Presentación Práctico Algoritmos Voraces (Greedy)

Nicolás Carrasco

Instituto de Computación, Facultad de Ingeniería, Universidad de la República
Basadas en Fundamentos de Algoritmia. G. Brassard and P. Bratley. Prentice Hall, 1998.

Programación 3, 26 de Octubre de 2010

Plan de la exposición

Repaso Algoritmos Voraces

- Los **algoritmos voraces** se basan en **construir** una **solución** utilizando **decisiones** que son **óptimos locales** logrando un óptimo global (en caso de ser correcto el algoritmo).
- C es un conjunto de **candidatos**. Una vez que es descartado un candidato $c \in C$ nunca más es considerado.
- S es una solución.
- **Función select:** se encarga de la selección del mejor de los candidatos basándose en una decisión local.
  - Busca el más prometedor.
- **Función objetivo:** Es aquella que queremos maximizar o minimizar (según el problema)

Repaso Algoritmos Voraces

- Los **algoritmos voraces** se basan en **construir** una **solución** utilizando **decisiones** que son **óptimos locales** logrando un óptimo global (en caso de ser correcto el algoritmo).
- C es un conjunto de **candidatos**. Una vez que es descartado un candidato $c \in C$ nunca más es considerado.
- S es una solución.
- **Función select:** se encarga de la selección del mejor de los candidatos basándose en una decisión local.
  - Busca el más prometedor.
- **Función objetivo:** Es aquella que queremos maximizar o minimizar (según el problema)

Repaso Algoritmos Voraces

- Los **algoritmos voraces** se basan en **construir** una **solución** utilizando **decisiones** que son **óptimos locales** logrando un óptimo global (en caso de ser correcto el algoritmo).
- C es un conjunto de **candidatos**. Una vez que es descartado un candidato $c \in C$ nunca más es considerado.
- S es una solución.
- **Función select:** se encarga de la selección del mejor de los candidatos basándose en una decisión local.
  - Busca el más prometedor.
- **Función objetivo:** Es aquella que queremos maximizar o minimizar (según el problema)

Repaso Algoritmos Voraces

- Los **algoritmos voraces** se basan en **construir** una **solución** utilizando **decisiones** que son **óptimos locales** logrando un óptimo global (en caso de ser correcto el algoritmo).
- C es un conjunto de **candidatos**. Una vez que es descartado un candidato $c \in C$ nunca más es considerado.
- S es una solución.
- **Función select:** se encarga de la selección del mejor de los candidatos basándose en una decisión local.
  - Busca el más prometedor.
- **Función objetivo:** Es aquella que queremos maximizar o minimizar (según el problema)

Repaso Algoritmos Voraces

- Los **algoritmos voraces** se basan en **construir** una **solución** utilizando **decisiones** que son **óptimos locales** logrando un óptimo global (en caso de ser correcto el algoritmo).
- C es un conjunto de **candidatos**. Una vez que es descartado un candidato $c \in C$ nunca más es considerado.
- S es una solución.
- **Función select:** se encarga de la selección del mejor de los candidatos basándose en una decisión local.
  - Busca el más prometedor.
- **Función objetivo:** Es aquella que queremos maximizar o minimizar (según el problema)

Repaso Algoritmos Voraces

```
Greedy ( C )
{
    S := {}
    while ( !esVacio(C) and !esSolucion(S) )
    {
        x := Select(C)
        C = C - {x}
        if ( esFactible (S U {x} ))
        {
        S := S U {x}
        }
    }
    if ( esSolucion(S))
        return S
    else
        return {}
}
```

Repaso Algoritmos Voraces

- En el algoritmo de:
  - Prim los candidatos son los vertices.
  - Kruskal los candidatos son las aristas.
  - ¿Cuales son los candidatos en el algoritmo de Dijsktra?
- Identificar cuales son los candidatos, función select y objetivo en los problemas vistos en el teórico.

Repaso Algoritmos Voraces

- En el algoritmo de:
  - Prim los candidatos son los vertices.
  - Kruskal los candidatos son las aristas.
  - ¿Cuales son los candidatos en el algoritmo de Dijsktra?
- Identificar cuales son los candidatos, función select y objetivo en los problemas vistos en el teórico.

Ejercicio 4

Sea $\{J_1, ..., J_n\}$ un conjunto de trabajos que deben ser ejecutados secuencialmente en un mismo procesador. Cada J_i consume una unidad de tiempo de procesador y produce una ganancia $r_i > 0$ en el caso en que sea concluido antes de su tiempo límite t_i. Si J_i se concluye luego de su t_i, no produce ganancia. El problema consiste en determinar el orden de procesamiento de los trabajos que produzca mayor ganancia. Considere dados los valores $r_1, \ldots, r_n$ y $t_1, \ldots, t_n$. Formular una estrategia ávida (Greedy) que resuelva el problema, justificando por qué conduce al óptimo.

Ejercicio 4

- Es un problema de Planificación con plazo fijo.
- Se dispone de un conjunto T de n tareas.
- La tarea i produce beneficio r_i si es ejecutada en un instante anterior a t_i.
- Se quiere encontrar S tal que $\sum_{J_i \in S} r_i$ sea máxima. (Todos los trabajos incluidos deben estar en hora).

Ejercicio 4

- Es un problema de Planificación con plazo fijo.
- Se dispone de un conjunto T de n tareas.
- La tarea i produce beneficio r_i si es ejecutada en un instante anterior a t_i.
- Se quiere encontrar S tal que $\sum_{J_i \in S} r_i$ sea máxima. (Todos los trabajos incluidos deben estar en hora).

Ejercicio 4

- Es un problema de Planificación con plazo fijo.
- Se dispone de un conjunto T de n tareas.
- La tarea i produce beneficio r_i si es ejecutada en un instante anterior a t_i.
- Se quiere encontrar S tal que $\sum_{J_i \in S} r_i$ sea máxima. (Todos los trabajos incluidos deben estar en hora).

Ejercicio 4

- Es un problema de Planificación con plazo fijo.
- Se dispone de un conjunto T de n tareas.
- La tarea i produce beneficio r_i si es ejecutada en un instante anterior a t_i.
- Se quiere encontrar S tal que $\sum_{J_i \in S} r_i$ sea máxima. (Todos los trabajos incluidos deben estar en hora).

Ejercicio 4

Conjunto de tareas factible

Se dice que un conjunto de tareas es factible si existe al menos una sucesión (que también se llama sucesión factible) que permite que todas las tareas del conjunto se ejecuten en sus respectivos plazos.

Algoritmo voraz

Un algoritmo voraz evidente consiste en construir una planificación paso a paso, añadiendo en cada paso la tarea que tenga el mayor r_i, entre las que aún no se hayan considerado, siempre y cuando el conjunto de tareas seleccionadas siga siendo factible.

Ejercicio 4

Conjunto de tareas factible

Se dice que un conjunto de tareas es factible si existe al menos una sucesión (que también se llama sucesión factible) que permite que todas las tareas del conjunto se ejecuten en sus respectivos plazos.

Algoritmo voraz

Un algoritmo voraz evidente consiste en construir una planificación paso a paso, añadiendo en cada paso la tarea que tenga el mayor r_i, entre las que aún no se hayan considerado, siempre y cuando el conjunto de tareas seleccionadas siga siendo factible.

Ejercicio 4

Conjunto de tareas factible

Se dice que un conjunto de tareas es factible si existe al menos una sucesión (que también se llama sucesión factible) que permite que todas las tareas del conjunto se ejecuten en sus respectivos plazos.

Algoritmo voraz

Un algoritmo voraz evidente consiste en construir una planificación paso a paso, añadiendo en cada paso la tarea que tenga el mayor r_i, entre las que aún no se hayan considerado, siempre y cuando el conjunto de tareas seleccionadas siga siendo factible.

Ejercicio 4

Queda por:

1. Demostrar que el algoritmo siempre encuentra una planificación óptima.
2. Encontrar una manera eficiente de implementarlo.

Ejercicio 4

Queda por:

1. Demostrar que el algoritmo siempre encuentra una planificación óptima.
2. Encontrar una manera eficiente de implementarlo.

Ejercicio 4

Lema

Sea J un conjunto de k tareas. Supongamos, sin pérdida de generalidad que las tareas están numeradas de tal forma que $t_1 \leq t_2 \leq \ldots t_k$. Entonces el conjunto J es factible si y sólo si la secuencia $1, 2, \ldots, k$ es factible.

Demostración

Si la secuencia S_J es factible es evidente que J es factible. Por otro lado, supongamos que la secuencia $1, 2, \ldots, k$ no es factible. Sea r cualquiera de estas tareas tal que $t_r \leq r - 1$. Dado que las tareas se ejecutan por orden no decrecientes de plazos, esto significa que al menos r tareas tienen fecha final $r - 1$ ó anterior. Sea cual fuere la forma en que se planifiquen, la última siempre llegará tarde contradiciendo que J es factible.

Ejercicio 4

Lema

Sea J un conjunto de k tareas. Supongamos, sin pérdida de generalidad que las tareas están numeradas de tal forma que $t_1 \leq t_2 \leq \ldots t_k$. Entonces el conjunto J es factible si y sólo si la secuencia $1, 2, \ldots, k$ es factible.

Demostración

Si la secuencia S_J es factible es evidente que J es factible. Por otro lado, supongamos que la secuencia $1, 2, \ldots, k$ no es factible. Sea r cualquiera de estas tareas tal que $t_r \leq r - 1$. Dado que las tareas se ejecutan por orden no decrecientes de plazos, esto significa que al menos r tareas tienen fecha final $r - 1$ ó anterior. Sea cual fuere la forma en que se planifiquen, la última siempre llegará tarde contradiciendo que J es factible.

Ejercicio 4

Lema

Sea J un conjunto de k tareas. Supongamos, sin pérdida de generalidad que las tareas están numeradas de tal forma que $t_1 \leq t_2 \leq \ldots t_k$. Entonces el conjunto J es factible si y sólo si la secuencia $1, 2, \ldots, k$ es factible.

Demostración

Si la secuencia S_J es factible es evidente que J es factible. Por otro lado, supongamos que la secuencia $1, 2, \ldots, k$ no es factible. Sea r cualquiera de estas tareas tal que $t_r \leq r - 1$. Dado que las tareas se ejecutan por orden no decrecientes de plazos, esto significa que al menos r tareas tienen fecha final $r - 1$ ó anterior. Sea cual fuere la forma en que se planifiquen, la última siempre llegará tarde contradiciendo que J es factible.

Correctitud del algoritmo

- Supongamos que el algoritmo voraz decide ejecutar un conjunto de tareas I en la secuencia S_I y se tiene otro conjunto de tareas J óptimo y una secuencia óptima de J llamada S_J.
- Reorganizando las tareas de S_I y S_J, que pueden tener huecos, tales que toda tarea común a I y a J se planifique en el mismo instante de tiempo. Sea a una tarea que aparece en las dos secuencias factibles S_I y S_J, en donde queda planificada en los instantes i y j respectivamente.

Correctitud del algoritmo

- Supongamos que el algoritmo voraz decide ejecutar un conjunto de tareas I en la secuencia S_I y se tiene otro conjunto de tareas J óptimo y una secuencia óptima de J llamada S_J.
- Reorganizando las tareas de S_I y S_J, que pueden tener huecos, tales que toda tarea común a I y a J se planifique en el mismo instante de tiempo. Sea a una tarea que aparece en las dos secuencias factibles S_I y S_J, en donde queda planificada en los instantes i y j respectivamente.

Correctitud del algoritmo

- Supongamos que el algoritmo voraz decide ejecutar un conjunto de tareas I en la secuencia S_I y se tiene otro conjunto de tareas J óptimo y una secuencia óptima de J llamada S_J.
- Reorganizando las tareas de S_I y S_J, que pueden tener huecos, tales que toda tarea común a I y a J se planifique en el mismo instante de tiempo. Sea a una tarea que aparece en las dos secuencias factibles S_I y S_J, en donde queda planificada en los instantes i y j respectivamente.

Correctitud del algoritmo

Si $i < j$, dado que S_J es factible, se sigue que el plazo de tarea a no es anterior a j.

	a		$?_I$	
	$?_J$		a	

Pasa a ser:

	$?_I$		a	
	$?_J$		a	

Correctitud del algoritmo

Si $i < j$, dado que S_J es factible, se sigue que el plazo de tarea a no es anterior a j.

	a		$?_I$	
	$?_J$		a	

Pasa a ser:

