UNIVERSITAT JAUME I

EI1022, Algoritmia

Problemas 2: Algoritmos Voraces

El problema del empaquetado (*bin packing*)

- Tenemos n objetos que queremos guardar en el mínimo número de contenedores
- Supondremos que todos los contenedores son iguales y que cada uno de ellos admite cualquier número de objetos siempre que su peso total sea menor o igual que su capacidad de carga, C
- También supondremos que los pesos de los objetos son mayores que cero y que ningún objeto tiene un peso mayor que la carga del contenedor

Formalización

- Si identificaremos los objetos con sus pesos, podemos representarlos como la tupla $w = (w_0, w_1, \ldots, w_{n-1})$, donde $w_i > 0$ es el peso del objeto i
- Una solución será una tupla de n componentes $x = (x_0, x_1, \ldots, x_{n-1})$, donde x_i es el número del contenedor en el que se almacenará el objeto i
- Si asumimos que los contenedores se numeran desde cero en adelante y que en nuestra solución no hay ninguno vacío, podemos escribir el número de contenedores de la solución como

$$\mathrm{NC}((x_0, x_1, \ldots, x_{n-1})) = 1 + \max_{0 \le i < n} x_i$$

Formalización (2)

- Por lo tanto, el conjunto de soluciones es

$$X = \left\{ x \in \mathbb{N}^n \;\middle|\; \forall\, 0 \le c < \mathrm{NC}(x) : 0 < \sum_{\substack{i<n \\ x_i = c}} w_i \le C \right\}$$

- Y nuestra función objetivo será:

$$f(x) = \mathrm{NC}(x)$$

- Queremos encontrar la solución $\hat{x}$ que minimiza f:

$$\hat{x} = \arg\min_{x \in X} f(x)$$

Ejemplo

- Disponemos de infinitos contenedores de capacidad 10 y seis objetos de pesos $(1, 2, 8, 7, 8, 3)$
- La solución $(1, 0, 0, 2, 1, 2)$ representa el siguiente reparto de objetos en tres contenedores:

Contenedor	Objetos	Peso
0	$\{1, 2\}$	$2 + 8 = 10$
1	$\{0, 4\}$	$1 + 8 = 9$
2	$\{3, 5\}$	$7 + 3 = 10$

Algoritmos voraces

- El problema de decisión asociado a *bin packing* es NP-Completo, por lo tanto no se conoce ningún algoritmo voraz que obtenga la solución óptima
- Existen varios algoritmos voraces de aproximación, por ejemplo:
  - Mientras quepa
  - En el primero en que quepa
  - En el primero en que quepa ordenado

Mientras quepa

- Pasos:
  - Se abre el primer contenedor.
  - Se depositan los objetos en él hasta que uno no quepa.
  - Se cierra el contenedor, se abre otro y se repite el proceso.
- Este es un algoritmo voraz de aproximación. Se garantiza que el número de contenedores que utiliza es menor o igual que $2f(\hat{x})$: como mucho utiliza el doble de contenedores que la solución óptima

En el primero en que quepa

- Se recorre la lista de objetos:
  - Se busca en la lista de contenedores ya creados el primer contenedor en el que quepa el objeto
  - Si se encuentra un contenedor con hueco suficiente, se deposita el objeto en él, si no, se deposita en un nuevo contenedor
- Este es un algoritmo voraz de aproximación. Se garantiza que el número de contenedores que utiliza es menor o igual que $\frac{17}{10} f(\hat{x})$: como mucho utiliza un 70 % más de contenedores que la solución óptima

En el primero en que quepa ordenado

- Idéntico al anterior, pero ordenando previamente los objetos de mayor a menor peso
- Este es un algoritmo voraz de aproximación. Se garantiza que el número de contenedores que utiliza es menor o igual que $\frac{6}{9} + \frac{11}{9} f(\hat{x})$: como mucho utiliza un 22 % más de contenedores que la solución óptima

Implementación

- Crea el fichero `binpacking_mq.py`
- Implementa la función

```
def process(C: int, w: list[int]) -> list[int]:
```

que resuelve el problema mediante el algoritmo "mientras quepa"

Implementación (2)

- La función `read_data` leerá de `f` una serie de enteros, uno por línea:
  - Un entero con la capacidad de los contenedores
  - Una lista de enteros con los pesos de cada objeto
- La función `show_results` mostrará los números de los contenedores uno por línea

Pruebas

- En el aula virtual tienes el fichero `binpacking.zip` que contiene algunas pruebas

Fichero	Objetos
`small.bpk`	6
`medium.bpk`	500
`large.bpk`	1000

- También tienes los ficheros `bpack_test.py` y `bpack_sol_viewer.py`

Pruebas (2)

