
- Làm thế nào để mở rộng mảng?
  - Chiến lược phát triển theo hằng số:Tăng thêm kích thước mảng theo một hằng số c
  - Chiến lược gấp đôi:Tăng gấp đôi số phần tử hiện có của mảng

Thêm phần tử vào cuối

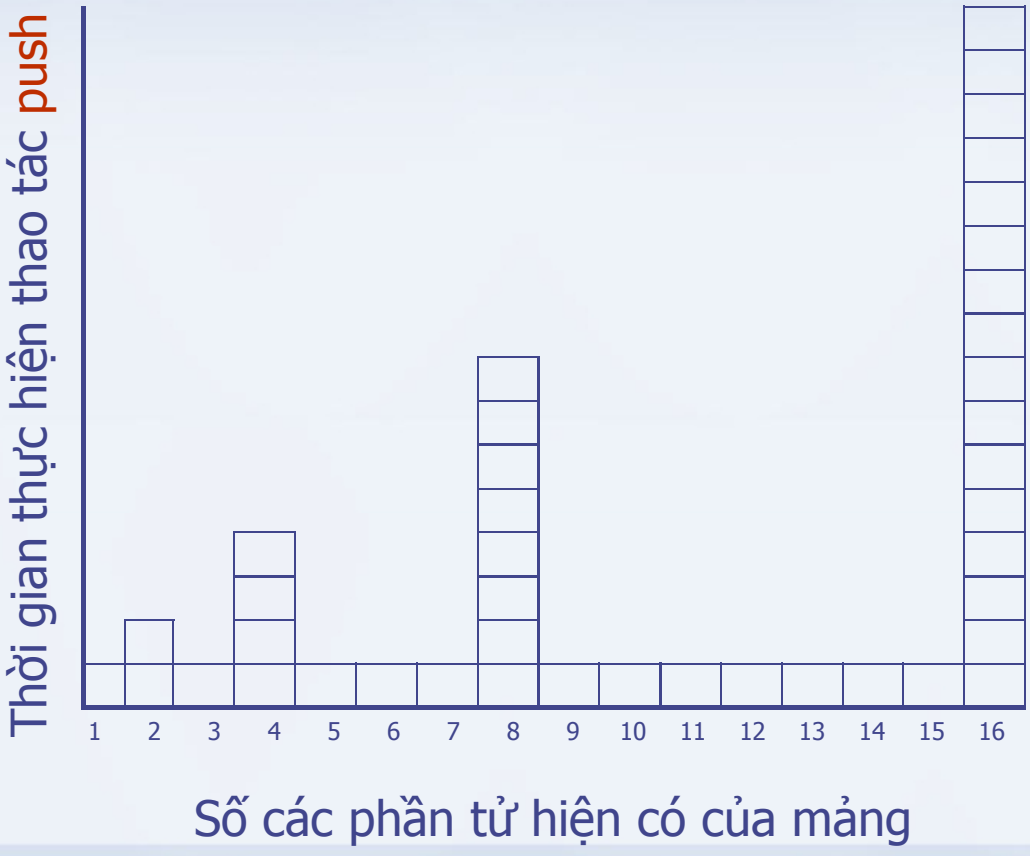
```
Algorithm push( o)
  if n = V.N – 1 then
     A ← Tạo mảng mới có kích thước …
     for i ← 0 to n-1 do
        A[i] ← V[i]
    delete V
    V ← A
    V[n] ←O
    n ← n+ 1
```



