Escuela de Computación

**Análisis de algoritmos**

Tarea Programada #1

Estudiantes: Erick Quesada Fonseca

José Barrientos Rojas

Geovanny Astorga López

Silvia Calderón Navarro

Contenido

[Análisis de algoritmos: 2](#_Toc495871012)

[1. Algoritmo “Vuelto de monedas”: 2](#_Toc495871013)

[2. Algoritmo de la mochila: 2](#_Toc495871014)

[3. Floyd y Dijkstra: 2](#_Toc495871015)

[4. Torres de Hannoi: 2](#_Toc495871016)

[5. Quicksort y Heapsort: 2](#_Toc495871017)

[6. Multiplicación de matrices en cadena: 2](#_Toc495871018)

[Preguntas: 2](#_Toc495871019)

# Análisis de algoritmos:

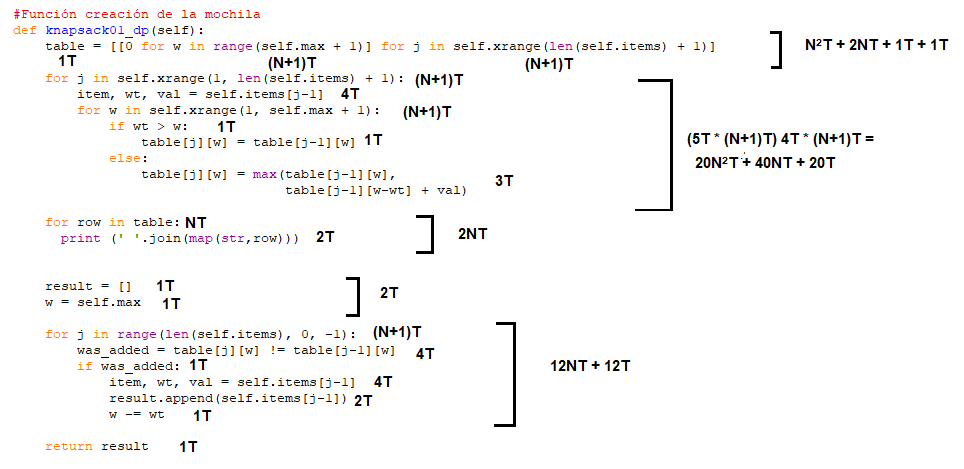
1. Algoritmo “Vuelto de monedas”:
2. Algoritmo de la mochila:

El algoritmo de la mochila es un problema que busca la mejor solución entre un conjunto finito de posibles soluciones. En este caso se tiene una mochila con una capacidad de peso máximo, y un conjunto de objetos con un peso y un valor cada uno. La idea del problema de la mochila es encontrar aquellos elementos que maximicen el valor sin exceder el peso máximo. En este caso se tratará con una mochila 0-1, lo cual significa que se tiene un solo elemento de cada tipo.

1. Análisis de los tiempos de ejecución en forma teórica:

* La decisión de incluir o descartar un elemento es del tipo NP completo, ya que no se conoce un algoritmo que sea correcto y rápido para todos los casos.
* El problema de optimización es un NP-duro, su solución es como mínimo tan difícil como la decisión del problema (la decisión de incluir o descartar un elemento), y no se conoce un algoritmo que dada una solución pueda decir si es óptima o no.
* Existe un algoritmo que produce un tiempo pseudo-polinomial utilizando programación dinámica, en este caso ese el algoritmo que se va a utilizar.
* Tiempo de ejecución: θ(NW).

Análisis:



* Si sumamos todos los resultados sería:
  + N2T + 2NT + 2T + 20N2T + 40NT + 20T + 2NT + 2T + 12NT + 12T + 1T = **21N2T + 56NT + 37T**
  + Lo cual se reduce a θ(N2).
* Si hablamos de casos intratables se reduciría a los casos donde todos los objetos que tratamos superan el peso máximo de la mochila, es decir cada peso de cada objeto es mayor que el máximo posible.

