Algoritmos Voraces o Greedy

Emanuel Lupi

Universidad Nacional de Córdoba

emanuel.lupi91@gmail.com

29 de Julio

Overview

Qué son los algoritmos greedy?

- Un algoritmo greedy es una estrategia de búsqueda.

Qué son los algoritmos greedy?

- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.

Qué son los algoritmos greedy?

- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.
- El algoritmo debe conducir a una solución óptima.

Qué son los algoritmos greedy?

- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.
- El algoritmo debe conducir a una solución óptima.
- Para saber que el algoritmo conduce a una solución óptima hay que demostrarlo. Y por lo general se hace un demostración formal.

Qué son los algoritmos greedy?

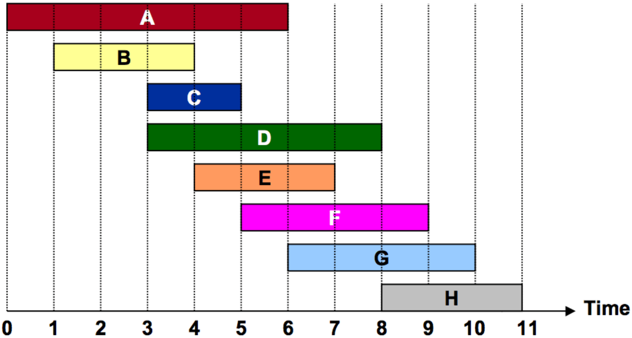
- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.
- El algoritmo debe conducir a una solución óptima.
- Para saber que el algoritmo conduce a una solución óptima hay que demostrarlo. Y por lo general se hace un demostración formal.
- Hay problemas greedy que son conocidos y por tanto no hay que demostrarlos. Pues ya fueron probados.

Problema de la selección de tareas

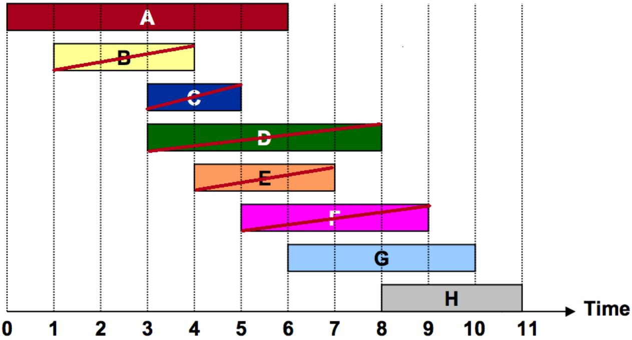
Problema de la selección de tareas

Tenemos actividades a realizar, dichas actividades tienen un principio y fin conocido. Las actividades se pueden superponer. El problema pide la máxima cantidad de actividades que podemos elegir sin que se superpongan.

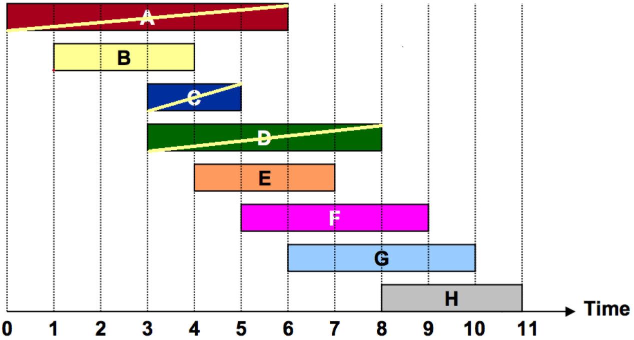
Problema de la selección de tareas



Problema de la selección de tareas



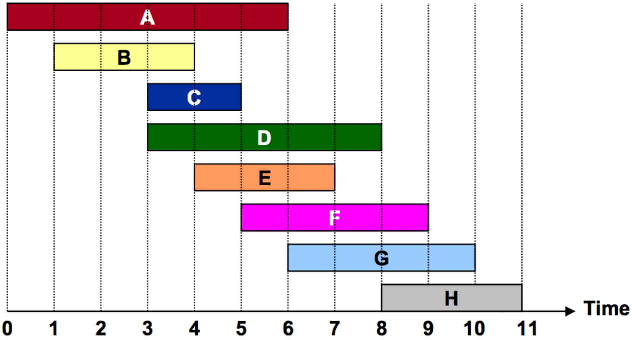
Problema de la selección de tareas



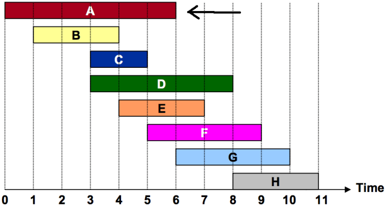
Problema de la selección de tareas

- Cuál es la estrategia de búsqueda?
- Que heurística consistente usamos?
- Cuál es la opción óptima en cada paso local?

