Cấu trúc dữ liệu

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

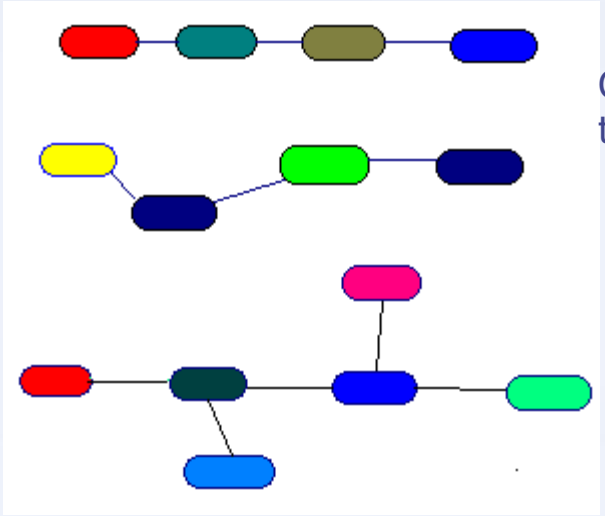
Bài 6

Mảng (Array) và 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 (list)
  - Vector, chuỗi (vector, sequence, string)
  - Danh sách kiểu ngăn xếp, danh sách kiểu hàng đợi (stack, queue)



