Lab7 - Backtracking y Programación Dinámica

Algoritmos y Estructuras de Datos II – Laboratorio

Descripción

En este proyecto se pide resolver el “Problema de la mochila” (o en inglés, “Knapsack problem”).

Dado un conjunto de items, cada uno con un valor y un peso asignado, y una mochila con cierta capacidad, se requiere elegir aquellos items que maximicen el valor total sin sobrepasar la capacidad de la mochila.

Este problema en su versión original no admite solución voraz, pero puede resolverse usando backtracking y programación dinámica. En este proyecto se pide implementar ambas soluciones.

Para implementar los algoritmos se puede usar como guía pseudo-código que se encuentra en el material teórico de la materia:

Backtracking y Programación Dinámica

TAD item

La cátedra provee la implementación del TAD item, que representa un objeto candidato a ingresar en la mochila. El TAD provee las siguientes operaciones:

```
/* Construye un item a partir de un identificador, valor y peso. */
item_t item_create(string_t id, value_t value, weight_t weight);

/* Devuelve el valor de un item */
value_t item_value(item_t item);

/* Devuelve el peso de un item */
weight_t item_weight(item_t item);

/* Devuelve el identificador (nombre) de un item */
string_t item_id(item_t id);

/* Lee un arreglo de items desde de un archivo
y almacena la cantidad de items en *array_length */
item_t *item_read_from_file(FILE *file, unsigned int *array_length);

/* Destruye el item */
item_t item_destroy(item_t item);
```

Knapsack

Se deben implementar las operaciones del archivo knapsack.h.

```
value_t knapsack_backtracking(item_t *items, unsigned int array_length, weight_t max_weight);
```

Esta función resuelve el problema de la mochila usando backtracking. Los parámetros son: el arreglo de items, la longitud del arreglo y la capacidad máxima de la mochila. Devuelve el valor máximo que se puede alcanzar seleccionando items del arreglo sin sobrepasar la capacidad de la mochila.

```
value_t knapsack_dynamic(item_t *items, unsigned int array_length, weight_t max_weight);
```

Esta función resuelve el problema de la mochila usando programación dinámica. Los parámetros son: el arreglo de items, la longitud del arreglo y la capacidad máxima de la mochila. Devuelve el

valor máximo que se puede alcanzar seleccionando items del arreglo sin sobrepasar la capacidad de la mochila.

value_t knapsack_dynamic_selection(item_t *items, bool *selected, unsigned int array_length, weight_t max_weight)

Esta función resuelve el problema de la mochila usando programación dinámica. Los parámetros son: el arreglo de items, un arreglo de booleanos, la longitud de ambos arreglos, y la capacidad máxima de la mochila. Devuelve el valor máximo que se puede alcanzar seleccionando items del arreglo sin sobrepasar la capacidad de la mochila. El arreglo de booleanos se utiliza para indicar cuáles son los items del arreglo que fueron seleccionados para colocarse en la mochila.

Si *value == knapsack_dynamic_selection(items, selected, array_length, max_weight)* entonces se cumplen las siguientes post-condiciones respecto al arreglo selected:

• *selected[i] == true* si y sólo si *items[i]* fue seleccionado para entrar en la mochila, para todo $0 \leq i < array\_length$.

• $(\Sigma\ i : selected[i] : item\_value(items[i])) == value$.

• $(\Sigma\ i : selected[i] : item\_weight(items[i])) \leq max\_weight$.

Interfaz de línea de comandos

La cátedra provee la implementación de la interfaz de línea de comandos, en el archivo main.c.

El programa lee listas de items desde archivos de texto con el siguiente formato:

```
# KNAPSACK:15 VALUE:14
A:13:15
B:6:7
C:8:4
```

La primera línea contiene la capacidad máxima de la mochila (15), y de manera opcional se puede colocar el valor máximo esperado (14). Las siguientes líneas contienen los items (uno por línea) en el formato nombre:valor:peso.
Si la primera línea contiene el valor esperado (14), y se le ingresa al programa la opción -t, se verificará que el resultado de las funciones coincida con el mismo.

Las opciones del programa son las siguientes:
-b : habilitar el algoritmo de backtracking
-d : habilitar el algoritmo de programación dinámica
-s : mostrar los items seleccionados (sólo junto con -d)
-w <arg>: forzar la capacidad de la mochila a "arg" (ignorando campo KNAPSACK de la primera línea del archivo)
-t : habilitar las verificaciones (lee VALUE desde primera línea)
-f <files> : lista de archivos para los cuáles se quiere correr los algoritmos habilitados. Si no se incluye esta opción se pide la lista de items desde el teclado en el formato nombre:valor:peso.

Algunos ejemplos:

Bactracking:
$ ./knapsack -b -f input/example0.in

Dinámica:
$./knapsack -d -f input/example0.in

Dinámica mostrando selección de items:
$./knapsack -ds -f input/example0.in

Backtracking y Dinámica:
$./knapsack -bd -f input/example0.in

Backtracking y Dinámica mostrando selección y verificando el valor obtenido:
$./knapsack -bdst -f input/example.in*

Se deben corregir los memory leaks y los errores de acceso a memoria:

$> valgrind --show-reachable=yes --leak-check=full ./knapsack -b -f input/example0.in

Importante
La carpeta *input/* contiene ejemplos de archivos con listas de items. Aquellos ejemplos que tienen 100 items o más pueden demorar mucho en resolverse si se utiliza backtracking, de hecho puede acabarse la memoria de la pila del sistema causando un error de violación de segmento. Sin embargo, el algoritmo de programación dinámica debe ser capaz de resolver todos los ejemplos dados. Si en el algoritmo de programación dinámica utilizan una matriz declarada estáticamente, por ejemplo, *value_t matriz[N][M]*; sólo funcionará para los ejemplos con pocos items, para otros archivos producirá un error de violación de segmento cuando se acabe la memoria de la pila del sistema. Para lograr que funcione para los ejemplos grandes, se debe usar una matriz creada dinámicamente (usando calloc) que utilizará memoria del heap y no de la pila. En clase repasaremos cómo hacer matrices dinámicas.

Opcional:
Modificar main.c para que los items seleccionados se muestren ordenados en forma decreciente según su valor.