24-5-2017

Práctica 3

Algoritmos Greedy

Pablo Álvarez Cabrera

Johanna Capote Robayna

Cristina de la Torre Valverde

Guille Galindo Ortuño

Adolfo Soto Werner

# Índice

# Introducción

# Diseño de la solución

# Esqueleto del Algoritmo greedy

# Ejemplo para comprender el funcionamiento del algoritmo.

# Ejemplo real para aplicar el algoritmo.

# Eficiencia teórica del algoritmo.

# Compilación y ejecución del algoritmo.

# Introducción.

El objetivo de esta práctica es implementar un algoritmo Greedy que solucione el problema conocido como Bin Packing Problem o problema del embalaje. Este problema consiste en buscar la mejor forma de introducir elementos con distintos volúmenes en cajas de un mismo volumen de forma que las cajas estén lo mas llenas posibles y tener el menor número posible de cajas.

El planteamiento del problema es que teniendo un conjunto de n elementos con un volumen vi cada uno de ellos debemos empaquetarlos en cajas con un volumen v de forma que la suma de todos los volúmenes de los objetos que hay en cada caja no debe superar el volumen de la caja en la que se encuentran. Es decir:

# Diseño de la solución.

Nuestro algoritmo está diseñado para que la solución que devuelva sea un vector de vectores, es decir un vector que en cada entrada tiene un puntero a otro vector.

El vector que contiene a los demás se corresponde con el conjunto de cajas que se han llenado de elementos al acabar al ejecución del algoritmo y cada uno de los vectores contenidos en el primer vector es un vector de pares de enteros, o dicho de otro modo, tiene un par de enteros en cada entrada los cuales son el volumen de cada elemento y la posición en la que se encuentra en el vector que se pasa como parámetro al algoritmo.

La lista de candidatos se corresponde con el vector de enteros en el que se encuentran los volúmenes de cada uno de los elementos que hay que empaquetar.

La función solución consiste en comprobar si todos los elementos han sido ya empaquetados.

El resto de componentes del algoritmo Greedy se indican en el punto siguiente.

# Esqueleto del algoritmo Greedy.

El funcionamiento de nuestro algoritmo se basa en buscar el elemento de mayor volumen de todos los candidatos mientras este conjunto no esté vacío para introducirlo en una caja ya que para este elemento es mas fácil encontrar otro elemento que encaje en el hueco que deja que encajarlo en un hueco que deje otro elemento en otra caja.

Una vez se hace esto se quita este elemento de la lista de candidatos y se introduce en un par de enteros que contendrá el indice en el que se encontraba en el vector de candidatos y el volumen que ocupa. Este par se introduce en un vector de pares que representaría el paquete que estamos llenando. Si resulta que por su volumen no este elemento no cabe en ninguno de los vectores que tenemos parcialmente llenos se crea otro nuevo, se introduce en él y se aumenta el tamaño del vector de vectores. Después de esto se pasa a buscar el siguiente mas grande y se va realizando el mismo proceso hasta que empaquetamos todos los candidatos que teníamos, es decir, cuando el vector P se queda vacio.

Las componentes greedy de nuestro algoritmo son:

* Lista de candidatos: Elementos que tenemos que empaquetar.
* Lista de candidatos utilizados: El vector de vectores que vamos rellenando que los elementos que vamos empaquetando.
* Función solución: Comprueba si la lista de candidatos está vacía o no.
* Función de selección: El candidato que se selecciona en cada momento es el que tenga el mayor volúmenes de todos los que aún no han sido utilizados.
* Criterio de factibilidad:
* Función objetivo: Devolver la forma de empaquetar los candidatos de forma que se utilicen el menor numero posible de paquetes.

El pseudocódigo de nuestro algoritmo es el siguiente:

|  |
| --- |
| C=Conjunto de Contenedores de tamaño V.  P=paquetes a embalar  C=0  Mientras P != 0 hacer:  x=mayor elemento de P  P=P\{x}  Si {x} cabe en Ci  Ci=Ci U {x}  Si no  C aumenta su tamaño en 1  Cn={x}  Fin-mientras  Devuelve C  n=índice del último vector de C  i=índice del vector de C |

# Ejemplo para comprender el funcionamiento del algoritmo.

Sea P un vector con los volúmenes de cada elemento a empaquetar tal que P = {8,5,4,7,3,1} y contamos con cajas para embalar con un volumen V = 10. Los colores son simplemente para ver en que caja se encuentra cada objeto.

El primer paso del algoritmo es buscar el elemento mas grande del vector en nuestro caso 8. y quitarlo del vector de candidatos. Como aún no hay ninguna caja creamos una y lo introducimos. Luego nuestra lista de candidatos sería ahora P = {5,4,7,3,1}.