Cấu trúc tuyến tính

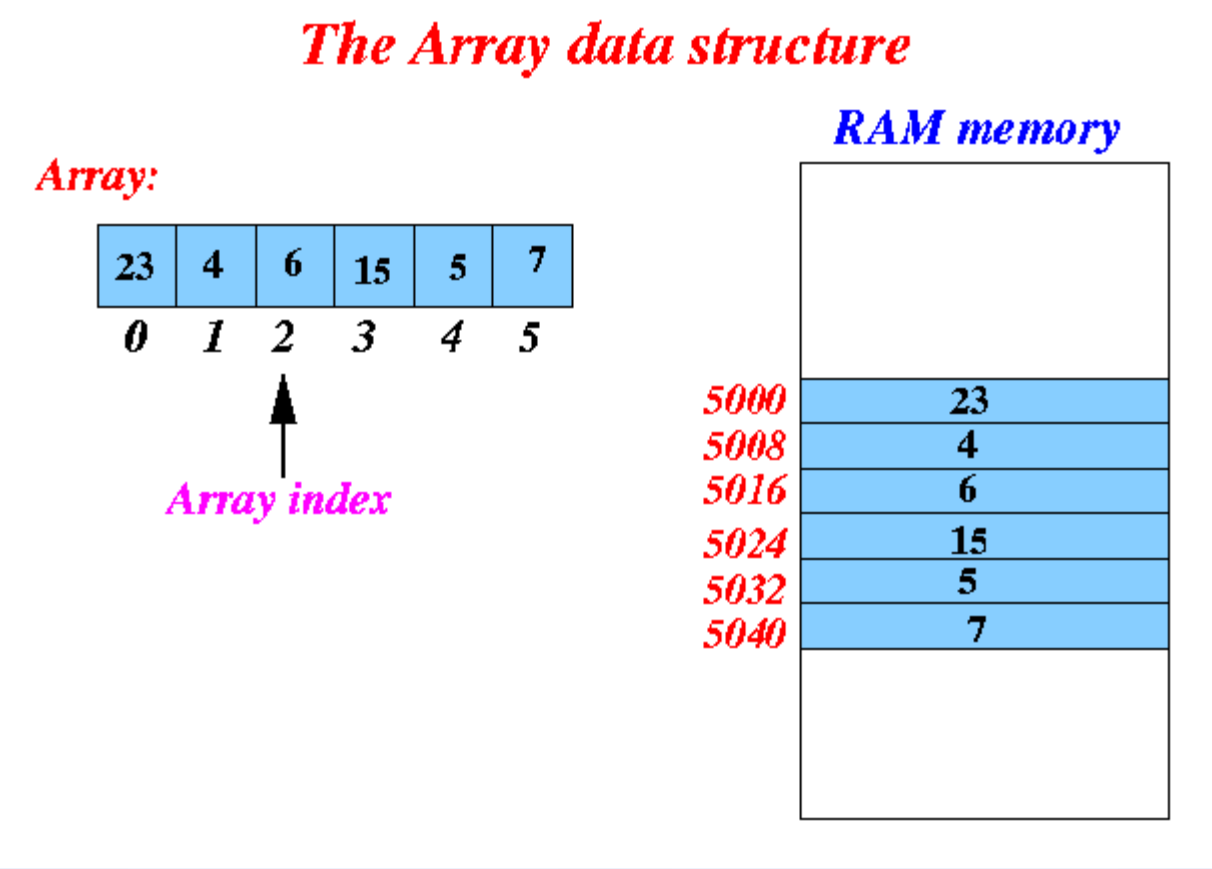
Cấu trúc phi tuyến

Mảng

- Mảng là một cấu trúc dữ liệu bao gồm một nhóm các phần tử, mỗi phần tử được xác định ít nhất bằng một chỉ số (index) hoặc khóa (key)
- Mảng được lưu theo cách có thể tính được vị trí của các phần tử từ giá trị của một bộ chỉ số bằng một biểu thức toán học

Mảng

Ví dụ, một mảng có 6 số nguyên với các chỉ số từ 0 đến 5, mỗi số có 8 byte và được lưu tại các địa chỉ bộ nhớ 5000, 5008, 50016,... 5040. Như vậy, phần tử có chỉ số i sẽ nằm ở địa chỉ 5000 + 8 × i



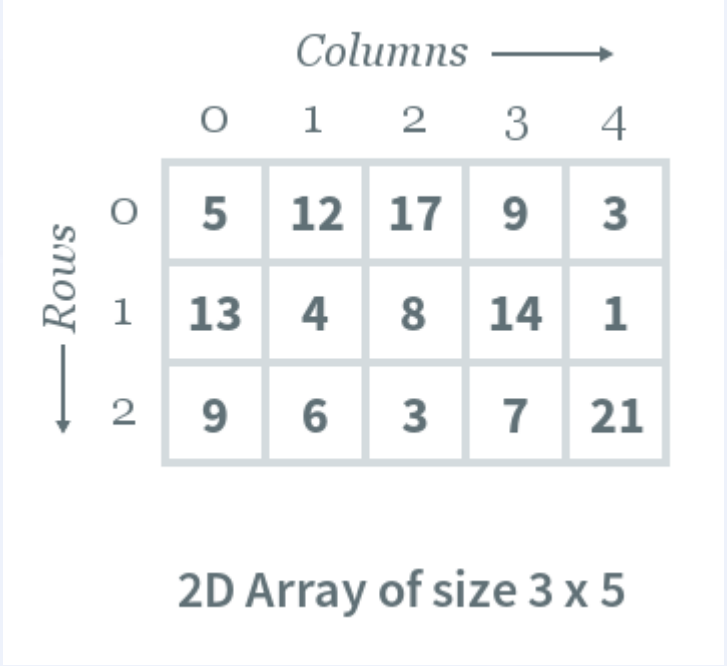
Mảng

Mảng nhiều chiều

Ví dụ, phần tử có chỉ số [0, 0] nằm ở địa chỉ 5000 trong bộ nhớ, mỗi phần tử có 4 byte và mảng có n cột thì