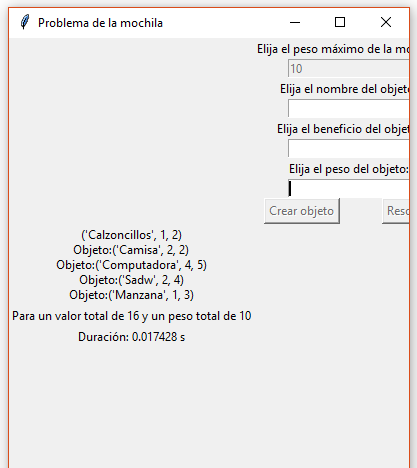
1. Análisis de los tiempos de ejecución con distintas entradas:

|  |  |  |
| --- | --- | --- |
| Nombre | Peso | Beneficio |
| Manzana | 1 | 3 |
| Sandwich | 2 | 4 |
| Computadora | 5 | 4 |
| Camisa | 2 | 2 |
| Pantalón | 3 | 2 |
| Coca Cola | 1 | 1 |
| Calzoncillos | 1 | 2 |

Peso máximo: 10

Elementos: 7

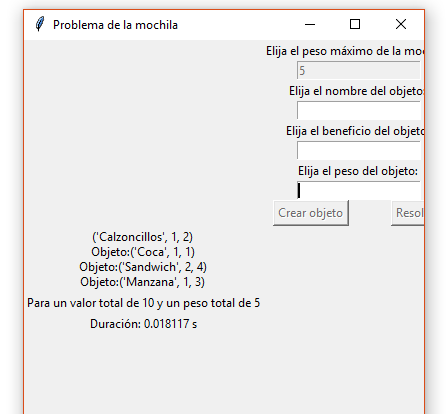
Tiempo: 0.017428s



Máximo peso: 5

Elementos: 7

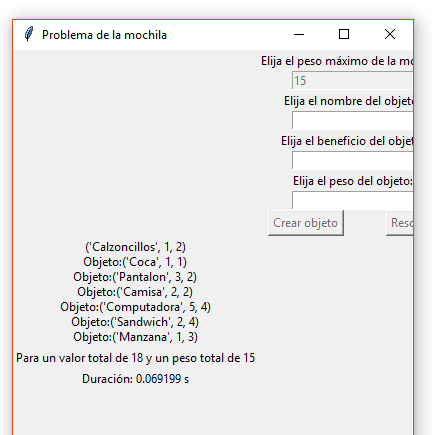
Tiempo: 0.018117s



Máximo peso: 15

Elementos: 7

Tiempo: 0.069199s

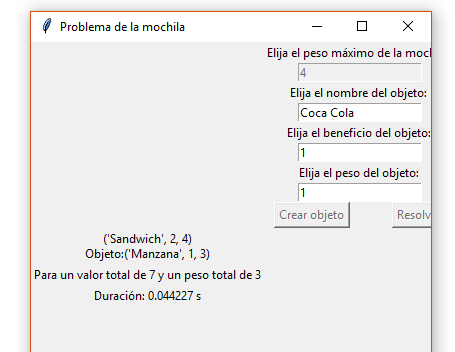


|  |  |  |
| --- | --- | --- |
| Nombre | Peso | Beneficio |
| Manzana | 1 | 3 |
| Sandwich | 2 | 4 |
| Coca Cola | 1 | 1 |

Máximo peso: 4

Elementos: 3

Tiempo: 0.044227s

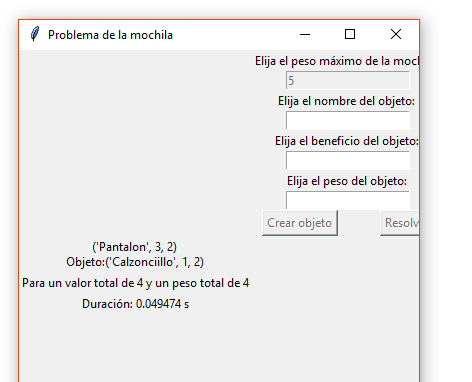


|  |  |  |
| --- | --- | --- |
| Nombre | Peso | Beneficio |
| Calzoncillos | 1 | 2 |
| Pantalón | 3 | 2 |
| Camisa | 2 | 2 |
| Computadora | 5 | 4 |

