LAB. 08
CÁC THUẬT TOÁN TÌM KIẾM

MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

- Hiểu các loại thuật toán tìm kiếm cơ bản.
- Thực hành được các loại thuật toán này.
- Áp dụng cho bài toán thực tế.

THỜI GIAN THỰC HÀNH

Từ 120phút – 240phút

NỘI DUNG THỰC HÀNH

Tìm kiếm: duyệt một danh sách và lấy ra phần tử thoả tiêu chuẩn cho trước.

Là thao tác phổ biến trên máy tính:

- Tìm mẫu tin trong cơ sở dữ liệu
- Tìm kiếm thông tin trên Internet…

Có hai loại tìm kiếm cơ bản:

- Tìm kiếm tuần tự (Sequential/ Linear Search)
- Tìm kiếm nhị phân (Binary Search)

Bài toán tìm kiếm tổng quát:

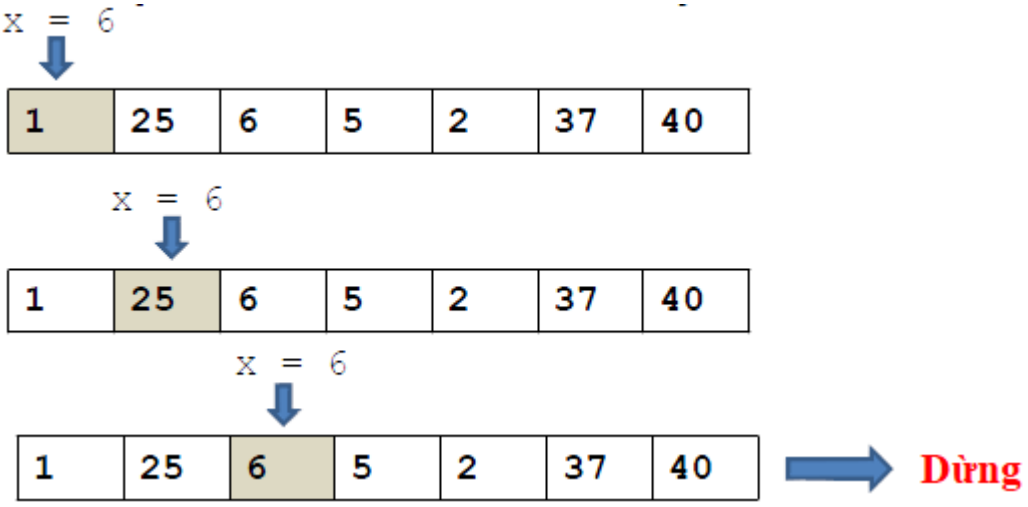
- Đầu vào: mảng A có n phần tử và giá trị x cần tìm
- Trả về: Vị trí phần tử x trong A hoặc –1 nếu x không xuất hiện

1. TÌM KIẾM TUẦN TỰ

Giải thuật: *Lần lượt so sánh x với các phần tử của mảng A cho đến khi gặp được phần tử cần tìm, hoặc hết mảng.*

1.1. Tìm kiếm tuần tự vét cạn (Exhaustive Linear)

Ví dụ: A = {1, 25, 6, 5, 2, 37, 40}, x = 6



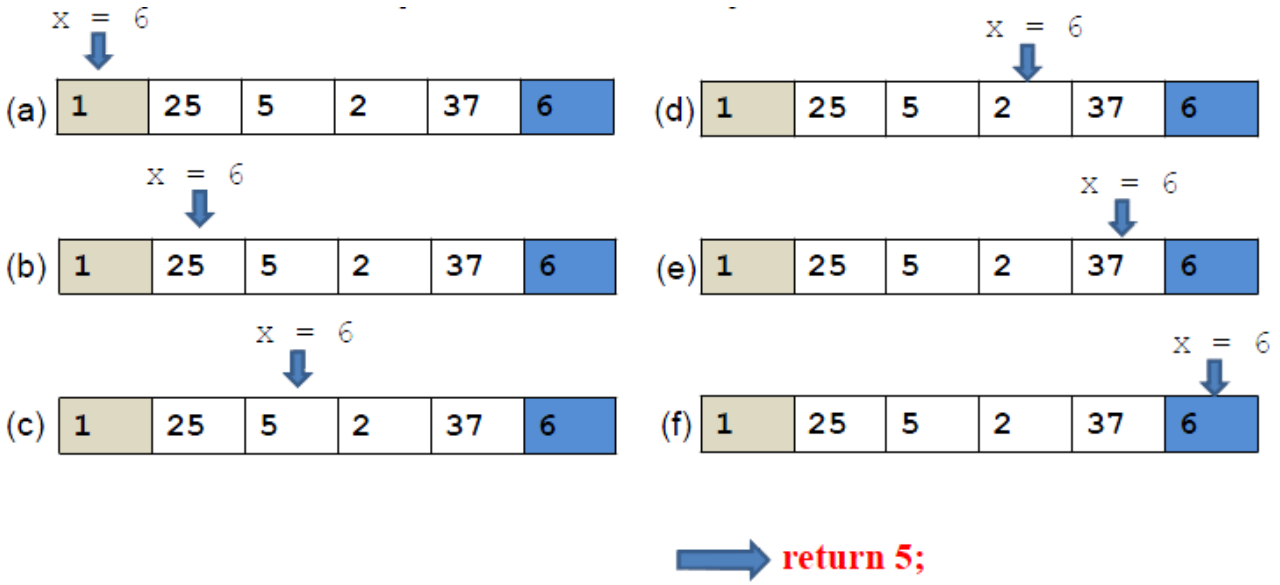

```
int LinearExhaustive(int a[], int n, int x){
      for(int i=0; i<n; i++){
            if(a[i] == x){
                  return i;
            }
      }
      return -1;
}
```

1.2. Tìm kiếm tuần tự lính canh (Sentinel Linear)

Nhận xét:

- Trường hợp x nằm ở 2 biên của mảng A rất hiếm khi xuất hiện (Xác suất là bao nhiêu???)
- Có thể bỏ việc kiểm tra điều kiện cuối mảng (trong vòng lặp for) bằng cách dùng "lính canh".
- Lính canh là phần tử có giá trị bằng với phần tử cần tìm và đặt ở cuối mảng.