Reviendo el ejemplo

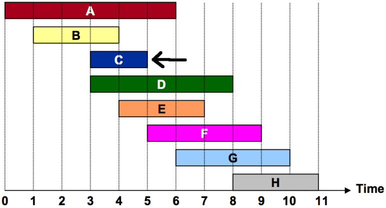


Reviendo el ejemplo



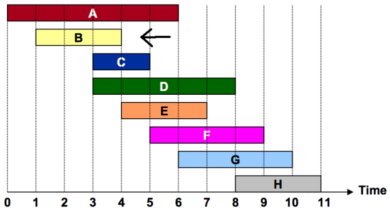
- Se impulsivo: tomar las tareas en forma ascendente en el tiempo de inicio.

Reviendo el ejemplo



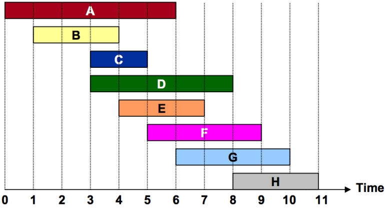
- Se impulsivo: tomar las tareas en forma ascendente en el tiempo de inicio.
- Evitar colisiones: Tomar la tarea más corta.

Reviendo el ejemplo



- Se impulsivo: tomar las tareas en forma ascendente en el tiempo de inicio.
- Evitar colisiones: Tomar la tarea más corta.
- Terminar rápido: Tomar la tarea que termina primera.

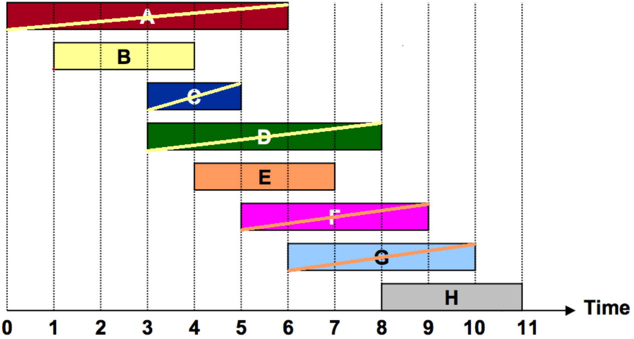
Reviendo el ejemplo



- Se impulsivo: tomar las tareas en forma ascendente en el tiempo de inicio.
- Evitar colisiones: Tomar la tarea más corta.
- Terminar rápido: Tomar la tarea que termina primera.
- Otra?

Solución

La solución es **Terminar rápido**.



Preguntas

Prueba

Para convencernos de que la solución S producida por el algoritmo es óptimo, usaremos el argumento de **se mantiene por delante**.

Prueba

Para convencernos de que la solución S producida por el algoritmo es óptimo, usaremos el argumento de **se mantiene por delante**.
Para facilitar la notación diremos que las actividades están ordenadas por tiempo de finalización. La funcion f indica el tiempo de final (finish time) y s el tiempo de comienzo de una actividad (start)
Lema: Suponga que la solución greedy seleccionó las actividades $G = \{G_1, ..., G_k\}$. Entonces para cualquier $0 \leq l \leq k$ hay una solución óptima de la forma $O = \{G_1, ..., G_l, O_{l+1}, ...O_m\}$

Prueba

Prueba por inducción en l. **Caso base** $l = 0$ la prueba se da solo. No se reemplaza nigún elemento de O

Caso inductivo

- Suponga que la declaración es válida para l. Por lo tanto, existe una solución óptima $O = \{G_1, ..., G_l, O_{l+1}, ...O_m\}$

Prueba

Prueba por inducción en l. **Caso base** $l = 0$ la prueba se da solo. No se reemplaza nigún elemento de O

Caso inductivo

- Suponga que la declaración es válida para l. Por lo tanto, existe una solución óptima $O = \{G_1, ..., G_l, O_{l+1}, ...O_m\}$
- Notar que: $s(O_{l+2}) \geq f(O_{l+1}))$

