UNIVERSIDAD DE GRANADA
E.T.S. de Ingenierías Informática y de Telecomunicación

Departamento de Ciencias de la Computación e Inteligencia Artificial

Teoría de Algoritmos

Guión de Prácticas

Práctica 1: Técnica Divide y Vencerás

Curso 2009-2010

Segundo Curso de Ingeniero Técnico en Informática de Gestión
Segundo Curso de Ingeniero Técnico en Informática de Sistemas

Práctica 1

Técnica Divide y Vencerás

1. Objetivos y Evaluación

El objetivo de la práctica es que el alumno comprenda y asimile el funcionamiento de la técnica de diseño de algoritmos "Divide y Vencerás". Esta técnica consiste en resolver un problema a partir de la solución de subproblemas del mismo tipo, pero de menor tamaño. Para ello se proponen dos problemas que deben ser resueltos utilizando algoritmos basados en dicha técnica y sobre los que se realizará un estudio de su eficiencia teórica, empírica e híbrida.

La práctica se evalúa sobre **1 punto**. Se valorará tanto la correcta implementación de los algoritmos como el análisis posterior. La no consideración de las especificaciones de formato y demás cuestiones expresadas en este guión podrán influir negativamente en la evaluación de la práctica.

2. Rango de Dominancia

2.1. Descripción del Problema

Dado un conjunto S de n puntos definidos en $\mathbb{R}^2$ (asumiendo que todos los puntos son distintos entre sí), diremos que un punto $p = (x_p, y_p)$ domina a otro punto $q = (x_q, y_q)$ si $x_p > x_q$ y $y_p > y_q$. El rango de un punto p es el número de puntos del conjunto S que domina. Se desea implementar un algoritmo que calcule el rango de todos los puntos del conjunto S de forma eficiente.

Este cálculo resulta útil en aplicaciones reales donde existen distintos criterios de calidad para evaluar soluciones a un problema. Un ejemplo claro son los algoritmos evolutivos multiobjetivo, que son técnicas aproximadas que permiten resolver problemas de optimización NP-completos con objetivos contradictorios.

2.2. Algoritmo Iterativo

Una primera solución al problema consiste en calcular el rango de cada punto comparándolo con los $n-1$ puntos restantes y contabilizando el número de soluciones que domina. Claramente, este algoritmo es de orden $O(n^2)$. Sin embargo, con un enfoque Divide y Vencerás es posible resolver el problema de forma más eficiente.

2.3. Descripción de la Técnica Divide y Vencerás

Versión 1: ordenando por y en cada paso

En primer lugar, ordenaremos de forma creciente al comienzo del algoritmo (sólo una vez) el vector de puntos según su coordenada x. En cada paso de la recursividad, dividiremos el vector S por la mitad en dos partes (S_1 y S_2) y resolveremos cada una de ellas. Una vez obtenido el rango de los puntos dentro del conjunto S_1 y del conjunto S_2, faltará por combinar ambos conjuntos para obtener los rangos del conjunto unión S.

En este caso, sabemos que ningún punto de S_1 puede dominar a algún punto de S_2 ya que las coordenadas x del primero son siempre menores que las del segundo. Por tanto, los rangos de los puntos del conjunto S_1 siguen siendo los mismos cuando consideramos el conjunto S.

Sin embargo, es posible que puntos de S_2 dominen a puntos de S_1, por lo que deberíamos sumar a los rangos ya calculados en S_2 aquellos nuevos puntos dominados en S_1. Sabemos que un punto de S_2 concreto dominará a todos los puntos de S_1 que tengan en su coordenada y un valor menor que el de éste.

Así, podemos actualizar los rangos de S_2 ordenando el conjunto unión S según su coordenada y y recorriéndolo de menor a mayor. En el momento que encontremos en este vector ordenado un punto p que pertenece a S_2, sabemos que éste dominará a todos los puntos anteriores (es decir, con valor de y menor que él) que pertenecen a S_1, por lo que bastará con añadir al rango previamente calculado para p el número de puntos dominados de S_1 para actualizar su valor.

