Algorithms for Problem Solving – 11650

Dynamic Programming

Jon Ander Gómez Adrián
jon@dsic.upv.es

Departament de Sistemes Informàtics i Computació
Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

23 de marzo de 2014

Dynamic Programming I

- Es similar a divide-y-vencerás.
- Se obtiene la solución a un problema a partir de combinar la solución a subproblemas (instancias del problema más pequeñas).
- Se debe aplicar *DP* cuando los subproblemas son interdependientes, es decir, los subproblemas comparten subsubproblemas.

  En este caso los algoritmos de divide-y-vencerás realizan muchos más cálculos de los estrictamente necesarios.
- *DP* soluciona cada subproblema una sola vez y guarda el resultado en una tabla para reutilizarlo sin tener que calcularlo de nuevo.

Dynamic Programming II

- *DP* se aplica a problemas de optimización.
- Pueden existir muchas soluciones distintas al problema, pero es necesario encontrar la óptima. O una de las óptimas.
- Las óptimas son aquellas que obtienen el valor máximo o mínimo, según interese, de una función objetivo.

Dynamic Programming III

Cuando se intente abordar la solución a un problema aplicando *DP* debemos seguir los siguientes pasos:

1. Caracterizar la estructura para una solución óptima.
2. Obtener la relación de recurrencia que permite obtener el valor óptimo de la función objetivo, es decir, obtener una solución óptima. Puede haber más de una solución óptima.
3. Calcular el valor para una solución óptima desde abajo hacia arriba (bottom-up), de los casos triviales hacia la solución completa.
4. Reconstrucción de la solución óptima.

   Siempre que además del valor se necesiten los elementos que se han utilizado en la solución y en qué orden.

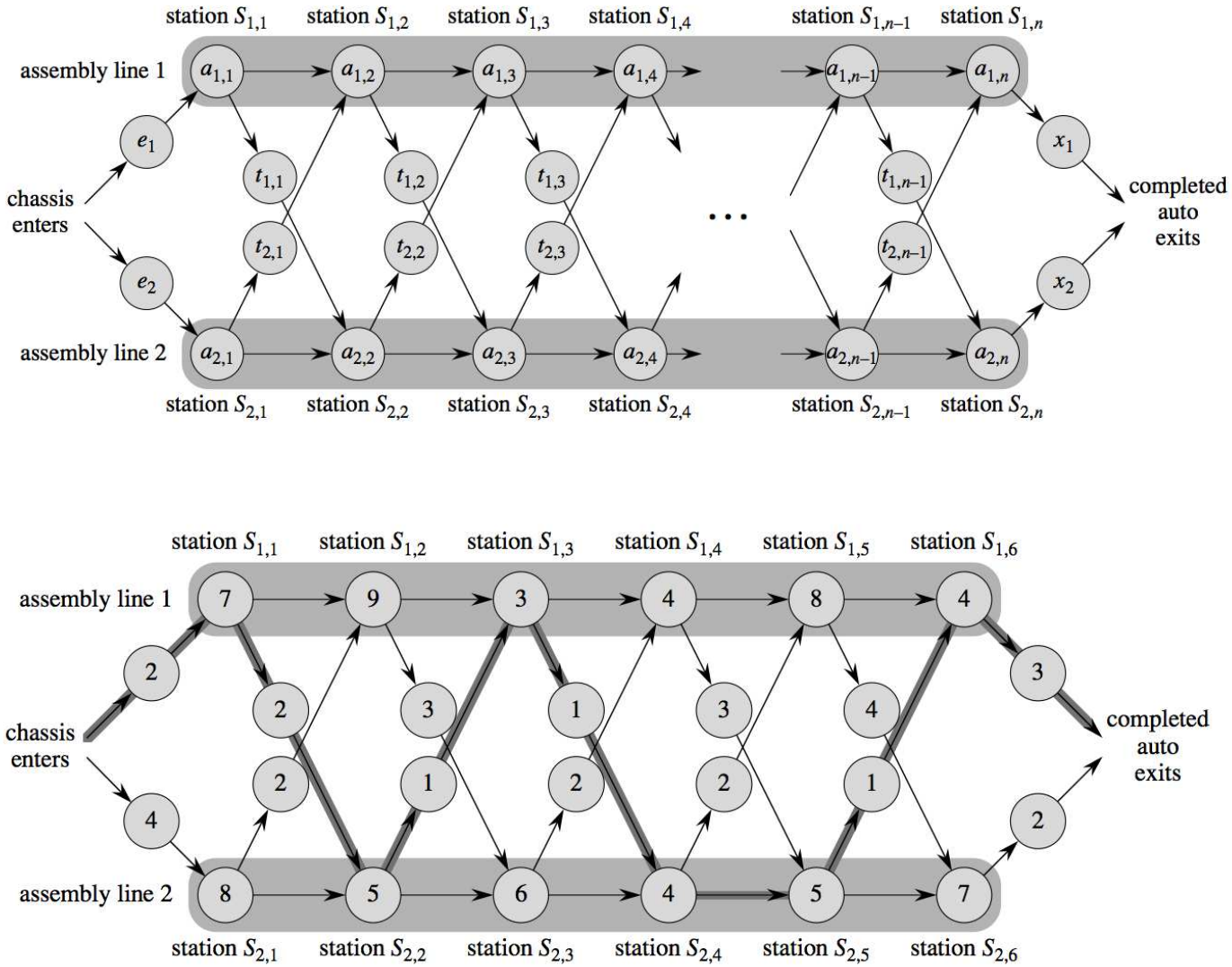
Elementos de la DP

- ¿Cuándo se puede aplicar DP en la solución a un problema?
- Optimal substructure y overlapping-subproblems.
- Optimal substructure:

  Un problema tiene subestructura óptima si la solución óptima del problema contiene, o se basa en, soluciones óptimas a subproblemas.
- Overlapping-subproblems:

  Cuando un algoritmo recursivo revisita los mismos subproblemas una y otra vez decimos que el problema tiene subproblemas solapados.

Ejemplo: líneas de ensamblaje



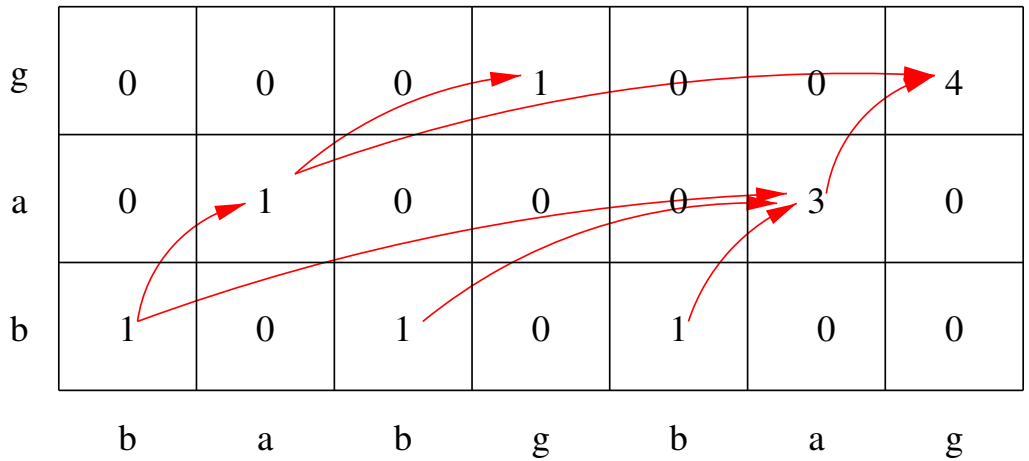
Is Bigger Smarter?

111101/10131

- Como en todos los problemas solucionables mediante programación dinámica, necesitamos encontrar la relación de recurrencia que permita obtener una solución parcial a partir de otras soluciones parciales.
- Un elefante puede preceder a otro si su peso es inferior y su índice de inteligencia superior.
- Encontrar la máxima secuencia es decidir que para llegar a un elefante desde cualquiera de los posibles predecesores nos quedamos con el predecesor que maximiza el número de elefantes en la secuencia.
- En este problema es importante el orden, debemos ordenarlos de menor a mayor peso, y en caso de pesos iguales de menor a mayor índice de inteligencia.
- Resulta más cómodo comenzar por el primero y marcar como alcanzados todos los posibles sucesores, y así sucesivamente.
- A uno alcanzado cambiaremos su predecesor cuando se aumente el número de predecesores.

111102/10069

- En este problema trabajaremos con números grandes porque el número de distintas subsecuencias puede ser bastante alto.



- Nótese como el valor de una casilla es la suma de los valores de las que la alcanzan.

Distinct Subsequences II

111102/10069

- En la primera fila (la de más abajo) solo hay ceros y unos.
- Los unos corresponden a las coincidencias de la primera letra de la cadena Z dentro de X.
- No es necesario mantener una matriz de $|X| \times |Z|$. Con dos filas de $|X|$ columnas es suficiente para el algoritmo.

111103/10154

- Una vez más el orden de los elementos es fundamental.
- Ordenaremos de menor a mayor capacidad diferencial, y en caso de empate de mayor a menor peso. Pensadlo bien.
- Iremos colocando tortugas debajo de pilas de tortugas que puedan soportar. La primera tortuga (la de menos capacidad) la colocaremos debajo de la pila vacía.
- Ojo, debajo de la pila vacía siempre podremos colocar a la nueva tortuga que intentamos colocar debajo de las pilas existentes.
- El peso total de una pila será el índice en el array. Por eso cada vez que intentamos colocar una nueva tortuga debajo las pilas existentes, gracias a la vacía se creará una pila con esa única tortuga.
- Para un mismo peso nos quedaremos con la pila que tenga más tortugas.