Máximo peso: 5

Elementos: 4

Tiempo: 0.049474s



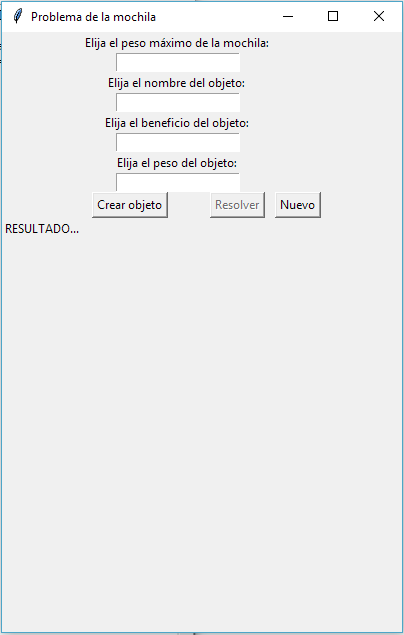
1. Descripción técnica:

**Algoritmo y estructura:**

Se utiliza un algoritmo de programación dinámica no recurrente, el cuál utiliza las siguientes estructuras:

* Un arreglo en el que se guarda la información de cada ítem o elemento, el nombre, el peso y el beneficio.
* Un arreglo bidimensional para guardar los valores máximos.
* Por último, se utiliza un arreglo para guardar los objetos escogidos después de analizar la matriz anterior. Estos elementos serán el resultado final del algoritmo.

**Interfaz:**



**1**

**2**

**Descripción:**

**7**

**6**

**5**

**3**

**4**

1. El primer paso es introducir el peso máximo de la mochila. Esto se realizará una única vez con cada mochila, al crear el primer objeto.
2. Al crear un objeto, se debe elegir un nombre que lo identifique, se recomienda que este no se repita, para poder diferenciar los objetos apropiadamente.
3. Cada objeto debe contar con un beneficio que lo caracteriza, este tiene que ser un valor numérico mayor que cero, en caso contrario el programa no permitirá que se introduzca.
4. Cada objeto debe contar con un peso que lo caracteriza, este tiene que ser un valor numérico mayor que cero, en caso contrario el programa no permitirá que se introduzca.
5. El botón crear objeto, permite añadir un objeto a la lista de elementos disponibles para la mochila.
6. Resuelve el algoritmo de la mochila, devolviendo el tiempo de ejecución en segundos, la lista de objetos utilizados, el peso total que se obtuvo de estos y el beneficio total.
7. El botón nuevo limpia la lista de objetos y crea una nueva mochila.
8. Dijkstra y Floyd:

Dado un grafo ponderado y dos vértices s y t se quiere hallar la distancia entre s y t, y el camino con dicha longitud. El algoritmo de Dijkstra obtiene todos los caminos de longitud mínima desde un vértice dado s al resto de vértices del grafo. El algoritmo de Floyd resuelve el problema para un par cualquiera de vértices de G.

1. Análisis de los tiempos de ejecución en forma teórica:

**Algoritmo de Dijkstra**

En cada iteración se añade un vértice a T, luego el número de iteraciones es n. En cada una se elige una etiqueta mínima, la primera vez entre n-1, la segunda entre n-2, ..., luego la complejidad total de estas elecciones es O(n^2). Por otra parte, cada arista da lugar a una actualización de etiqueta, que se puede hacer en tiempo constante O(1). Por tanto, la complejidad total del algoritmo es O(n^2).