phần tử có chỉ số [i, j] nằm ở địa chỉ: $5000 + 4 \times (i \times n + j)$

Hình dung một mảng hai chiều dưới dạng hàng và cột



Mảng

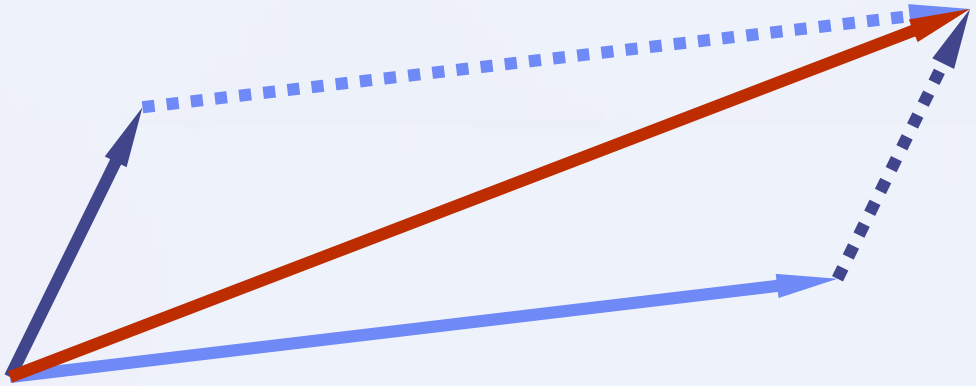
Mảng nhiều chiều

Ví dụ, phần tử có chỉ số [0, 0] nằm ở địa chỉ 5000 trong bộ nhớ, mỗi phần tử có 4 byte và mảng có n cột thì

phần tử có chỉ số [i, j] nằm ở địa chỉ: $5000 + 4 \times (i \times n + j)$

Một mảng hai chiều trong bộ nhớ máy tính

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 tạo Vector

```
int main ()
{
  // constructors used in the same order as described above:
  std::vector<int> first;                                // empty vector of ints
  std::vector<int> second (4,100);                       // four ints with value 100
  std::vector<int> third (second.begin(),second.end());  // iterating through second
  std::vector<int> fourth (third);                       // a copy of third
  // the iterator constructor can also be used to construct from arrays:
  int myints[] = {16,2,77,29};
  std::vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );
  std::cout << "The contents of fifth are:";
  for (std::vector<int>::iterator it = fifth.begin(); it != fifth.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';
}
```

Các thao tác làm việc của Vector

Các thao tác chính trên Vector:

- void push_back (const value_type& val) thêm phần tử vào cuối
- void pop_back() Bớt phần tử cuối vector
- reference operator[] (size_type n) tham chiếu phần tử vị trí n (có thể lấy hoặc gán giá trị -> đọc, ghi)
- reference front() tham chiếu vào phần tử đầu tiên (đọc, ghi)
- reference back() tham chiếu vào phần tử cuối cùng (đọc, ghi)

Thêm vào đó là các phép toán:

- int size() cho biết kích thước (số phần tử hiện tại)
- int capacity() cho biết khả năng lưu trữ của Vector
- bool empty() cho biết Vector có rỗng hay không?
- void clear() xóa hết tất cả các phần tử

Các thao tác làm việc của Vector

Các thao tác bộ lặp Vector:

- iterator begin() vị trí bắt đầu bộ lặp xuôi
- iterator end () vị trí kết thúc bộ lặp xuôi
- reverse_iterator rbegin() vị trí bắt đầu bộ lặp ngược
- reverse_iterator rend () vị trí kết thúc bộ lặp ngược

Các phép toán với vị trí bộ lặp:

- iterator insert (const_iterator position, const value_type& val) chèn 1 phần tử vào vị trí bộ lặp trỏ tới (có 4 hàm chèn nữa – tự tìm hiểu)
- iterator erase (const_iterator position) xóa 1 phần tử tại vị trí bộ lặp trỏ tới (có 1hàm xóa nữa -> tự tìm hiểu)

Cài đặt Vector bằng mảng

- Sử dụng mảng V có kích thước ***cap (capacity)***
- Một biến ***num*** (number) lưu trữ kích thước của vector (số phần tử được lưu trữ)
- Phép toán reference operator[] được thực hiện trong thời gian $O(1)$ bằng việc trả lại $V[r]$

