Aclaraciones Generales

Ejercicio 1: Matching Máximo

Introducción

En este ejercicio se pedía encontrar el matching máximo dentro de un grafo. Se define como matching a un subconjunto de aristas que comparten vértices. Si bien este ejercicio podría ser de gran complejidad, el mismo se encuentra en una presentación mas accesible dado que no hay que aplicar el algoritmo sobre cualquier tipo de grafos, sino que solamente tiene que ser aplicable a grafos de 3 o más nodos que sean ciclos simples, es decir, grafos que en su isomorfismo planar sean simplemente un dibujo de un polígono.

/

Algoritmo

A continuación se muestra el pseudocódigo del algoritmo propuesto como resolución del problema.

Aca va el pseudocodigo.

Este algoritmo consiste en transformar el grafo en un vector, dado que al poder ser representado como un polígono, se puede ver que, quitando una sola arista, el grafo se convierte en una sucesión de aristas con peso, lo cual se puede representar con un vector de números. La idea del algoritmo es poder encontrar el máximo matching posible, una primera aproximación a la solución final, podría ser la siguiente:

1. Se toma una arista cualquiera para comenzar. Luego, el matching buscado tiene dos opciones: contener esa arista, o no contenerla. Entonces el resultado será el máximo entre el matching máximo del grafo sin esa arista(no se utiliza la primer arista tomada) y el matching máximo del grafo sin esa arista, ni ninguno de sus dos aristas vecinas, más el valor de la arista elegida en primer término(si se utiliza la primer arista tomada). De esta manera, el problema sobre un grafo, se convierte en 2 problemas similares pero sobre vectores.

2. En este momento, se necesita sacar el matching máximo, pero sobre vectores, para hacer esto se piensa de manera parecida al punto anterior comenzando con el último elemento: o bien el matching máximo lo contiene, o bien no lo contiene. Luego, el matching buscado para el vector, sera el máximo entre el matching máximo del vector sin el último elemento(caso en el que no se usa el último elemento) y el matching

máximo del vector sin los últimos dos elementos más el valor del último elemento(caso en el que si se utiliza el último).

$matchingMaximo(v_1, v_2, .., v_n) = maximo(matchingMaximo(v_1, v_2, .., v_{n-1}), matchingMa$
$v_n)$

3. De esta manera, se puede ver que el problema se torna recursivo, siendo solucionado mediante la técnica de dividir y conquistar, teniendo como caso base los vectores de dos o un elemento resolubles trivialmente.

Si bien el algoritmo propuesto retorna el valor esperado, la complejidad del mismo no es óptima. Al hacer los llamados recursivos sucede que hay varios matching máximos que se realizan sobre los mismos vectores, generando así más cálculos de los necesarios. Fue por esto, que el siguiente paso fue modificar el algoritmo para que no calcule las cosas de modo *top down*, sino que fuese un algoritmo *bottom up*, para así evitar los cálculos repetidos, utilizando así la mayor ventaja de la programación dinámica, técnica que define al algoritmo final.

De esta manera, se pensó como teniendo instancias más peque nas del problema, se puede conocer la solución de una instancia mayor. Se utilizó que, dadas las soluciones óptimas para vectores de n-2 y n-1 elementos, la solución para el vector de n elementos es el máximo entre la solucion de n-1 elementos, y la solución de n-2 elementos más el elemento n-ésimo. Es asque el cálculo se torna *bottom up*, calculando los máximos de los subvectores, una sola vez.

Aca podria ir un ejemplo hecho paso por paso o algo asi.

Demostración de correctitud

Complejidad

Análisis de resultados

Para analizar este algoritmo, se basó el enfoque en dos aspectos diferentes. Por un lado, se encuentran los análisis sobre la correctitud de la solución propuesta y, por otro lado, se encuentra el análisis sobre el tiempo de ejecución para diferentes archivos de entrada, para poder realizar así, una correlación entre el tiempo de ejecución y la complejidad teórica calculada anteriormente.

Casos de correctitud

Con el fin de realizar una comprobación empírica de la solución propuesta se genero un archivo de entrada con diez casos de prueba.

- Los primeros nueve casos de prueba, estan conformados por los entregados por la cátedra, teniendo de esta manera las soluciones reales para contrastar con las arrojadas por el algoritmo.
- El último caso de prueba esta conformado por un grafo de 30 aristas, con pesos de 1 a 30, ordenados consecutivamente. En este caso, se puede ver que el matching máximo esta dado por tomar las aristas con valor par.

El análisis en este tipo de casos se baso solamente en la correctitud de los mismos y no en el tiempo de ejecución debido a que son grafos de tamaños muy pequeños y el tiempo de ejecucion no sobrepasa los 2 microsegundos en ninguno de los casos.

Cabe destacar, que para todos los casos propuestos, los resultados fueron satisfactorios al ser contrastados con la soluciones previamente obtenidas

Casos de Stress

.

Conclusiones

Ejercicio 2: Se inunda la isla

Introducción

En este ejercicio se pedia encontrar el area que no se inunde en una isla plana luego de ciertas condiciones.

