Algoritmos Greedy

Ingaramo Gastón[1]

[1] Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

Training Camp 2012

Contenidos

Contenidos

Que son?

- Diremos que un algoritmo es greedy cuando en cada paso, elige la ‘mejor‘ solución local.
- Dicha función de elección puede conducirnos o no a una solución óptima.
- Podemos observar una relación entre los algoritmos greedy y los DP, los algoritmos greedy parecen ser mas ‘inteligentes‘ (cuando funcionan).

Contenidos

Ejemplos - decisiones greedy.

- En el algoritmo de Kruskal(MST), elegimos en cada paso la arista mas liviana que no genera un ciclo.
- En el problema de la moneda, elegimos en cada paso la mas pesada que no sea superior al monto a devolver.

Contenidos

Cuando podemos utilizar una estrategia greedy?

- Cuando un problema exhibe la propiedad ‘optimal-substructure‘. Es decir que toda solución óptima a un problema puede ser construida considerando soluciones optimas de los subproblemas.
- Los problemas que exhiben dicha propiedad pueden ser resueltos de forma greedy o con programación dinamica.
- Existe también un conjunto de teoremas para demostrar que cuando un problema exhibe las propiedades de un matroide, siempre un algoritmo greedy nos llevará a una solución maximal que es óptima. (Kruskal)

Contenidos

Mas ejemplos

- 0-1 knapsack problem. Podemos solucionar este problema con programación dinámica, no asi con un algoritmo greedy.
- Fractional knapsack problem. Podemos solucionar este problema con un algoritmo greedy.
- Problema del cambio. No podemos solucionarlo siempre con greedy pero podemos con programación dinamica.
- Problema de selección de actividades compatibles. Con ambos.

Contenidos

Activity Selection Problem

- Dado un conjunto S de n tareas, $S = a_1, a_1, a_2, ..., a_n$.
- Diremos que la actividad a_i comienza en el instante s_i y finaliza en el instante t_i.
- Diremos también que dos tareas i, j con $1 \leq i < j \leq n$ son compatibles sii $f_i \leq s_j$.
- El problema pide encontrar uno de los conjuntos de tareas compatibles con mas elementos (puede haber varias soluciones optimas).

Activity Selection Problem - Optimal substructure

- Definamos S_{ij} como $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- Creamos nuevas tareas a_0 y a_{n+1} con $f_0 = -inf$ y $s_{n+1} = inf$. Luego, una solución de $S_{0(n+1)}$ es una solución a nuestro problema original.

Activity Selection Problem - Optimal substructure

- Definamos S_{ij} como $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- Creamos nuevas tareas a_0 y a_{n+1} con $f_0 = -inf$ y $s_{n+1} = inf$. Luego, una solución de $S_{0(n+1)}$ es una solución a nuestro problema original.

Activity Selection Problem - Optimal substructure

- Veamos la subestructura del problema. Si tenemos una solución a S_{ij} que utiliza a_k, vemos que nuestra solución es igual a una solución de S_{ik}, mas una solución de S_{kj}, mas a_k.

- Veamos la subestructura óptima del problema. Si tenemos una solución optima A_{ij} de S_{ij} que utiliza a_k, tenemos que las soluciones A_{ik} de S_{ik} y A_{kj} de S_{kj} deben ser optimas también. (prueba por contradicción en clase).

Activity Selection Problem - Optimal substructure

- Veamos la subestructura del problema. Si tenemos una solución a S_{ij} que utiliza a_k, vemos que nuestra solución es igual a una solución de S_{ik}, mas una solución de S_{kj}, mas a_k.
- Veamos la subestructura óptima del problema. Si tenemos una solución optima A_{ij} de S_{ij} que utiliza a_k, tenemos que las soluciones A_{ik} de S_{ik} y A_{kj} de S_{kj} deben ser optimas también. (prueba por contradicción en clase).

Activity Selection Problem - Optimal substructure

- Por lo tanto si tenemos la solución óptima a todos los subproblemas de S_{ij}, podemos encontrar a A_{ij}

- $A_{ij} = A_{ik} U\{a_k\} UA_{kj}$, para alguna actividad k tal que $f_i \leq s_k$ y $f_k \leq s_j$.

Activity Selection Problem - Optimal substructure

- Por lo tanto si tenemos la solución óptima a todos los subproblemas de S_{ij}, podemos encontrar a A_{ij}
- $A_{ij} = A_{ik} U \{a_k\} U A_{kj}$, para alguna actividad k tal que $f_i \leq s_k$ y $f_k \leq s_j$.

Activity Selection Problem - Solución greedy

- Con el enunciado anterior podemos facilmente programar una dinámica que encuentre la solución en O(n^3).
- Demostraremos en clase lo siguiente:
- Sea S_{ij} un subproblema no vacío y a_m la tarea dentro de S_{ij} que termina antes:
- 1) La actividad a_m pertenece a alguna solución óptima.
- 2) El subproblema S_{im} es vacío, por lo que elegír a_m nos deja solo a S_{mj} como subproblema no vacío.

Activity Selection Problem - Solución greedy

- Con el enunciado anterior podemos facilmente programar una dinámica que encuentre la solución en O(n^3).
- Demostraremos en clase lo siguiente:
- Sea S_{ij} un subproblema no vacío y a_m la tarea dentro de S_{ij} que termina antes:
- 1) La actividad a_m pertenece a alguna solución óptima.
- 2) El subproblema S_{im} es vacío, por lo que elegír a_m nos deja solo a S_{mj} como subproblema no vacío.

Activity Selection Problem - Solución greedy

```
int ASP(int s[], int f[], int n) {
  // los eventos estan ordenados de forma creciente por su finalizaci n.
  int selected = 0;
  int F = -(1<<30);
  for (int i = 0; i < n; i++) {
    if(s[i] >= F) {
      selected++;
      F = f[i];
    }
  }
  return selected;
}
```

- Complejidad del algoritmo : O(n).

Conclusión

- Es conveniente verificar que el problema exhiba la optimal-substructure property antes de intentar pensar una estrategia greedy.
- Una idea greedy puede llegar a ser facil de implementar, pero puede fallar.
- Antes de codear una dinamica muy compleja, investigar la subestructura del problema para ver si podemos deducir una propiedad que sea ‘inteligente‘.