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

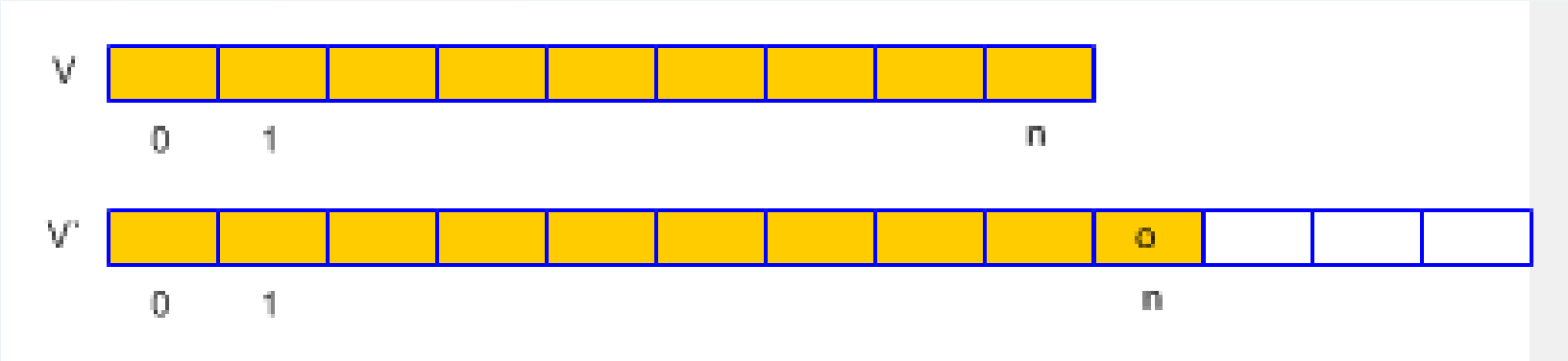
- Cài đặt Vector bằng mảng:
  - Không gian sử cho cấu trúc dữ liệu là $O(n)$
  - ***Các phép toán size***, ***isEmpty***, ***getAtRank*** và ***replaceAtRank*** chạy trong thời gian $O(1)$
  - ***insertAtRank*** và ***removeAtRank*** chạy trong thời gian $O(n)$
- Nếu chúng ta sử dụng một mảng quay vòng 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 mạng hiện tại bằng mảng lớn hơn

Phát triển mảng

