UGR

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones
Grado en Ingeniería Informática

Asignatura: Algorítmica

Práctica 4 (Primera parte): División en dos equipos

Autores:
Míriam Mengíbar Rodríguez
Néstor Rodríguez Vico
Ángel Píñar Rivas

Granada, 18 de mayo de 2016

Índice.

1. Enunciado del problema.

Se desea dividir un conjunto de n personas para formar dos equipos que competirán entre sí. Cada persona tiene un cierto nivel de competición, que viene representado por una puntuación (un valor numérico entero). Con el objeto de que los dos equipos tengan una capacidad de competición similar, se pretende construir los equipos de forma que la suma de las puntuaciones de sus miembros sea la misma.

Hemos resuelto el problema usando dos técnicas distintas; Backtracking y Branch&Bound.

2. Backtracking.

Para empezar, tenemos que calcular cual sería la puntuación de cada equipo. Para ello, sumamos las puntuaciones de todos los jugadores y dividimos entre dos.
Nuestro algoritmo va construyendo una solución parcial introduciendo y sacando jugadores, todo ello mientras haya elementos en esta solución parcial. Antes de empezar este ciclo, introducimos el primer jugador en la solución parcial y en puntuación guardamos su nivel de competición. Comprobamos si la puntuación acumulada es igual que la meta, en caso que lo sea, retornamos esta solución parcial como solución. Si en la solución parcial ya hemos acumulado menos de lo que realmente debería haber, asignamos la solución parcial que llevamos construida a la solución y actualizamos la puntuación de dicha solución.
Posteriormente, comprobamos que el último jugador de la solución parcial sea distinto del último jugador de nuestro grupo de jugadores inicial. En este caso, añadimos a la solución parcial el siguiente al último jugador de la solución parcial. En caso contrario, realizamos un bucle quitando jugadores de la solución parcial y actualizando la puntuación debidamente hasta estar por debajo de la meta.

```
def Backtracking {
	solucionParcial[];
	mejorSolucion[];
	meta = sumaTotalPuntacionesJugadores / 2;
	mejorPuntuacion = meta * 2;

	puntuacion = puntuacionPrimerJugador();
	solucionParcial.añadir(primerJugador);

	while(solucionParcial no vacia){
		if(puntuacion == meta){
			return mejorSolucion;
		}
		else if(puntuacion se acerca mas a la meta que mejorPuntuacion){
			mejorSolucion = solucionParcial; mejorPuntuacion = puntuacion;
		}

		if(ultimoJugadorSolParcial != ultimoJugador()){
			solucionParcial.añadirSiguienteJugador();
		}
		else{
			ultimo = ultimoJugador();
			while(ultimo == ultimoJugador && solucionParcial tiene al menos dos
elementos){
				solucionParcial.sacarUltimoJugador(); //Elimina al ultimo y resta su
puntuacion a "puntuacion"
				ultimo = ultimoJugador();
			}
			solucionParcial.sacarUltimoJugador();
			if(solucionParcial vacia) acaba la iteracion;
			solucionParcial.añadirSiguienteJugador();
		}
		puntuacion += solucionParcial.puntuacionUltimoJugador();
	}
```

```
        return mejorSolucion;
}
```

3. Branch&Bound.

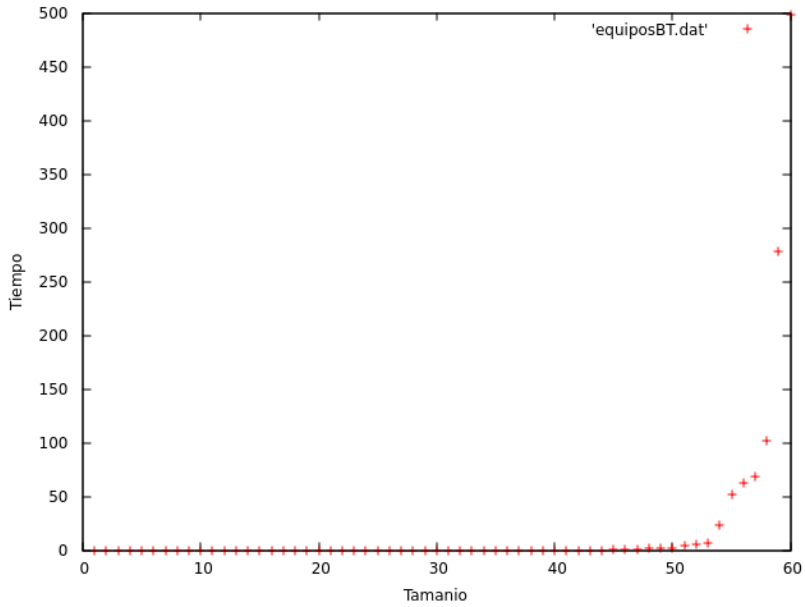
El algoritmo implementado es similar al de BackTracking, pero la diferencia está en el uso de una pila de enteros auxiliar en la que se almacenan los IDs de los jugadores. La idea de poda se basa en el que si el siguiente jugador a añadir sobrepasa la meta, se poda esa rama. Para ello, sacamos los elementos necesarios de la pila para no explorar de nuevo esa rama.

```
def BranchBound {
    solucionParcial[], mejorSoucion[];
    pila<int> auxiliar;
    meta = sumaTotalPuntacionesJugadores / 2;

    for(todos los jugadores){
        auxiliar.push(jugador_actual.ID);
        solucionParci.push_back(jugador_actual);
        while(!auxiliar.empty()){
            if(puntuacion == meta){
                return mejorSolucion;
            }
            else if (puntuacion se acerca mas a la meta que mejorPuntuacion) {
                mejorSolucion = solucionParcial; mejorPuntuacion = puntuacion;
            }

            if(ultimoJugadorSolParcial != ultimoJugador && putuacion < meta){
                solucionParcial.aniadir(jugadorSiguienteAlUltimoSolParcial)
                auxiliar.push(jugadorSiguienteAlUltimoSolParcial.ID)
                puntuacion += jugadorSiguienteAlUltimoSolParcial.competicion
            }
            else {
                ciclo:
                    ultID = auxiliar.top();
                    puntuacion -= solucionParcial.back().competicion;
                    solucionParcial.pop_back(); auxiliar.pop();
                    if (ultID == ultimoJugador && !auxiliar.empty()) goto ciclo;
                if(auxiliar.empty()) break;
                solucionParcial.push_bak(jugadores[ultID + 1]);
                auxiliar.push(aniadir.ID);
                puntuacion += jugadores[ultID + 1].competicion;
            }
        }
    }

    return mejorSolucion;
}
```

4. Análisis empírico.

4.1. Backtracking.

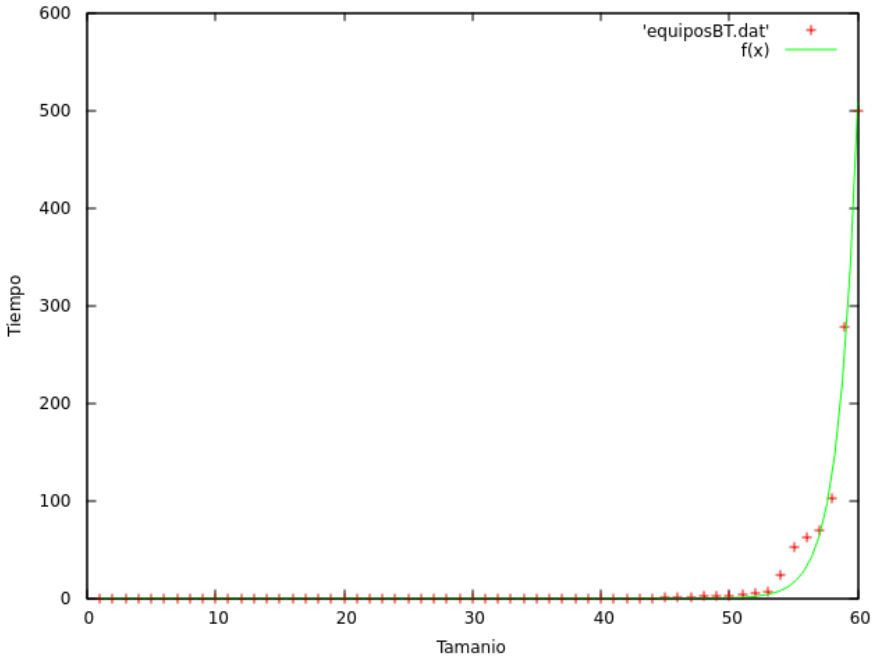
El algoritmo de Backtracking, teóricamente, tiene un orden de ejecución de 2^n. Para ver que el algoritmo se ajusta correctamente en tiempos de ejecución hemos realizado la siguiente gráfica:



El ajuste de los datos obtenidos con la función f(x) = a*2**x nos devuelve la constante oculta que sigue:

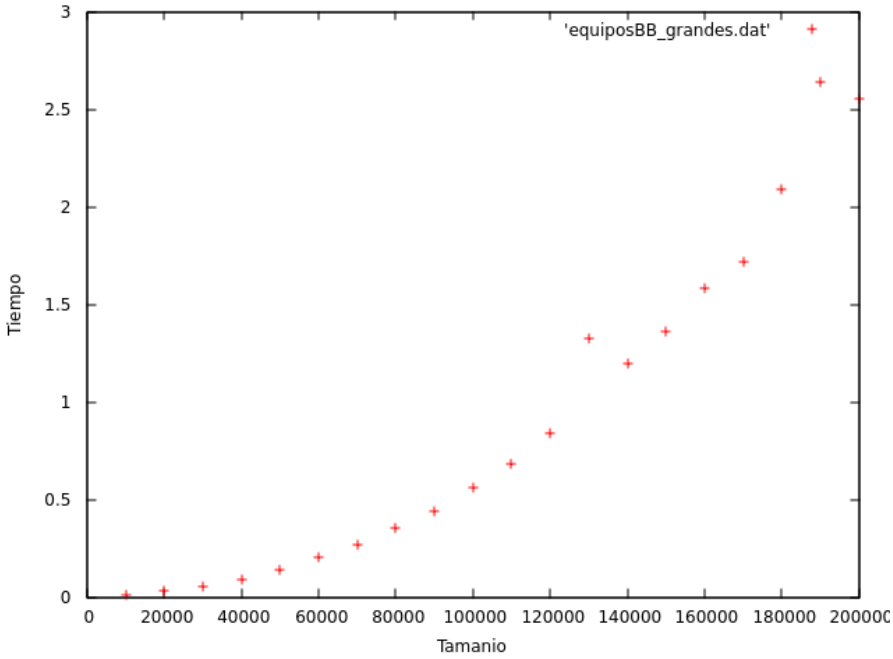
a	= 4.41306e-16	+/- 6.084e-18	(1.379%)

Para ver que realmente se comporta como esperábamos hemos pintado la función ajustada junto con los datos obtenidos empíricamente. Este es el resultado:



4.2. Branch&Bound.

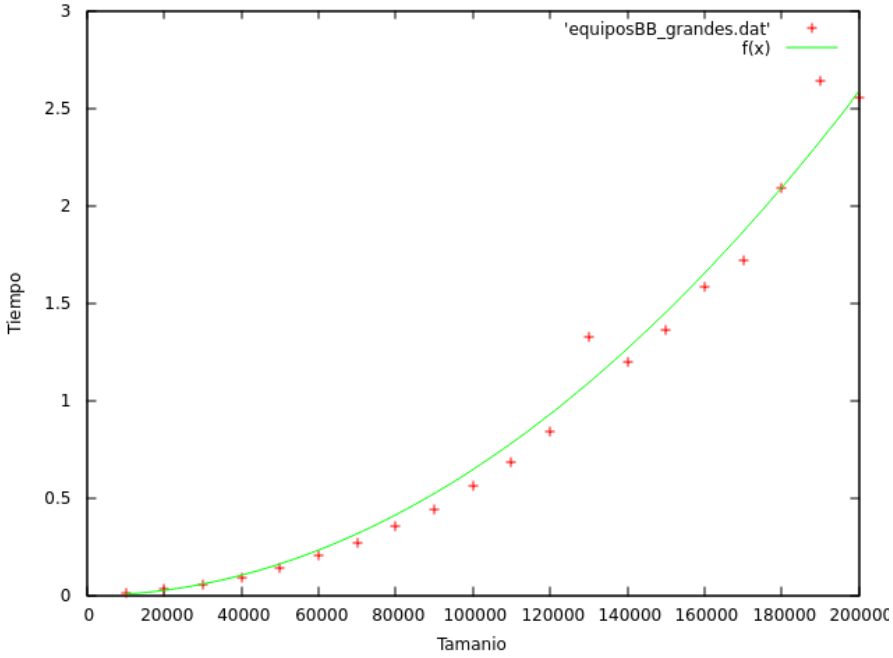
El algoritmo de Backtracking, teóricamente, tiene un orden de ejecución de x^2. Para ver que el algoritmo se ajusta correctamente en tiempos de ejecución hemos realizado la siguiente gráfica:



El ajuste de los datos obtenidos con la función f(x) = a*x*x nos devuelve la constante oculta que sigue:

a	= 6.45889e-11	+/- 1.296e-12	(2.006%)

Para ver que realmente se comporta como esperábamos hemos pintado la función ajustada junto con los datos obtenidos empíricamente. Este es el resultado:



4.3. Comparación de los dos algoritmos.

Para ver la diferencia que hay entre los tiempos de ejecución de ambos algoritmos a la hora de realizar la misma tarea, hemos ejecutado los algoritmos con equipos del mismo tamaño. Estos son los tiempos obtenidos:

Tamaño	Backtracking	Branch&Bound
51	4.30079	2.0903e-05
52	5.3923	1.7196e-05
53	6.36779	2.1329e-05
54	23.5523	2.4292e-05
55	52.0758	1.5957e-05
56	63.2138	4.1667e-05
57	69.2986	2.7149e-05
58	102.418	2.7704e-05
59	277.833	4.255e-05
60	499.175	3.9529e-05