Jose , Julio , Adrian

2016

Practica 3 ED

Ingenieria Informatica

Practica 3 ED

Contenido

Eficiencia teórica:

Analizaremos la eficiencia teórica de los métodos que pide la practica.

Inserción:

```
bool conjunto::insert( const conjunto::value_type & e){
    if ( vm.size() == 0 ){
        vm.push_back(e);
        return true;
    }
    if( vm[vm.size()-1] < e ){
        vm.push_back(e);
        return true;
    }
    else{
      int Iarriba = vm.size()-1;
      int Iabajo = 0;
      int Icentro;
      while (Iabajo <= Iarriba){
      Icentro = (Iarriba + Iabajo)/2;
      if (vm[Icentro] == e)
                return false;
      else
                if (e < vm[Icentro])
                Iarriba=Icentro-1;
                else
                Iabajo=Icentro+1;
      }
        vm.insert(vm.begin()+Icentro,e);
        return true;
    }
}
```

Este método lo primero que comprueba es si el vector esta vacía y así meter el primer elemento.

Cuando el vector ya tenga elementos comprobamos primero si el elemento a insertar es mayor que el ultimo del vector y si lo es, inserta el elemento al final siendo O(1) su eficiencia y así ganamos en eficiencia.

en el peor caso posible, tendríamos que por cada llamada, accediese al else y realiza de la búsqueda binaria que es O(log(n)) y después el método insert de el vector que tiene una eficiencia de O(n) por lo tanto al realizar n inserciones tenemos que la eficiencia final del método es **$O(n^2)$**

Búsqueda:

```
pair<conjunto::value_type,bool> conjunto::find (const string & ID) const{
  pair<conjunto::value_type,bool> resultado;
  bool encontrado = false;

  for( int i = 0; i < vm.size() && !encontrado; i++ ){
    if(vm[i].getID() == ID){
      resultado.first = vm[i];
      resultado.second = true;
      encontrado = true;
    }
  }
  if( !encontrado ){
    mutacion a;
    resultado.first = a;
    resultado.second = false;
  }
  return resultado;
}
```

En este método busca una mutación en un vector de mutaciones cuya ID coincida con otra ID que debe ser indicada como parámetro del método.

Nuestro algoritmo recorre el vector desde la primera posición, hasta que encuentre una mutación cuya ID coincida con la que queremos buscar, llegando hasta la última posición en el caso de que no la encuentre.

Nuestro bucle itera desde i = 0, hasta i < n, y la eficiencia del código que se encuentra en su interior es O(1), por lo tanto, podemos decir que la eficiencia de nuestro algoritmo de búsqueda es **O(n).**

Somos conscientes de que el algoritmo implementado no es lo mas eficiente que podría ser, podríamos haber usado cualquier otro algoritmo de búsqueda mas eficiente que la búsqueda secuencial, como la búsqueda binaria entre otros, pero como el find no se ejecutaba muchas veces en nuestro programa principal(2 veces), hemos decidido dejarlo con búsqueda secuencial, mientras que otros métodos que son muchas mas veces llamados en nuestro main, los hemos implementado con algoritmos más eficientes, como por ejemplo el metodo insert, que es llamado una vez por cada linea a leer del fichero de datos.

Borrado:

```
bool conjunto::erase(const string & ID){
    bool esta = false;
    for( int i=0; i<vm.size() && !esta; i++ ){
        if( vm[i].getID() == ID )
            esta = true;
            vm.erase(vm.begin()+i);
    }
    return esta;
}
```

El método erase es un método que toma como argumento una ID, busca su mutación asociada y la elimina.

Se usa un bucle desde 0 hasta el tamaño del vector, que para si se ha encontrado la mutación que buscamos, eliminándola con vm.erase().

La eficiencia teórica sería de O(n) por la búsqueda y O(n) por vm.erase(), por lo tanto la eficiencia teórica final del método sería **$O(n^2)$**

Eficiencia empírica:

Vamos a mostrar el resultado de los métodos midiendo el tiempo que tarda en ejecutarse modificando el número de elementos que utiliza.

Insert:

```
500 1.224e-06
1000 1.059e-06
2000 9.47e-07
4000 8.48e-07
8000 8.98e-07
16000 1.167e-06
32000 1.001e-06
64000 9.6e-07
128000 9.48e-07
```

Find:

```
584 3.1098e-05
585 3.101e-05
586 3.1169e-05
587 3.1115e-05
1000 5.5776e-05
1001 5.6451e-05
1002 5.7326e-05
1003 5.6734e-05
1004 5.8492e-05
6530 0.000401677
6531 0.000398966
6532 0.000403546
6533 0.000403634
21509 0.00282053
21510 0.00284326
21511 0.0028251
21512 0.00285331
40377 0.00536598
40378 0.00544777
40379 0.00538928
63478 0.00861126
63479 0.0085716
63480 0.00841433
63481 0.00874236
```

Erase:

```
10 23698e-05
100 49878e-05
500 52772e-05
1000 68788e-05
5000 276975e-05
10000 516313e-05
25000 1159786e-05
50000 2731869e-05
75000 3412568e-05
100000 4510500e-05
130000 5839102e-05
```