Programación Dinámica

Corte de tubos (recursiva con almacén):

Complejidad espacial: O(n)
Complejidad temporal: O (n2)

Corte de tubos (iterativa):

Complejidad espacial: O(n)
Complejidad temporal: O (n2)

Principio de optimalidad (necesario para programación dinámica):

Un problema tiene una subestructura óptima si una solución óptima puede construirse eficientemente a partir de las soluciones óptimas de sus subproblemas.

Problemas clásicos para los que resulta eficaz la programación dinámica:

El problema de la mochila 0-1

Calculo de los números de Fibonacci.

Problemas con cadenas:

- La subsecuencia común máxima (longest common subsequence) de dos cadenas.
- La distancia de edición (edit distance) entre dos cadenas.

Problemas sobre grafos:

- El viajante de comercio (travelling salesman problem)
- Caminos más cortos en un grafo entre un vértice y todos los restantes (alg. de Dijkstra)
- Existencia de camino entre cualquier par de vértices (alg. de Warshall)
- Caminos más cortos en un grafo entre cualquier par de vértices (alg. De Floyd)

Mochila versión recursiva: Complejidad temporal

Mejor: n
Peor: 2^n

Algoritmos Voraces

Un algoritmo voraz es aquel que, para resolver un determinado problema, sigue una heurística consistente en elegir la opción local óptima en cada paso con la esperanza de llegar a una solución general óptima. Dicho de otra forma:

- ✓ Descompone el problema en un conjunto de decisiones . . .
- ✓ . . . y elige la más prometedora
- ✓ Nunca reconsidera las decisiones ya tomadas
- ✓ Son algoritmos eficientes y fáciles de implementar
- ✓ Puede que no se encuentre la solución optima
- ✓ Incluso pude que no se encuentre ninguna solución
- ✓ Es necesario un buen criterio de selección para tener garantías
- ✓ Se aplican mucho:
  - En problemas con muy alta complejidad
  - Cuando es suficiente una solución aproximada

Problema del cambio:

La solución voraz es tomar en cada momento la moneda de mayor valor posible.

Kruskal. ¿Cómo implementar la función noCreaCiclo(…)?

No nos sirve marcar cada vértice que se selecciona. Una arista forma parte de la solución si sus dos vértices no están ya marcados. Solución: Las estructuras de conjuntos disjuntos (Disjoint-set)

Conjuntos disjuntos:

También llamado TAD union-busqueda (union-find) por las operaciones que comprende.
Tenemos una partición de un conjunto de datos y queremos:
Inicializar la partición: cada elemento en un bloque distinto
Poder unir dos bloques de la partición (union)
Saber a qué bloque pertenece un elemento (find)

Dijkstra

En un grafo no dirigido y ponderado con pesos no negativos, calcular el coste de los caminos más cortos desde un vértice u a todos los demás.

Vuelta atrás

Problema de la mochila:

¿Cómo obtener la solución optima?

- ✓ Programación dinámica: objetos no fragmentables y pesos discretos.
- ✓ Algoritmos voraces: objetos fragmentables.

Vuelta atrás

- Algunos problemas solo se pueden resolver mediante el estudio exhaustivo del conjunto de posibles soluciones al problema.
- De entre todas ellas, se podrá seleccionar un subconjunto o bien, aquella que consideremos la mejor (la solución optima).
- Vuelta atrás proporciona una forma sistemática de generar todas las posibles soluciones a un problema.
- Generalmente se emplea en la resolución de problemas de selección u optimización en los que el conjunto de soluciones posibles es finito.
- En los que se pretende encontrar una o varias soluciones que sean:
  - Factibles: que satisfagan unas restricciones y/o.
  - Optimas: optimicen una cierta función objetivo.
- La solución debe poder expresarse mediante una tupla de decisiones.
- Las decisiones pueden pertenecer a dominios diferentes entre sí pero estos dominios siempre serán discretos o discretizables.
- Es posible que se tenga que explorar todo el espacio de soluciones.
- Los mecanismos de poda van dirigidos a disminuir la probabilidad de que esto ocurra.
- La estrategia puede proporcionar:
- Una solucion factible,
  - todas las soluciones factibles,
  - la solucion optima al problema.
  - las n mejores soluciones factibles al problema.
- A costa, en la mayoria de los casos, de complejidades prohibitivas.
- La generación y búsqueda de la solucion se realiza mediante un sistema de prueba y error.
- Se trata de un recorrido sobre una estructura arbórea imaginaria.

Ramificación y poda

¿El recorrido es importante?

Adecuando el orden de exploración del árbol de soluciones según nuestros intereses. Se priorizara para su exploración aquellos nodo más prometedores.

Variante del diseño de Vuelta Atrás

Realiza una enumeración parcial del espacio de soluciones mediante la generación de un árbol de expansión

Uso de cotas para podar ramas que no son de interés

Nodo vivo: aquel con posibilidades de ser ramificado (visitado pero no completamente expandido)

Los nodos vivos se almacenan en estructuras que faciliten su recorrido y eficiencia de la búsqueda:

En profundidad (estrategia LIFO)	$\Rightarrow$	pila
En anchura (estrategia FIFO)	$\Rightarrow$	cola
Dirigida (primero el mas prometedor)	$\Rightarrow$	cola de prioridad

Funcionamiento:

Funcionamiento de un algoritmo de ramificación y poda.

Partimos del nodo inicial del árbol.

Se asigna una solucion pesimista (subóptima, soluciones voraces).

Selección

Extracción del nodo a expandir del conjunto de nodos vivos.

La elección depende de la estrategia empleada.

Se actualiza la mejor solucion con las nuevas soluciones encontradas.

Ramificación

Se expande el nodo seleccionado en la etapa anterior dando lugar al conjunto de sus nodos hijos.

Poda

Se eliminan (podan) nodos que no contribuyen a la solucion.

El resto de nodos se añaden al conjunto de nodos vivos.

El algoritmo analiza cuando se agota el conjunto de nodos vivos.

Cotas:

Cota optimista:

Estima, a mejor, el mejor valor que podrá alcanzarse al expandir el nodo. Puede que no haya ninguna solucion factible que alcance ese valor. Normalmente se obtienen relajando las restricciones del problema. Si la cota optimista de un nodo es peor que la solucion en curso, se puede podar el nodo.

Cota pesimista:

Estima, a peor, el mejor valor que podrá alcanzarse al expandir el nodo. Ha de asegurar que existe una solucion factible con un valor mejor que la cota. Normalmente se obtienen mediante soluciones voraces del problema. Se puede eliminar un nodo si su cota optimista es peor que la mejor cota pesimista. Permite la poda aun antes de haber encontrado una solucion factible.

Cuanto más ajustadas sean las cotas, más podas se producirán.

Características Ramificación y poda:

La estrategia puede proporcionar:

- ✓ Todas las soluciones factibles
- ✓ Una solucion al problema
- ✓ La solucion optima al problema
- ✓ Las n mejores soluciones

Objetivo de esta técnica

- ✓ Mejorar la eficiencia en la exploración del espacio de soluciones

Desventajas/Necesidades

- ✓ Encontrar una buena cota optimista (problema relajado)
- ✓ Encontrar una buena solucion pesimista (estrategias voraces)
- ✓ Encontrar una buena estrategia de exploración (como ordenar)
- ✓ Mayor requerimiento de memoria que los algoritmos de Vuelta Atrás las complejidades en el peor caso suelen ser muy altas

Ventajas

- ✓ Suelen ser más rápidos que Vuelta Atrás