Universidad de La Hábana

Facultad de Matemáticas y Ciencias de la Computación

Proyecto Final de DAA

Presentado por:

Alex Sánchez Saez C412
Carlos Manuel González C411
Jorge Alberto Aspiolea González C412

Septiembre 2024

Índice

1. Grid

1.1. Definición del problema

Un día iba Alex por su facultad cuando ve un cuadrado formado por $n \times n$ cuadraditos de color blanco. A su lado, un mensaje ponía lo siguiente: "Las siguientes tuplas de la forma (x_1, y_1, x_2, y_2) son coordenadas para pintar de negro algunos rectángulos. (coordenadas de la esquina inferior derecha y superior izquierda)" Luego se veían k tuplas de cuatro enteros. Finalmente decía: "Luego de tener el cuadrado coloreado de negro en las secciones pertinentes, su tarea es invertir el cuadrado a su estado original. En una operación puede seleccionar un rectángulo y pintar todas sus casillas de blanco. El costo de pintar de blanco un rectángulo de $h \times w$ es el mínimo entre h y w. Encuentre el costo mínimo para pintar de blanco todo el cuadrado."

En unos 10 minutos Alex fue capaz de resolver el problema. Desgraciadamente esto no es una película y el problema de Alex no era un problema del milenio que lo volviera millonario. Pero, ¿sería usted capaz de resolverlo también?

1.1.1. Entrada

La entrada del problema sería un entero n y m tuplas, tal que n sería la longitud de los lados del cuadrado; y las tuplas serían de la forma (x_1, y_1, x_2, y_2), con $0 \leq x1, x2, y1, y2 \leq n-1$, donde (x_1, y_1) sería las coordenadas de la esquina inferior derecha, y (x_2, y_2) las de las esquina superior izquierda.

1.1.2. Salida

Un número entero indicando el costo mínimo de revertir el color de todos los rectángulos a blanco.

1.1.3. Técnicas de solución empleadas

- Backtrack
- Greedy

1.2. Backtrack

Del problema tenemos un cuadrado en donde fueron pintados algunos rectángulos los cuales pueden solaparse entre ellos tanto parcial como completamente. Nuestra primera solución para atacar el problema fue crear la solución de fuerza bruta para poder probar nuestras posteriores soluciones.

Para esto implementamos un backtrack clásico, en donde iteramos por todas las tuplas de los rectángulos, y calculamos el costo de borrarlo o no borrarlo. En cada iteración guarda la mejor solución hasta el momento. El caso de parada sería cuando no haya más cuadrados negros, en ese caso se regresa por la rama del backtrack, hasta que se hayan probado todas las combinaciones de rectángulos.

1.2.1. Análisis de correctitud

Claramente este algoritmo da la solución correcta pues prueba todas las combinaciones posibles de borrar los rectángulos y se queda con la mejor, lo que en un tiempo no polinomial.

1.2.2. Analisis de complejidad

La complejidad de este algoritmo sería exponencial respecto a la cantidad de rectángulos, siendo de m!.

1.2.3. Algoritmo

```
optimal_solution=[]
min_cost = float('inf')
def backtrack(matriz, rectangulos, solution=[],
    solution_cost=0):
    global optimal_solution
    global min_cost
    # Caso base: si la matriz esta completamente vacia,
    # devolver el costo de la solucion
    if matriz_esta_vacia(matriz):
        if solution_cost < min_cost:
            optimal_solution= solution.copy()
            min_cost = solution_cost
        return solution_cost

    # Inicializamos la respuesta como infinito
    response = float('inf')

    for i, rect in enumerate(rectangulos):
        if i not in solution:
            # Agregamos el indice del rectangulo actual
            # a la solucion
            solution.append(i)

            # Eliminamos el rectangulo actual de la matriz
            eliminar_rectangulo(matriz, rect[0], rect[1],
                rect[2], rect[3])

            # Llamamos recursivamente con la solucion
            # actualizada
            a = backtrack(matriz, rectangulos, solution,
                solution_cost + peso_rectangulo(rect[0],
                rect[1], rect[2], rect[3]))

            # Restauramos el estado de la matriz
            restaurar_rectangulo(matriz, rect[0], rect[1],
                rect[2], rect[3])
```

```
35
36              # Eliminamos el indice del rectangulo actual
37              # de la solucion
38              solution.pop()
39
40              # Actualizamos la respuesta con el minimo entre
41              # la solucion actual y la respuesta
42              response = min(response, a)
43
44      return response
```

