PENSANDO EN GREEDY

Programacion Competitiva

Friday 22nd February, 2019

Santiago Hincapie Potes

Universidad EAFIT

EL DIA DE HOY VEREMOS

1. Algoritmos greedy

2. Greedy is god

3. Proxima sesion

ALGORITMOS GREEDY

¿QUE SON?

→ Diremos que un algoritmo es greedy cuando en cada paso, elige la “mejor” solución local.

¿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.

¿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.
- Cuando el algoritmo conduzca a una solución óptima diremos que el greedy “funciona”.

¿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.
- Cuando el algoritmo conduzca a una solución óptima diremos que el greedy “funciona”.
- Beneficio inmediato.

PROBLEMA DE LA MONEDA

→ El problema de cambio de monedas aborda la forma de encontrar el número mínimo de monedas (de ciertas denominaciones) tales que entre ellas suman una cierta cantidad.

PROBLEMA DE LA MONEDA

- → El problema de cambio de monedas aborda la forma de encontrar el número mínimo de monedas (de ciertas denominaciones) tales que entre ellas suman una cierta cantidad.
- → Elegimos en cada paso la moneda de mayor denominacion que no supere el monto.

PROBLEMA DE LA MONEDA

- El problema de cambio de monedas aborda la forma de encontrar el número mínimo de monedas (de ciertas denominaciones) tales que entre ellas suman una cierta cantidad.
- Elegimos en cada paso la moneda de mayor denominacion que no supere el monto.
- ¿Funciona esta idea?

PROBLEMA DE LA MONEDA

- → El problema de cambio de monedas aborda la forma de encontrar el número mínimo de monedas (de ciertas denominaciones) tales que entre ellas suman una cierta cantidad.
- → Elegimos en cada paso la moneda de mayor denominacion que no supere el monto.
- → ¿Funciona esta idea?
- → Consideremos que tenemos monedas de $(25, 15, 1)$ y deseamos dar un cambio de 30

PROBLEMA DE LA MONEDA

- El problema de cambio de monedas aborda la forma de encontrar el número mínimo de monedas (de ciertas denominaciones) tales que entre ellas suman una cierta cantidad.
- Elegimos en cada paso la moneda de mayor denominacion que no supere el monto.
- ¿Funciona esta idea?
- Consideremos que tenemos monedas de $(25, 15, 1)$ y deseamos dar un cambio de 30
- El algoritmo encontraria la secuencia $\{25, 1, 1, 1, 1, 1\}$, sin embargo, la secuencia optima es $\{15, 15\}$

PROBLEMA DE LA SELECCIÓN DE TAREAS

Juan tiene n actividades que realizar y sabe cuándo empieza y cuándo termina cada una. Lamentablemente algunas se superponen y por lo tanto no puede realizarlas todas. El problema pide la máxima cantidad de actividades que Juan puede realizar sin que se le superpongan dos de ellas.

PROBLEMA DE LA SELECCIÓN DE TAREAS

Juan tiene n actividades que realizar y sabe cuándo empieza y cuándo termina cada una. Lamentablemente algunas se superponen y por lo tanto no puede realizarlas todas. El problema pide la máxima cantidad de actividades que Juan puede realizar sin que se le superpongan dos de ellas.

→ Por ejemplo si tenemos tres tareas de rangos $(1, 3)$, $(2, 9)$ y $(8, 10)$

PROBLEMA DE LA SELECCIÓN DE TAREAS

Juan tiene n actividades que realizar y sabe cuándo empieza y cuándo termina cada una. Lamentablemente algunas se superponen y por lo tanto no puede realizarlas todas. El problema pide la máxima cantidad de actividades que Juan puede realizar sin que se le superpongan dos de ellas.

- → Por ejemplo si tenemos tres tareas de rangos $(1, 3)$, $(2, 9)$ y $(8, 10)$
- → ... la respuesta sería 2 tareas, la primera y la última.

PROBLEMA DE LA SELECCIÓN DE TAREAS

→ ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?

PROBLEMA DE LA SELECCIÓN DE TAREAS

→ ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?

→ ¿elegir la tarea que dure menos tiempo?

PROBLEMA DE LA SELECCIÓN DE TAREAS

- ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?
- ¿elegir la tarea que dure menos tiempo?
- ¿la tarea que empiece primero?

PROBLEMA DE LA SELECCIÓN DE TAREAS

- ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?
- ¿elegir la tarea que dure menos tiempo?
- ¿la tarea que empiece primero?
- No funcionan.

PROBLEMA DE LA SELECCIÓN DE TAREAS

- → ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?
- → ¿elegir la tarea que dure menos tiempo?
- → ¿la tarea que empiece primero?
- → No funcionan.
- → **Clave:** Escoger la tare que te deje el mayor tiempo posible para realizar las próximas.

PROBLEMA DE LA SELECCIÓN DE TAREAS

→ La forma correcta de ordenarlas es por horario de finalización

PROBLEMA DE LA SELECCIÓN DE TAREAS

→ La forma correcta de ordenarlas es por horario de finalización

→ Siempre que podamos realizar la próxima tarea la realizamos, sino la ignoramos.

PROBLEMA DE LA SELECCIÓN DE TAREAS

→ La forma correcta de ordenarlas es por horario de finalización

→ Siempre que podamos realizar la próxima tarea la realizamos, sino la ignoramos.

→ De esta forma, intuitivamente vamos realizando una a una las tareas con el objetivo de que nos sobre mayor tiempo para realizar las otras.

PROBLEMA DE LA SELECCIÓN DE TAREAS

- → La forma correcta de ordenarlas es por horario de finalización
- → Siempre que podamos realizar la próxima tarea la realizamos, sino la ignoramos.
- → De esta forma, intuitivamente vamos realizando una a una las tareas con el objetivo de que nos sobre mayor tiempo para realizar las otras.
- → ¿Funciona esto?

¿POR QUÉ ES CORRECTO ESTE ALGORITMO?

→ Supongamos que el algoritmo no es óptimo

¿POR QUÉ ES CORRECTO ESTE ALGORITMO?

- Supongamos que el algoritmo no es óptimo
- Con la selección de tareas que nosotros realizamos vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer desde que terminaron las primeras i actividades?

¿POR QUÉ ES CORRECTO ESTE ALGORITMO?

- Supongamos que el algoritmo no es óptimo
- Con la selección de tareas que nosotros realizamos vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer desde que terminaron las primeras i actividades?
- Supongamos que en ese subproblema, no hay solución eligiendo como primer tarea la que finaliza primero dentro de las posibles.

¿POR QUÉ ES CORRECTO ESTE ALGORITMO?

- Supongamos que el algoritmo no es óptimo
- Con la selección de tareas que nosotros realizamos vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer desde que terminaron las primeras i actividades?
- Supongamos que en ese subproblema, no hay solución eligiendo como primer tarea la que finaliza primero dentro de las posibles.
- Borremos la primer tarea elegida, y pongamos la que finaliza primero de las posibles. Todas las otras claramente van a poder realizarse.

¿POR QUÉ ES CORRECTO ESTE ALGORITMO?

→ Supongamos que el algoritmo no es óptimo

→ Con la selección de tareas que nosotros realizamos vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer desde que terminaron las primeras i actividades?

→ Supongamos que en ese subproblema, no hay solución eligiendo como primer tarea la que finaliza primero dentro de las posibles.

→ Borremos la primer tarea elegida, y pongamos la que finaliza primero de las posibles. Todas las otras claramente van a poder realizarse.

→ Por lo tanto hay una solución óptima que elije la primer tarea que finaliza. Contradicción.

¿POR QUÉ ES CORRECTO ESTE ALGORITMO?

- Supongamos que el algoritmo no es óptimo
- Con la selección de tareas que nosotros realizamos vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer desde que terminaron las primeras i actividades?
- Supongamos que en ese subproblema, no hay solución eligiendo como primer tarea la que finaliza primero dentro de las posibles.
- Borremos la primer tarea elegida, y pongamos la que finaliza primero de las posibles. Todas las otras claramente van a poder realizarse.
- Por lo tanto hay una solución óptima que elije la primer tarea que finaliza. Contradicción.
- Luego, el algoritmo es óptimo

TEOREMA DE NICO ALVAREZ

“Todos los problemas Greedies salen igual. Hay que ordenar ‘las tareas’ y después resolverlas en ese orden. Para ver en que orden se resuelven tenes que agarrar dos tareas y ver cual es la que greedymente se tiene que hacer primero”

TEOREMA DE NICO ALVAREZ

"Todos los problemas Greedies salen igual. Hay que ordenar 'las tareas' y después resolverlas en ese orden. Para ver en que orden se resuelven tenes que agarrar dos tareas y ver cual es la que greedymente se tiene que hacer primero"
Eso quiere decir que el código será simplemente:

- → Hacer una función de comparación entre 2 tareas
- → Ordenar el 'arreglo de tareas'
- → Hacer un `for`

La parte más difícil claramente es la función de comparacion

THE HERO

Dado un héroe llamado Foronda con sus puntos de vida inicial y dados los monstruos que Foronda tiene que matar, queremos saber si puede matarlos a todos sin quedarse en ningún momento sin energía.
Los monstruos se simbolizan con la vida c_i que le cuesta al héroe matar al i-ésimo monstruo. Además, cada monstruo cuida un cofre que contiene una poción, la cual Foronda sólo puede beber luego de matar al monstruo que la cuida y que le hace recuperar r_i puntos de vida al héroe. Foronda tiene vida máxima infinita.
Constraints: $N \leq 10^5$ Monstruos, $1 \leq Z \leq 10^5$ vida inicial, $c_i, r_i \leq 10^5$ naturales

SOLUCION

→ Por el teorema anterior hay que buscar una forma de ordenar los monstruos para saber cuál matar primero.

SOLUCION

- → Por el teorema anterior hay que buscar una forma de ordenar los monstruos para saber cuál matar primero.
- → Lo primero que hay que suponer es que podemos matar a todos y ver si llegamos a una contradicción.

SOLUCION

- Por el teorema anterior hay que buscar una forma de ordenar los monstruos para saber cuál matar primero.
- Lo primero que hay que suponer es que podemos matar a todos y ver si llegamos a una contradicción.
- Ahora... ¿En qué orden los matamos?

SOLUCION

- → Por el teorema anterior hay que buscar una forma de ordenar los monstruos para saber cuál matar primero.
- → Lo primero que hay que suponer es que podemos matar a todos y ver si llegamos a una contradicción.
- → Ahora... ¿En qué orden los matamos?
- → Empecemos matando a los monstruos buenos, los que te dan diferencia positiva de vida, es decir, los que la poción te da más vida que la que te saca el monstruo.

SOLUCION

- Por el teorema anterior hay que buscar una forma de ordenar los monstruos para saber cuál matar primero.
- Lo primero que hay que suponer es que podemos matar a todos y ver si llegamos a una contradicción.
- Ahora... ¿En qué orden los matamos?
- Empecemos matando a los monstruos buenos, los que te dan diferencia positiva de vida, es decir, los que la poción te da más vida que la que te saca el monstruo.
- Supongamos que no los podemos matar al principio, no los vamos a poder matar teniendo menos vida y ademas después vamos a tener más vida para los otros.

SOLUCION

- Pero… ¿En qué orden matamos a los monstruos buenos?
- No parece muy complejo, como cada vez vamos a tener mayor vida si antes podíamos matar a un monstruo bueno, nunca vamos a dejar de poder matarlo, por lo que una estrategia “matar al que podamos” va a funcionar.

SOLUCION

- → Pero... ¿En qué orden matamos a los monstruos buenos?
- → No parece muy complejo, como cada vez vamos a tener mayor vida si antes podíamos matar a un monstruo bueno, nunca vamos a dejar de poder matarlo, por lo que una estrategia “matar al que podamos” va a funcionar.
- → Si en algún momento queda algún monstruo bueno y no podemos matar a ningun otro bueno, nunca podremos matarlo.

SOLUCION

- Pero... ¿En qué orden matamos a los monstruos buenos?
- No parece muy complejo, como cada vez vamos a tener mayor vida si antes podíamos matar a un monstruo bueno, nunca vamos a dejar de poder matarlo, por lo que una estrategia “matar al que podamos” va a funcionar.
- Si en algún momento queda algún monstruo bueno y no podemos matar a ningun otro bueno, nunca podremos matarlo.
- Pensando un poquito más para simplificar el algoritmo, podemos matarlos en orden creciente de la vida c_i que nos cuesta matarlos.

SOLUCION

- → Pero... ¿En qué orden matamos a los monstruos buenos?
- → No parece muy complejo, como cada vez vamos a tener mayor vida si antes podíamos matar a un monstruo bueno, nunca vamos a dejar de poder matarlo, por lo que una estrategia "matar al que podamos" va a funcionar.
- → Si en algún momento queda algún monstruo bueno y no podemos matar a ningun otro bueno, nunca podremos matarlo.
- → Pensando un poquito más para simplificar el algoritmo, podemos matarlos en orden creciente de la vida c_i que nos cuesta matarlos.
- → Si en algún momento no podemos matar al monstruo bueno i-ésimo no podremos matar a ningún otro bueno ya que nunca podremos poseer más vida de la que tenemos.

SOLUCION

→ Nos quedan los malos.

SOLUCION

- → Nos quedan los malos.
- → ¿Podemos matarlos en el mismo orden que a los buenos?

SOLUCION

- → Nos quedan los malos.
- → ¿Podemos matarlos en el mismo orden que a los buenos?
- → Antes de programar esa idea, intentemos buscar un caso borde que nos destruya ese greedy...

SOLUCION

- Nos quedan los malos.
- ¿Podemos matarlos en el mismo orden que a los buenos?
- Antes de programar esa idea, intentemos buscar un caso borde que nos destruya ese greedy...
- Si un monstruo malo cuesta mucho pero te recupera casi la misma cantidad quizá convenga matarlo antes, ¿no?

SOLUCION

- Nos quedan los malos.
- ¿Podemos matarlos en el mismo orden que a los buenos?
- Antes de programar esa idea, intentemos buscar un caso borde que nos destruya ese greedy...
- Si un monstruo malo cuesta mucho pero te recupera casi la misma cantidad quizá convenga matarlo antes, ¿no?
- ```
  2 120
  100 99
  50 0
  ```
  ```

SOLUCION

- Nos quedan los malos.
- ¿Podemos matarlos en el mismo orden que a los buenos?
- Antes de programar esa idea, intentemos buscar un caso borde que nos destruya ese greedy...
- Si un monstruo malo cuesta mucho pero te recupera casi la misma cantidad quizá convenga matarlo antes, ¿no?
- ```
  2 120
  100 99
  50 0
  ```
- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!
  ```

SOLUCION

- Nos quedan los malos.
- ¿Podemos matarlos en el mismo orden que a los buenos?
- Antes de programar esa idea, intentemos buscar un caso borde que nos destruya ese greedy...
- Si un monstruo malo cuesta mucho pero te recupera casi la misma cantidad quizá convenga matarlo antes, ¿no?
- ```
  2 120
  100 99
  50 0
  ```
- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!
- ```
  2 X
  100 90
  50 40
  ```

SOLUCION

→ Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.

SOLUCION

- → Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- → Pensemos un caso para probarlo.

SOLUCION

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- Pensemos un caso para probarlo.
-

```
2 X           2 X
10000 20      10000 10
100 20        100 10
```

SOLUCION

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- Pensemos un caso para probarlo.
-
```
2 X          2 X
10000 20     10000 10
100 20       100 10
```
- ¿Como podemos estar seguros de que esto funciona?

GREEDY IS GOD

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.

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.

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.

MATROIDE

Es una estructura estructura que toma y generaliza el concepto de independencia lineal en los espacios vectoriales.

MATROIDE

Es una estructura estructura que toma y generaliza el concepto de independencia lineal en los espacios vectoriales.
Formalmente un matroide M es un par ordenado de elementos (E, I) donde E es un conjunto finito e I es un subconjunto del conjunto potencia de E que cumplen las siguientes propiedades

- $\emptyset \in I$
- Si $A \in I$ y $B \subseteq A$ entonces $B \in I$
- Si $A, B \in I$ y $|B| < |A|$ entonces existe $e \in A - B$ tal que $B \cup \{e\} \in I$

MATROIDE

Es una estructura estructura que toma y generaliza el concepto de independencia lineal en los espacios vectoriales.
Formalmente un matroide M es un par ordenado de elementos (E, I) donde E es un conjunto finito e I es un subconjunto del conjunto potencia de E que cumplen las siguientes propiedades

- $\emptyset \in I$
- Si $A \in I$ y $B \subseteq A$ entonces $B \in I$
- Si $A, B \in I$ y $|B| < |A|$ entonces existe $e \in A - B$ tal que $B \cup \{e\} \in I$

Es la manera formal de probar que algo puede ser resuelto por un algoritmo Greedy

"DEMOSTRACIÓN" DE UN ALGORITMO GREEDY

→ Para tener garantizado un “Accepted”, los Greedys hay que demostrarlos.

"DEMOSTRACIÓN" DE UN ALGORITMO GREEDY

- Para tener garantizado un “Accepted”, los Greedys hay que demostrarlos.
- Es raro que alguien utilice matroides en programacion competitiva.

"DEMOSTRACIÓN" DE UN ALGORITMO GREEDY

- Para tener garantizado un “Accepted”, los Greedys hay que demostrarlos.
- Es raro que alguien utilice matroides en programacion competitiva.
- Usualmente lo que se hace es trabajar con reduccion al absurdo.

"DEMOSTRACIÓN" DE UN ALGORITMO GREEDY

- Para tener garantizado un “Accepted”, los Greedys hay que demostrarlos.
- Es raro que alguien utilice matroides en programacion competitiva.
- Usualmente lo que se hace es trabajar con reduccion al absurdo.
- O.... se puede probar con casos de prueba inteligentes.

"DEMOSTRACIÓN" DE UN ALGORITMO GREEDY

- Para tener garantizado un “Accepted”, los Greedys hay que demostrarlos.
- Es raro que alguien utilice matroides en programacion competitiva.
- Usualmente lo que se hace es trabajar con reduccion al absurdo.
- O.... se puede probar con casos de prueba inteligentes.
- Casos de pruebas inteligentes son unos pocos casos chicos o bien pensados, donde poder analizar que ideas sirven.

"DEMOSTRACIÓN" DE UN ALGORITMO GREEDY

→ Para tener garantizado un “Accepted”, los Greedys hay que demostrarlos.

→ Es raro que alguien utilice matroides en programacion competitiva.

→ Usualmente lo que se hace es trabajar con reduccion al absurdo.

→ O.... se puede probar con casos de prueba inteligentes.

→ Casos de pruebas inteligentes son unos pocos casos chicos o bien pensados, donde poder analizar que ideas sirven.

→ Este enfoque es riesgoso, uno tiene que estar preparado para dejar un problema si no te da Accepted.

"DEMOSTRACIÓN" DE UN ALGORITMO GREEDY

- → Para tener garantizado un “Accepted”, los Greedys hay que demostrarlos.
- → Es raro que alguien utilice matroides en programacion competitiva.
- → Usualmente lo que se hace es trabajar con reduccion al absurdo.
- → O.... se puede probar con casos de prueba inteligentes.
- → Casos de pruebas inteligentes son unos pocos casos chicos o bien pensados, donde poder analizar que ideas sirven.
- → Este enfoque es riesgoso, uno tiene que estar preparado para dejar un problema si no te da Accepted.
- → Sin embargo suele ser muy efectivo.

OTROS USOS DE LOS CASOS DE PRUEBAS

→ Podemos usarlos antes de codear para garantizar que no estemos programando cualquier cosa.

OTROS USOS DE LOS CASOS DE PRUEBAS

- → Podemos usarlos antes de codear para garantizar que no estemos programando cualquier cosa.
- → Podemos usarlos para comprobar y buscar algoritmos.

OTROS USOS DE LOS CASOS DE PRUEBAS

- → Podemos usarlos antes de codear para garantizar que no estemos programando cualquier cosa.
- → Podemos usarlos para comprobar y buscar algoritmos.
- → Y podemos usarlos para entender los problemas (poco comun en greedy, pero muy importante).

OTROS USOS DE LOS CASOS DE PRUEBAS

→ Podemos usarlos antes de codear para garantizar que no estemos programando cualquier cosa.

→ Podemos usarlos para comprobar y buscar algoritmos.

→ Y podemos usarlos para entender los problemas (poco comun en greedy, pero muy importante).

RESUMEN

- La mejor forma de aprender a resolver Greedys es realizando problemas.
- Para demostrar que andan, se procede por el absurdo. Se supone que el greedy no es óptimo y se llega a una contradicción
- Otra forma es probar casos extremos, y confiar en que funciona
- Como los greedys son muy simples de codear, lo mandamos y probamos. Si pasan estamos melos, sino lo volvemos a pensar.
- Para encontrar un greedy en un ejercicio es fundamental probar como resolverías el ejercicio para casos extremos simples.

PROXIMA SESION

CONTEST

https://vjudge.net/contest/284280

PROXIMA SEMANA

Binary search