

PROYECTO

FUNDAMENTOS DE ANÁLISIS & DISEÑO DE ALGORITMOS

JESÚS ALEXANDER ARANDA

Santiago Hernández Arias 1631281

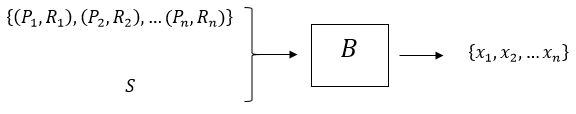
Cristian Camilo Vallecilla Cuellar 1628790

Esneider Arbey Manzano Arango 1628373

Christian Camilo Taborda Campiño 1632081

Ingeniería de Sistemas

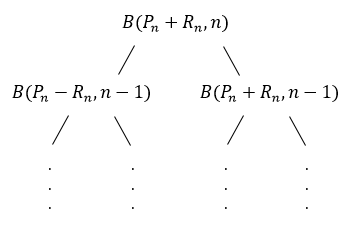
Programación Dinámica:

Analizando el problema tenemos lo siguiente:

Teniendo a y como las dimensiones del cuadro donde se ubican los blancos, las entradas son un conjunto de blancos , donde cada blanco es una dupla conformada por su posición en el eje horizontal y su radio , y un número que me indica la posición donde termina el alcance del último blanco para servir como criterio de solapamiento. Es necesario asumir que y son números enteros para cada . Además se debe cumplir que , , , y para cada . La salida es un conjunto de valores binarios tal que , donde . El objetivo de la función es seleccionar aquellos blancos que suman la mayor área sin solaparse entre ellos, es decir:

* Estructura de la solución óptima:

Dependiendo del blanco actual que estemos evaluando y la posición donde se encuentre el criterio de solapamiento, habrá un máximo de dos posibles subproblemas. Si ilustramos de forma recursiva la estructura de la solución tenemos lo siguiente:



El anterior árbol puede llegar a tener dos aristas por nodo como máximo, y el problema se sigue dividiendo hasta llegar a los casos triviales, donde se elegirá el camino con mayor beneficio. Los casos triviales serían aquellos donde no nos quedan blancos por evaluar o donde el criterio de solapamiento es .

Sea un problema de bombardeo considerando los primeros blancos y con un criterio de solapamiento . Como el problema principal toma desde los primeros blancos hasta los primeros blancos y con un criterio de solapamiento desde hasta , la cantidad de subproblemas es: . Sea la solución del problema , Tenemos lo siguiente:

Si :

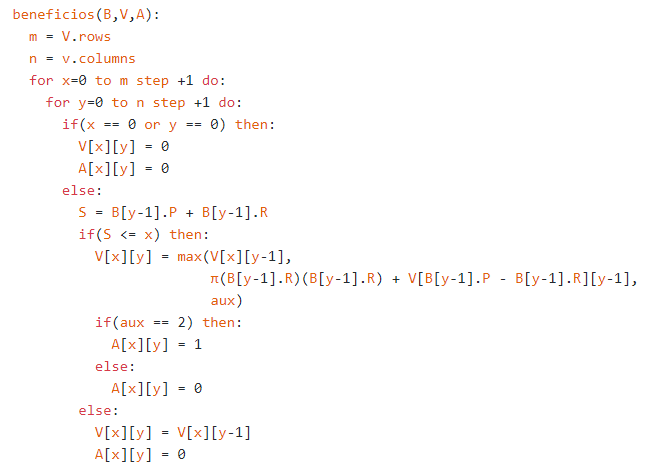
Si :

* Valor de la solución de manera recursiva:

Sea la función que calcula el valor de la solución , entonces:

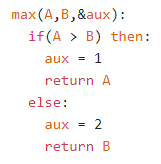
* Si o :
* Si :
* Si :
* Algoritmo para llenar la matriz de valores:

Un aspecto importante acerca del algoritmo es que recibe el arreglo de blancos ordenado, tomando como criterio de ordenamiento la posición sobre el eje horizontal de cada blanco. Para ello se utilizó el algoritmo de mergesort, ya que su complejidad es , el cual no sería un factor dominante al analizar la complejidad del algoritmo principal que se visualizará a continuación. El siguiente algoritmo se encargará de llenar la matriz de valores y una matriz auxiliar para la construcción de la solución a la vez:



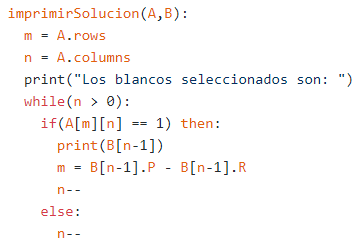
Donde es un arreglo con los blancos a evaluar, es la matriz que me va a almacenar todos los valores de las soluciones dadas a los subproblemas presentados y es la matriz auxiliar que va a contener valores binarios para permitir la construcción de la solución. Las dimensiones de ambas matrices son , donde y la cantidad de blancos a evaluar.

La función utilizada no sólo retorna el máximo, también recibe una variable que indica cuál de los valores evaluados fue elegido como el máximo, su algoritmo es:



* Algoritmo para determinar la solución óptima:

El siguiente algoritmo determina cuál fue el subconjunto de blancos que permitieron maximizar el área de alcance de los misiles imprimiendo en pantalla uno por uno:



Donde es la matriz auxiliar que contiene los valores binarios usados para determinar la solución y es el arreglo con los blancos a evaluar.

* Modificando el problema:

Si modificamos el problema de tal forma que no se busque maximizar el área de impacto de los misiles sino maximizar la cantidad de misiles que se disparan, lo único que cambiaría en el enfoque de la programación dinámica es el valor calculado para llenar la matriz de valores, pues en vez de sumar el área que proporciona cada blanco evaluado, estaríamos sumando un para indicar que se dispara un misil. Al final, el algoritmo haría exactamente lo mismo, seleccionar el subconjunto de blancos que permitan disparar la mayor cantidad de misiles sin solaparse entre ellos

* Eficiencia en tiempo:

La complejidad del algoritmo se deriva de la complejidad de las dos partes que lo conforman, llenar la matriz de valores y construir la solución.

Si analizamos la complejidad de llenar la matriz de valores nos damos cuenta que la cantidad de operaciones realizadas depende directamente de la cantidad de iteraciones que realicen los bucles , pues dentro y fuera de los bucles tenemos una cantidad constante de operaciones. El tiempo de ejecución está dado por:

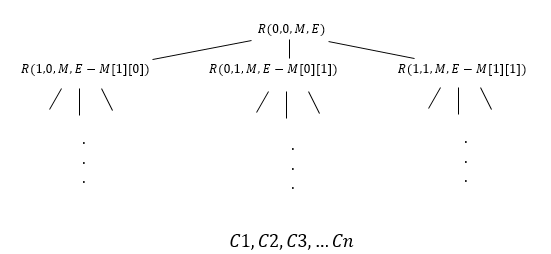
Donde corresponde a la cantidad de filas de la matriz y corresponde a la cantidad de columnas de la matriz.

Si analizamos la complejidad de determinar la solución óptima nos damos cuenta que la cantidad de operaciones realizadas depende directamente de la cantidad de iteraciones que realice el bucles , pues dentro y fuera del bucle tenemos una cantidad constante de operaciones. Como dentro del siempre hay un decremento en la variable y la condición de parada está cuando , la cantidad de iteraciones realizadas por el bucle será . El tiempo de ejecución está dado por:

Donde corresponde a la cantidad de blancos que se tuvieron en cuenta para la evaluación de los posibles subconjuntos.

Donde son la fila y columna correspondientes a la ubicación actual del robot, es el tablero con los costos por donde se mueve el robot y es la energía actual del robot. Al utilizar el método con estos tres parámetros obtenemos los movimientos que el robot realizaría para llegar a su destino con la mayor energía posible.

1. Estructura de la solución:

Dependiendo de la posición donde se encuentre el robot y de la energía disponible, habrá un máximo de tres posibles movimientos. Si ilustramos de forma recursiva la estructura de la solución tenemos lo siguiente:

El anterior árbol puede llegar a tener tres aristas por nodo como máximo, y el problema se sigue dividiendo hasta llegar a los casos triviales, donde se elegirá el camino con el menor costo. Los casos triviales serían aquellos donde el robot llegó a su destino o donde el robot no pudo avanzar más por falta de energía.