- Khi thực hiện phép toán. Nếu mảng đầy sẽ dẫn đến xảy ra 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

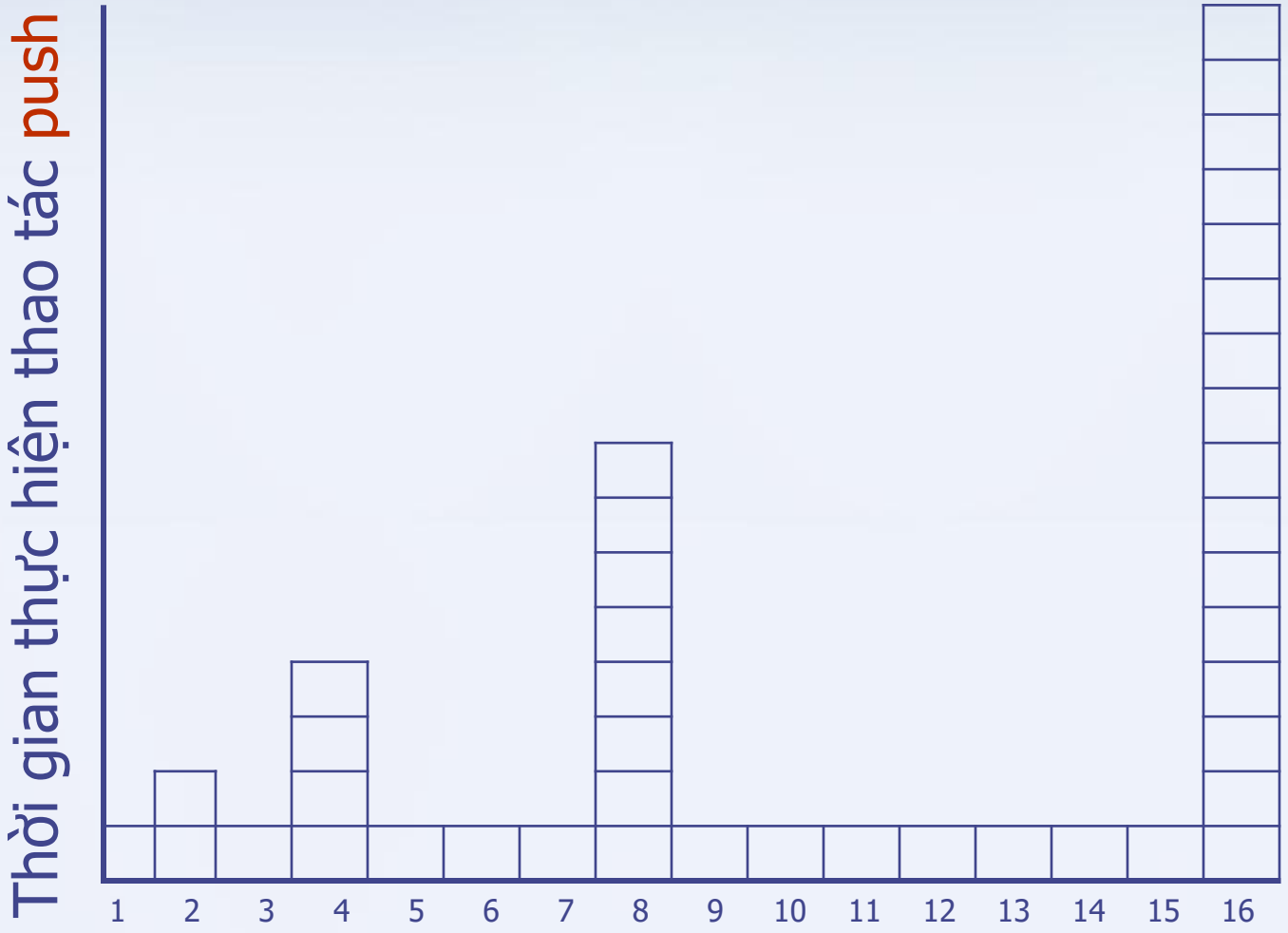
```
Algorithm push(o)
    if n = V.N – 1 then
        A ← Tạo mảng mới (trung gian) có kích thước …
        for i ← 0 to n-1 do
            A[i] ← V[i]          //Sao chép các phần tử của vector sang mảng tạm
        delete V
        V ← A                    //tham chiếu lại mảng V tới A
        V[n] ← o                 //Gán o cho phần tử cuối cùng
        n ← n+ 1                 // Tăng số phần tử của vector
```