En primer lugar, se tiene una isla que posee la misma altura en todos sus puntos, es por esto que si la marea sube la isla se inunda por completo. La solución para que las partes importantes de la isla no se inunde cuando sube la marea, es poner una serie de vallas rectangulares que no dejen pasar el agua hasta cierta altura. La idea sería entonces, colocando adecuadamente estas vallas, encerrar partes de la isla para que el agua no pueda entrar.

Aca se podria poner un dibujo de ejemplo de la isla con ciertas vallas o algo asi.

El problema consiste en, dado un conjunto de vallas y el nivel de la marea. Calcular cual es el area de la isla que no va a ser inundada.

Algoritmo

A continuación se presenta el pseudocódigo y la explicación de la solución propuesta.

Dado que, por restricción del problema, las vallas no pueden estar en cualquier lado, sino que su coordenada (x,y) del punto inferior izquierdo esta formada por x e y enteros. Asimismo, como la longitud de una valla tambien es entera, la coordenada del vértice restante tambien será entera. De esta manera, podemos ver que la isla se puede pensar como una grilla de cuadrados de 1x1. Luego, esta grilla fue pensada como un grafo, donde cada cuadrado de la grilla es un nodo y las aristas estan dadas por la relación entre un cuadrado y sus 4 posibles vecinos. Para armar esta grilla, se calcularon los mínimos y máximos valores de las vallas en x y en y, ya que por fuera de estos los cuadrados que pertenezcan a la isla se inundarán de todos modos (ya que no hay vallas que los cubran).

Al comenzar el algoritmo, todos los cuadrados estan relacionados con sus vecinos con aristas de peso 0. Esto quiere decir que si un cuadrado se inunda, los cuadrados que esten relacionados con ese por una arista de peso 0 tambien se va a inundar.

Luego, se recorren todas las vallas dadas por el problema y se setean las nuevas relaciones entre los cuadrados, es decir, si existe una valla de altura 4 entre el nodo v_1 y el nodo v_2, lo que se hace es ponerle un peso de 4 a la arista que los relaciona. Indicando así que solamente si la marea es mayor a 4, el agua pasara de ese v_1 a v_2 directamente.

Por último, lo que se hace es crear una circunvalación de nodos que se inundan al rededor del grafo real y realizar BFS desde uno de estos (que seguro se inunda) y contar cuantos nodos tiene la componente conexa que el BFS recorre por completo. Obviamente, este BFS toma que un nodo es vecino de otro si la arista que los une tiene peso menor a la marea, sino podremos decir que la valla es efectiva entre esos dos nodos y no hay inundación de uno hacia el otro. Una vez obtenida la cantidad total de nodos dados por el recorrido en anchura, el último paso es realizar la sustracción entre los nodos totales de la grilla, y los nodos inundados; obteniendo así, la cantidad total de nodos que no fueron inundados gracias a la protección de las vallas. Por último, como cada nodo representa un cuadrado de área 1, la cantidad de

nodos no alcanzados por el BFS es igual al área no inundada.

```
{
vector vallas;

leer(vallas)

maxmin(vallas)

matriz nodo[alto][ancho];

seteamosMatriz(matriz)

int inundadas  <- bfsContador(matriz)


return (ancho*alto - inundadas)
}
```

- leer(vallas): Carga en el vector vallas, todas las vallas del input.
- maxmin(vallas): Setea ciertas variables con el tamaño máximo que tendr la isla delimitada por las vallas mas externas.
- seteamosMatriz(matriz): Usando las variables seteadas por maxmin crea un grafo (representado con una matriz) donde establece 4 relaciones por nodo (con sus vecinos) donde el peso del a arista es la altura de la valla que debe atravesar.
- bfsContador(matriz): Recorre el grafo usando bfs, por cada nodo que visita suma uno a una variable que al final devuelve. Recorre todos los nodos que se inundan dado q son los que se relacionan (si la marea supera el peso de las aristas). De este modo al finalizar obtenemos cuantos nodos visitamos coincidiendo con cuantos nodos se inundan.

Complejidad

Para analizar la complejidad se utilizó el modelo uniforme. En este modelo el análisis no esta centrado en el tamaño de los operandos, por lo que el tiempo de ejecución de cada operación se considera constante.

Si tomamos como tamaño de entrada la cantidad de vallas (CV, en adelante), y como sabemos que estas no se solapan en mas de un punto, podemos

acotar (por arriba) a la cantidad de vallas, por la cantidad de nodos en el grafo. Por consiguiente vamos a analizar el algoritmo usado en funcion a la cantidad de nodos. TODO JUSTIFICAR BIEN ESTO La función leer, leer del archivo de entrada toda una instancia del problema ciclando CV para poder relevar la informacion de las vallas, por lo cual esta operación la realizamos en O(CV) La función maxmin, busca máximos y mínimos en un vector de vallas linealmente, por lo cual es de orden O (CV). ... habia q seguir

Análisis de resultados

Conclusiones

Ejercicio 3: Bernardo Armando Pandillas

Introducción

En este problema lo que se quiere es, teniendo un conjunto de personas y sabiendo si se conocen entre sí, armar dos grupos de tres personas cada uno, un grupo en el cual las 3 personas se conoscan mutuamente, y otro grupo en el cual ninguno este relacionado con otro.

Algoritmo

Complejidad

Análisis de resultados

Conclusiones