Pensando en formato Greedy

Pablo Zimmermann

Universidad Nacional de Rosario

11th Caribbean Camp

Contenidos

1. Algoritmos Greedy
   - Qué son los algoritmos greedy
2. Ejemplos
   - Activity Selection Problem
   - The Hero
   - The Agency
   - Demostración de un algoritmo Greedy
   - Woodworms
3. Resumen
4. Links a Ejercicios

Contenidos

1. Algoritmos Greedy
   - Qué son los algoritmos greedy
2. Ejemplos
   - Activity Selection Problem
   - The Hero
   - The Agency
   - Demostración de un algoritmo Greedy
   - Woodworms
3. Resumen
4. Links a Ejercicios

Definición de algoritmo greedy

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

Definición de algoritmo greedy

- 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.

Definición de algoritmo greedy

- 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”.

Ejemplos

- En el algoritmo de Kruskal (MST), elegimos en cada paso la arista más liviana que no genera un ciclo.

Ejemplos

- En el algoritmo de Kruskal (MST), elegimos en cada paso la arista más liviana que no genera un ciclo.
- Para calcular el máximo Matching en un árbol, elegimos en cada paso cualquier arista a la que pertenezca una hoja.

Ejemplos

- En el algoritmo de Kruskal (MST), elegimos en cada paso la arista más liviana que no genera un ciclo.
- Para calcular el máximo Matching en un árbol, elegimos en cada paso cualquier arista a la que pertenezca una hoja.
- Los greedys conocidos ya sabemos que andan, no tenemos que demostrarlos cada vez que los usamos.

Ejemplos

- En el algoritmo de Kruskal (MST), elegimos en cada paso la arista más liviana que no genera un ciclo.
- Para calcular el máximo Matching en un árbol, elegimos en cada paso cualquier arista a la que pertenezca una hoja.
- Los greedys conocidos ya sabemos que andan, no tenemos que demostrarlos cada vez que los usamos.
- **Demostrar que un greedy funciona no es una tarea simple**, generalmente requiere una demostración formal.

Ejemplos

- En el algoritmo de Kruskal (MST), elegimos en cada paso la arista más liviana que no genera un ciclo.
- Para calcular el máximo Matching en un árbol, elegimos en cada paso cualquier arista a la que pertenezca una hoja.
- Los greedys conocidos ya sabemos que andan, no tenemos que demostrarlos cada vez que los usamos.
- **Demostrar que un greedy funciona no es una tarea simple**, generalmente requiere una demostración formal.
- Veámoslo con un ejemplo conocido...

¿Para dónde tiene que patear Marco Ruben?

Contenidos

1. Algoritmos Greedy
   - Qué son los algoritmos greedy
2. Ejemplos
   - Activity Selection Problem
   - The Hero
   - The Agency
   - Demostración de un algoritmo Greedy
   - Woodworms
3. Resumen
4. Links a Ejercicios

Enunciado

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.

Enunciado

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)...

Enunciado

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

Posibles soluciones

- Creando un estado para los subproblemas que simbolize **(número de tarea, inicio de rango, final de rango)**, se puede realizar una dinámica en O(n^3):
  - Con una tarea (l, r) y $a \leq l, r \leq b$:
    - dp(i, a, b) = max(dp(i+1,a,b), dp(i+1,a,l)+ dp(i+1, r, b)+1

Posibles soluciones

- Creando un estado para los subproblemas que simbolize **(número de tarea, inicio de rango, final de rango)**, se puede realizar una dinámica en O(n^3):
  - Con una tarea (l, r) y $a \leq l, r \leq b$:
    - dp(i, a, b) = max(dp(i+1,a,b), dp(i+1,a,l)+ dp(i+1, r, b)+1
- Sin embargo, esto puede ser muy lento para las cotas de nuestro problema.

Posibles soluciones

- Creando un estado para los subproblemas que simbolize **(número de tarea, inicio de rango, final de rango)**, se puede realizar una dinámica en $O(n^3)$:
  - Con una tarea (l, r) y $a \leq l, r \leq b$:
    - dp(i, a, b) = max(dp(i+1,a,b), dp(i+1,a,l)+ dp(i+1, r, b)+1
- Sin embargo, esto puede ser muy lento para las cotas de nuestro problema.
- Pensemos en otro sentido... ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?

Posibles soluciones

- Creando un estado para los subproblemas que simbolize **(número de tarea, inicio de rango, final de rango)**, se puede realizar una dinámica en $O(n^3)$:
    - Con una tarea (l, r) y $a \leq l, r \leq b$:
        - dp(i, a, b) = max(dp(i+1,a,b), dp(i+1,a,l)+ dp(i+1, r, b)+1
- Sin embargo, esto puede ser muy lento para las cotas de nuestro problema.
- Pensemos en otro sentido... ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?
- ¿elegir la tarea que dure menos tiempo? ¿la tarea que empiece primero?

Posibles soluciones

- Creando un estado para los subproblemas que simbolize **(número de tarea, inicio de rango, final de rango)**, se puede realizar una dinámica en O(n^3):
  - Con una tarea (l, r) y $a \leq l, r \leq b$:
    - dp(i, a, b) = max(dp(i+1,a,b), dp(i+1,a,l)+ dp(i+1, r, b)+1
- Sin embargo, esto puede ser muy lento para las cotas de nuestro problema.
- Pensemos en otro sentido... ¿Hay alguna forma de decidir rápidamente qué tarea hacer primero?
- ¿elegir la tarea que dure menos tiempo? ¿la tarea que empiece primero? No funcionan...