So sánh hai chiến lược

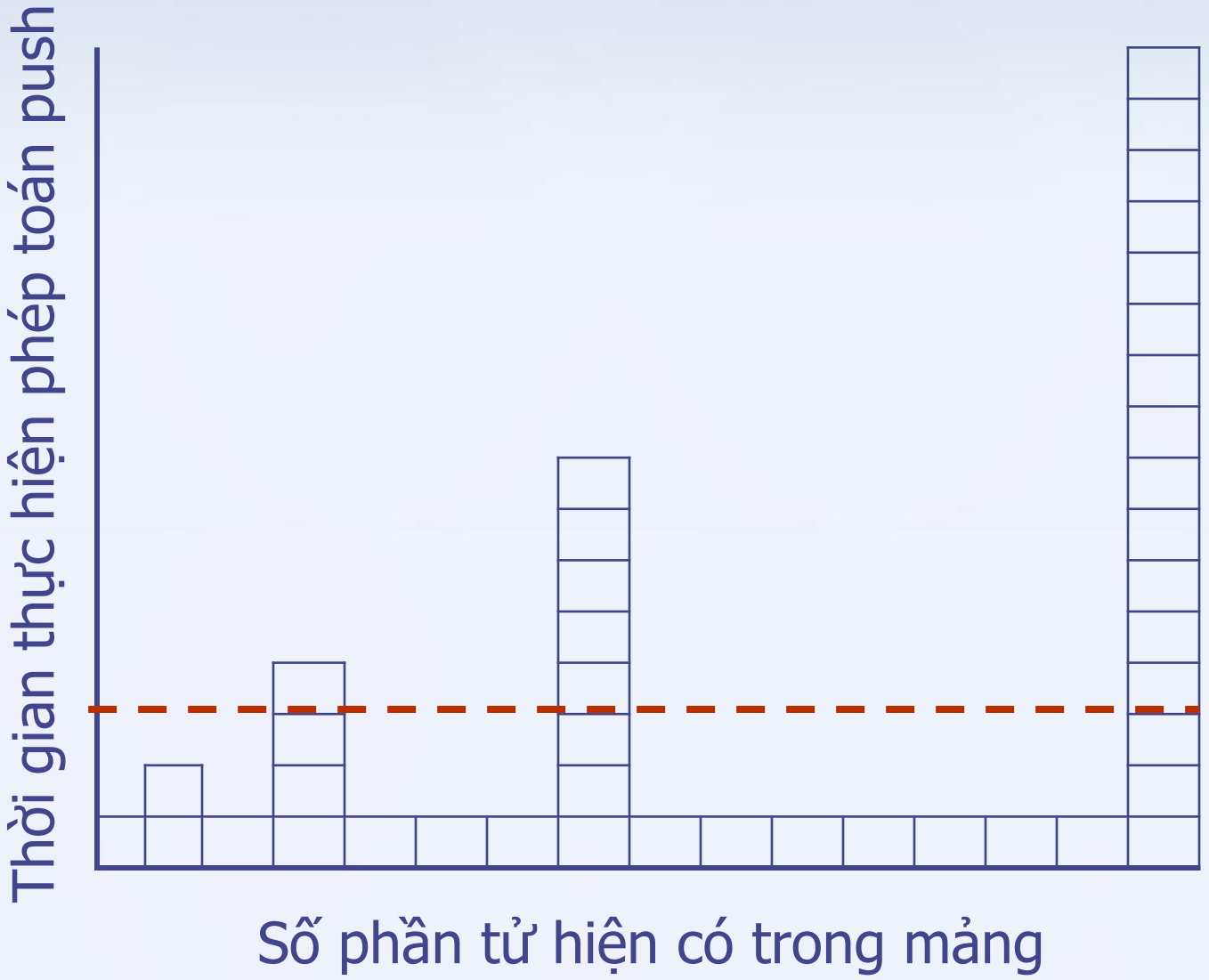
- Ta so sánh chiến lược phát triển theo hằng số và chiến lược gấp đôi bằng cách phân tích tổng thời gian $T(n)$ cần thiết để thực hiện thao tác push một dãy n phần tử vào mảng.
- Chúng ta thực hiện bắt đầu với mảng có 1 phần tử
- Và đi xác định thời gian trung bình khi push một phần tử vào mảng là $T(n)/n$