La solución general sería de la forma:

Si :

Si :

Si :

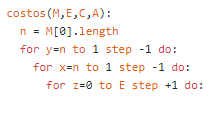
1. Función de costo de la solución:

Sea la función que me calcula los costos de las soluciones:

* Si , entonces:
* Si y , entonces:
* Si y , entonces:
* Si y , entonces:
* Si y , entonces:
* Si y , entonces:
* Si y , entonces sería el valor mínimo entre:
* Si , y , entonces sería el valor mínimo entre:
* Si , y , entonces sería el valor mínimo entre:
* Si , y , entonces sería el valor mínimo entre:
* Si , y , entonces sería:
* Si , y , entonces sería:
* Si , y , entonces sería:

1. Algoritmo para el costo de la solución:

El siguiente algoritmo se encargará de llenar la matriz de costos y una matriz auxiliar a la vez:



Espacio para las condiciones de llenado

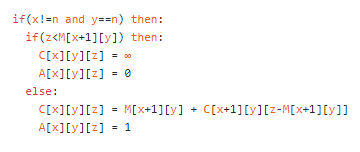
Donde es la matriz con los costos de energía que debe consumir el robot, E la energía inicial del robot, C y A corresponden a las matrices de costos y la auxiliar a llenar, ambas tienen dimensiones , siendo la dimensión de la matriz .

Los siguientes son los bloques de pseudocódigo con las condiciones de llenado de las matrices y :

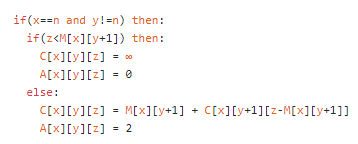
* Si el robot se encuentra en la posición :



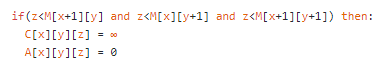
* Si el robot se encuentra en la última columna:

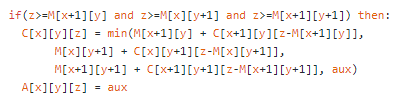


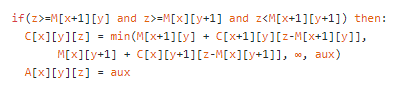
* Si el robot se encuentra en la última fila:

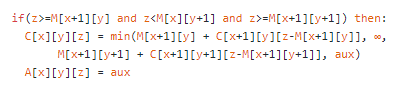


Si el robot está en cualquier otra posición tenemos las siguientes condiciones:

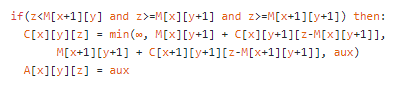
* Si no puede tomar ninguno de los tres caminos:
* Si puede tomar los tres caminos:



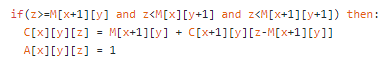
* Si sólo puede ir por abajo y por la derecha:
* Si sólo puede ir por abajo y diagonal:

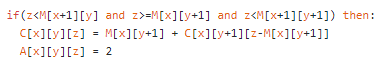


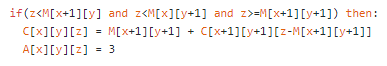
* Si sólo puede ir por la derecha y diagonal:



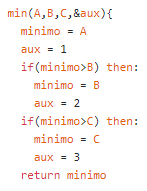
* Si sólo puede ir por abajo:



* Si sólo puede ir por la derecha:
* Si sólo puede ir diagonal:



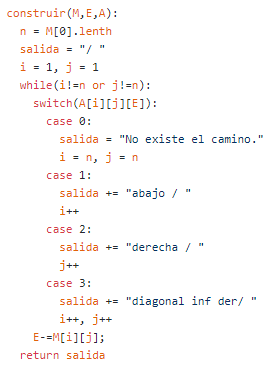
La función utilizada no sólo retorna el mínimo, también recibe una variable que indica cuál de los valores evaluados fue elegido como el mínimo, su algoritmo es:



La complejidad del algoritmo final depende directamente de la cantidad de iteraciones realizadas por los bucles , dentro de los bucles tenemos una cantidad constante de operaciones, por lo tanto:

Con lo anterior podemos observar que la complejidad del algoritmo es igual a la cantidad de subproblemas que surgen del planteamiento inicial.

