Universidad de Buenos Aires
Facultad De Ingeniería
Año 2019 – 2.do Cuatrimestre

(75.29 / 95.06) Teoría de Algoritmos

TRABAJO PRÁCTICO N.º 3
Grupo: "Los Fieles de Cardozo"
23 de Octubre de 2019

Integrante	Padrón	Correo electrónico
Salvador, Yago	# 99 725	yagosalvador@gmail.com
Nenadovit, Emmanuel Angel	# 91 734	emmanuelnenadovit@gmail.com
Untrojb, Kevin	# 97 866	oliverk12@hotmail.com
Bocchio, Guido	# 101 819	guido@bocch.io

1. Resumen

En el presente informe se expone el análisis y resultados de cada consigna. La primera parte presenta un problema de geometría computacional e informática gráfica en el cual se utilizó la estrategia de dividir y conquistar para la resolución del problema. se expone la solución, el análisis del algoritmo utilizado y se programa la solución. La segunda parte consiste en optimizar y minimizar la cantidad de rotaciones de una caja desde un punto de inicio hasta un punto final. Se analizó varios casos del problema y finalmente se utilizó un algoritmo de programación dinámica para poder llegar a la solución óptima. La ultima parte es optativa, consiste en analizar y explicar el *paper "Fast detection of polyhedral intersection"* publicado por David P. Doeskin y David C. Kirkpatrick. Además se proporciona el pseudocódigo del algoritmo y proporcionar dos ejemplos completos de funcionamiento.

2. Rectángulo contenedor

Se desea hallar el rectángulo contenedor de un conjunto de puntos que forman un polígono convexo. Un enfoque directo del problema consiste en hallar cuatro puntos construidos a partir de aquellos puntos cuyas coordenadas x e y sean máximas y mínimas. Para esto se opta por recorrer cuatro veces el arreglo de puntos dado. Luego, se construye el rectángulo a partir de estos puntos. En el Algoritmo 1 se muestra el pseudocódigo de esta solución. Encontrar el mínimo y máximo son operaciones $\mathcal{O}(n)$. De esto se tiene que algoritmo es $\mathcal{O}(n)$.

Además se propone una solución utilizando la estrategia de división y conquista. Esta consiste en contar con los puntos ordenados según su coordenada x. Luego al construir los subproblemas se obtienen dos rectángulos contenedores. Como un rectángulo tiene puntos cuyas coordenadas x son estrictamente menores a las del otro rectángulo se tienen $x_{mín}$ y $x_{máx}$. Para obtener $y_{mín}$ e $y_{máx}$ simplemente se comparan los valores máximos y mínimos para la coordenada y en cada rectángulo. En el Algoritmo 2 se muestra el pseudocódigo de esta solución. Se muestra en la Ecuación (1) la recurrencia que modela esta solución. De esta recurrencia, utilizando el teorema maestro, se aprecia que la solución es de orden $\mathcal{O}(n)$.

La primera solución utiliza memoria constante para obtener el rectángulo contenedor y únicamente requiere tener acceso a los puntos del polígono. Por esto se obtiene que su complejidad espacial es constante: $\mathcal{O}(1)$. Sin embargo, la segunda solución realiza sucesivas llamadas recursivas. Puesto que divide el problema en partes iguales se concluye que la profundidad del árbol de llamadas es de orden $\mathcal{O}(\log n)$ y la complejidad espacial resulta del mismo orden.

Puesto que ninguna solución utiliza el hecho de que los puntos formen un polígono convexo el problema no se ve alterado por esto.

$$T(n) = \begin{cases} 1, & \text{si } n = 1. \\ 2\mathrm{T}(\mathrm{n}/2) + 1, & \text{en otro caso.} \end{cases} \tag{1}$$

Algoritmo 1 Solución inicial para obtener el rectángulo contenedor

Sea P un conjunto de puntos
$x_{mín} \leftarrow$ la menor coordenada x en para todo punto en P
$x_{máx} \leftarrow$ la mayor coordenada x en para todo punto en P
$y_{mín} \leftarrow$ la menor coordenada y en para todo punto en P
$y_{máx} \leftarrow$ la mayor coordenada y en para todo punto en P
rectángulo $\leftarrow [(x_{mín}, y_{mín}), (x_{mín}, y_{máx}), (x_{máx}, y_{máx}), (x_{máx}, y_{mín})]$
devolver rectángulo