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

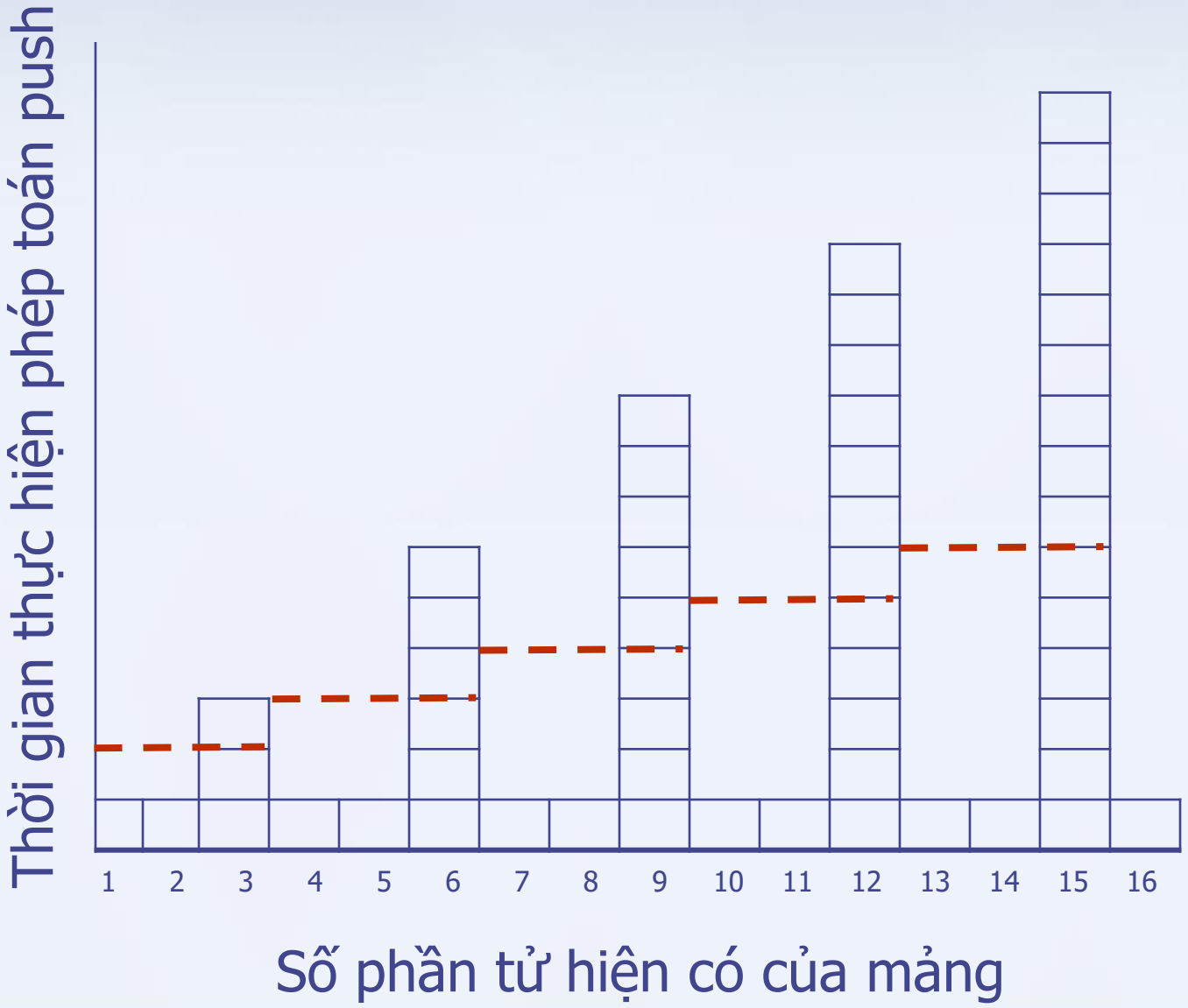


Thời gian thực hiện đưa một dãy các phần tử vào mảng bằng cách sử dụng chiến lược phát triển theo hằng số

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



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

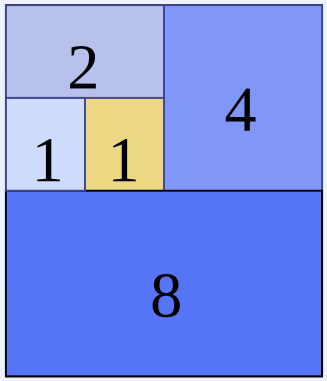
- Chúng ta thay thế mảng $k = \log 2\ n$ **lần**
- Tổng thời gian thực hiện phép toán push n phần tử vào mảng là $T(n)$ và tương ứng là:

$$n + 1 + 2 + 4 + 8 + \ldots + 2^k =$$

$$n + 2^{k+1} - 1 = 3n - 1$$

- $T(n)$ là $O(n)$
- Thời gian trung bình phải trả 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.
- Các ngôn ngữ thủ tục được chia thành 2 lớp ngôn ngữ dựa vào khía cạnh này:
  - Các ngôn ngữ như là: Ada, Algol, C và 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 và 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.

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 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