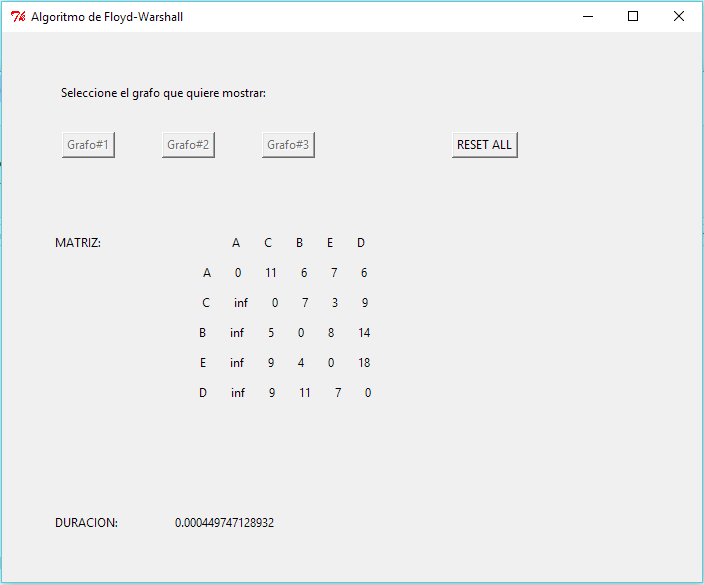
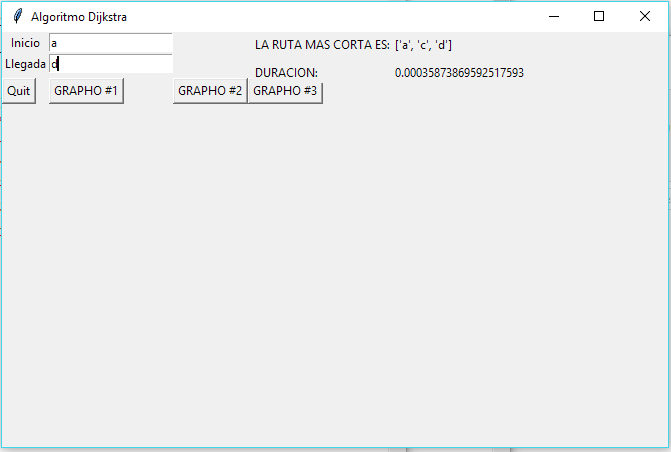
**Algoritmo de Floyd**

Se deben construir n matrices de tamaño nxn y cada elemento se halla en tiempo constante. Por tanto, la complejidad del algoritmo es O(n^3).

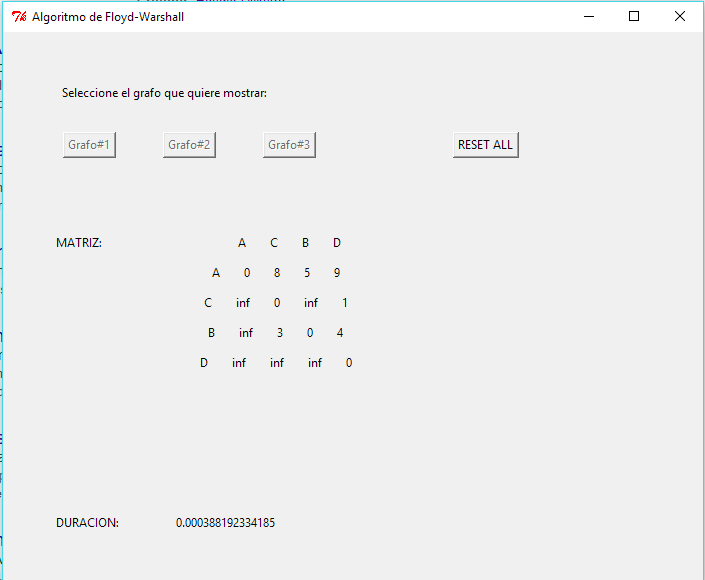
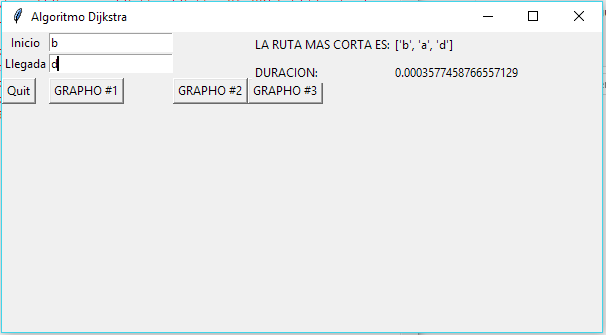
El algoritmo de Floyd es mucho más eficiente desde el punto de vista de almacenamiento dado que puede ser implementado una vez actualizado la distancia de la matriz con cada elección en k; no hay ninguna necesidad de almacenar matrices diferentes. En muchas aplicaciones específicas, es más rápido que cualquier versión de algoritmo de Dijkstra.