Posibles soluciones

- Creando un estado para los subproblemas que simbolize **(número de tarea, inicio de rango, final de rango)**, se puede realizar una dinámica en $O(n^3)$:
  - Con una tarea (l, r) y $a \leq l, r \leq b$:
    - dp(i, a, b) = max(dp(i+1,a,b), dp(i+1,a,l)+ dp(i+1, r, b)+1
- Sin embargo, esto puede ser muy lento para las cotas de nuestro problema.
- Pensemos en otro sentido... ¿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: De las tareas elegidas en la solución que sea, una empieza primero

Posibles soluciones

- Creando un estado para los subproblemas que simbolize **(número de tarea, inicio de rango, final de rango)**, se puede realizar una dinámica en O(n^3):
  - Con una tarea (l, r) y $a \leq l, r \leq b$:
    - dp(i, a, b) = max(dp(i+1,a,b), dp(i+1,a,l)+ dp(i+1, r, b)+1
- Sin embargo, esto puede ser muy lento para las cotas de nuestro problema.
- Pensemos en otro sentido... ¿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: De las tareas elegidas en la solución que sea, una empieza primero
- Intuitivamente, uno quiere realizar esa tarea y que te quede el mayor tiempo posible para realizar las próximas

Solución que utilizaremos

- La forma correcta de ordenarlas es por horario de finalización.

Solución que utilizaremos

- La forma correcta de ordenarlas es por horario de finalización.
- Siempre que podamos realizar la próxima tarea la realizamos, sino la ignoramos.

Solución que utilizaremos

- 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.

Solución que utilizaremos

- 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? ¿Cuál es la próxima tarea a hacer?

¿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? ¿Cuál es la próxima tarea a hacer?
- 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? ¿Cuál es la próxima tarea a hacer?
- 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? ¿Cuál es la próxima tarea a hacer?
- 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? ¿Cuál es la próxima tarea a hacer?
- 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.
- El algoritmo es óptimo.

Contenidos

1 Algoritmos Greedy
- Qué son los algoritmos greedy

2 Ejemplos
- Activity Selection Problem
- The Hero
- The Agency
- Demostración de un algoritmo Greedy
- Woodworms

3 Resumen

4 Links a Ejercicios

Enunciado

Problema

Dado un héroe llamado *Leopoldus* con sus puntos de vida inicial y dados los monstruos que *Leopoldus* 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 *Leopoldus* sólo puede beber luego de matar al monstruo que la cuida y que le hace recuperar r_i puntos de vida al héroe. Leopoldus 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

Teorema de Nico Alvarez

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

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 cmp (de comparación entre 2 tareas)
- Ordenar el “arreglo de tareas”
- Hacer un for

La parte más difícil claramente es la función cmp

Solución

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

Solución

- 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.

Solución

- 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?

Solución

- 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.

Solución

- 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.
- Se puede demostrar que estos van primero, por el absurdo. Supongamos que no los podemos matar al principio, no los vamos a poder matar teniendo menor vida y después vamos a tener más vida para los otros (No esperen algo más formal que esto en lo que quede de la clase).

Solución

- Pero... ¿En qué orden matamos a los monstruos buenos?

Solución

- 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.

Solución

- 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 ninguno, nunca podremos matarlo.

Solución

- 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 ninguno, 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.

Solución

- 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 ninguno, 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 y nunca podremos poseer más vida de la que ya tenemos $\Rightarrow$ **No podemos matar a todos**

Solución

- Nos quedan los monstruos malos.

Solución

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

Solución

- Nos quedan los monstruos 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...

Solución

- Nos quedan los monstruos 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?

Solución

- Nos quedan los monstruos 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
  ```
  ```

Solución

- Nos quedan los monstruos 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
  ```
- Nuestro algoritmo mata primero al que nos cuesta 50 de vida, sin dejarnos vida suficiente para matar al primero. Matándolos al revés podríamos matarlos.
  ```

Solución

- ¿Otra idea?

Solución

- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!

Solución

- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!

Solución

- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!
- Mejor frenemos un cacho, generemos un caso de prueba con 2 monstruos y analicemos como deberíamos resolverlo (nuevamente teorema de Nico Alvarez).

Solución

- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!
- Mejor frenemos un cacho, generemos un caso de prueba con 2 monstruos y analicemos como deberíamos resolverlo (nuevamente teorema de Nico Alvarez).
- ```
  2 X
  100 90
  50 40
  ```
  ```

Solución

- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!
- Mejor frenemos un cacho, generemos un caso de prueba con 2 monstruos y analicemos como deberíamos resolverlo (nuevamente teorema de Nico Alvarez).
- ```
  2 X
  100 90
  50 40
  ```
  ¿A cuál de estos monstruos deberíamos matar primero? ¿Es lo mismo?
  ```

Solución

- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!
- Mejor frenemos un cacho, generemos un caso de prueba con 2 monstruos y analicemos como deberíamos resolverlo (nuevamente teorema de Nico Alvarez).
- ```
  2 X
  100 90
  50 40
  ```
  ¿A cuál de estos monstruos deberíamos matar primero? ¿Es lo mismo?
- Si pudiéramos tomar la poción antes, sería mucho más simple. (Pensarlo)
  ```

Solución

- ¿Otra idea?
- ¡Matemos al que te cueste menor diferencia de vida!
- ¡Matemos al que te saque mayor vida!
- Mejor frenemos un cacho, generemos un caso de prueba con 2 monstruos y analicemos como deberíamos resolverlo (nuevamente teorema de Nico Alvarez).
- ```
  2 X
  100 90
  50 40
  ```
  ¿A cuál de estos monstruos deberíamos matar primero? ¿Es lo mismo?
- Si pudiéramos tomar la poción antes, sería mucho más simple. (Pensarlo)
- El problema es que la última poción la estamos desperdiciando, ¿y entonces qué hacemos?
  ```

Solución

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

Solución

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- De esta forma, usamos las mejores pociones al principio, para poder luchar contra la mayor cantidad de monstruos restantes!

Solución

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- De esta forma, usamos las mejores pociones al principio, para poder luchar contra la mayor cantidad de monstruos restantes!
- ¡Demostrémoslo!

Solución

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- De esta forma, usamos las mejores pociones al principio, para poder luchar contra la mayor cantidad de monstruos restantes!
- ¡Demostrémoslo! Nah, mejor pensemos un caso para probarlo.