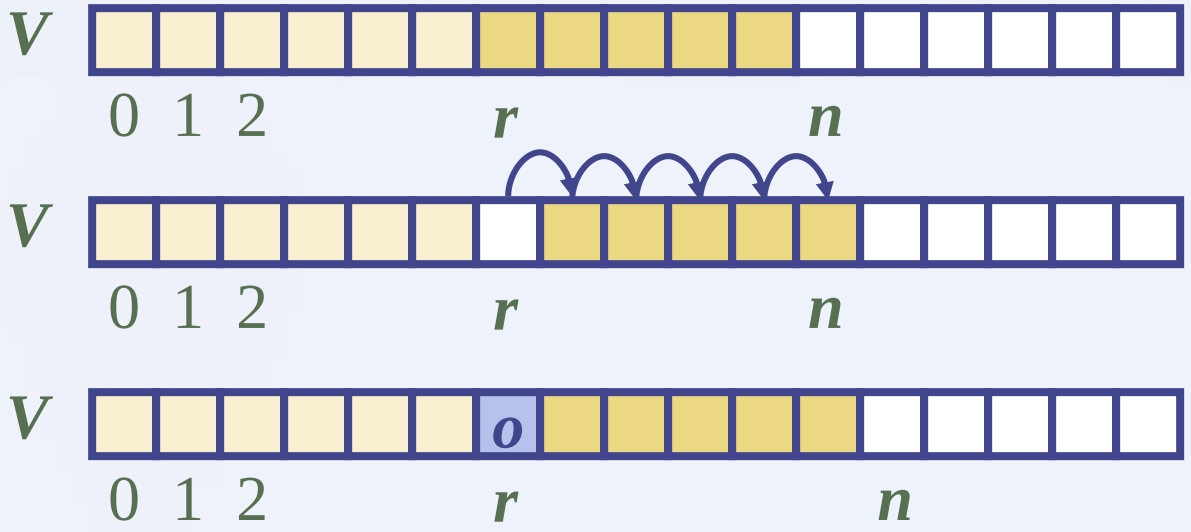
  - thao tác gán =

  - thao tác tham chiếu lấy giá trị tại vị trí nó trỏ tới *it

  - thao tác ++it và it++

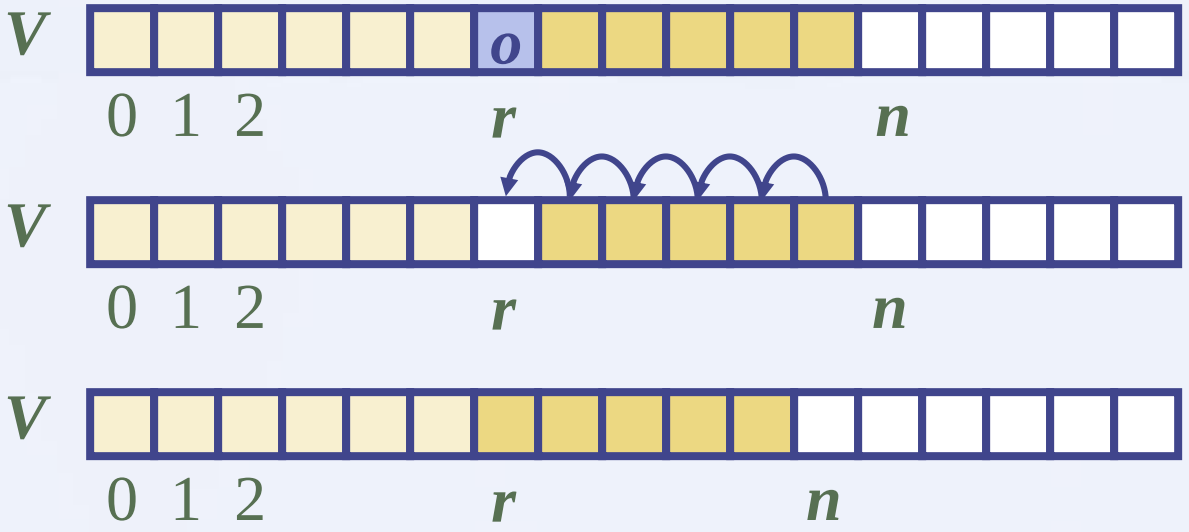
Chèn thêm phần tử

- iterator insert (const_iterator position, const value_type& val)
- 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 iterator erase (const_iterator position) chúng ta cần đẩy n – r – 1 phần tử từ $V[r+1], \ldots, V[n-1]$ về trước một vị trí
- Trong trường hợp xấu nhất ($r = 0$), phép toán thực hiện trong thời gian $O(n)$



Cài đặt Vector bằng C++

Tham khảo code tại

https://ideone.com/LyeviE

Ứng dụng

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

Ví dụ tính n! với n <= 1000

- Ta dùng vector lưu các đảo ngược chữ số của n!
- Ví dụ 7!=5040 thì ta lưu 0 4 0 5 để tính khi xuất ra thì đảo ngược lại

Code minh họa n!

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    long long n,memory=0;
    cin>>n;
    vector<int> V(1,1);
    for(int i=2;i<=n;i++)
    {
        memory=0;
        for(auto it=V.begin();it!=V.end();it++)
        {
            memory+=*it*i;
            *it=memory%10;
            memory/=10;
        }
        while(memory) {V.push_back(memory%10);memory/=10;}
    }
    for(auto it=V.rbegin();it!=V.rend();it++) cout<<*it;
}
```

Hết