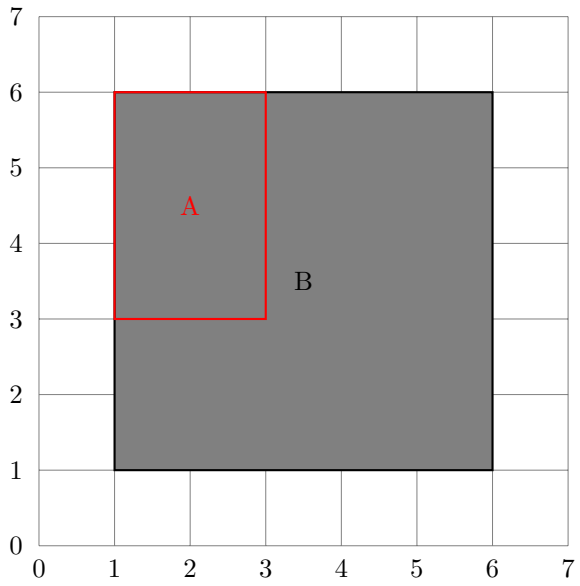
1.3. Primeras ideas

Podemos observar que si un rectángulo es cubierto por otros rectágunos, este no es necesario pintarlo ya que pintando a los que lo cubren se pintaría este también, evitándonos así un costo innecesario.

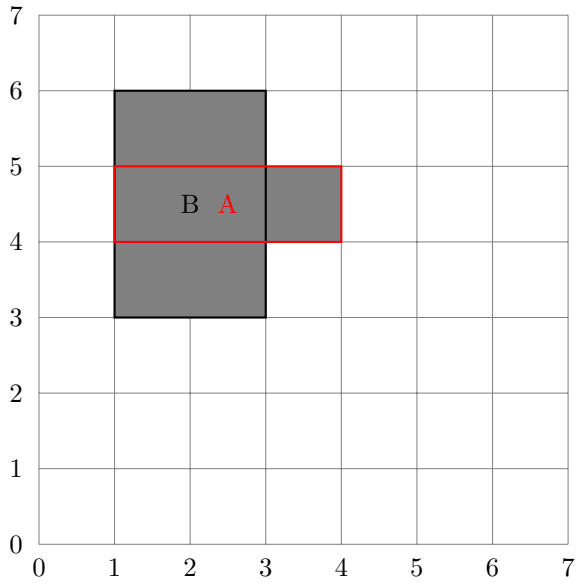
Aquí tendríamos otro problema y es en que orden se pintan los cuadrados para asegurarnos de que siempre se pintan solo los necesarios?

Para esto definamos algúnos puntos importantes.

- Si un rectángulo A esta completamente contenido dentro de otro B, no es necesario pintar A y sólo pintaríamos B. Esto es así porque aunque pintemos A, igual necesitaríamos pintar B, sin embargo si pintamos solo B este cubriría A, por lo que solo tendríamos el costo de pintar uno solo.



- Si un rectángulo A tiene al menos una casilla la cual no es cubierta por ningún otro rectangulo, entonces hay que pintar A obligatioriamente. Esto es obvio ya que pintando los demas rectángulos solo pintaríamos una parte de A quedando algunas casillas en negro todavía que solo serán pintadas de blanco si y solo si pintamos A directamente.



Sabiendo esto se nos ocurrio ordenar los rectángulos por área con un costo de $O(\log m)$ y elegir en cada momento el más grande, siempre que no haya sido borrado.

Este método nos acercó a la solución con un costo computacional polinomial, pues solo sería el costo de ordenar los rectángulos($O(\log m)$) más el costo de borrarlos todos ($O(m) \cdot O(k) = O(max(m, k))$, donde k es la cantidad de cuadrados que conforman el rectángulo. Por tanto este algoritmo tendría un tiempo lineal.