So sánh hai chiến lược

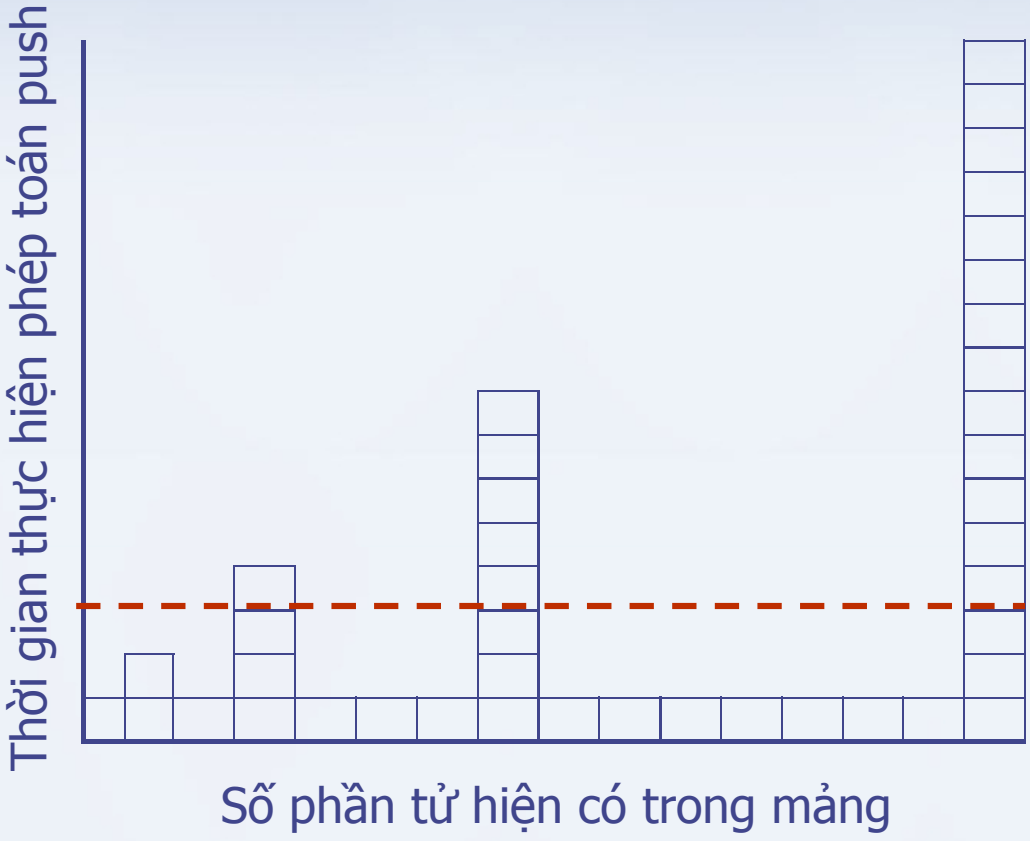
- Ta so sánh chiến lược phát triển theo hằn số và chiến lược gấp đôi bằng cách phân tích tổng thời gian $T(n)$ cần thiết để thực hiện thao tác push một dãy n phần tử vào mảng.
- Chúng ta thực hiện bắt đầu với mảng có 1 phần tử
- Và đi xác định thời gian trung bình khi push một phần tử vào mảng là $T(n)/n$

Thời gian thực hiện đưa một dãy các phần tử vào mảng bằng cách sử dụng chiến lược gấp đôi

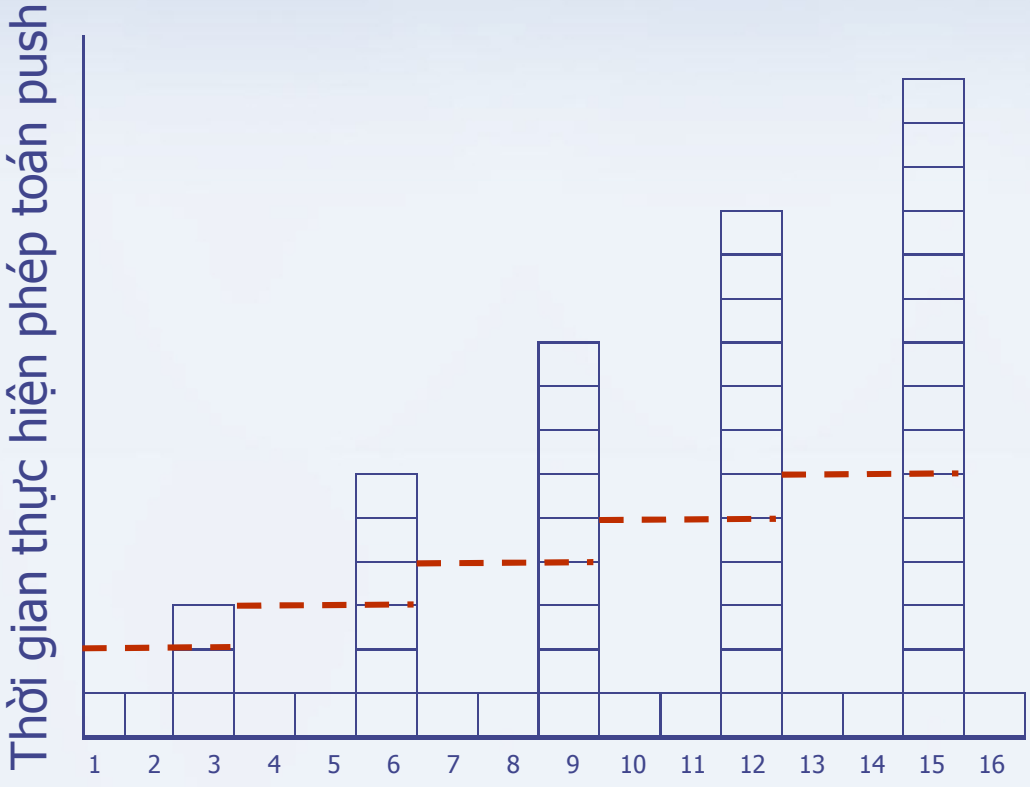


Thời gian thực hiện đưa một dãy các phần tử vào mảng bằng cách sử dụng chiến lược phát triển theo hằng số

Thời gian thực hiện đưa một dãy các phần tử vào mảng bằng cách sử dụng chiến lược gấp đôi



Thời gian thực hiện đưa một dãy các phần tử vào mảng bằng cách sử dụng chiến lược phát triển theo hằng số



- Đếm số lượng phép gán và so sánh trong 2 chiến lược: mở rộng theo hằng số c, mở rộng theo chiến lược tăng gấp đôi.