Caja N.º1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

El siguiente elemento que encontramos es el que tiene volumen 7. Como no cabe en la primera caja que tenemos parcialmente llena creamos otra e introducimos este objeto. Luego nuestra lista de candidatos sería ahora P = {5,4,3,1}.

Caja N.º1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

El siguiente elemento que encontramos es el que tiene volumen 5. Como no cabe en la primera caja ni tampoco en la segunda que tenemos parcialmente llena creamos otra e introducimos este objeto. Luego nuestra lista de candidatos sería ahora P = {4,3,1}.

Caja N.º1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

El siguiente elemento que encontramos es el que tiene volumen 4. No cabe en la primera caja que tenemos parcialmente llena, tampoco en la segunda pero si en la tercera. Introducimos entonces este elemento en la caja N.º 3. Luego nuestra lista de candidatos sería ahora P = {3,1}.

Caja N.º1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

El siguiente elemento que encontramos es el que tiene volumen 3. No cabe en la primera caja que tenemos parcialmente llena pero si en la segunda. Introducimos entonces este elemento en la caja N.º2. Luego nuestra lista de candidatos sería ahora P = {1}.

Caja N.º1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

El último elemento que nos queda por empaquetar es el que tiene volumen 1. Como encaja en la caja N.º1 lo introducimos en ella. Luego nuestra lista de candidatos sería ahora P = {}.

Caja N.º1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Caja N.º3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Como nuestra lista de candidatos se ha quedado vacía significa que el algoritmo ha terminado y generaría un vector (al que denotaremos por V) de vectores (que denotaremos por Vi ) de pares en el que habría tres entradas en el vector de vectores. En la primera entrada de V1 estarán los pares <8,0> y <1,5> en la segunda. En la primera entrada de V2 estarán los pares <7,3> y <3,4> en la segunda. Por último en la primera entrada de V3 estarán los pares <5,1> y <4,2> en la segunda.

# Ejemplo real para aplicar el algoritmo.

Una situación en la que este algoritmo se podría aplicar sería en la empresa cervecera Alhambra. A la hora de hacer realizar el transporte de las cajas de cerveza que van saliendo de la fábrica hacia el almacén que tengan en el polígono industrial. Como todas las cervezas van al mismo destino optimizar estos trayectos no dependen de la ruta que hagan sino del número de trayectos que realicen los camiones.

Suponiendo que todos los camiones tienen la misma capacidad y que todas las cervezas se empaquetan en cajas de un volumen determinado dependiendo de su tipo. Aplicar este algoritmo supondría encontrar la manera de guardar las cajas de cerveza en los camiones de manera que se necesiten el menor numero de camiones posible y de esta forma la empresa ahorre en el gasto de gasolina que supone el transporte.

# Eficiencia teórica del algoritmo.

# Compilación y ejecución del algoritmo.

Para compilar el programa que incluye el algoritmo hemos creado un Makefile que compila los ficheros Problema.cpp (contiene una clase que encapsula los datos que necesita el algoritmo, los getters y las funciones para leer desde un fichero), Solucion.cpp (que contiene el formato de la solución del algoritmo y las funciones para añadir y devolver elementos) y Algoritmos.cpp (que contiene solamente el algoritmo greedy) con sus correspondientes ficheros .h y el main.cpp (Es el programa en sí, lee desde un fichero los datos ejecuta el algoritmo y muestra los datos). El Makefile es:

SRC = src

INC = include

OBJ = obj

BIN = bin

CXX = g++

CPPFLAGS = -O2 -Wall -g -I./$(INC) -std=c++11

all: $(BIN)/binPacking

$(BIN)/binPacking : $(SRC)/main.cpp $(INC)/Problema.h $(SRC)/Problema.cpp $(INC)/Solucion.h $(SRC)/Solucion.cpp $(INC)/Algoritmos.h $(SRC)/Algoritmos.cpp

$(CXX) $(CPPFLAGS) -o $(BIN)/binPacking $(SRC)/main.cpp

# \*\*\*\*\*\*\*\*\*\*\*\* Limpieza \*\*\*\*\*\*\*\*\*\*\*\*

clean :

-rm $(OBJ)/\* $(SRC)/\*~ $(INC)/\*~ ./\*~

mrproper : clean

-rm $(BIN)/\*

La ejecución se realiza sin parámetros ya que por simplicidad solo lee datos de un archivo por defecto llamado Problema.dat que tiene el siguiente formato:

10 //Volumen máximo de cada caja

7 //Número de elementos que se van a introducir

1 //A partir de este número representan el volumen de cada objeto

2

8

3

6

5

2