1. Algoritmo para determinar la solución:



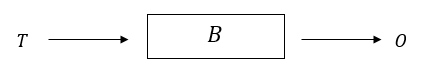
Donde es la matriz con los costos de energía que debe consumir el robot, es la matriz auxiliar que se llena simultáneamente con la de costos y es la energía inicial del robot.

La complejidad del algoritmo anterior depende directamente de la cantidad de iteraciones que realice el bucle , la cual depende de cómo aumenten las variables y en cada iteración. Dentro del bucle hay una cantidad constante de operaciones, en el peor de los casos, cada iteración debe aumentar sólo a una de las dos variables, por lo tanto:

El bucle realizaría iteraciones por cada variable en juego, en este caso son dos variables, es decir, el doble de iteraciones: .

Programación Voraz:

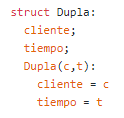
Analizando el problema tenemos lo siguiente:



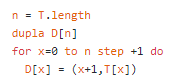
Donde es un arreglo con los tiempos de atención de los clientes en la fila y será un arreglo con números desde hasta en el orden en que deberían ser atendidos en el banco, siendo la cantidad de clientes en la fila. Se busca minimizar el tiempo promedio que gasta un cliente en el banco, el cual está dado por:

Para garantizar que la relación anterior siempre sea mínima debemos asegurarnos que la suma del numerador sea mínima. Siguiendo el esquema anterior podemos ver que el único sumando que aparece veces sería , luego aparece veces y así sucesivamente hasta llegar a , el cual sólo aparece una vez. La idea es ordenar los elementos del arreglo de menor a mayor, de esta forma nos aseguramos que los elementos más grandes aparezcan menos veces que los más pequeños y seguidamente nos aseguramos de que la suma será mínima y la solución óptima. Esta es una estrategia voraz por el hecho de tomar decisiones locales factibles y que no dependen de la solución de subproblemas.

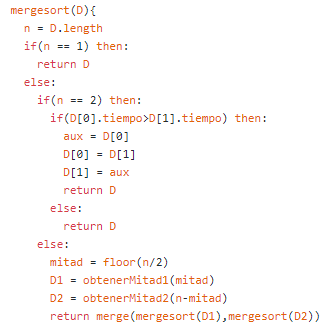
Lo primero que hace el algoritmo es crear un arreglo de duplas de igual tamaño al arreglo de entrada, para ello se usa la estructura dupla:

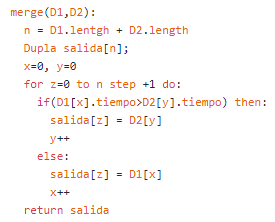


La inicialización del arreglo de duplas queda de la siguiente forma:



Luego se ordena el arreglo de duplas usando y tomando como criterio de comparación el primer atributo de cada dupla:



Las funciones y se encargan de dividir el arreglo en dos. La función se encarga de combinar dos arreglos ordenados en uno solo que también conserve el orden:

Finalmente nos queda imprimir el segundo atributo de cada dupla una vez que el arreglo de duplas está ordenado:



Para analizar la complejidad del algoritmo hace falta analizar el costo computacional de cada paso a realizar:

* El costo de crear el arreglo de duplas, el de la función y el de imprimir la solución es el mismo, pues los tres dependen directamente de la cantidad de iteraciones del bucle , ya que dentro y fuera del bucle tenemos operaciones constantes:
* El costo de la función depende directamente de la cantidad de llamados recursivos que haga y el costo equivalente a invocar a la función :

Aplicando método maestro vemos que:

Por lo tanto la complejidad del es:

Ahora determinamos el costo computacional total del algoritmo:

El factor dominante en la ecuación anterior nos indica que en general la complejidad del algoritmo es: