UNIVERSIDAD DE GRANADA
E.T.S. DE INGENIERIA INFORMATICA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Teoría de Algoritmos

Segunda Práctica

Backtracking y Branch and Bound

Curso 2009-10

Ingeniería Técnica en Informática de Gestión
Ingeniería Técnica en Informática de Sistemas

Autores: Jesús Alcalá Fernández y Rafael Alcalá Fernández

1.1. Técnica

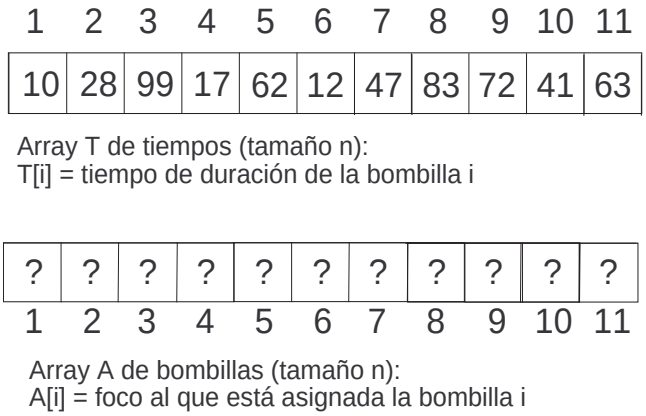
El objetivo de esta práctica es estudiar el comportamiento de las técnicas exactas de diseño de algoritmos *Backtracking* y *Branch and Bound*, basadas en la exploración de grafos. Para ello se requerirá que el alumno implemente algoritmos basados en estas técnicas y realice un estudio de su eficiencia al resolver un problema. El problema consistirá en distribuir un conjunto de bombillas de 200w entre distintos puntos de luz para que se mantenga una iluminación de 2 Kw el mayor tiempo posible (problema de asignación para maximización).

1.2. Descripción del Problema

El Director de la Escuela de Informática quiere celebrar una macrofiesta para todo el personal docente y administrativo y los alumnos. Aprovechando el buen tiempo, la fiesta tendría lugar en el jardín. Se prevée que la fiesta se prolongue hasta horas nocturnas. Para evitar sorpresas (desagradables) se quieren instalar 10 focos para iluminar todo el jardín. A tal efecto se han recabado n bombillas ($n \geq 10$) de 200w del almacén de la E.T.S.I.I.T., pero como dichas bombillas pertenecen a marcas, calidades y remesas diferentes, cada una tiene una duración (en tiempo) t_i distinta ($10 \leq t_i \leq 100$ minutos). El objetivo es mantener una iluminación de un total de 2 Kw durante el mayor tiempo posible.

Para ello, la dirección de la escuela quiere desarrollar una aplicación informática que asigne las bombillas de manera automática y resuelva el problema para futuros grandes eventos. Es decir, el algoritmo desarrollado deberá funcionar para distintos valores de n. En general, este algoritmo o una extensión adecuada sería útil para resolver problemas de asignación en los que se produzcan beneficios sobre los objetos asignados.

Una forma de representar una instancia de este problema es mediante un vector de tiempos T (de tamaño n). Por otro lado, una solución al problema consistiría en un vector A (de tamaño n) que represente el foco (o punto de luz) al que ha sido asignada cada una de las bombillas (con valores entre 1 y 10). El vector A indicaría como hay que distribuir las bombillas. En la siguiente figura, se muestra un ejemplo para una instancia de tamaño 11:



Un algoritmo *Greedy* que puede resolver este problema de manera aproximada consiste en asignar en cada momento la bombilla que tenga mayor duración al foco que acumule menor tiempo de luz en cada momento. Una forma de implementar dicha idea de manera eficiente es trabajar bajo la premisa de tener las bombillas ordenadas de mayor a menor durabilidad. De esta manera, para cada bombilla (siguiendo el orden establecido) se buscará el foco con menor tiempo acumulado hasta el momento y se le asignará en el vector solución A.

Sea T' el vector ordenado y A' la solución según dicho orden. Adicionalmente, consideremos la existencia de un vector F' de tamaño 10 que permita almacenar el tiempo acumulado para cada foco, y un vector de índices $OrdF'$ que permita en todo momento recorrer F' de manera ordenada (de menor a mayor tiempo total acumulado). El algoritmo se describe en la figura 1.

Greedy:

```
- A' = ∅
- Para (i=1; i<=10; i++)
      - F'[i] = 0
      - OrdF'[i] = i
- A'[1] = OrdF'[1] //se asigna al primer foco
- F'[OrdF'[1]] += T'[1] //acumular tiempo al foco
- Para (i=2; i<=10 && F'[OrdF'[i]]<F'[OrdF'[i-1]]; i++)
      - swap (OrdF'[i], OrdF'[i-1])

- Para (b=2; b<=n; b++) //para cada bombilla
      - A'[b] = OrdF'[1] //se asigna al peor foco
      - F'[OrdF'[1]] += T'[b] //acumular tiempo al foco
      - Para (i=2; i<=10 && F'[OrdF'[i]]<F'[OrdF'[i-1]]; i++)
            - swap (OrdF'[i], OrdF'[i-1])
```

FIGURA 1.

Desgraciadamente, este algoritmo no nos puede asegurar una solución óptima. Necesitamos utilizar una técnica de exploración de soluciones como Backtracking o Branch and Bound, que explore en el espacio de soluciones en busca de la mejor solución posible (utilizando si es posible la información que nos proporciona la solución Greedy para reducir dicho espacio de búsqueda).

1.3. Tareas a realizar

Las tareas a realizar en esta práctica son las siguientes:

1. Implementar un algoritmo Backtracking: Es bastante simple resolver este problema utilizando un algoritmo backtracking. Una aproximación directa intentaría asignar un foco válido a cada una de las bombillas disponibles en el vector solución A. Cuando no queden más focos que chequear para una bombilla, el algoritmo hace una vuelta atrás seleccionando otro foco válido para la bombilla que le precede en el nivel anterior.

   Para agilizar la búsqueda de la solución se deberán considerar como valores válidos para una posición sólo aquéllos que satisfacen las restricciones del problema, eliminando valores no permitidos (en este problema valores que ya no tiene sentido asignar, por ejemplo, que el número de focos sin bombillas sea mayor que el número de bombillas que quedan, etc.).

2. Implementar un algoritmo Branch and Bound: Para resolver el problema esta técnica puede utilizar como cota superior (cota local) la duración total estimada al repartir de la mejor manera posible el tiempo total de las bombillas que aún no tienen asignado un foco, *Resto*. Dada una solución parcial A' con sus respectivos tiempos acumulados por foco F' y dado un vector de índices $OrdF'$ con la ordenación de menor a mayor tiempo acumulado en F', el algoritmo para el cálculo de la cota local se describe en la figura 2.

   Cota Local (estimación de lo mejor):

   ```
   - R = Resto //Tiempo total de las bombillas restantes
   - Obj  = F'[OrdF'[1]] //minimo acumulado
   - Para (i=1; i<10 && R>0; i++)
         - tmp = i * (F'[OrdF'[i+1]] - Obj ) //necesario para equiparar focos
         - if (R < tmp) tmp = R
         - R -= tmp
         - Obj  += tmp / i //división real
   - if (R > 0)  Obj += R / 10 //división real
   ```

FIGURA 2.

Como cota inferior (actualización de cota global) se considerará la duración del foco con menor tiempo acumulado para la solución aproximada obtenida completando el vector mediante el algoritmo greedy para la parte que falta. Este valor se calculará cada vez que se explore un nodo para actualizar la cota global si la mejorase antes de aplicar la poda.

Como criterio para seleccionar el siguiente nodo a expandir se empleará el LC o "más prometedor". Hay distintas posibilidades para determinar qué nodos son más prometedores. *Se considerará como nodo más prometedor aquél que presente el mejor valor medio entre la cota superior e inferior*. Para ello debemos hacer uso de una cola ordenada *LC-fifo* que almacene los nodos ya generados (nodos vivos). Esta cola quedará ordenada en cada momento por el criterio que se ha indicado pero dejando los nodos en el mismo orden en el que entraron cuando tengan igual prioridad.

1.4. Obtención de resultados

Cada implementación se ejecutará al menos sobre 20 casos distintos del problema (varios casos generados aleatoriamente para un tamaño de problema fijado por el alumno dependiendo de la velocidad de su máquina). Se construirá una tabla que recoja para cada caso el valor de la solución obtenida, el número de nodos que han sido expandidos, el tamaño máximo de la cola ordenada *LC-fifo* de nodos vivos (sólo en Branch and Bound), número de veces que se realiza la poda y el tiempo empleado para resolver cada uno de ellos.

Utilizando la media de los valores mostrados en dicha tabla se debe tener una aproximación del tiempo de ejecución del algoritmo. No es posible obtener una ecuación exacta de este tiempo en algoritmos basados en las técnicas Backtracking y Branch and Bound, ya que la expresión del mismo depende del número de nodos generados, el cual varía en función de la instancia concreta del problema.

La ecuación de tiempos respondería a la siguiente fórmula:

$$t(n) = N \cdot t_{nodo}(n)$$

donde:

- N es el número medio de nodos generados en la ejecución del algoritmo sobre las distintas instancias del problema.
- $t_{nodo}(n)$ es el tiempo medio de ejecución de las operaciones efectuadas en cada nodo.

De este modo, para obtener una aproximación general de esta fórmula para nuestro algoritmo deberemos proceder del siguiente modo:

1. Aproximar una expresión polinómica en función de n que aproxime el tiempo medio por nodo ($t_{nodo}(n)$) calculado para las distintas instancias. Para ello, efectuar cinco regresiones polinómicas: de orden uno ($f_t(n) = a \cdot n + b$), de orden dos ($f_t(n) = a \cdot n^2 + b \cdot n + c$), ..., hasta de orden cinco. Escoger aquélla que devuelva menor error con respecto al conjunto de datos.
2. Aproximar una expresión polinómica en función de n ($f_N(n)$) que estime el número medio de nodos (N) calculado para las distintas instancias. Para ello, proceder de la misma manera que en el punto 1.

Una vez efectuados los pasos anteriores, la ecuación de tiempo sería:

$$t(n) = f_N(n) \cdot f_t(n)$$

1.5. Entrada y Salida

Se pide diseñar e implementar los algoritmos comentados para resolver el problema. Toda la implementación que deba hacer el alumno en relación a cada algoritmo deberá estar contenida en un único fichero. 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 en varios algoritmos, estas funciones deberán duplicarse y usar nombres distintos. Los ficheros no podrán contener la función *main()*. Naturalmente, esto no impide que durante la implementación y depuración el alumno pueda usarla para crear un ejecutable, aunque no deberá incluirse en la versión final que se entregue. La función principal deberá estar al final del fichero.

Los prototipos de la función principal y los nombres de ficheros para cada algoritmo se indican a continuación **y deben respetarse**, donde T es el vector de tiempos, A es el vector en el que se devuelve la solución y `'tiempo'` es el tiempo máximo que la iluminación tendrá un total de 2 Kw. Tanto T como A se crearán antes de llamar a la función, por tanto la función no crea ni

destruye las matrices. Además, T se supone inicializado fuera con los valores apropiados para cada instancia. Es importante destacar que las cabeceras de **estas funciones no tienen por qué coincidir** con la función utilizada para implementar el algoritmo, se trata sólo de las funciones que sirven de interfaz, en las que, la primera tarea consistiría en ordenar las bombillas de mayor a menor durabilidad en un vector temporal que será el utilizado para aplicar los algoritmos implementados. Por lo tanto, habrá que considerar unos índices que nos permitan asignar los resultados en las posiciones originales para cada bombilla (al estilo de los índices usados en la práctica 1, ejercicio 1). Las cabeceras de estas funciones son:

a) Algoritmo Backtracking (fichero `bombillas_BK.cpp`). El prototipo de la función será:

```
void bombillas_BK (const vector<int> T,
vector<int> &A, int &tiempo);
```

b) Algoritmo Branch and Bound (fichero `bombillas_BB.cpp`). El prototipo será:

```
void bombillas_BB (const vector<int> T,
vector<int> &A, int &tiempo);
```

1.6. Documentación y Entrega de la Práctica

La práctica deberá entregarse antes del **Viernes 15 de Enero de 2010 a las 23:59**.

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 de eficiencia. Igualmente, **en un anexo al final de la documentación, se entregará un listado del código** de los ficheros *.cpp* realizados.

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

```
<apellidos_nombre>_practica_2.pdf
bombillas_BK.cpp
bombillas_BK_fN.dat
bombillas_BK_ft.dat
bombillas_BB.cpp
bombillas_BB_fN.dat
```

`bombillas_BB_ft.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_2.pdf`