- Si ejecutas `bpack_test.py`, intentará probar las implementaciones con las instancias de prueba
- De momento, solo probará `binpacking_mq.py` y dará mensajes de error para `binpacking_pqq.py` y `binpacking_pqqo.py`, que implementarás luego
- Las líneas "Cómo líquido" indica cuántos contenedores se necesitarían si los objetos se pudieran fraccionar (si fueran *líquidos*), son cotas inferiores del número de contenedores

Pruebas (3)

- Con `bpack_sol_viewer.py` puedes ver los objetos agrupados por contenedor
- Para ello, debes pasarle como parámetro de la línea de órdenes el problema y por la entrada estándar la solución.
- Por ejemplo

```
python3 binpacking_mq binpacking_bab/small.bpk > small.sol
python3 bpack_sol_viewer.py binpacking_bab/small.bpk < small.sol
```

- También puedes encadenar la salida de tu programa para ahorrarte el fichero intermedio:

```
python3 binpacking_mq.py < binpacking_bab/small.bpk
    | python3 bpack_sol_viewer.py binpacking_bab/small.bpk
```


Pruebas (4)

- Los contenedores que necesitará `binpacking_mq` serán:

Problema	Contenedores
small	5
medium	217
large	439

En el primero en que quepa

- Copia el programa `binpacking_mq.py` en `binpacking_pqq.py`
- Modifica `process` en `binpacking_pqq.py` para que use el algoritmo "en el primero en que quepa"
- Los contenedores que necesitará `binpacking_pqq` serán:

Problema	Contenedores
small	4
medium	173
large	341

En el primero en que quepa ordenado

- Copia el programa `binpacking_pqq.py` en `binpacking_pqqo.py`
- Modifica `process` en `binpacking_pqqo.py` para que use el algoritmo "en el primero en que quepa ordenado"
- Los contenedores que necesitará `binpacking_pqqo` serán:

Problema	Contenedores
small	3
medium	168
large	334

Ejercicio adicional

- Resuelve el ejercicio 58 del libro de algoritmia.
- Dicho ejercicio se compone, en realidad, de tres ejercicios diferentes: 58.1, 58.2 y 58.3

Ejercicio 58.1

La señora Amanita tiene una lucrativa afición: la recolección de setas. Conoce muy bien el bosque que hay cerca de su casa y todos los años recoge las setas que crecen en él. A lo largo de los años ha apuntado, con ayuda de un GPS, las coordenadas de cada uno de los n puntos en los que hay setas.
La señora Amanita siempre camina a la misma velocidad (unos cinco kilómetros a la hora) y este año quiere recolectar todas las setas del bosque en el menor tiempo posible. Ha ideado el siguiente sistema: partiendo de su casa, va al lugar más cercano en el que hay setas y las recoge; a continuación, va al siguiente punto más cercano; y así sucesivamente.
La estrategia seguida es, evidentemente, voraz.

- a) ¿Qué coste computacional presenta su estrategia? (Exprésalo en función de n.)
- b) ¿Es una estrategia que garantiza una solución óptima? Es decir, ¿crees que así recolectará todas las setas en el menor tiempo posible?

Ejercicio 58.2

El señor Bolet compra setas a la señora Amanita y las vende por bares y restaurantes. Hay p variedades de seta que identificamos con números entre 1 y p. El precio de compra de la variedad i, para i entre 1 y p, es de c_i euros el kilo y su precio de venta es de v_i euros el kilo, donde $v_i > c_i$. La señora Amanita dispone de k_i kilos de la variedad i-ésima. La camioneta del señor Bolet puede cargar hasta K kilos de setas.

- c) ¿Qué estrategia debe seguir el señor Bolet para maximizar su beneficio con la carga que le cabe en la camioneta? Diseña una estrategia voraz. (Explica con la mayor claridad posible el algoritmo diseñado.)
- d) ¿Constituye tu estrategia un algoritmo óptimo? ¿Qué coste computacional tiene?

Ejercicio 58.3

El señor Champiñón, al igual que la señora Amanita, recoge sus propias setas. En el bosque del señor Champiñón hay n puntos de recogida. En el punto i-ésimo hay k_i kilogramos de setas de una variedad que el señor Champiñón vende a v_i euros el kilo. Cada vez que el señor Champiñón va a un punto de recogida, se hace con todas las setas del lugar y vuelve a su casa para descargar.

- e) ¿Qué debe hacer si quiere ganar la mayor cantidad posible de dinero y sólo puede permitirse hacer m viajes? Diseña una estrategia voraz.
- f) ¿Le garantiza tu estrategia que recolectará la mayor cantidad posible de setas?
- g) ¿Qué coste computacional tiene la estrategia del señor Champiñón? (Trata de ajustar el cálculo del coste temporal tanto como te sea posible y recurre a la estructura o estructuras de datos que consideres más apropiadas.)