Prueba

Prueba por inducción en l. **Caso base** $l = 0$ la prueba se da solo. No se reemplaza nigún elemento de O

Caso inductivo

- Suponga que la declaración es válida para l. Por lo tanto, existe una solución óptima $O = \{G_1, ..., G_l, O_{l+1}, ...O_m\}$
- Notar que: $s(O_{l+2}) \geq f(O_{l+1}))$
- Notar que: $s(G_{l+1}) \leq f(G_{l+1}) \leq f(O_{l+1}))$

Prueba

Prueba por inducción en l. **Caso base** $l = 0$ la prueba se da solo. No se reemplaza nigún elemento de O

Caso inductivo

- Suponga que la declaración es válida para l. Por lo tanto, existe una solución óptima $O = \{G_1, ..., G_l, O_{l+1}, ...O_m\}$
- Notar que: $s(O_{l+2}) \geq f(O_{l+1}))$
- Notar que: $s(G_{l+1}) \leq f(G_{l+1}) \leq f(O_{l+1}))$
- Por lo tanto, G_{l+1} puede ser sustituto de O_{l+1} en la solución O, produciendo la solución O'.

Prueba

El algoritmo greedy siempre encuentra una solución óptima.

Prueba

El algoritmo greedy siempre encuentra una solución óptima.
Usando el lema anterior para $l = k$, sabemos que existe una solución óptima de la forma $O = \{G_1, ..., G_k, O_{k+1}, ...O_m\}$.
Si $m > k$ entonces esto significa que el tiempo de inicio $S(O_{k+1}) \geq f(G_k)$, pero O_{k+1} se agregaría a G por como es el algoritmo. Contradicción.

Preguntas

Formas para probar algoritmos Greedys

- Primero, debe demostrar que su algoritmo produce una solución factible, o sea, una solución al problema que obedece las restricciones.

Formas para probar algoritmos Greedys

- Primero, debe demostrar que su algoritmo produce una solución factible, o sea, una solución al problema que obedece las restricciones.
- Vamos a explicar dos formas de probar un algoritmo greedy.

Factibilidad

Por lo general, es mucho más fácil demostrar que la solución cumple con las restricciones que demostrar la optimidad.
Sin embargo, al redactar una prueba formal de corrección aveces nos saltamos este paso. Típicamente, estas pruebas funcionan por inducción, mostrando que en cada paso que la elección no viola las restricciones y el algoritmo termina con una solución correcta.

Las dos principales técnicas de prueba de optimalidad

- Se mantiene por delante: Este estilo de prueba funciona demostrando que, de alguna manera, el algoritmo siempre está a la altura de la solución óptima durante cada iteración de el algoritmo.

Las dos principales técnicas de prueba de optimalidad

- Se mantiene por delante: Este estilo de prueba funciona demostrando que, de alguna manera, el algoritmo siempre está a la altura de la solución óptima durante cada iteración de el algoritmo.
- Intercambio de argumentos: Funcionan demostrando que puede transformar iterativamente cualquier solucin óptima en la solución producida por el algoritmo sin cambiar el costo de la solución óptima, por lo tanto demostrando que la solución es óptima.

Se mantiene por delante

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.

Se mantiene por delante

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.
- **Define tu medida**: El objetivo es encontrar una serie de medidas que se puedan hacer a la solución greedy y la solución óptima. Definir algunas series de medidas tales como $m_1(G), m_2(G), ..., m_n(G)$ tal que $m_1(O), m_2(O), ..., m_k(O)$ también se define para algunas opciones de m y n (no se asumen que son iguales).

Se mantiene por delante

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.
- **Define tu medida**: El objetivo es encontrar una serie de medidas que se puedan hacer a la solución greedy y la solución óptima. Definir algunas series de medidas tales como $m_1(G), m_2(G), ..., m_n(G)$ tal que $m_1(O), m_2(O), ..., m_k(O)$ también se define para algunas opciones de m y n (no se asumen que son iguales).
- **Probar que se mantiene por delante**: probar que $m_i(G) \geq m_i(O)$ o al revez dado el caso.

Se mantiene por delante

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.
- **Define tu medida**: El objetivo es encontrar una serie de medidas que se puedan hacer a la solución greedy y la solución óptima. Definir algunas series de medidas tales como $m_1(G), m_2(G), ..., m_n(G)$ tal que $m_1(O), m_2(O), ..., m_k(O)$ también se define para algunas opciones de m y n (no se asumen que son iguales).
- **Probar que se mantiene por delante**: probar que $m_i(G) \geq m_i(O)$ o al revez dado el caso.
- **Probar optimalidad**: Usando el hecho de que **se mantiene por delante**, hay que probar que el algoritmo debe producir una solución óptima. Este argumento a menudo se hace por prueba del absurdo.

Intercambio de argumentos

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.

Intercambio de argumentos

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.
- **Comparar las Soluciones**: Ver que si $G \neq O$. Esto es devido a: que hay elementos de G que no estan en O, que están en diferente orden, u otras razones.

Intercambio de argumentos

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.
- **Comparar las Soluciones**: Ver que si $G \neq O$. Esto es devido a: que hay elementos de G que no estan en O, que están en diferente orden, u otras razones.
- **Intercambiar piezas**: Mostrar como transformar O reemplazando algunas piezas de esta por piezas de G y probar que haciendo esto no se empeora la solución.

Intercambio de argumentos

- **Define la Solución**: El algoritmos produce una solución G y se compara con una solución óptima O. Hay que agregar algunas variables que describan las soluciones.
- **Comparar las Soluciones**: Ver que si $G \neq O$. Esto es devido a: que hay elementos de G que no estan en O, que están en diferente orden, u otras razones.
- **Intercambiar piezas**: Mostrar como transformar O reemplazando algunas piezas de esta por piezas de G y probar que haciendo esto no se empeora la solución.
- **Iterar**: Habiendo reducido el número de diferencias entre G y O realizando el intercambio, y que al repetir este proceso puede convertir O en G sin impactando la calidad de la solución. Por lo tanto, G debe ser óptimo.

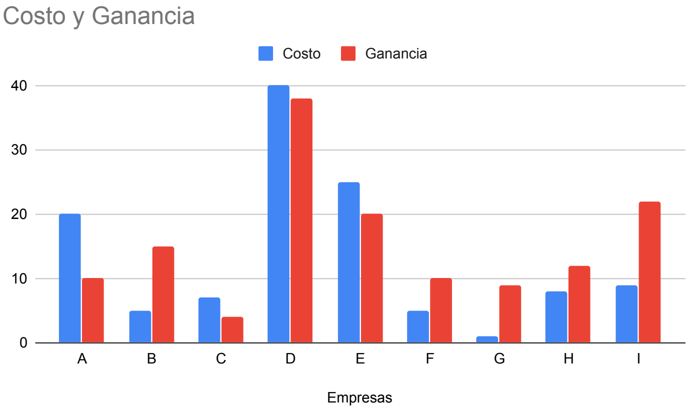
Preguntas

The Investor

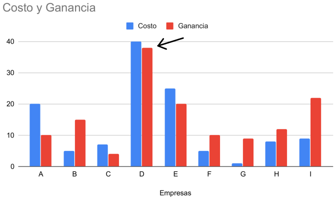
The Investor

Un inversionista es exitoso si compra todas las empresas, las empresas cuestan una cierta cantidad de dinero y una vez adquiridas dan una cierta cantidad de dinero (única ganancia). Diga si pepito comenzando con tanto dinero D puede ser un inversionista exitoso.

The Investor

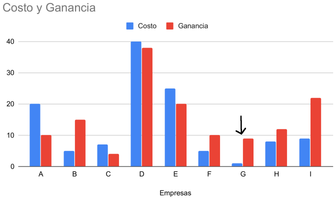


Cuál es la mejor estrategia?



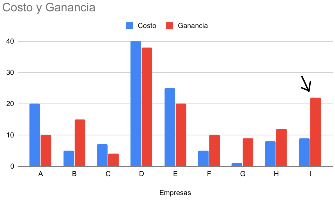
- Comprar las empresas que dan más ganancia.

Cuál es la mejor estrategia?



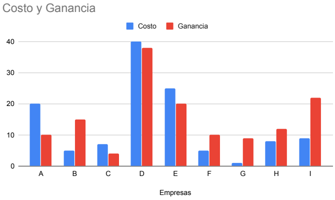
- Comprar las empresas que dan más ganancia.
- Comprar las empresas más baratas.

Cuál es la mejor estrategia?