Versión 2: ordenando por y sólo al principio

Otra posibilidad consiste en evitar ordenar el vector S por su coordenada y antes de aplicar cada combinación. Para ello, podemos mantener dos copias del conjunto de puntos inicial que, al comienzo del algoritmo, serán ordenadas por la coordenada x y la coordenada y respectivamente. De este modo, cada vez que se realiza una combinación, se extrae del conjunto global ordenado por y aquellos puntos que estén incluidos en S (basta con comprobar que su coordenada x está contenido en S hasta que se encuentren todos los puntos implicados). Nótese que estos puntos extraídos ya se encuentran ordenados por y.

2.4. Tareas

Diseñar y analizar el algoritmo iterativo y las dos versiones del enfoque Divide y Vencerás que resuelven este problema. Para ello, se deberá:

1. Diseñar el algoritmo iterativo (sección 2.2) que resuelve el problema y describirlo en forma de pseudocódigo.

2. Para cada una de las dos versiones de la técnica Divide y Vencerás, proveer una descripción del algoritmo en forma de pseudocódigo. En estos dos algoritmos, se considerará un parámetro que define el número de puntos umbral, de tal forma que emplee el algoritmo iterativo implementado en el punto anterior para casos con una dimensión menor o igual que este umbral.

3. Realizar un análisis teórico, empírico e híbrido de los tres algoritmos, considerando un valor umbral de uno (caso base trivial) en los dos algoritmos de Divide y Vencerás. En la documentación deberá describirse el razonamiento seguido en el cálculo de la eficiencia empírica, el modo de realizar los experimentos y los datos del ajuste en el estudio híbrido.

4. A partir de la eficiencia teórica y empírica obtenida en las dos versiones de la técnica Divide y Vencerás con umbral igual a uno, analizar qué algoritmo es mejor y justificar los resultados.

5. Realizar un análisis empírico del valor umbral óptimo para las dos versiones del Divide y Vencerás. Deberá documentarse el experimento realizado y los tiempos obtenidos con los diferentes umbrales.

Implementación

Toda la implementación que haga el alumno en relación a este problema deberá estar contenida en el fichero `dyv_rango.cpp`. No podrá emplearse ninguna librería que no se considere estándar de C/C++. En caso de que algunas funciones auxiliares diseñadas por el alumno se empleen también en el resto de los problemas, estas funciones deberán duplicarse y usar nombres distintos. El fichero no podrá contener la función `main()`. Naturalmente, esto no impide que durante su implementación y depuración el alumno pueda usarla para crear un ejecutable, aunque no deberá incluirse en la versión final que se entregue.

Los prototipos de las tres funciones principales (una por algoritmo) deberán estar al final del fichero y tendrán la siguiente forma:

```
void iter_rango (const vector< pair<float,float> > S, vector<float> &rango);

void dyv_rango_1 (const vector< pair<float,float> > S, vector<float> &rango,
                  const int umbral);

void dyv_rango_2 (const vector< pair<float,float> > S, vector<float> &rango,
                  const int umbral);
```

donde `S` será el vector de puntos (que **puede no estar ordenado inicialmente**); los puntos serán de tipo `pair`, siendo el primer elemento el x y el segundo el y; `rango` es un vector creado

previamente a la llamada a la función donde se devolverá el rango de cada punto (**el orden corresponde con el de los puntos iniciales** almacenados en `S`); y `umbral` define el tamaño máximo para aplicar el caso base.

Es importante destacar que las cabeceras de **estas funciones no tienen porqué coincidir** con las funciones utilizadas para las llamadas recursivas de los algoritmos, se trata sólo de las funciones que sirven de interfaz.

Análisis

Una vez terminada la implementación, deberá realizarse un análisis de eficiencia teórico, empírico e híbrido de los tres algoritmos. Para obtener las tablas de tiempos, el alumno deberá diseñar una batería de experimentos de la forma que estime más oportuna. Se valorará la calidad de la experimentación realizada.

Deberán entregarse tres ficheros de texto ASCII de nombre `iter_rango.dat`, `dyv_rango_1.dat` y `dyv_rango_2.dat` que recojan el comportamiento de los tres algoritmos (con umbral igual a uno en la técnica Divide y Vencerás) para distintos valores de n mediante un formato en dos columnas, la primera con el valor de n y la segunda con el tiempo requerido para la ejecución del algoritmo.