Phân tích chiến lược phát triển theo hằng số

- Chúng ta thay thế mảng $k = n/c$ *lần*
- Tổng thời gian $T(n)$ của phép toán push n phần tử vào mảng tương ứng là

$$n + c + 2c + 3c + 4c + \ldots + kc =$$
$$n + c(1 + 2 + 3 + \ldots + k) =$$
$$n + ck(k + 1)/2$$

- Trong đó c là một hằng số, $T(n)$ là $O(n + k^2)$, hay là $O(n^2)$
- Vậy thời gian trung bình phải trả cho phép toán push là $O(n)$

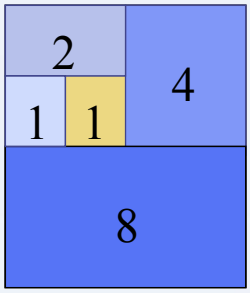
Phân tích chiến lược gấp đôi

- Chúng ta thay thế mảng $k = \log 2\ n$ lần
- Tổng thời gian thực hiện phép toán push n phần tử vào mảng là $T(n)$ và tương ứng là:

$$n + 1 + 2 + 4 + 8 + \ldots + 2^k =$$
$$n + 2^{k+1} - 1 = 3n - 1$$

- $T(n)$ là $O(n)$
- Thời gian trung bình phải tra cho phép toán push một phần tử mảng là $O(1)$.

Mô tả bằng hình học



Một số chú ý khi cài đặt bằng mảng

- Khi sử dụng mảng để cài đặt Vector thì việc sử dụng ngôn ngữ có khả năng dễ dàng cấp phát bộ nhớ là rất quan trọng.
- Trong một số ngôn ngữ, mảng không thể mở rộng được sau khi nó đã được tạo ra..

Một số chú ý khi cài đặt bằng mảng

- Các ngôn ngữ thủ tục được chia thành 2 lớp ngôn ngữ dựa vào khia cạnh này:
  - Các ngôn ngữ như là: Ada, Algol, C and JAVA cho phép xác định kích thước mảng vào thời gian chạy chương trình khi mảng được tạo ra. Như vậy, mảng có thể mở rộng tùy ý.
  - Các ngôn ngữ như là: Pascal, Modula-2 and Fortran thì rất hạn chế, nó yêu cầu kích thước của mảng phải được xác định khi dịch chương trình.
  - Nếu không thể thay đổi động thì khi cài đặt các cấu trúc dữ liệu bằng mảng phải có phương pháp mở rộng mảng.

Cài đặt Vector bằng C++

BTVN: Thực hiện chương trình trên dãy số nguyên

```
template <class Object>
class Vector{
    private:
        int  N;   //Số chiều tối đa của Vector
        Object *V;     //lưu trữ dữ liệu
        int  n;          //Số phần tử hiện có trong Vector
    public:
        Vector();
        ~Vector();
        int getAtRank(int r, Object &o);
        int replaceAtRank(int r, Object o);
        int insertAtRank(int r, Object o);
        int removeAtRank(int r, Object &o);
        int size();
        int isEmpty();
};
```

Iterator – Bộ lặp

- Trong các ADT không hỗ trợ phép tìm duyệt các phần tử của nó.
- Bộ lặp được sử dụng để tìm duyệt lần lượt các phần tử của các ADT như: Vector, List, Tree,…
- Đối tượng bộ lặp sẽ cung cấp một phần tử của đối tượng ADT sử dụng nó theo thứ tự nào đó đã được xác định
- Tại một thời điểm có thể có nhiều đối tượng bộ lặp trên cùng một đối tượng ADT

Cấu trúc bộ lặp

- Thuộc tính

  - Một thuộc tính có kiểu con trỏ lưu địa chỉ của thuộc tính quản lý các phần tử của đối tượng sử dụng bộ lặp

  - Một thuộc tính lưu địa chỉ của phần tử hiện tại của đối tượng sử dụng bộ lặp

- Phương thức

  - int hasnext(): Cho biết có còn phần tử tiếp theo không?

  - Object next(): trả lại phần tử hiện tại và chuyển con trỏ đến phần tử kế tiếp

Bộ lặp cho các đối tượng Vector

```
template <class Object>
class VectorItr{
    private :
        Vector<Object>* theVector;
         int current_Index;

    public:
        VectorItr(Vector<Object>*v)
       {
         theVector = v;
         current_Index = 0;
        }
```

```
int hasNext(){
  if (current_Index< theVector->size())
       return 1;
  else
        return 0;
 }
 Object next(){
  Object o;
  theVector->getAtRank(current_index, o);
  current_Index++;
  return o;
}
};//End of class VectorItr
```


Bài tập

Sử dụng lớp Vector và lớp bộ lặp của lớp vector xây dựng chương trình có các chức năng sau:

1. Chèn 1 phần tử vào vector
2. Xóa 1 phần tử của vector
3. Thay thế một phần tử của vector
4. Lấy giá trị của một phần tử của vector
5. In danh sách các phần tử hiện có trong vector

Các phần tử lưu vào vector là các số thực

Hết