Solución

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- De esta forma, usamos las mejores pociones al principio, para poder luchar contra la mayor cantidad de monstruos restantes!
- ¡Demostrémoslo! Nah, mejor pensemos un caso para probarlo.
- ```
  2 X
  10000 10
  100 20
  ```
  ```
  2 X
  10000 10
  100 10
  ```
  ```

Solución

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- De esta forma, usamos las mejores pociones al principio, para poder luchar contra la mayor cantidad de monstruos restantes!
- ¡Demostrémoslo! Nah, mejor pensemos un caso para probarlo.
-
```
2 X                2 X
10000 10         10000 10
100 20           100 10
```
- ¿¿Por qué estos casos??

Solución

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- De esta forma, usamos las mejores pociones al principio, para poder luchar contra la mayor cantidad de monstruos restantes!
- ¡Demostrémoslo! Nah, mejor pensemos un caso para probarlo.
-
```
2 X            2 X
10000 10       10000 10
100 20         100 10
```
- ¿¿Por qué estos casos?? Porque queremos ver si realmente lo único que importa es la poción que nos dan.

Solución

- Matemos los monstruos malos en orden decreciente de lo que nos recuperan r_i.
- De esta forma, usamos las mejores pociones al principio, para poder luchar contra la mayor cantidad de monstruos restantes!
- ¡Demostrémoslo! Nah, mejor pensemos un caso para probarlo.
- ```
  2 X              2 X
  10000 10       10000 10
  100 20         100 10
  ```
- ¿¿Por qué estos casos?? Porque queremos ver si realmente lo único que importa es la poción que nos dan.
- Probando todos los órdenes, queda claro que conviene arrancar con el monstruo de mayor poción para minimizar la mínima vida necesaria que debe tener *Leopoldus* al final.
  ```

Solución

- También se puede demostrar.

Solución

- También se puede demostrar.
- La demostración es similar a la de todos los greedys. Agarremos dos monstruos malos, supongamos que los matamos en orden creciente de las pociones y veamos que también los podemos matar al revés.
- Con cualquier solución podemos ir swicheando los pares de monstruos malos que estén en orden inverso de pociones.

Solución

- También se puede demostrar.
- La demostración es similar a la de todos los greedys. Agarremos dos monstruos malos, supongamos que los matamos en orden creciente de las pociones y veamos que también los podemos matar al revés.
- Con cualquier solución podemos ir switcheando los pares de monstruos malos que estén en orden inverso de pociones.
- Queda como ejercicio la demostración

Contenidos

1 Algoritmos Greedy
- Qué son los algoritmos greedy

2 Ejemplos
- Activity Selection Problem
- The Hero
- The Agency
- Demostración de un algoritmo Greedy
- Woodworms

3 Resumen

4 Links a Ejercicios

Enunciado

Problema

En una galaxia lejana, cada planeta se representa como una secuencia de 0's y 1's y a cada secuencia le corresponde un planeta. Viajar de un planeta a otro sólo es posible si difieren en 1 bit y el único costo está dado por aterrizar en el planeta entrante. Cada bit tiene una tasa fija, y el costo de ir a un planeta es la suma de los costos de los bits que ese planeta destino tiene en 1.

Input/Output

Dado $N \leq 10^3$ Cantidad de bits en la galaxia, S y E dos strings que simbolizan los códigos de los planetas, y $a_i \leq 10^6$ costos de cada bit, **se pide que responder el costo mínimo de viajar entre S y E.**

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

Rta : 35

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

Rta : 35

Case 2

```
5 11111 00000
1 2 3 4 5
```

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

$Rta : 35$

Case 2

```
5 11111 00000
1 2 3 4 5
```

$Rta : 20$

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

Rta : 35

Case 2

```
5 11111 00000
1 2 3 4 5
```

Rta : 20

Case 3

```
3 110 011
3 1 2
```

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

Rta : 35

Case 2

```
5 11111 00000
1 2 3 4 5
```

Rta : 20

Case 3

```
3 110 011
3 1 2
```

Rta : 4

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

Rta : 35

Case 2

```
5 11111 00000
1 2 3 4 5
```

Rta : 20

Case 3

```
3 110 011
3 1 2
```

Rta : 4

Case 4

```
4 1111 1000
100 1 1 1
```

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

$Rta : 35$

Case 2

```
5 11111 00000
1 2 3 4 5
```

$Rta : 20$

Case 3

```
3 110 011
3 1 2
```

$Rta : 4$

Case 4

```
4 1111 1000
100 1 1 1
```

$Rta : 106 \rightarrow Hack!$

Casos de Ejemplo

Case 1

```
5 00000 11111
1 2 3 4 5
```

$Rta: 35$

Case 2

```
5 11111 00000
1 2 3 4 5
```

$Rta: 20$

Case 3

```
3 110 011
3 1 2
```

$Rta: 4$

Case 4

```
4 1111 1000
100 1 1 1
```

$Rta: 106 \rightarrow Hack!$

Solución

- El problema se puede traducir a que hay que ir prendiendo y apagando bits para llegar de un inicio a un fin.

Solución

- El problema se puede traducir a que hay que ir prendiendo y apagando bits para llegar de un inicio a un fin.
- El primer ejemplo nos muestra que para pagar menor costo claramente conviene prender los bits que necesitamos de menor a mayor.

Solución

- El problema se puede traducir a que hay que ir prendiendo y apagando bits para llegar de un inicio a un fin.
- El primer ejemplo nos muestra que para pagar menor costo claramente conviene prender los bits que necesitamos de menor a mayor.
- El segundo ejemplo nos muestra que para pagar menor costo conviene apagar los bits que necesitamos de mayor a menor.

Solución

- El problema se puede traducir a que hay que ir prendiendo y apagando bits para llegar de un inicio a un fin.
- El primer ejemplo nos muestra que para pagar menor costo claramente conviene prender los bits que necesitamos de menor a mayor.
- El segundo ejemplo nos muestra que para pagar menor costo conviene apagar los bits que necesitamos de mayor a menor.
- El tercer ejemplo nos muestra que para pagar menor costo conviene apagar primero antes de prender los que necesitemos.

Solución

- El problema se puede traducir a que hay que ir prendiendo y apagando bits para llegar de un inicio a un fin.
- El primer ejemplo nos muestra que para pagar menor costo claramente conviene prender los bits que necesitamos de menor a mayor.
- El segundo ejemplo nos muestra que para pagar menor costo conviene apagar los bits que necesitamos de mayor a menor.
- El tercer ejemplo nos muestra que para pagar menor costo conviene apagar primero antes de prender los que necesitemos.
- ¡El cuarto ejemplo nos rompe todo! Algo de todo los 3 puntos anteriores debe estar mal, ¿no?

Solución

- El problema se puede traducir a que hay que ir prendiendo y apagando bits para llegar de un inicio a un fin.
- El primer ejemplo nos muestra que para pagar menor costo claramente conviene prender los bits que necesitamos de menor a mayor.
- El segundo ejemplo nos muestra que para pagar menor costo conviene apagar los bits que necesitamos de mayor a menor.
- El tercer ejemplo nos muestra que para pagar menor costo conviene apagar primero antes de prender los que necesitemos.
- ¡El cuarto ejemplo nos rompe todo! Algo de todo los 3 puntos anteriores debe estar mal, ¿no?
- ¡No! Los tres puntos parecen bastante claros. ¿Qué nos estamos olvidando?

Solución

- El problema se puede traducir a que hay que ir prendiendo y apagando bits para llegar de un inicio a un fin.
- El primer ejemplo nos muestra que para pagar menor costo claramente conviene prender los bits que necesitamos de menor a mayor.
- El segundo ejemplo nos muestra que para pagar menor costo conviene apagar los bits que necesitamos de mayor a menor.
- El tercer ejemplo nos muestra que para pagar menor costo conviene apagar primero antes de prender los que necesitemos.
- ¡El cuarto ejemplo nos rompe todo! Algo de todo los 3 puntos anteriores debe estar mal, ¿no?
- ¡No! Los tres puntos parecen bastante claros. ¿Qué nos estamos olvidando?
- Que aunque parezca anti-intuitivo, puede convenir apagar bits que tanto en el inicio como en el final estén prendidos.

Solución

- ¿Qué hacemos? Podemos intentar pensarlo con dinámica...

Solución

- ¿Qué hacemos? Podemos intentar pensarlo con dinámica...
- ¡No! Tenemos muchas cosas greedy para abandonarlo, solo queda solucionar el problema de apagar bits que al principio y al final están prendidos. ¿¿Se puede saber cuándo conviene hacer esto??

Solución

- ¿Qué hacemos? Podemos intentar pensarlo con dinámica...
- ¡No! Tenemos muchas cosas greedy para abandonarlo, solo queda solucionar el problema de apagar bits que al principio y al final están prendidos. ¿¿Se puede saber cuándo conviene hacer esto??
- No, pero como tenemos $N \leq 10^3$ podemos probar todas las posibilidades en $O(N^2)$.

Solución

- ¿Qué hacemos? Podemos intentar pensarlo con dinámica...
- ¡No! Tenemos muchas cosas greedy para abandonarlo, solo queda solucionar el problema de apagar bits que al principio y al final están prendidos. ¿¿Se puede saber cuándo conviene hacer esto??
- No, pero como tenemos $N \leq 10^3$ podemos probar todas las posibilidades en $O(N^2)$.
- Para cada cantidad i de bits prendidos inicio-fin que vamos a apagar y prender deberíamos elegir los i más grandes (para ahorrar el costo).

Solución

- ¿Qué hacemos? Podemos intentar pensarlo con dinámica...
- ¡No! Tenemos muchas cosas greedy para abandonarlo, solo queda solucionar el problema de apagar bits que al principio y al final están prendidos. ¿¿Se puede saber cuándo conviene hacer esto??
- No, pero como tenemos $N \leq 10^3$ podemos probar todas las posibilidades en $O(N^2)$.
- Para cada cantidad i de bits prendidos inicio-fin que vamos a apagar y prender deberíamos elegir los i más grandes (para ahorrar el costo).
- Cuando los apagamos seguimos el orden de los otros que tenemos que apagar, y cuando los prendemos el orden de los que tenemos que prender.

Solución

- ¿Qué hacemos? Podemos intentar pensarlo con dinámica...
- ¡No! Tenemos muchas cosas greedy para abandonarlo, solo queda solucionar el problema de apagar bits que al principio y al final están prendidos. ¿¿Se puede saber cuándo conviene hacer esto??
- No, pero como tenemos $N \leq 10^3$ podemos probar todas las posibilidades en $O(N^2)$.
- Para cada cantidad i de bits prendidos inicio-fin que vamos a apagar y prender deberíamos elegir los i más grandes (para ahorrar el costo).
- Cuando los apagamos seguimos el orden de los otros que tenemos que apagar, y cuando los prendemos el orden de los que tenemos que prender.
- Y listo, $O(N^2)$

Solución

- ¿Qué hacemos? Podemos intentar pensarlo con dinámica...
- ¡No! Tenemos muchas cosas greedy para abandonarlo, solo queda solucionar el problema de apagar bits que al principio y al final están prendidos. ¿¿Se puede saber cuándo conviene hacer esto??
- No, pero como tenemos $N \leq 10^3$ podemos probar todas las posibilidades en $O(N^2)$.
- Para cada cantidad i de bits prendidos inicio-fin que vamos a apagar y prender deberíamos elegir los i más grandes (para ahorrar el costo).
- Cuando los apagamos seguimos el orden de los otros que tenemos que apagar, y cuando los prendemos el orden de los que tenemos que prender.
- Y listo, $O(N^2)$
- Ejercicio: Formalizar la demostración de porque funciona esto.

Contenidos

1. Algoritmos Greedy
   - Qué son los algoritmos greedy
2. Ejemplos
   - Activity Selection Problem
   - The Hero
   - The Agency
   - Demostración de un algoritmo Greedy
   - Woodworms
3. Resumen
4. Links a Ejercicios

Demostración de un algoritmo Greedy

- Acabamos de ver tres problemas que se solucionan con “ideas greedy”

Demostración de un algoritmo Greedy

- Acabamos de ver tres problemas que se solucionan con “ideas greedy”
- Para tener garantizado un “Accepted”, los *Greedys* hay que demostrarlos.

Demostración de un algoritmo Greedy

- Acabamos de ver tres problemas que se solucionan con “ideas greedy”
- Para tener garantizado un “Accepted”, los *Greedys* hay que demostrarlos.
- Demostrar greedys es siempre muy similar. Suponés que no seguís esa estrategia localmente, y ves que siguiendo esa estrategia también llegaríamos a un algoritmo óptimo.

Teorema de la Fruta Golosa

- O... podemos probarlos con casos de prueba inteligentes.

Teorema de la Fruta Golosa

- O... podemos probarlos con casos de prueba inteligentes.
- Casos de pruebas inteligentes son unos pocos casos chicos o bien pensados, donde poder analizar que ideas sirven.

Teorema de la Fruta Golosa

- O... podemos probarlos con casos de prueba inteligentes.
- Casos de pruebas inteligentes son unos pocos casos chicos o bien pensados, donde poder analizar que ideas sirven.
- Si el algoritmo pasa los casos, lo programamos rápido y lo mandamos.

Teorema de la Fruta Golosa

- O... podemos probarlos con casos de prueba inteligentes.
- Casos de pruebas inteligentes son unos pocos casos chicos o bien pensados, donde poder analizar que ideas sirven.
- Si el algoritmo pasa los casos, lo programamos rápido y lo mandamos.
- Este enfoque es riesgoso, uno tiene que estar preparado para **dejar** un problema si no te da *Accepted*.

Teorema de la Fruta Golosa

- O... podemos probarlos con casos de prueba inteligentes.
- Casos de pruebas inteligentes son unos pocos casos chicos o bien pensados, donde poder analizar que ideas sirven.
- Si el algoritmo pasa los casos, lo programamos rápido y lo mandamos.
- Este enfoque es riesgoso, uno tiene que estar preparado para **dejar** un problema si no te da *Accepted*.
- Sin embargo es muy efectivo, y yo lo sigo sin dudar.
- Como un ejemplo, el problema de la mochila “puede tener pinta” pero no sale con greedy

Otros usos de los casos de pruebas

- Generar casos de pruebas tienen otras utilidades.

Otros usos de los casos de pruebas

- Generar casos de pruebas tienen otras utilidades.
- Podemos usarlos antes de codear para garantizar que no estemos programando cualquier cosa.

Otros usos de los casos de pruebas

- Generar casos de pruebas tienen otras utilidades.
- 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

- Generar casos de pruebas tienen otras utilidades.
- 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.

Otros usos de los casos de pruebas

- Generar casos de pruebas tienen otras utilidades.
- 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.
- Esta última, según mi opinión, es el uso menos dado y uno de los más importantes, sobre todo al realizar algoritmos greedy.

Otros usos de los casos de pruebas

- Generar casos de pruebas tienen otras utilidades.
- 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.
- Esta última, según mi opinión, es el uso menos dado y uno de los más importantes, sobre todo al realizar algoritmos greedy.
- Veamos un último ejemplo de este uso.

Contenidos

1. Algoritmos Greedy
   - Qué son los algoritmos greedy
2. Ejemplos
   - Activity Selection Problem
   - The Hero
   - The Agency
   - Demostración de un algoritmo Greedy
   - Woodworms
3. Resumen
4. Links a Ejercicios

Enunciado

Problema

Fidel y Agustín juegan a un juego. Dada una lista de $N \leq 10^6$ números, tal que $1 \leq a_i \leq 10^9$ cada jugador puede retirar de la lista el número de la izquierda, el número de la derecha o ambos. El juego termina cuando no quedan más números. El objetivo de ambos jugadores es obtener la mayor suma posible. Calcular la mayor suma que puede asegurarse obtener Fidel, sin importar lo bien Agustín.

Unico caso de ejemplo dado

```
4
5 2 9 3
Rta: 14
```

Interpretando el enunciado

- Ideas???

Interpretando el enunciado

- Ideas???
- Analicemos algunos ejemplos.

Interpretando el enunciado

- Ideas???
- Analicemos algunos ejemplos.
- Cómo generamos ejemplos?

Interpretando el enunciado

- Ideas???
- Analicemos algunos ejemplos.
- Cómo generamos ejemplos?
- Busquemos los más simples posible.

Interpretando el enunciado

- Ideas???
- Analicemos algunos ejemplos.
- Cómo generamos ejemplos?
- Busquemos los más simples posible.
- Definimos p_1 puntaje óptimo de Fidel y p_2 puntaje óptimo de Agustín.

Interpretando el enunciado

- Ideas???
- Analicemos algunos ejemplos.
- Cómo generamos ejemplos?
- Busquemos los más simples posible.
- Definimos p_1 puntaje óptimo de Fidel y p_2 puntaje óptimo de Agustín.
- Aclaración: En este ejercicio voy a intentar mostrar de que forma creo que hay que pensar los "greedys". No se esperen algo formal.

N = 1

- N = 1
  a_1

N = 1

- N = 1
  a_1
- Fidel solo puede llevarse a_1, Agustín no se lleva nada.
  $p_1 = a_1$
  $p_2 = 0$

N = 2

- N = 2

  $a_1 \, a_2$

N = 2

- N = 2
  $a_1 a_2$
- Trivialmente a Fidel le conviene llevarse ambos, al ser positivos
  $p_1 = a_1 + a_2$
  $p_2 = 0.$

N = 3

- N = 3

  $a_1 a_2 a_3$

N = 3

- N = 3
  $a_1 a_2 a_3$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_3$.

N = 3

- N = 3
  $a_1 a_2 a_3$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_3$.
- Si Fidel se lleva solo a_3, por lo dicho antes p_2 será $a_1 + a_2$.

N = 3

- N = 3
  $a_1 a_2 a_3$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_3$.
- Si Fidel se lleva solo a_3, por lo dicho antes p_2 será $a_1 + a_2$.
- Si Fidel se lleva a_1 y a_3, por lo dicho antes p_2 será a_2.

N = 3

- N = 3
  $a_1 a_2 a_3$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_3$.
- Si Fidel se lleva solo a_3, por lo dicho antes p_2 será $a_1 + a_2$.
- Si Fidel se lleva a_1 y a_3, por lo dicho antes p_2 será a_2.
- Trivialmente a Fidel le conviene llevarse ambos a_1 y a_3, al ser positivos,para minimizar el puntaje de Agustín.
  $p_1 = a_1 + a_3$
  $p_2 = a_2$.

N = 4

- N = 4

  $a_1 a_2 a_3 a_4$

N = 4

- N = 4
  $a_1 a_2 a_3 a_4$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_4$.

N = 4

- N = 4
  $a_1 a_2 a_3 a_4$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_4$.
- Si Fidel se lleva solo a_4, por lo dicho antes p_2 será $a_1 + a_3$.

N = 4

- N = 4
  $a_1 a_2 a_3 a_4$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_4$.
- Si Fidel se lleva solo a_4, por lo dicho antes p_2 será $a_1 + a_3$.
- Si Fidel se lleva a_1 y a_4, por lo dicho antes p_2 será $a_2 + a_3$.

N = 4

- N = 4
  $a_1 a_2 a_3 a_4$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_4$.
- Si Fidel se lleva solo a_4, por lo dicho antes p_2 será $a_1 + a_3$.
- Si Fidel se lleva a_1 y a_4, por lo dicho antes p_2 será $a_2 + a_3$.
- $p_1 = max(a_1 + a_3, a_1 + a_4, a_2 + a_4)$

N = 4

- N = 4
  $a_1 a_2 a_3 a_4$
- Si Fidel se lleva solo a_1, por lo dicho antes p_2 será $a_2 + a_4$.
- Si Fidel se lleva solo a_4, por lo dicho antes p_2 será $a_1 + a_3$.
- Si Fidel se lleva a_1 y a_4, por lo dicho antes p_2 será $a_2 + a_3$.
- $p_1 = max(a_1 + a_3, a_1 + a_4, a_2 + a_4)$
- Realmente esto no me da mucha información.

N = 5

- N = 5
  $a_1 a_2 a_3 a_4 a_5$

N = 5

- N = 5
  $a_1\, a_2\, a_3\, a_4\, a_5$
- Si Fidel se lleva a_1 y a_5 , por lo dicho antes $p_2 = a_2 + a_4$ =>
  $p_1 = a_1 + a_3 + a_5$.

N = 5

- N = 5
  $a_1\, a_2\, a_3\, a_4\, a_5$
- Si Fidel se lleva a_1 y a_5 , por lo dicho antes $p_2 = a_2 + a_4$ => $p_1 = a_1 + a_3 + a_5$.
- Si Fidel se lleva solo a_1, por lo dicho antes $p_2 = max(a_2 + a_4, a_2 + a_5, a_3 + a_5)$.

N = 5

- N = 5
  $a_1 a_2 a_3 a_4 a_5$
- Si Fidel se lleva a_1 y a_5 , por lo dicho antes $p_2 = a_2 + a_4$ =>
  $p_1 = a_1 + a_3 + a_5$.
- Si Fidel se lleva solo a_1, por lo dicho antes
  $p_2 = max(a_2 + a_4, a_2 + a_5, a_3 + a_5)$.
- => $p_2 \geq a_2 + a_4$.

N = 5

- N = 5
  $a_1 a_2 a_3 a_4 a_5$
- Si Fidel se lleva a_1 y a_5 , por lo dicho antes $p_2 = a_2 + a_4$ => $p_1 = a_1 + a_3 + a_5$.
- Si Fidel se lleva solo a_1, por lo dicho antes $p_2 = max(a_2 + a_4, a_2 + a_5, a_3 + a_5)$.
- => $p_2 \geq a_2 + a_4$.
- Si Fidel se lleva a_5, podemos obtener con un razonamiento análogo que $p_2 \geq a_2 + a_4$.

N = 5

- N = 5
  $a_1 a_2 a_3 a_4 a_5$
- Si Fidel se lleva a_1 y a_5 , por lo dicho antes $p_2 = a_2 + a_4$ => $p_1 = a_1 + a_3 + a_5$.
- Si Fidel se lleva solo a_1, por lo dicho antes $p_2 = max(a_2 + a_4, a_2 + a_5, a_3 + a_5)$.
- => $p_2 \geq a_2 + a_4$.
- Si Fidel se lleva a_5, podemos obtener con un razonamiento análogo que $p_2 \geq a_2 + a_4$.
- Acá tenemos un greedy! A Fidel le conviene sacar a_1 y a_5 y obligar a que $p_2 = a_2 + a_4$.

Análisis de Casos

Informalmente tenemos que:

- N = 1 => $p_1 = a_1$

Análisis de Casos

Informalmente tenemos que:

- N = 1 => $p_1 = a_1$
- N = 2 => $p_1 = a_1 + a_2$

Análisis de Casos

Informalmente tenemos que:

- N = 1 => $p_1 = a_1$
- N = 2 => $p_1 = a_1 + a_2$
- N = 3 => $p_1 = a_1 + a_3$

Análisis de Casos

Informalmente tenemos que:

- N = 1 => $p_1 = a_1$
- N = 2 => $p_1 = a_1 + a_2$
- N = 3 => $p_1 = a_1 + a_3$
- N = 4 => $max(a_1 + a_3, a_1 + a_4, a_2 + a_4)$

Análisis de Casos

Informalmente tenemos que:

- N = 1 => $p_1 = a_1$
- N = 2 => $p_1 = a_1 + a_2$
- N = 3 => $p_1 = a_1 + a_3$
- N = 4 => $max(a_1 + a_3, a_1 + a_4, a_2 + a_4)$
- N = 5 => $p_1 = a_1 + a_3 + a_5$

Notan algo?

Análisis de Casos

Informalmente tenemos que:

- N = 1 => $p_1 = a_1$
- N = 2 => $p_1 = a_1 + a_2$
- N = 3 => $p_1 = a_1 + a_3$
- N = 4 => $max(a_1 + a_3, a_1 + a_4, a_2 + a_4)$
- N = 5 => $p_1 = a_1 + a_3 + a_5$

  Notan algo?
- Acá tiene que haber un greedy general para los impares.

Demostración para N impar

- Si N es impar, Agustín se puede asegurar sacar todos los números de las posiciones pares.

Demostración para N impar

- Si N es impar, Agustín se puede asegurar sacar todos los números de las posiciones pares.
- Simplementa repite la jugada de Fidel.

Demostración para N impar

- Si N es impar, Agustín se puede asegurar sacar todos los números de las posiciones pares.
- Simplementa repite la jugada de Fidel.
- A su vez Fidel se puede asegurar sacar todos los números de las posiciones impares.

Demostración para N impar

- Si N es impar, Agustín se puede asegurar sacar todos los números de las posiciones pares.
- Simplementa repite la jugada de Fidel.
- A su vez Fidel se puede asegurar sacar todos los números de las posiciones impares.
- Simplemente saca a_1 y a_n y repite la jugada de Agustín!

Demostración para N impar

- Si N es impar, Agustín se puede asegurar sacar todos los números de las posiciones pares.
- Simplementa repite la jugada de Fidel.
- A su vez Fidel se puede asegurar sacar todos los números de las posiciones impares.
- Simplemente saca a_1 y a_n y repite la jugada de Agustín!
- Por lo tanto ambos son óptimas! Ninguno puede asegurar sacar más

Demostración para N impar

- Si N es impar, Agustín se puede asegurar sacar todos los números de las posiciones pares.
- Simplementa repite la jugada de Fidel.
- A su vez Fidel se puede asegurar sacar todos los números de las posiciones impares.
- Simplemente saca a_1 y a_n y repite la jugada de Agustín!
- Por lo tanto ambos son óptimas! Ninguno puede asegurar sacar más
- Esto ya es suficiente para pensar el caso par también.

N par

N es par

- Fidel, como siempre, tiene 3 opciones

N par

N es par

- Fidel, como siempre, tiene 3 opciones
- Si saca el a_1, por lo demostrado antes, Agustín terminará sacando como puntaje la sumatoria de los pares (la llamaremos PAR). Por lo tanto Fidel tendrá la suma de los impares ($p_1 = IMP$)

N par

N es par

- Fidel, como siempre, tiene 3 opciones
- Si saca el a_1, por lo demostrado antes, Agustín terminará sacando como puntaje la sumatoria de los pares (la llamaremos PAR). Por lo tanto Fidel tendrá la suma de los impares ($p_1 = IMP$)
- Si saca el a_n, similarmente, Fidel tendrá $p_1 = PAR$

N par

N es par

- Fidel, como siempre, tiene 3 opciones
- Si saca el a_1, por lo demostrado antes, Agustín terminará sacando como puntaje la sumatoria de los pares (la llamaremos PAR). Por lo tanto Fidel tendrá la suma de los impares ($p_1 = IMP$)
- Si saca el a_n, similarmente, Fidel tendrá $p_1 = PAR$
- Si Fidel saca p_1 y p_n, Agustín queda con una cantidad par de elementos y no tenemos una fórmula

N par

N es par

- Fidel, como siempre, tiene 3 opciones
- Si saca el a_1, por lo demostrado antes, Agustín terminará sacando como puntaje la sumatoria de los pares (la llamaremos PAR). Por lo tanto Fidel tendrá la suma de los impares ($p_1 = IMP$)
- Si saca el a_n, similarmente, Fidel tendrá $p_1 = PAR$
- Si Fidel saca p_1 y p_n, Agustín queda con una cantidad par de elementos y no tenemos una fórmula
- Pero entonces, solo tenemos un caso que no sabemos que hacer (y es un caso menor).

N par

N es par

- Fidel, como siempre, tiene 3 opciones
- Si saca el a_1, por lo demostrado antes, Agustín terminará sacando como puntaje la sumatoria de los pares (la llamaremos PAR). Por lo tanto Fidel tendrá la suma de los impares ($p_1 = IMP$)
- Si saca el a_n, similarmente, Fidel tendrá $p_1 = PAR$
- Si Fidel saca p_1 y p_n, Agustín queda con una cantidad par de elementos y no tenemos una fórmula
- Pero entonces, solo tenemos un caso que no sabemos que hacer (y es un caso menor).
- Intuitivamente se puede pensar que los dos van sacando las puntas hasta que uno decide elegir IMP o PAR

N par

N es par

- Fidel, como siempre, tiene 3 opciones
- Si saca el a_1, por lo demostrado antes, Agustín terminará sacando como puntaje la sumatoria de los pares (la llamaremos *PAR*). Por lo tanto Fidel tendrá la suma de los impares ($p_1 = IMP$)
- Si saca el a_n, similarmente, Fidel tendrá $p_1 = PAR$
- Si Fidel saca p_1 y p_n, Agustín queda con una cantidad par de elementos y no tenemos una fórmula
- Pero entonces, solo tenemos un caso que no sabemos que hacer (y es un caso menor).
- Intuitivamente se puede pensar que los dos van sacando las puntas hasta que uno decide elegir *IMP* o *PAR*
- Si hacemos una función recursiva actualizando los valores *IMP* y *PAR*, podremos saber la respuesta

Solución

- Para N impar es la suma de las posiciones impares

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (*IMP* y *PAR*)

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (*IMP* y *PAR*)
- $p_1 = f(0, IMP, PAR)$
- $f(0, i, p) = max(i, p, a_1 + a_n + (tot' - f(1, i', p')))$

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (*IMP* y *PAR*)
- $p_1 = f(0, IMP, PAR)$
- $f(0, i, p) = max(i, p, a_1 + a_n + (tot' - f(1, i', p')))$
- $f(0, i, p) = max(i, p, i + p - f(1, i', p'))$

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (*IMP* y *PAR*)
- $p_1 = f(0, IMP, PAR)$
- $f(0, i, p) = max(i, p, a_1 + a_n + (tot' - f(1, i', p')))$
- $f(0, i, p) = max(i, p, i + p - f(1, i', p'))$
- $f(0, i, p) = max(i, p, i + p - f(1, i - a_1, p - a_n))$

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (IMP y PAR)
- $p_1 = f(0, IMP, PAR)$
- $f(0,i,p) = max(i,p,a_1 + a_n + (tot' - f(1,i',p')))$
- $f(0,i,p) = max(i,p,i+p-f(1,i',p'))$
- $f(0,i,p) = max(i,p,i+p-f(1,i-a_1,p-a_n))$
- $f(1,i,p) = max(i,p,i+p-f(2,i-a_{n-1},p-a_2))$

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (IMP y PAR)
- $p_1 = f(0, IMP, PAR)$
- $f(0, i, p) = max(i, p, a_1 + a_n + (tot' - f(1, i', p')))$
- $f(0, i, p) = max(i, p, i + p - f(1, i', p'))$
- $f(0, i, p) = max(i, p, i + p - f(1, i - a_1, p - a_n))$
- $f(1, i, p) = max(i, p, i + p - f(2, i - a_{n-1}, p - a_2))$
- $f(ki, i, p) = max(i, p, i + p - f(ki + 1, i - a_{1+ki}, p - a_{n-ki}))$

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (IMP y PAR)
- $p_1 = f(0, IMP, PAR)$
- $f(0, i, p) = max(i, p, a_1 + a_n + (tot' - f(1, i', p')))$
- $f(0, i, p) = max(i, p, i + p - f(1, i', p'))$
- $f(0, i, p) = max(i, p, i + p - f(1, i - a_1, p - a_n))$
- $f(1, i, p) = max(i, p, i + p - f(2, i - a_{n-1}, p - a_2))$
- $f(ki, i, p) = max(i, p, i + p - f(ki + 1, i - a_{1+ki}, p - a_{n-ki}))$
- $f(kp, i, p) = max(i, p, i + p - f(kp + 1, i - a_{n-kp}, p - a_{1+kp}))$

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (*IMP* y *PAR*)
- $p_1 = f(0, IMP, PAR)$
- $f(0, i, p) = max(i, p, a_1 + a_n + (tot' - f(1, i', p')))$
- $f(0, i, p) = max(i, p, i + p - f(1, i', p'))$
- $f(0, i, p) = max(i, p, i + p - f(1, i - a_1, p - a_n))$
- $f(1, i, p) = max(i, p, i + p - f(2, i - a_{n-1}, p - a_2))$
- $f(ki, i, p) = max(i, p, i + p - f(ki + 1, i - a_{1+ki}, p - a_{n-ki}))$
- $f(kp, i, p) = max(i, p, i + p - f(kp + 1, i - a_{n-kp}, p - a_{1+kp}))$
- $f(k, 0, 0) = 0$

Solución

- Para N impar es la suma de las posiciones impares
- Para N par empezamos calculando la suma inicial de los impares y los pares (IMP y PAR)
- $p_1 = f(0, IMP, PAR)$
- $f(0, i, p) = max(i, p, a_1 + a_n + (tot' - f(1, i', p')))$
- $f(0, i, p) = max(i, p, i + p - f(1, i', p'))$
- $f(0, i, p) = max(i, p, i + p - f(1, i - a_1, p - a_n))$
- $f(1, i, p) = max(i, p, i + p - f(2, i - a_{n-1}, p - a_2))$
- $f(ki, i, p) = max(i, p, i + p - f(ki + 1, i - a_{1+ki}, p - a_{n-ki}))$
- $f(kp, i, p) = max(i, p, i + p - f(kp + 1, i - a_{n-kp}, p - a_{1+kp}))$
- $f(k, 0, 0) = 0$

Aclaración: Cuidado que está 1-indexado

Resumen

- La mejor forma de aprender a resolver Greedys es realizando problemas.

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.

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 bordes, y confiar en que funciona.

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 bordes, y confiar en que funciona.
- Como los greedys son muy simples de codear, lo mandamos y probamos. Si anda un golazo, sino lo volvemos a pensar. Esta estrategia seguimos nosotros como equipo.

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 bordes, y confiar en que funciona.
- Como los greedys son muy simples de codear, lo mandamos y probamos. Si anda un golazo, sino lo volvemos a pensar. Esta estrategia seguimos nosotros como equipo.
- Para encontrar un greedy en un ejercicio es fundamental probar como resolverías el ejercicio para casos bordes simples.

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 bordes, y confiar en que funciona.
- Como los greedys son muy simples de codear, lo mandamos y probamos. Si anda un golazo, sino lo volvemos a pensar. Esta estrategia seguimos nosotros como equipo.
- Para encontrar un greedy en un ejercicio es fundamental probar como resolverías el ejercicio para casos bordes simples.
- Sino es imposible poder llegar a encontrar una solución intuitiva.

Problema Extra para pensar

At Most Twice

Dado un entero positivo $1 \leq U \leq 10^{18}$, determinar el mayor $L \leq U$ tal que L no contenga más de una vez cada dígito

Ejemplos

2210102960	RTA: 2210099887
1000000000000000000	RTA: 998877665544332211
1001223343	RTA: 998877665
20152015	RTA: 20152015

Links a Ejercicios

- The Agency
- The Hero y Woodworms - No los van a poder submitear
- Emoticons (Medio)
- Flip and Shift (Medio)

¿Preguntas?