Unidirectional TSP

111104/116

- Este es más fácil de lo que aparenta.
- Seguro que todos tenéis ya en mente el algoritmo que encuentra el camino o los caminos de menor coste.
- Pero, ¿cómo hacemos para quedarnos con el primer camino lexicográfico?
- Intentemos realizar el algoritmo desde la última columna hacia la primera. El orden en que procesemos las filas por cada columna nos permitirá quedarnos con el camino que nos piden en caso de empate.

111105/10003

- La relación de recurrencia para este problema es:

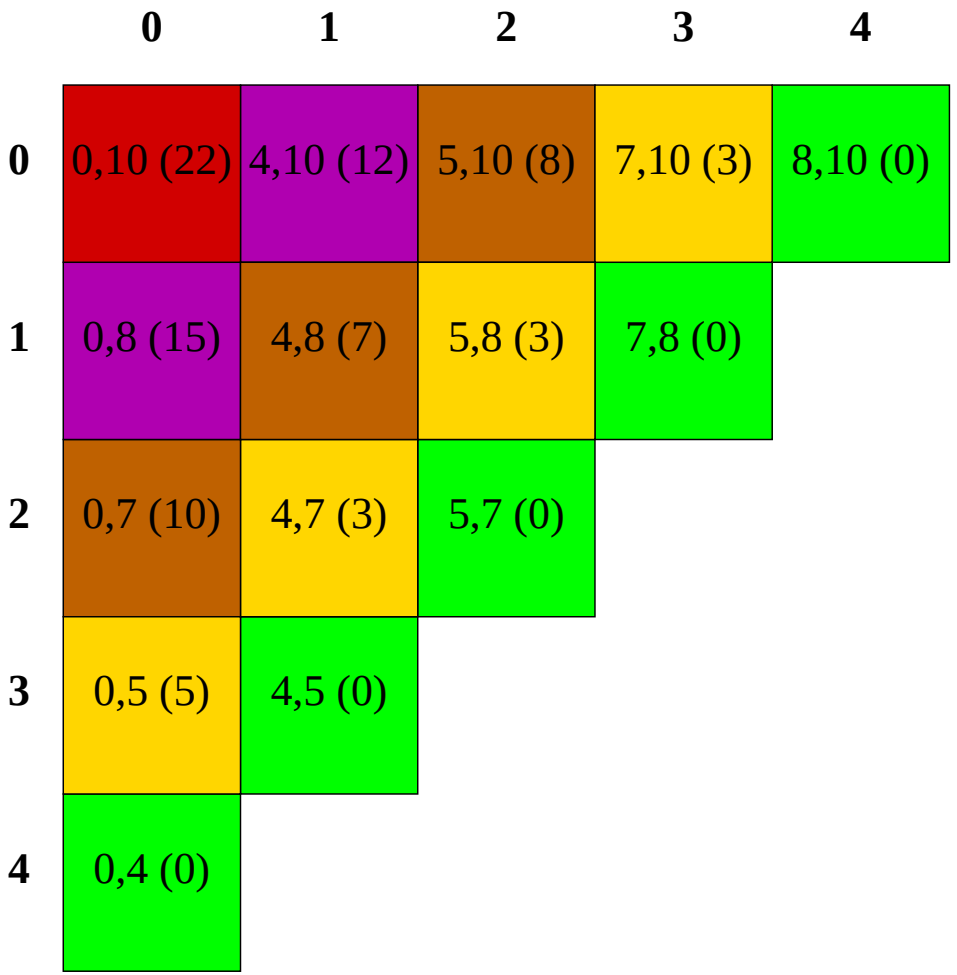
$$corte(i,j) = (pos_j - pos_i) + \underset{\forall k = i+1 \ \ldots \ j-1}{minimo}\{corte(i,k) + corte(k,j)\}$$

$$corte(i,j) = 0 \quad \texttt{siempre que} \quad i > j$$

- donde i y j son los índices para la matriz triangular que figura en la siguiente transparencia,
- $(pos_j - pos_i)$ representa el coste de cortar una vara cuya longitud es precisamente esa diferencia y que es resultado de haber cortado previamente por pos_i y pos_j.
- Inicialmente la vara está completa y por tanto $pos_i = 0$ y $pos_j = L$
- L es la longitud de la vara antes de realizar ningún corte.

Cutting Sticks II

111105/10003



Cutting Sticks III

111105/10003

- Los cuadros verdes se calculan primero, son los casos triviales.
- A continuación los amarillos, que necesitan de los verdes.
- Después los marrones, que necesitan a los amarillos y a los verdes.
- Los violeta que necesitan a todos los colores anteriores.
- Y finalmente el rojo, que necesita a todos los colores anteriores.
- Nótese que para calcular el mejor coste para la celda 0,0 que corresponde al trozo completo, únicamente son necesarias las celdas de la primera fila y de la primera columna.

Resumen problemas

111101/10131	"Is Bigger Smarter?"
111102/10069	"Distinct Subsequences"
111103/10154	"Weights and Measures"
111104/116	"Unidirectional TSP"
111105/10003	"Cutting Sticks"