Además, deberá realizarse un **análisis empírico del valor umbral óptimo** para cada una de las dos versiones implementadas que permita la combinación más eficiente del algoritmo iterativo y la técnica Divide y Vencerás. Para ello, el alumno escogerá un valor de n relativamente alto donde el tiempo de ejecución sea relativamente elevado y, por tanto, la posible ganancia de eficiencia sea significativa. Entonces, probará con distintos valores umbrales aplicados sobre **diferentes vectores de la misma dimensión** n escogida. En cada caso, se considerará el tiempo medio necesario para resolver los problemas. Es de esperar que conforme se aumenta el valor del umbral (comenzando desde uno) se observe una ganancia de eficiencia en el algoritmo Divide y Vencerás híbrido hasta llegado un punto en el cual seguir aumentando ese valor umbral empieza a empeorar la eficiencia. Ese punto de inflexión determinará el valor umbral óptimo. Este estudio se realizará para los dos algoritmos de Divide y Vencerás.

Una vez fijado el valor umbral óptimo en cada versión, se analizará su eficiencia empírica probando con los distintos valores de n considerados en el análisis para umbral uno y se comparará con éstos. Deberán entregarse dos ficheros de texto ASCII más, de nombre `dyv_rango_1_umbral.dat` y `dyv_rango_2_umbral.dat`, que incluyan estos resultados.

3. Tornillos y Tuercas

3.1. Descripción del Problema

Se dispone de un conjunto de n tornillos de diferente tamaño y sus correspondientes n tuercas, de forma que cada tornillo encaja perfectamente con una y sólo una tuerca. Dado un tornillo y una tuerca, uno es capaz de determinar si el tornillo es menor que la tuerca, mayor, o encaja exactamente. Sin embargo, **no hay forma de comparar dos tornillos o**

dos tuercas entre ellos para decidir un orden. Se desea ordenar los dos conjuntos de forma que los elementos que ocupan la misma posición en los dos conjuntos emparejen entre sí.

3.2. Descripción de la Técnica Divide y Vencerás

Se procederá mediante un algoritmo Divide y Vencerás de la siguiente forma. Representaremos los conjuntos de tuercas y tornillos mediante dos vectores de tamaño n que contienen dos permutaciones.

El algoritmo comienza escogiendo un tornillo aleatoriamente. Usando este tornillo, divide el vector de tuercas en tres grupos de elementos: un primer grupo con las tuercas menores que el tornillo escogido, otro con la tuerca que encaja perfectamente con el tornillo y finalmente un tercer grupo con las tuercas mayores que el tornillo elegido. A continuación, usando la tuerca que encaja perfectamente con el tornillo elegido, realizamos una partición similar en el conjunto de los tornillos dividiéndolo en tres grupos (aquellos tornillos menores que la tuerca, el tornillo elegido inicialmente que encaja con la tuerca y los tornillos mayores que la tuerca).

El algoritmo, de forma recursiva, aplica el mismo procedimiento a cada uno de los dos grupos que se han formado, es decir, el de los tornillos y tuercas menores que el par encontrado, y el de los mayores.

3.3. Tareas a Realizar

Diseñar y analizar un algoritmo Divide y Vencerás que resuelva este problema siguiendo una representación de vectores.

Implementación

Toda la implementación que deba hacer el alumno en relación a este problema deberá estar contenida en el fichero `dyv_tornillos_y_tuercas.cpp`. No podrá emplearse ninguna librería que no se considere estándar de C/C++. En caso de que algunas funciones auxiliares diseñadas por el alumno se empleen también en el resto de los problemas, estas funciones deberán duplicarse y usar nombres distintos. El fichero no podrá contener la función `main()` ni la función `compara()` que se emplea para comprobar si un tornillo/tuerca es mayor, menor o igual a otro dado. Naturalmente, esto no impide que durante su implementación y depuración el alumno pueda usar estas funciones para crear un ejecutable, aunque **no deberán incluirse en la versión final** que se entregue.

