Modelado y Programación 2017-1: Práctica 09, Introducción a la Programación Dinámica.

Cinthia Rodriguez Maya (cinthia.rodriguez@ciencias.unam.mx)
María del Carmen Sánchez Almanza (carmensanchez@ciencias.unam.mx)
Ricchy Alaín Pérez Chevanier (alain.chevanier@ciencias.unam.mx)

Fecha de Publicación: 20 de Octubre de 2016
Fecha de Entrega: 27 de Octubre de 2016

1 Introducción

La programación dinámica tipicamente aplica a problemas de optimización, en los cuales hacemos un conjunto de elecciones que nos hacen llegar a una solución óptima. Cada vez que hacemos una elección, surgen subproblemas con la misma forma. La programación dinámica es efectiva cuando un subproblema puede surgir de más de un conjunto parcial de opciones; la técnica clave es guardar la solución de cada subproblema en caso de que esta pueda reaparecer durante la ejecución del algoritmo. En muchas ocasiones la programación dinámica nos ayuda a encontrar algoritmos a problemas cuya solución obvia es exponencial a una mejor solución que utiliza tiempo polinomial.

La programación dinámica, así como los algoritmo divide y vencerás, resuelven problemas combinando las soluciones a subproblemas. En la estrategia divide y vencerás partimos el problema en subproblemas disjuntos, luego resolvemos recursivamente los subproblemas, y finalmente combinamos la soluciones para resolver el problema original; en contraste la programación dinámica aplica cuando los subproblemas que tenenos que resolver se traslapan, es de decir que los subproblemas en los que dividimos el problema original comparten a su vez subproblemas. En este contexto, un algoritmo divide y vencerás hace más trabajo que el que es realmente necesario, pues repetidamente resuelve los subproblemas comunes, mientras un algoritmo de programación dinámica resuleva cada subproblema solamente una vez y guarda su respuesta en una "tabla", así evitando recalcular la respuesta cada vez que resuleve el mismo subproblema.

Como ya mencionamos aplicamos algoritmos de programación dinámica a problemas de optimización. Dichos problemas pueden tener muchas posibles soluciones, y a cada solución se le asocia un valor, y buscamos encontrar la solución con un valor óptimo (mínimo o máximo). Llamamos a esa solución una solución óptima al problema, opuesto a "la solución óptima", ya que pueden existir varias soluciones que alcanzan el valor óptimo asociado.

Cuando desarrollamos un algoritmo de programación dinámica, seguimos la siguiente secuencia de 4 pasos:

1. Caracterizamos la estructura de una solución óptima.
2. Definimos recursivamente el valor de una solución óptima.
3. Computamos el valor de una solución óptima, tipicamente de manera ascendente.

4. Construimos una solución óptima a partir de la información que computamos en el paso anterior.

Los pasos 1 al 3 forman la base de la solución de programación dinámica a un problema. Si solamente necesitamos el valor de una solución óptima, y no la solución per se, podemos omitir el paso 4. Cuando realizamos el paso 4, a veces es necesario mantener información adicional durante el paso 2, de tal menera que fácilmente podamos resconstruir una solución óptima.

2 Descripción del Problema

http://acm.tju.edu.cn/toj/showp2827.html

El final del verano es una temporada lenta, ... muy lente. Betsy tiene muy poco que hacer por lo que se la pasa jugando solitario de vacas. Por razones obvias, el solitario de vacas no es tan complicado como otros juegos de solitario jugados por humanos. El solitario de vacas es jugado en un tablero de NxN ($2 \leq N \leq 7$) de cartas jugables ordinarias con cuatro categorías (Clubes, Diamantes, Corazones y Espadas) de 13 cartas cada una (As, 2, 3, 4, ... , 10, Jota, Reina y Rey). Nombramos cada carta con dos caracteres: su valor (A, 2, 3, ..., 10, T, J, Q, K) seguido por su categoría (C, D, H, S) ambos por sus siglas en inglés. Abajo un ejemplo de un tablero de 4x4.

8S AD 3C AC

8C 4H QD QS

5D 9H KC 7H

TC QC AS 2D

Para jugar este juego de solitario, Betsy inicia en la esquina inferior izquierda (la que tiene el valor TC es decir 10 de Club) y procede utilizando exactamente $2 * N - 2$ movimientos a la derecha y hacia arriba hasta llegar a la esquina superior derecha. A lo largo del camino, Betsy acumula puntos por cada carta por la que pasa (As vale 1 punto, 2 vale 2 puntos, ..., 9 vale 9 puntos, T vale 10 puntos, J vale 11, Q vale 12, y K vale 13). Su objetivo es recolectar el mayor puntaje posible. Si el camino que elige Betsy es $TC - QC - AS - 2C - 7H - QS - AC$, el valor de su recorrido sería $10+12+1+2+7+12+1 = 45$. Si toma el camino $TC - 5D - 8C - 8S - AD - 3C - AC$ el resultado sería $10+5+8+8+1+3+1 = 36$, que no es tan buena como la otra ruta. De hecho el puntaje máximo para este tablero es de 69 puntos ($TC - QC - 9H - KC - QD - QS - AC \rightarrow 10 + 12 + 9 + 13 + 12 + 12 + 1$).

Betsy quiere saber el mejor puntaje que puede lograr para cada tablero que juegue. Una de las vacas geek una vez le dijo algo sobre "trabajar desde el final hacia el inicio", pero ella no entendió qué es lo que esto significa.

Entrada

- Linea 1: Un entero N

- Lineas 2.. $N+1$: La linea $i+1$ lista las cartas en el renglón i (el renglón 1 es el primer renglón del tablero) utilizando N cartas separadas por espacios en blanco y dadas en el orden en le que estas aparecen el el tablero.

Salida

- Linea 1: Un linea que contiene un entero que representa el mejor puntaje posible que Betsy puede obtener.

3 Entrega

Este reto lo tiene que resolver en alguno de los siguientes lenguajes (les recomiendo intentar resolverlo en C):

- C
- C++
- Java

Pueden evaluar su código en la siguiente liga:

http://acm.tju.edu.cn/toj/submit.php?pid=2827

Además del entregar el código fuente en el archivo readme debes de dar una explicación de cómo y por qué funciona tu algoritmo. Además debes de explicar la complejidad (el tiempo y espacio en memoria) del algoritmo .