Ví dụ: A = {1, 25, 5, 2, 37}, x = 6



```
int LinearSentinel(int a[], int n, int x){
	a[n] = x;
	for (int i = 0; ;i++){
		if (a[i] == x){
			return i;
		}
	}
}
```

2. TÌM KIẾM NHỊ PHÂN

Nhận xét:

- Với mảng A đã được sắp xếp tăng dần, độ phức tạp của tìm kiếm tuần tự không đổi
- Tận dụng thông tin của mảng đã được sắp xếp để giới hạn vị trí của giá trị cần tìm trong mảng

⇨ Thuật toán tìm kiếm nhị phân

Giải thuật:

- So sánh x với phần tử chính giữa của mảng A.
- Nếu x là phần tử giữa thì dừng
- Nếu không, xác định xem x có thể thuộc nửa trái hay nửa phải của A.
- Lặp lại 2 bước trên với nửa đã xác định.

```
int BinarySearch(int a[], int n, int x){
	int l = 0, r = n-1;
	while (l <= r){
		int m = (l + r)/2;
		if (a[m] == x){
			return m;
		}
		else{
			if (x < a[m]){
				r = m - 1;
			}
			else{
				l = m + 1;
			}
		}
	}
	return -1;
}
```

3. SOURCE CODE MẪU

Bài tập (code mẫu: Timkiem.cpp)

```
#include <stdio.h>

int LinearExhaustive(int a[], int n, int x){
	for(int i=0; i<n; i++){
		if(a[i] == x){
			return i;
		}
```

```
	}
	return -1; //-1 ở đây có ý nghĩa là gì?
}

int LinearSentinel(int a[], int n, int x){
	a[n] = x; //Tại sao gán x cho a[n] mà không phải cho a[n-1]?
	for (int i = 0; ;i++){ //Tại sao vòng lặp for là lặp vô tận?
		if (a[i] == x){
			return i;
		}
	}
	//Tại sao không có return ở cuối hàm?
}

int BinarySearch(int a[], int n, int x){
	int l = 0, r = n-1;
	while (l <= r){
		int m = (l + r)/2;
		if (a[m] == x){
			return m;
		}
		else{
			if (x < a[m]){
				r = m - 1;
			}
			else{
				l = m + 1;
			}
		}
	}
	return -1;
}

int main(int argc, char* argv[])
{
	int n;
	printf("Nhap so luong phan tu: ");
	scanf("%d", &n);

	int*  a = new int[n];

	for (int i = 0; i < n; i++)
	{
		printf("Nhap a[%d]: ",i);
		scanf("%d", &a[i]);
	}

	int x;
	printf("Nhap gia tri phan tu can tim kiem: ");
	scanf("%d", &x);

	int i = LinearExhaustive(a, n, x);
	if (i == -1){
		printf("Khong tim thay x trong mang A\n");
	}
	else{
		printf("Vi tri x trong mang A la: %d\n", i + 1);
		//Tại sao lại xuất i+1 ở đây
	}
	return 0;
}
```

1. Biên dịch đoạn chương trình trên.

2. Cho biết kết quả in ra màn hình với các trường hợp mảng A và x sau:

A = 3 6 8 9 0 -1

X = 3

A = 1 2 3 4

X = 5

A = 6 6 6 6 6 6

X = 6

3. Nếu dòng

```
	int i = LinearExhaustive(a, n, x);
```

trong hàm main ta thay hàm LinearExhaustive bằng hàm LinearSentinel. Cho biết kết quả in ra màn hình với A={1,2,3,4} và x=5.

4. Sửa lại code trong hàm main để có thể chạy đúng khi gọi hàm LinearSentinel ở câu 3.

5. Hãy ghi chú các thông tin bằng cách trả lời các câu hỏi ứng với các dòng lệnh có yêu cầu ghi chú (//Ghi chú)

6. Ứng dụng hàm BinarySearch trong hàm main. Diễn giải ý nghĩa code của hàm này.

7. Viết lại hàm BinarySearch dùng đệ quy.

8. (Nâng cao) Đo thời gian tính toán của mỗi thuật toán tìm kiếm. Gợi ý: hàm clock_t của thư viện C/C++ (Xem code mẫu)

```
#include <time.h>

clock_t start, end;
double cpu_time_used;

start = clock();
... /* Do the work. */
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
printf("Thoi gian xu ly la: %f\n", cpu_time_used);
```

9. (Nâng cao) Xây dựng cấu trúc WORD trong từ điển (gồm tên của từ và nghĩa của từ) và áp dụng thuật toán tìm kiếm để xây dựng chương trình tra từ điển. Hàm so sánh chuỗi có thể dùng hàm strcmp (http://www.cplusplus.com/reference/clibrary/cstring/strcmp/)

```
struct WORD{
	char Name[256];
	char Meaning[512];
}
```