Algoritmo 2 Solución para obtener el rectángulo contenedor utilizando la estrategia de división y conquista

Sea P un conjunto de puntos ordenados según su coordenada x
si $|P| = 1$ **entonces**
    Sea p el único punto de P
    devolver $[p, p, p, p]$
izquierda $\leftarrow$ El rectángulo contenedor de la primera mitad de puntos de P
derecha $\leftarrow$ El rectángulo contenedor de la segunda mitad de puntos de P
$x_{mín} \leftarrow x_{mín}$ de izquierda
$x_{máx} \leftarrow x_{máx}$ de derecha
$y_{mín} \leftarrow$ mín$\{y_{mín}$ de izquierda, $y_{mín}$ de derecha$\}$
$y_{máx} \leftarrow$ máx$\{y_{máx}$ de izquierda, $y_{máx}$ de derecha$\}$
rectángulo $\leftarrow [(x_{mín}, y_{mín}), (x_{mín}, y_{máx}), (x_{máx}, y_{máx}), (x_{máx}, y_{mín})]$
devolver rectángulo

3. Acomodando cajas

El enunciado pide construir un algoritmo que permite trasladar una caja de almacenamiento de dimensiones estándar de una ubicación inicial a una final determinada teniendo en cuenta que la caja es un prisma rectangular que mide 2 de alto x 1 de ancho x 1 de largo. La caja solo la puedan hacer girar en el piso en cualquiera de sus ejes. Se considera el depósito como una grilla de n filas por m columnas, tanto n como m valores enteros. Cada celda es de 1x1 metro. Algunas de las celdas podrían estar ocupadas por otras cajas que no podemos trasladar. La caja se encuentra inicialmente parada. Para resolver este problema, se puede modelar las celdas como un grafo y la caja se puede modelar con dos estados, parada y acostada.

Para cada estado posible la caja puede girar en cuatro direcciones ilustradas en la Figura 1

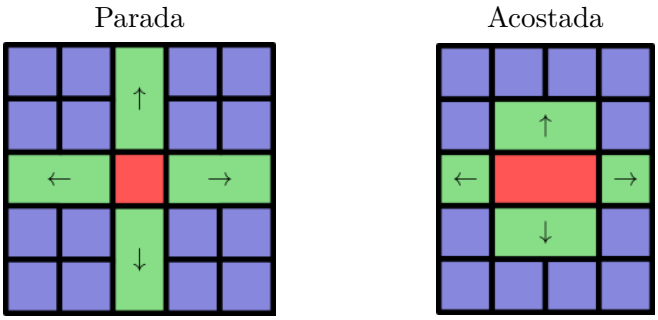

Figura 1: Posiciones en las cuales se puede girar una caja. La posición actual de la caja se indica en rojo. Las posiciones a las que esta puede moverse se indican en verde.

Se puede representar cada posición como un vértice en un grafo. En este grafo un giro es representado por una arista. Como cada giro es un proceso reversible el grafo resultante es no-dirigido. Dado un depósito de n celdas libres las posiciones en las que la caja se encuentra parada son exactamente n. Esto puede verse en la Figura 2. Asimismo, es posible listar todas las posiciones en las que la caja puede encontrarse acostada. Es posible ver que no hay más de n posiciones en las que la caja puede encontrarse acostada horizontalmente

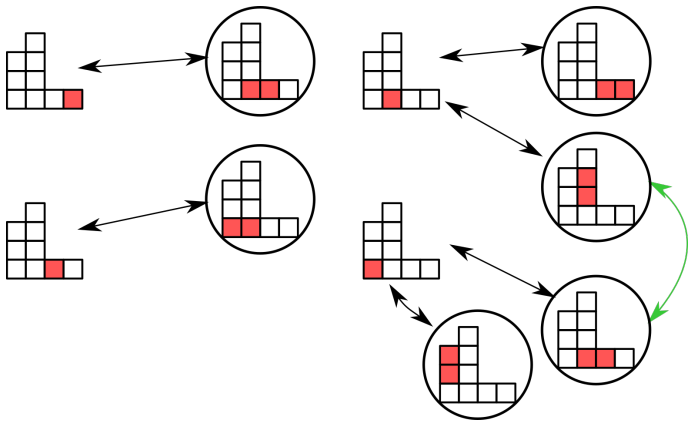

Figura 3: Primeros pasos en la creación de un grafo para el depósito dado. La arista marcada en verde se construirá posteriormente pero existe en el grafo final.