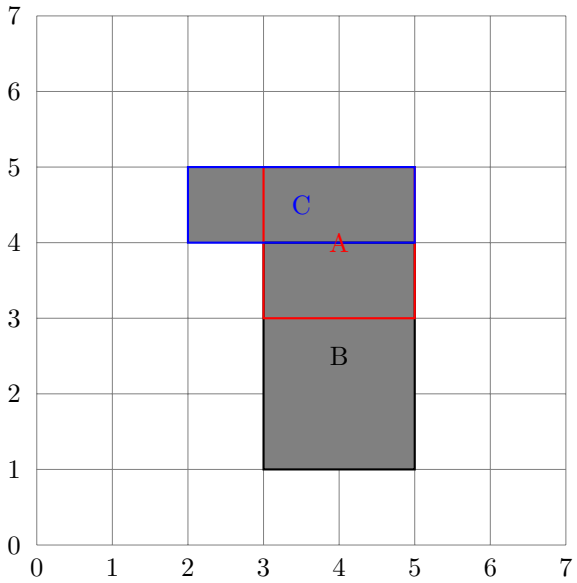
Esta solución tiene un caso que nos falla, el cual abordaremos más adelante.

1.3.1. Problemas con este enfoque

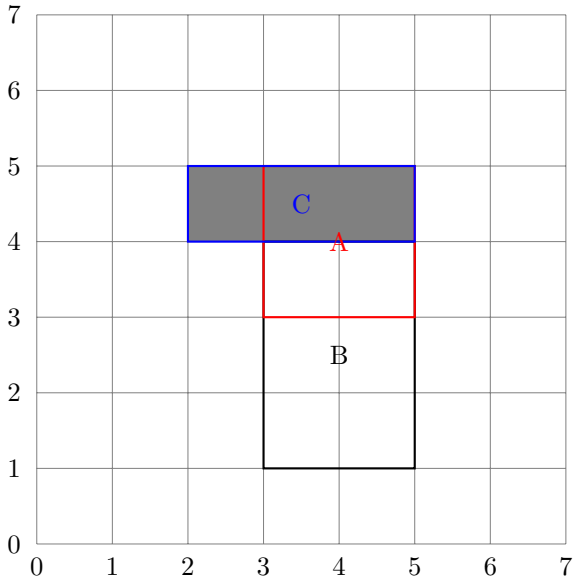
El enfoque planteado hasta ahora, aunque no define una solución completa, si especifica como podemos enfrentar los posibles casos al plantear la solución final. Pero este tiene un problema.

Pensemos que siguiendo este enfoque empezando del rectángulo más pequeño al más grande, si encontrarmos un rectángulo A de tamaño $i \times j$, y rectángulo mayor B de tamaño $i \times j + 1$, desplazado una posición en alguno de los ejes, por tanto B cubre a A, completamente, exepto una línea. Pero a su

vez está linea es cubierta por un rectángulo C de tamaño $i + 1 \times 1$.



Observámos que si borramos el cuadrado más grande osea B, nos quedaría solo C por borrar, sin embargo A no se ha borrado completamente, por tanto el algoritmo borraría primero A y luego C, lo cual es incorrecto.



Para arreglar este problema se nos ocurrión en vez de borrar el rectángulo de mayor área, borrar el que mas área pinte en cada momento. Esto aumentaría la complejidad el algoritmo pues cada vez que se borre un rectángulo habría que recalcular cuanto están pintando en cada momento los rectángulos restantes.

1.4. Greedy

Al recorrer los rectángulos de menor a mayor, si este tiene al menos un uno, entonces lo pintamos, sino restamos uno a todas sus casillas del rectángulo. Esto garantiza que si cuando un rectángulo mas grande pase por una casilla que previamente era mayor que uno pero para ese momento ya su valor es uno, indica que él es el ultimo rectángulo que queda que debería pintar dicha casilla, ya que los mas pequeños que él "borraron"su color de esa casilla con la seguridad que un rectángulo más adelante lo pintará por él.

