TIPS PARA SOLUCIÓN DE ALGUNOS DE LOS PROBLEMAS

TAXIS

El truco para resolver este problema fácilmente, es ver a Taxtlán como un grafo en el que cada región es un nodo y las fronteras entre dos regiones son las aristas. Una vez teniendo el mapa representado por un grafo la solución de ambos incisos resulta evidente. Para el inciso (1) hay que hacer una búsqueda a lo ancho, empezando ya sea por la región que contiene al hotel o la región que contiene a la terminal, hasta que se encuentre el otro punto. Para resolver el inciso (2) se puede utilizar la misma técnica (búsqueda a lo ancho), solamente que se tiene que hacer partiendo desde cada una de las regiones para poder escoger la que tenga un mínimo costo.

Quizás la parte más difícil del problema está entonces en representar el mapa como un grafo. Se puede lograr ésto revisando los vértices de cada una de las regiones rectangulares y buscando con cuales coincide, pero éste es un trabajo de implementación un poco largo y propenso a errores. Una forma más fácil de implementarlo, aunque tal vez un poco más lenta es la siguiente: Haz un arreglo del tamaño de Taxtlán y conforme vayan llegando las regiones, ve llenando el mapa con números, de acuerdo al número y localización de la región, una vez lleno el mapa basta con irlo recorriendo recursivamente (como un fill), claro que cuando estés llenando una región (digamos la 0), lo haces hasta encontrarte un número diferente a 0, si encuentras este número entonces en una matriz de adyacencia marcas que hay frontera entre la región 0 y la región número *x*, cuando todos los cuadros del mapa hayan sido recorridos tu grafo estará completo. El número de cuadros máximo en Taxtlán es un poco mayor a 10,000 y si marcas correctamente, cada cuadro se recorre solo una vez, por lo que el tiempo no es gran problema y sin embargo la implementación se facilita mucho y hay menor posibilidad de errores.

CIRCUITO

Este no es un problema particularmente bueno, su solución es bastante sencilla pero requiere de una implementación un poco larga.

Primero hay que localizar las compuertas, ésto es fácil ya que puedes buscar las E´s o las S´s en la matriz del circuito, así tendrás la primera parte de la solución del problema.

Para resolver la segunda parte, hay que simular el circuito, pero primero será necesario buscar una estructura de datos que nos facilite la simulación. Todas las compuertas tienen dos entradas y una salida lo que nos lleva a pensar en un árbol binario, en que los nodos representen el valor de salida de la compuerta y los hijos estén conectados a las entradas de la misma. Una vez teniendo éste árbol basta recorrerlo en post-orden (primero los hijos y luego el nodo) para obtener las salidas en cada una de las compuertas.

Una vez localizadas las compuertas hay que ver a donde están conectadas para poder crear el árbol de simulación, hay que buscar las letras C´s y haciendo una búsqueda recursiva encontrar los puntos que unen, una vez conociendo los puntos hay que ver si pertenecen a la entrada o salida de alguna compuerta. Esto nos permitirá ir creando un árbol. Aunque en general no se recomienda el uso de memoria dinámica para estructuras de datos, en este caso resulta mucho más fácil armar el árbol con una estructura de apuntadores que con un árbol en memoria estática.

Una cosa importante es tomar en cuenta que el circuito puede quedar dividido en más de un árbol de simulación, de hecho para cada compuerta puede estar en su propio árbol, por lo que resulta importante verificar que todos los árboles hayan sido simulados antes de terminar.

SUBMARINO

La solución de este problema puede encontrarse en cualquier libro de algoritmos de grafos. Para la primera parte hay que utilizar l algoritmo de *Djikstra* y para la segunda parte hay que utilizar el algoritmo de puntos de articulación.

ÁTOMOS

Este es uno de los problemas más difíciles que se ha aplicado en un examen nacional, es copia exacta de un problema utilizado en la Olimpiada Internacional de Holanda, con la dificultad añadida de que el número de compuestos es limitado, por lo tanto se hace bastante más difícil que el usado en la Olimpiada Internacional.

Este problema se resuelve usando los métodos de programación dinámica con elementos limitados, por lo que es necesario que sepas usarlos o al

menos que conozcas y entiendas la solución del *problema de la mochila o knapsack.*

La dificultad real del problema, radica en el hecho de que la cantidad de compuestos de que dispone cada proveedor es limitada. Si la cantidad de compuestos no fuera limitada, entonces el problema se resolvería utilizando la técnica del *knapsack* pero ampliada a tres dimensiones, en vez de llenar un arreglo unidimensional se iría llenando un arreglo tridimensional.

Primero que nada, llena el arreglo con los precios correspondientes para comprar solo átomos por separado, con esto tendrás al inicio un precio máximo para comprar. Uno a uno ve tomando los compuestos y empezando desde el punto al que quieres llegar (6.3.4), cuyo valor inicial para el ejemplo es 6*8 + 3*9 + 4*7 = 103, escoge la mejor opción entre las que hay (103) en este caso y la opción de tomar el compuesto que se esta utilizando una, dos, tres, ….hasta las veces que ese compuesto este disponible. Por ejemplo, si tomamos el primer compuesto del ejemplo y estamos en (6,3,4) podemos escoger entre dejar lo que esta (103) y utilizar el compuesto una vez, que sería lo que esta en (4,2,4), más lo que vale el compuesto (96) , o utilizar el compuesto dos veces, que sería lo que esta en (2,1,4), más dos veces el valor del compuesto (89). Obviamente el mejor valor es el (89), este valor nos indica la solución óptima para llegar a (6,3,4), si solo dispusiéramos de átomos separados y del compuesto uno. Ahora, no basta con calcular el óptimo para (6,3,4), es necesario rellenar la tabla con los óptimos de usar el compuesto 1, de modo que cuando utilicemos el compuesto 2, el compuesto 1 ya este tomado en cuenta. Una vez que se haya rellenado la tabla con todos los compuestos existentes se tienen los valores óptimos para cada combinación de átomos.

