Auxiliatura INF-143 "A"
Ordenación y Búsqueda

Univ. Miguel Angel Quispe Mamani
Universidad Mayor de San Andrés
Carrera de Informática

I/2022

1 Búsqueda

Es una técnica en la que resolvemos un problema atravesando completamente el espacio de búsqueda para obtener la solución. Mientras buscamos una solución, podemos eliminar partes del espacio de búsqueda si determinamos que la solución no se encuentra en esas partes.

Un algoritmo de búsqueda es un conjunto de instrucciones que están diseñadas para localizar un elemento con ciertas propiedades dentro de una estructura de datos; por ejemplo, ubicar el registro correspondiente a cierta persona en una base de datos, o el mejor movimiento en una partida de ajedrez.

La variante más simple del problema es la búsqueda de un número en un vector.

1.1 Búsqueda Completa

1.1.1 Espacio de búsqueda complejo

¿Y si el espacio de búsqueda es más complejo?. Ejemplo:

- Todas las permutaciones de n elementos.
- Todos los subconjuntos de n elementos

1.1.2 Búsqueda Secuencial

Consiste en ir comparando el elemento a buscar con cada elemento del vector hasta encontrarlo o hasta que se llegue al final, esto hace que la búsqueda sea secuencialmente (de ahí su nombre).
La existencia se puede asegurar cuando el elemento es localizado, pero no podemos asegurar la no existencia hasta no haber analizado todos los elementos del vector. Se utiliza sin importar si el vector está previamente ordenado o no.

```cpp
int sequential_search(vector <int> &v, int n, int x){
    for(int i = 0; i < n; i++)
        if(v[i] == x)
            return i;
    return -1;
}
```

1.2 Búsqueda Binaria(Divide y vencerás)

Esta basado en la técnica divide y venceras. Se usa sobre un conjunto de elementos ordenados y tiene complejidad de $O(log(n))$.
En una forma general el algoritmo de busqueda binaria se puede usar en cualquier funcion "binaria" o creciente, es decir que sea falsa para un primer intervalo y verdadera para el resto, o viceversa, ya que lo que haremos sera buscar el punto de inflexion donde la funcion cambie su estado.

Búsqueda Binaria sobre arreglos:
Sus principales características:

- Es un algoritmo eficiente de Búsqueda
- El arreglo debe estar ordenado
- Tiene algunas variantes

Funcionamiento
La búsqueda binaria comienza por comparar el elemento del medio del arreglo con el valor buscado. Si el valor buscado es igual al elemento del medio, su posición en el arreglo es retornada. Si el valor buscado es menor o mayor que el elemento del medio, la búsqueda continua en la primera o segunda mitad, respectivamente, dejando la otra mitad fuera de consideración.

```cpp
int binary_search(vector <int> &v, int n, int x){
    int low = 0, high = n;
    while(high - low > 1){
        int mid = (low + high) >> 1;//(low + high) / 2
        if(v[mid] <= x)
            low = mid;
        else
            high = mid;
    }
    if(v[low] == x) return low;
    else return -1;
}
```

2 Ordenación

Los algoritmos de ordenamiento nos permiten, como su nombre lo dice, ordenar vectores con valores ya asignados. Para poder ordenar una cantidad determinada de números almacenados en un vector o lista, existen distintos métodos (algoritmos) con distintas características y complejidad.