1.5. Analisis de correctitud

Hasta ahora tenemos claro que:

- Si un rectángulo $A_{w\times h}$ tiene al menos un cuadrado que solo es pintado de negro por él mismo entonces este rectángulo hay que pintarlo de completamente con un costo de mín (w, h).
- Si un rectángulo $A_{w_1\times h_1}$ es **cubierto completamente** por al menos un rectángulo $B_{w_2\times h_2}$ tal que $w_1 \leq w_2$ y $h1 \leq h_2$, o sea, por un rectángulo mayor o igual a él que lo cubra completamente, entonces no es necesario pintar A ya que pintando B se cubre A, y el costo sería mín (w_2, h_2).
- Si un rectángulo $A_{w\times h}$ está cubierto completamente pero por más de un rectángulo, entonces si pintamos A sólo pintaríamos parcialmente los demás rectángulos, por lo que el costo total sería el costo de pintar A más el costo de pintar los demás, pero si delegamos la tarea de pintar A a los restantes rectángulos, el costo total solo sería el de pintar los demás.
- Se garantiza que pintando los rectángulos de menor a mayor con criterio de ordenación por área éste dará una respuesta correcta porque:
- Los más pequeños solo hay que pintarlos si no están cubiertos completamente por uno mas grande.
- Si un rectangulo grande está cubierto por varios rectángulos más pequeños y a su vez, el rectángulo grande está cubriendo a los más pequeños (vease como que el grande es una composición de rectángulos mas pequeños), entonces el coste óptimo es pintar el grande. Sea $A_{w\times h}$ el rectángulo grande y asumamos que hay n rectángulos más pequeños $B^i_{w_i\times h_i}$, tal que $\forall B^i_{w_i\times h_i}, w_i = w \vee h_i = h$, o sea, que todos los rectángulos B cubran a A a todo los alto o a todo lo ancho.

Con este emplo garantizamos que solo hallan rectángulos en horizontal o en vertial, pero que no hallan intermedios (figura de abajo). Si se cumple esto entonces el pintar los rectangulos mas pequeños entonces la suma de sus lados más pequeños h_i o w_i, entonces $\sum h_i = h \wedge \sum w_i = w$. En este caso, si $\forall w_i \leq h_i$, pero $h \leq w$ entonces la solución óptima sería w (porque es la suma de los lados más pequeños), sin embargo esto es un error ya que la solución optimas es h, por lo que es mejor pintar A directamente. Si por el contrario, $\forall w_i \leq h_i$, y $w \leq h$ entonces la solución óptima si es w, entonces da igual pintar A antes que los B^i que viceversa, la solución es la misma.

Si el caso anterior no se cumple y A está formado por varios rectángulos B^i pero no todos sus lados son iguales a w o a h como en el caso anterior, quiere decir que hay rectángulos intermedios, (TODO: poner la foto debajo), por lo que la $\sum w_i \geq w \vee \sum h_i \geq h$, y la solución optima en estos dos casos simpre va a ser pintar A primero.

Con lo anterior y siguendo la estrategia de descontar uno cada vez que encontramos un rectángulo que es cubierto por uno o varios mas grandes aseguramos como bien dijimos anteriormente, que en cada paso que restamos uno a cada casilla es como si estubiéramos .[el]iminando.[es]e rectángulo sin ningún costo, ya que uno o varios mas grandes que él lo cubrirán.

2. El Laberinto

2.1. Definición del problema

En tiempos antiguos, esos cuando los edificios se derrumbaban por mal tiempo y la conexión mágica era muy lenta, los héroes del reino se aventuraban en el legendario laberinto, un intrincado entramado de pasillos, cada uno custodiado por una bestia mágica. Los pasillos sólo podían caminarse en un sentido pues un viento muy fuerte no te dejaba regresar. Se decía que las criaturas del laberinto, uniendo sus fuerzas mágicas (garras y eso), habían creado ciclos dentro de este, atrapando a cualquiera que entrara a ellos en una especie de montaña rusa sin final en la que un monstruo se reía de ti cada vez que le pasabas por al lado, una locura.