Otra interesante tarea para el lector, es ver porque esta solución funciona ¡¡suerte¡¡ >☺.

CINTAS

Para resolver este problema hay que observar un poco como pueden ir los discos en una cinta. Cada disco solo puede ser grabado de dos formas, solo ocupando de preferencia ambos lados de la cinta (obviamente porque se aprovecha mejor el espacio), o compartiendo la cinta con otro disco.

Aquí es donde entra el axioma que resuelve el problema ☺.

Si los discos están ordenados según su tamaño en una lista, entonces, para cualquier solución que encuentres en la que dos discos no consecutivos en la lista compartan una cinta, existe una solución igual o mejor en la que sólo discos consecutivos en la lista comparten cintas.

El porque lo anterior es cierto debe ser fácil de comprobar y queda como tarea para el lector ☺.

Para encontrar la solución óptima, basta que para cada disco escojamos la mejor opción, entre ponerlo solo en la cinta más chica en que pueda caber, o ponerlo en la misma cinta que con el disco que esta junto a él en la lista.

SUPERCONEJO

Este es un problema clásico de búsqueda a lo ancho. Muy parecido al problema del caballo de ajedrez. La primera parte es trivial, haciendo una búsqueda por nivel. Para la segunda parte, basta con ir marcando los cuadros que has visitado y abstenerte de volverlos a introducir en la cola del proceso, cuando la cola quede vacía, entonces habrás visitado todos los cuadros que podías visitar.

SUMAS

Este problema se puede resolver por una búsqueda en profundidad, pero como el espacio se soluciones es de 1,000,000,000,000, será necesario podarlo, pero por suerte puedes usar la siguiente poda: una vez que te pases del número al que quieres llegar, corta la rama en la que te encuentres, esa poda debe ser suficiente para resolver el problema en tiempo.

OMICEL

Este es otro problema más de búsqueda a lo ancho, con la leve variante de que en este problema deben efectuarse cuatro búsquedas, una por cada canal y seleccionar la mejor de ellas.

PRÁCTICA

Los problemas como este, se conocen como problemas heurísticos, ya que no es posible encontrar un algoritmo que asegure la solución óptima en el tiempo que se da. Por lo tanto debes hacer un algoritmo que se acerque lo más posible a la solución óptima en el tiempo dado.

Para este problema se implementaron cuatro soluciones oficiales, cada una con una heurística diferente, dos de las heurísticas eran algoritmos

glotones, es decir, toman lo mejor que hay en ese momento y las otras dos eran algoritmos genéticos. Para hacer un algoritmo genético, es necesario partir de un algoritmo rápido que te acerque a la solución, mientras más se acerque a la solución tu algoritmo inicial, mejor será el desempeño de tu algoritmo (para los algoritmos genéticos que se propusieron, se tomo como punto de partida los algoritmos glotones de las dos primeras soluciones), una vez que se tiene una solución inicial, el algoritmo genético, mientras cuente con tiempo, debe hacer cambios aleatorios en su solución y verificar si la solución mejora o no. No da el mismo resultado utilizar cualquier tipo de cambio, mientras más rápido sea el proceso de cambio y verificación del resultado, más cambios podrás realizar en tiempo y tu algoritmo tendrá mejores oportunidades, también hay cambios que tienen una alta probabilidad de mejorar el resultado, mientras que otros por más que se hagan, difícilmente van a brindar alguna mejora. Experimenta con diferentes cambios para ver cual da mejores resultados.

TORNEO

Este problema es enteramente matemático, aunque se puede buscar la solución por medio de búsqueda, la verdad es que no vale la pena, ya que para la mayoría de los casos no resolvería el problema en tiempo.

Para resolver el problema hay que dividirlo en casos. El primer caso va a ser cuando los dos jugadores se encuentren en la misma mitad, es decir, el número de ambos es mayor que la mitad de los jugadores o menor que la mitad de los jugadores. Cuando dos jugadores de encuentran en la misma mitad, siempre juegan la final. Para que dos jugadores se enfrenten es necesario que se encuentren en mitades opuestas, pero si se encuentran en la misma mitad basta con que en esa ronda pierdan todos los jugadores que se encuentran entre los jugadores A y B para que estos queden juntos al final o al principio de la lista y no sea posible que se enfrenten antes de la final.

El segundo caso, es cuando A y B se encuentran en mitades opuestas. Si es posible llevarlos a una mitad de la lista, entonces el problema se convierte en el caso anterior y queda resuelto. Si no es posible llevar a ambos jugadores a un solo lado de la lista, es necesario evitar lo más posible que se encuentren, resulta evidente que si no se encuentran en posiciones completamente opuestas tienen *X* jugadores en medio, pero cada ronda que pasa se acercan al menos un jugador, ya que aunque todos los jugadores entre ellos permanezcan, A y B forzosamente tienen que ganar, para que sigan en el juego. Debido a que el torneo completo dura $\log_2(N)$ rondas, entonces los dos jugadores se enfrentan antes de la final, si ambos se encuentran en mitades opuestas y los dos están a una distancia menor al número total de rondas de sus extremos, en cuyo caso la ronda en la que se enfrenten es igual a la distancia del extremo a la que se encuentre el jugador que esté más lejos de su respectivo extremo.