El prototipo de la función principal deberá estar al final del fichero y tendrá la siguiente forma:

```
void dyv_tornillos_y_tuercas (const vector<tornillo> tor,
                              const vector<tuerca> tuer,
                              vector<tornillo> &tor_ordenado,
                              vector<tuerca> &tuer_ordenado);
```

en el que `tor` y `tuer` contienen los vectores que representan el orden inicial de los tornillos y tuercas, y `tor_ordenado` y `tuer_ordenado` son dos vectores creados antes de la llamada a la función y que contienen los conjuntos de tornillos y tuercas ordenados de tal forma que emparejan por posición.

Es importante destacar que la cabecera de **esta función no tiene porqué coincidir** con la función utilizada para las llamadas recursivas del algoritmo, se trata sólo de la función que sirve de interfaz.

La implementación de la función anterior debe utilizar dos tipos de datos, `tornillo` y `tuerca`, que el alumno puede definir a su antojo, y una función auxiliar que permita comparar dos objetos. Esta función auxiliar debe tener la siguiente signatura:

```
int compara (tornillo tor, tuerca tuer);
```

y devolverá `0` cuando ambos objetos encajen perfectamente, `1` cuando el tornillo sea mayor que la tuerca y `-1` en caso de que el tornillo sea menor que la tuerca.

Tanto los tipos de datos `tornillo` y `tuerca` como la función `compara` deben estar definidos en un fichero distinto a `dyv_tornillos_y_tuercas.cpp`. Para probar el correcto funcionamiento del algoritmo crearemos un fichero auxiliar `objeto.hpp` al que haremos referencia mediante el `#include` correspondiente en `dyv_tornillos_y_tuercas.cpp`. Por ejemplo, nuestro fichero auxiliar podría contener algo similar al fragmento de código que se muestra en el siguiente recuadro:

```
typedef struct {
   int grosor;
} tornillo;

typedef struct {
   int anchura;
} tuerca;

int compara (tornillo tor, tuerca tuer)
{
   if (tor.grosor == tuer.anchura)
      return 0;
   else
      if (tor.grosor > tuer.anchura)
         return 1;
      else
         return -1;
}
```

Análisis

Una vez terminada la implementación, deberá realizarse un análisis de eficiencia teórico, empírico e híbrido del algoritmo. Para obtener las tablas de tiempos, el alumno deberá diseñar una batería de experimentos de la forma que estime más oportuna. Se valorará la calidad de la experimentación realizada.

Deberá entregarse un fichero de texto ASCII de nombre `dyv_tornillos_y_tuercas.dat` que incluya los resultados para distintos tamaños con un formato en dos columnas, una primera que contiene el tamaño del problema (es decir, el valor de n) y una segunda que muestra el tiempo requerido. En caso de considerar varios problemas del mismo tamaño, la segunda columna mostrará el tiempo medio.

4. Documentación y Entrega de la Práctica

La práctica deberá entregarse antes del **viernes 4 de diciembre de 2009 a las 23:59**. La entrega se realizará telemáticamente a través de la Web de la asignatura accesible desde

`https://decsai.ugr.es`

Junto con la implementación realizada, se entregará un fichero en formato `pdf` que contendrá la documentación. En ella se indicará todo aquello que el alumno desee destacar en relación al diseño y programación de los algoritmos, así como el análisis teórico, empírico e híbrido. Igualmente, **en un anexo al final de la documentación, se entregará un listado de código** de los ficheros `.cpp` realizados.

A través de la página GAP se subirán los siguientes nueve ficheros:

```
<apellidos_nombre>_practica_1.pdf
dyv_rango.cpp
iter_rango.dat
dyv_rango_1.dat
dyv_rango_2.dat
dyv_rango_1_umbral.dat
dyv_rango_2_umbral.dat
dyv_tornillos_y_tuercas.cpp
dyv_tornillos_y_tuercas.dat
```

Para la evaluación de la práctica no se considerará ningún otro fichero que no corresponda con los nombres indicados anteriormente.

El nombre del fichero que contiene la documentación se compondrá añadiendo los apellidos y nombre del alumno separados por el símbolo de subrayado `_`, sin tildes y sustituyendo la letra 'ñ' por la letra 'n'. Por ejemplo, el alumno José Caños Pérez subirá el siguiente fichero: `Canos_Perez_Jose_practica_1.pdf`