El joven héroe Carlos, se enfrentaba a una prueba única: desmantelar los ciclos eternos y liberar los pasillos del laberinto para que su gente pudiera cruzarlo sin caer en los bucles infinitos de burla y depravación.

Cada vez que el héroe asesinaba cruelmente (no importa porque somos los buenos) a la criatura que cuidaba una un camino, este se rompía y desaparecía. Orión era fuerte, pero no tanto, debía optimizar bien a cuántos monstruos enfrentarse. Ayude al héroe encontrando la mínima cantidad de monstruos que debe matar para eliminar todas las montañas rusas de burla y depravación.

2.2. Primer contacto

El problema puede ser resumido a encontrar la menor cantidad de pasillos del laberinto que debemos eliminar para eliminar los ciclos del laberinto , modelando el laberinto como un conjunto de salas conectadas por pasillos unidireccionales, podr´ıamos tomar las salas como nodos y los pasillos como arcos de un Digrafo c´ıclico (si no hay ciclios no tiene sentido el ejercicio)

2.2.1. Comprobación en tiempo polinomial

Si queremos comprobar una solución ¡G,k¿es decir si eliminando k arcos eliminamos los ciclos. Simplemente eliminamos esos k arcos y comprobamos que no queden ciclos en el grafo resultante. Usando algoritmos como Floyd Warshall $O(V^3)$, verificando aristas de retroceso con Dfs O(V(G)+E(G)) etc). Por lo que verificar una solución es posible en tiempo polinomial.

2.3. Reduciendo a Vertex Cover

Supongamos que tenemos un grafo G,convirtámoslo en un grafo dirigido. Siguiendo las siguientes reglas:

1. Por cada vertice $u \in V(G)$ creamos en Gf(grafo dirigido resultante) dos vértices u_in y u_out y conectamos u_in con u_out
2. Por cada arista $u, v \in E(G)$ conectamos u_out con v_in y v_out con u_in en Gf

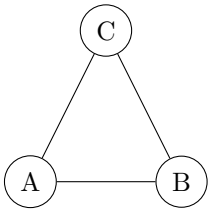


Figura 1: Grafo G

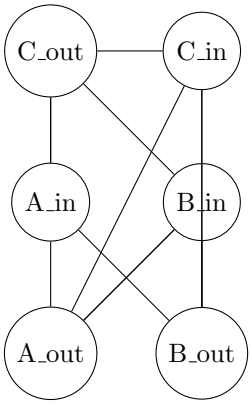


Figura 2: Grafo resultante G_f

Esta transformación es posible hacerla en tiempo polinomial siguiendo las dos reglas anteriores por cada nodo , por lo que la transformación se haría en O(V(G))

Por cada vértice en G se construye un ciclo en Gf, asumimos que el grafo no tiene vertices aislados de tener estos sería imposible hacer vertex cover y no tendría sentido la reducción. Por tanto si existiera un algoritmo que resolviera este problema en una complejidad polinomial, se pudiera transformar a vertex

Cover en una complejidad polinomial, por tanto se resolvería Vertex Cover en una complejidad polinomial , no puede pasar pq Vertex Cover está demostrado es NP-Hard. Por tanto el problema en cuestión es NP-Completo

3. El Profe

3.1. Definición del problema

Jorge es profesor de programación. En sus ratos libres, le gusta divertirse con las estadísticas de sus pobres estudiantes reprobados. Los estudiantes están separados en n grupos. Casualmente, este año, todos los estudiantes reprobaron alguno de los dos exámenes finales: P (POO) y R (Recursividad).

Esta tarde, Jorge decide entretenerse separando a los estudiantes suspensos en conjuntos de tamaño k que cumplan lo siguiente: En un mismo conjunto, todos los estudiantes son del mismo grupo i ($1 \leq i \leq n$) o suspendieron por el mismo examen P o R.

Conociendo el grupo y la prueba suspendida de cada estudiante, y el tamaño de los conjuntos, ayude a Jorge a saber cuántos conjuntos de estudiantes suspensos puede formar.