1. Análisis de los tiempos de ejecución con distintas entradas:

**Caso 1:**

* Floyd Grafo 1
* Tiempo: 0.000449747128932 s
* Dijkstra Grafo 1
* Tiempo: 0.00035873869592517593 s

**Caso 2:**

* Floyd Grafo 2
* Tiempo: 0.000388192334185 s
* Dijkstra Grafo 2
* Tiempo: 0.00035873869592517593 s

1. Descripción técnica:
   1. Dijkstra:

* **Algoritmo y estructura:**
* **Interfaz:**
* **Descripción:**
  1. Floyd:
* **Algoritmo y estructura:**
* **Interfaz:**
* **Descripción:**

1. Torres de Hannoi:
2. Análisis de los tiempos de ejecución en forma teórica:
3. Análisis de los tiempos de ejecución con distintas entradas:
4. Descripción técnica:

* **Algoritmo y estructura:**
* **Interfaz:**
* **Descripción:**

1. Quicksort y Heapsort:
2. Multiplicación de matrices en cadena:

# Preguntas:

1) ¿Funciona el algoritmo de Floyd en un grafo que tenga algunas aristas cuyas longitudes sean negativas pero que no contengan ningún ciclo negativo? Cuál es la relación con los problemas clase P, NP o NP completos. Justifique.

Sí funciona, sin embargo, si permitimos que el grafo tenga aristas con valores negativos, el algoritmo de Floyd pierde parte de su significado al buscar le camino más corto. No se conoce un algoritmo eficiente que halle los caminos simples más cortos en este tipo de grafos, este problema se clasifica como NP completo, a diferencia del algoritmo solamente con aristas positivas, este se resuelve en un tiempo polinomial.

2) Bajo qué circunstancias el algoritmo de Floyd es mejor al de Dijkstra. Justifique.

El algoritmo de Floyd tiene una complejidad de O(V^3), y con esto encuentra las mejores rutas para todos los pares, mientras que Dijkstra tiene una complejidad de O(E + Vlog(V)). Al correr el algoritmo de Dijkstra sobre todos los vértices, para poder encontrar todas las mejores rutas, la complejidad se convierte en O(EV + V^2log(V)). E representa la cantidad de aristas, por lo tanto, si E es del orden O(V^2) Dijkstra va a ser tan rápido como Floyd, sin embargo, si E es del orden O(V), el tiempo de ejecución del algoritmo de Dijkstra es menor.

Por lo tanto, para un grafo altamente denso Floyd va a tener un mejor rendimiento, además que se puede utilizar con aristas de valor negativo y Dijkstra no.

3) ¿Cuáles serían los tiempos de respuesta estimados para el algoritmo de las Torres de Hanoi con N = 50 discos y corriendo en una máquina que procesa diez mil millones de instrucciones por segundo? Justifique.

El algoritmo de Torres de Hanoi tiene una complejidad de O(2^N), siendo N la cantidad de discos.

Por lo tanto, tomando N = 50, la complejidad será de 1125899906842624.

La fórmula para obtener el tiempo de ejecución es la siguiente:

complejidad /cantidad de instrucciones por segundo = tiempo de ejecución en segundos

1125899906842624 / 10000000000 = 112589.990 s

112589.990 s = 1876.50 m = 31.275 h

El tiempo de respuesta del algoritmo de Hanoi es de 31.275 h, cuando se usan 50 discos.

Referencias:

[www.lab.dit.upm.es/~lprg/material/apuntes/o/index.html](http://www.lab.dit.upm.es/~lprg/material/apuntes/o/index.html)

<https://en.wikipedia.org/wiki/Knapsack_problem>

<https://es.wikipedia.org/wiki/Problema_de_la_mochila>

<https://es.wikibooks.org/wiki/Algoritmia/Programaci%C3%B3n_din%C3%A1mica>

<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&ved=0ahUKEwjct7H9n_zWAhVOySYKHagOAhIQFghVMAc&url=https%3A%2F%2Fwww.dc.uba.ar%2Fmaterias%2Faed3%2F2013%2F1c%2Fteorica%2Fdinamica.pdf&usg=AOvVaw22z65C9QjnwVeE4ro9Uy6f>