	$?_I$		a	
	$?_J$		a	

Correctitud del algoritmo

Si $i > j$

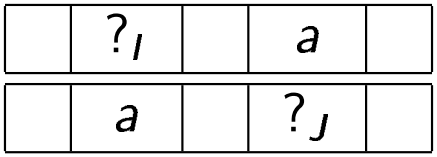
	$?_I$		a	
	a		$?_J$	

Pasa a ser:

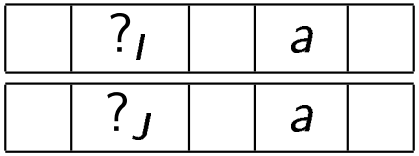
	$?_I$		a	
	$?_J$		a	

Correctitud del algoritmo

Si $i > j$



Pasa a ser:



Correctitud del algoritmo

- Una vez que se han tratado las tareas de esta manera está claro que nunca será preciso volver a transladarlas.
- Por lo tanto, si S_I y S_J tienen m tareas en común, después de un máximo de m modificaciones podemos asegurar que todas las tareas comunes estarán planificadas al mismo tiempo en ambas secuencias.
- Las secuencias resultantes pueden no ser iguales si $I \neq J$.
- Por lo tanto supongamos que existe un instante en el cual la tarea planificada S_J es distinta de la S_I.

Correctitud del algoritmo

- Una vez que se han tratado las tareas de esta manera está claro que nunca será preciso volver a transladarlas.
- Por lo tanto, si S_I y S_J tienen m tareas en común, después de un máximo de m modificaciones podemos asegurar que todas las tareas comunes estarán planificadas al mismo tiempo en ambas secuencias.
- Las secuencias resultantes pueden no ser iguales si $I \neq J$.
- Por lo tanto supongamos que existe un instante en el cual la tarea planificada S_J es distinta de la S_I.

Correctitud del algoritmo

- Una vez que se han tratado las tareas de esta manera está claro que nunca será preciso volver a transladarlas.
- Por lo tanto, si S_I y S_J tienen m tareas en común, después de un máximo de m modificaciones podemos asegurar que todas las tareas comunes estarán planificadas al mismo tiempo en ambas secuencias.
- Las secuencias resultantes pueden no ser iguales si $I \neq J$.
- Por lo tanto supongamos que existe un instante en el cual la tarea planificada S_J es distinta de la S_I.

Correctitud del algoritmo

- Una vez que se han tratado las tareas de esta manera está claro que nunca será preciso volver a transladarlas.
- Por lo tanto, si S_I y S_J tienen m tareas en común, después de un máximo de m modificaciones podemos asegurar que todas las tareas comunes estarán planificadas al mismo tiempo en ambas secuencias.
- Las secuencias resultantes pueden no ser iguales si $I \neq J$.
- Por lo tanto supongamos que existe un instante en el cual la tarea planificada S_J es distinta de la S_I.

Correctitud del algoritmo

- Si alguna tarea a está planificada en $S_I{}'$ frente a un hueco de $S_J{}'$ entonces no pertenece a J. El conjunto $J \cup \{a\}$ es factible contradiciendo que J sea óptimo.
- Si alguna tarea a esta planificada en $S_J{}'$ frente a un hueco de $S_I{}'$, el conjunto $I \cup \{a\}$ es factible. Lo anterior no puede suceder debido a que el algortimo voraz lo hubiece incluido en I.
- Alguna tarea a esta planificada en $S_J{}'$ frente a otra tarea distinta b de $S_I{}'$.
  - $r_a > r_b$, se podría sustituir a por b en J y mejorarla. Esto es imposible porque J es óptimo.
  - $r_a < r_b$, el algoritmo voraz habría seleccionado b antes de considerar a a, puesto que $(I \setminus \{a\}) \cup \{b\}$sería factible. Esto es imposible.
  - $r_a = r_b$ es la única posibilidad restante.

Correctitud del algoritmo

- Si alguna tarea a está planificada en S_I´ frente a un hueco de S_J´ entonces no pertenece a J. El conjunto $J \cup \{a\}$ es factible contradiciendo que J sea óptimo.
- Si alguna tarea a esta planificada en S_J´ frente a un hueco de S_I´, el conjunto $I \cup \{a\}$ es factible. Lo anterior no puede suceder debido a que el algortimo voraz lo hubiece incluido en I.
- Alguna tarea a esta planificada en S_J´ frente a otra tarea distinta b de S_I´.
  - $r_a > r_b$, se podría sustituir a por b en J y mejorarla. Esto es imposible porque J es óptimo.
  - $r_a < r_b$, el algoritmo voraz habría seleccionado b antes de considerar a a, puesto que $(I \setminus \{a\}) \cup \{b\}$sería factible. Esto es imposible.
  - $r_a = r_b$, es la única posibilidad restante.

Correctitud del algoritmo

- Si alguna tarea a está planificada en $S_I{}'$ frente a un hueco de $S_J{}'$ entonces no pertenece a J. El conjunto $J \cup \{a\}$ es factible contradiciendo que J sea óptimo.
- Si alguna tarea a esta planificada en $S_J{}'$ frente a un hueco de $S_I{}'$, el conjunto $I \cup \{a\}$ es factible. Lo anterior no puede suceder debido a que el algortimo voraz lo hubiece incluido en I.
- Alguna tarea a esta planificada en $S_J{}'$ frente a otra tarea distinta b de $S_I{}'$.
  - $r_a > r_b$, se podría sustituir a por b en J y mejorarla. Esto es imposible porque J es óptimo.
  - $r_a < r_b$, el algoritmo voraz habría seleccionado b antes de considerar a a, puesto que $(I \setminus \{a\}) \cup \{b\}$sería factible. Esto es imposible.
  - $r_a = r_b$ es la única posibilidad restante.

Correctitud del algoritmo

- Si alguna tarea a está planificada en $S_I{}'$ frente a un hueco de $S_J{}'$ entonces no pertenece a J. El conjunto $J \cup \{a\}$ es factible contradiciendo que J sea óptimo.
- Si alguna tarea a esta planificada en $S_J{}'$ frente a un hueco de $S_I{}'$, el conjunto $I \cup \{a\}$ es factible. Lo anterior no puede suceder debido a que el algortimo voraz lo hubiece incluido en I.
- Alguna tarea a esta planificada en $S_J{}'$ frente a otra tarea distinta b de $S_I{}'$.
  - $r_a > r_b$, se podría sustituir a por b en J y mejorarla. Esto es imposible porque J es óptimo.
  - $r_a < r_b$, el algoritmo voraz habría seleccionado b antes de considerar a a, puesto que $(I \setminus \{a\}) \cup \{b\}$sería factible. Esto es imposible.
  - $r_a = r_b$ es la única posibilidad restante.

Correctitud del algoritmo

- Si alguna tarea a está planificada en $S_I{}'$ frente a un hueco de $S_J{}'$ entonces no pertenece a J. El conjunto $J \cup \{a\}$ es factible contradiciendo que J sea óptimo.
- Si alguna tarea a esta planificada en $S_J{}'$ frente a un hueco de $S_I{}'$, el conjunto $I \cup \{a\}$ es factible. Lo anterior no puede suceder debido a que el algortimo voraz lo hubiece incluido en I.
- Alguna tarea a esta planificada en $S_J{}'$ frente a otra tarea distinta b de $S_I{}'$.
  - $r_a > r_b$, se podría sustituir a por b en J y mejorarla. Esto es imposible porque J es óptimo.
  - $r_a < r_b$, el algoritmo voraz habría seleccionado b antes de considerar a a, puesto que $(I \setminus \{a\}) \cup \{b\}$sería factible. Esto es imposible.
  - $r_a = r_b$ es la única posibilidad restante.

Correctitud del algoritmo

- Si alguna tarea a está planificada en $S_I{}'$ frente a un hueco de $S_J{}'$ entonces no pertenece a J. El conjunto $J \cup \{a\}$ es factible contradiciendo que J sea óptimo.
- Si alguna tarea a esta planificada en $S_J{}'$ frente a un hueco de $S_I{}'$, el conjunto $I \cup \{a\}$ es factible. Lo anterior no puede suceder debido a que el algortimo voraz lo hubiece incluido en I.
- Alguna tarea a esta planificada en $S_J{}'$ frente a otra tarea distinta b de $S_I{}'$.
  - $r_a > r_b$, se podría sustituir a por b en J y mejorarla. Esto es imposible porque J es óptimo.
  - $r_a < r_b$, el algoritmo voraz habría seleccionado b antes de considerar a a, puesto que $(I \setminus \{a\}) \cup \{b\}$sería factible. Esto es imposible.
  - $r_a = r_b$ es la única posibilidad restante.