2.1 Burbuja(Bubble Sort)

También llamado "Bubble Sort" es el más sencillo de los métodos, pero muy ineficiente. Lo que hace es que recorre el arreglo intercambiando los valores adyacentes que estén ordenados. Se recorre una y otra ves dicho arreglo hasta que sea imposible realizar cambios. Concretamente lo que hace es que agarra el valor mayor y lo recorre de posición en posición hasta ponerlo en su lugar.

```
void Bubble_Sort(vector<int> &v, int n){
    for(int i = 0; i < n - 1; i++){
        for(int j = i + 1; j < n; j++){
            if(v[j] < v[i]){
                int aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        }
    }
}
```

Complejidad: $O(n^2)$

2.2 Inserción(Insertion Sort)

La clasificación por inserción es un algoritmo de clasificación simple que funciona de manera similar a la forma en que clasifica las cartas de juego en sus manos. El vector se divide virtualmente en una parte ordenada y otra sin clasificar. Los valores de la parte sin clasificar se seleccionan y colocan en la posición correcta en la parte clasificada.

```
void Insertion_Sort(vector<int> &v, int n){
    int key, j;
    for(int i = 1; i < n; i++){
        key = v[i];
        j = i - 1;
        while(j >= 0 && v[j] > key){
            v[j + 1] = v[j];
            j = j - 1;
        }
        v[j + 1] = key;
    }
}
```

Complejidad: $O(n^2)$

2.3 Selección(Selection Sort)

El algoritmo de ordenación por selección ordena una matriz encontrando repetidamente el elemento mínimo (considerando el orden ascendente) de la parte no ordenada y colocándolo al principio. El algoritmo mantiene dos submatrices en una matriz determinada.

- El subarreglo que ya está ordenado.
- Subarreglo restante que no está clasificado.

En cada iteración del ordenamiento por selección, el elemento mínimo (considerando el orden ascendente) del subarreglo sin clasificar se elige y se mueve al subarreglo ordenado.

```
void Selection_Sort(vector<int> &v, int n){
    for(int i = 0; i < n - 1; i++){
        int minimum = i;
        for(int j = i + 1; j < n; j++)
            if(v[j] < v[minimum])
                minimum = j;
        int aux = v[minimum];
        v[minimum] = v[i];
        v[i] = aux;
    }
}
```

Complejidad: $O(n^2)$

2.4 Conteo(Counting Sort)

La ordenación por recuento(Counting Sort) es un algoritmo de ordenación que ordena los elementos de un vector contando el número de apariciones de cada elemento único en el vector. El recuento se almacena en un vector auxiliar y la clasificación se realiza mapeando la cuenta como un índice del vector auxiliar.

```
void counting_sort(int v[], int n, int mn, int mx){
    int k = mx - mn + 1;//nro de elementos del vector de ocurrencias
    int occ[k];
    for(int i = 0; i < k; i++)
        occ[i] = 0;
    for(int i = 0; i < n; i++)
        occ[v[i] - mn] += 1;
    int pos = 0;
    for(int i = mn; i <= mx; i++){
        while(occ[i - mn] > 0){
            v[pos] = i;
            pos += 1;
            occ[i - mn] -= 1;
        }
    }
}
```

Complejidad: $O(n+k)$, donde k es el valor máximo encontrado en los elementos del vector.

2.5 Merge Sort

Merge Sort es un algoritmo "Divide and Conquer". Divide el vector de entrada en dos mitades, se llama a sí mismo para las dos mitades y luego fusiona las dos mitades ordenadas.

```
void merge_sort(int v[], int low, int high){
    if(low == high) return;
    int mid = (low + high) >> 1;
    merge_sort(v, low, mid);
    merge_sort(v, mid + 1, high);
    queue<int> L, H;
    for(int i = low; i <= mid; i++)
        L.push(v[i]);//push(elem): inserta elemento a la cola
    for(int i = mid + 1; i <= high; i++)
        H.push(v[i]);
    for(int i = low; i <= high; i++){
        if(L.size() == 0){//size(): tamanio de la cola
            v[i] = H.front();//front(), al frente de la cola
            H.pop();//pop(), elimina el tope
        }else if(H.size() == 0){
            v[i] = L.front();
            L.pop();
        }else{
            if(L.front() <= H.front()){
                v[i] = L.front();
                L.pop();
            }else{
                v[i] = H.front();
                H.pop();
            }
        }
    }
}
```

Complejidad: $O(nlog(n))$