o verticalmente. Por lo tanto, hay como mucho, $2n$ posiciones en las que la caja puede encontrarse acostada. Finalmente las posiciones en las que la caja puede encontrarse son $\mathcal{O}(3n) = \mathcal{O}(n)$.

Para construir el grafo por cada posición se lista las cuatro posiciones a las que la caja puede girar. En caso de encontrar un obstáculo en alguna celda de estas posiciones, se ignora dicha posición. En caso de no existir una arista entre la posición actual y alguna de las posiciones listadas, esta es creada. Esto puede realizarse en $\mathcal{O}(n)$ puesto que obtener las posiciones a las que una caja puede girar son constantes y la creación de una arista se realiza en $\mathcal{O}(1)$. Se puede ver en la Figura 3 un ejemplo de los primeros pasos en la creación de dicho grafo.

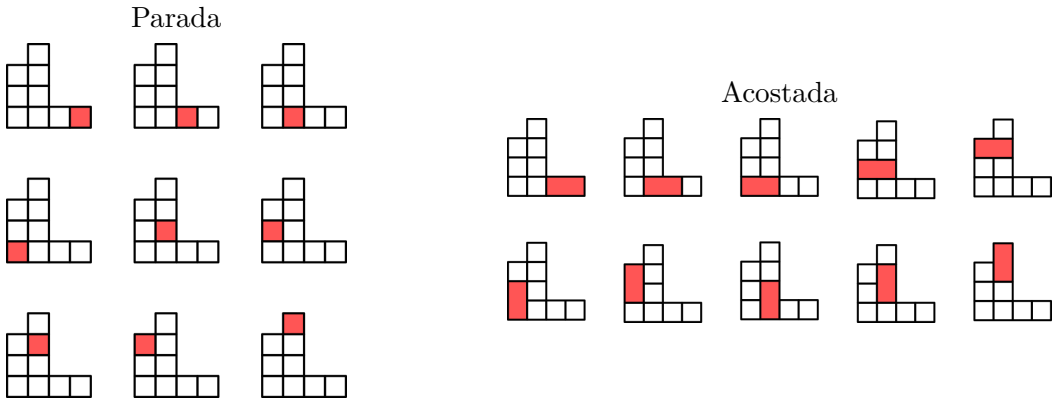

Figura 2: Ejemplo de un depósito y todas las posiciones en las que la caja se puede encontrar acostada en él.

Se desea resolver el problema utilizando programación dinámica para ello se propone utiliza el algoritmo de Bellman-Ford. Este algoritmo calcula en cada iteración si existe un camino más corto desde la posición inicial a la final utilizando una posición intermedia conocida. En caso de existir dicho camino actualiza el camino mínimo y su costo desde la posición inicial a la final.

Algoritmo 3 Coste mínimo para mover la caja

Construir el grafo
Sea $peso(v, w) = 1$ *de existir arista entre* v *y* w *o* ∞ *en otro caso*
Sea V las posiciones
$n \leftarrow |V| - 1$
devolver Distancia(posición inicial, posición final, n)

función DISTANCIA(v, w, n)
  si $v = w$ **entonces**
    devolver 0
  para cada posición u a la que se puede acceder desde v **hacer**
    si $peso(u, w) < \infty$ **entonces**
      $D_u \leftarrow$ Distancia$(v, u, n-1) + peso(u, w)$
    si no
      $D_u \leftarrow \infty$
  devolver mín{Distancia$(u, w, n-1)$, $\min_u\{D_u\}$}

Referencias

[1] David P. Doeskin & David C. Kirkpatrick *"Fast detection of polyhedral intersection"*. `https://www.cs.jhu.edu/ misha/Spring16/Dobkin83.pdf`

[2] Jon Kleinberg, Éva Tardos *Algorithm Design* . Addison Wesley,2006.