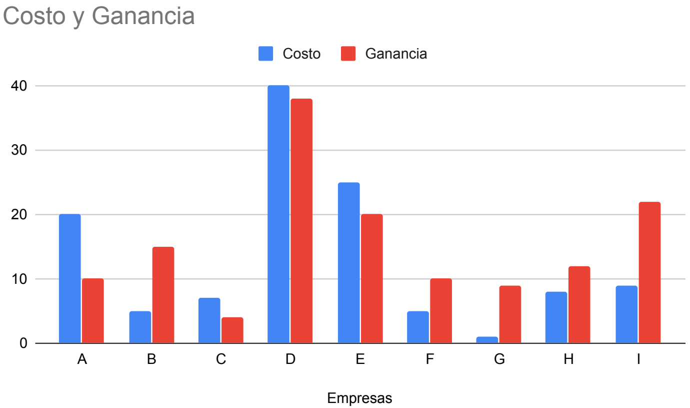
- Comprar las empresas que dan más ganancia.
- Comprar las empresas más baratas.
- La mejor en costo beneficio.

Cuál es la mejor estrategia?



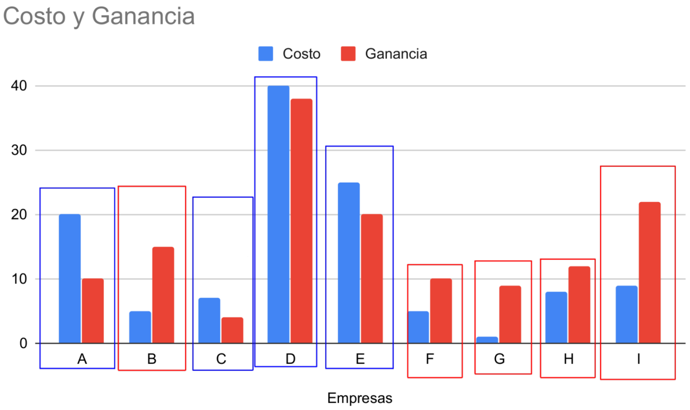
- Comprar las empresas que dan más ganancia.
- Comprar las empresas más baratas.
- La mejor en costo beneficio.
- Otra?

Volviendo al ejemplo

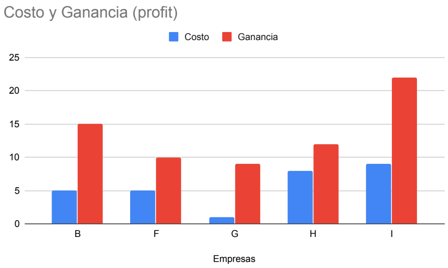


Volviendo al ejemplo

Podríamos pensar que tal vez sea bueno separar el problema en dos

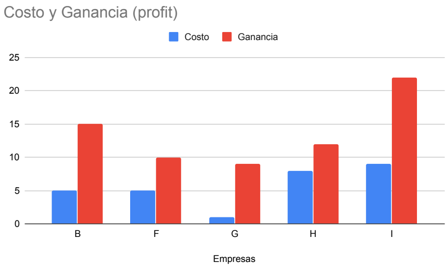


Estrategia para empresas que dan ganancias



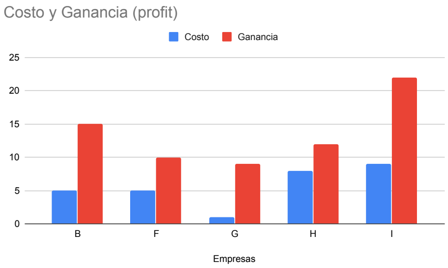
- Tomar cualquiera?

Estrategia para empresas que dan ganancias



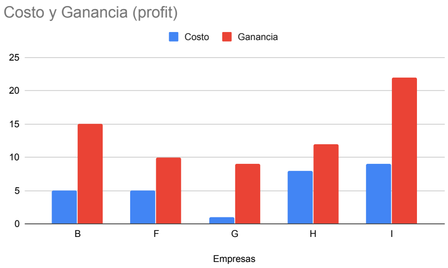
- Tomar cualquiera?
- Tomar la que más ganancias da?

Estrategia para empresas que dan ganancias



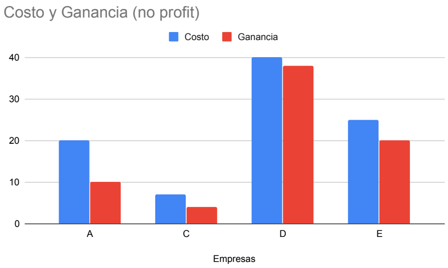
- Tomar cualquiera?
- Tomar la que más ganancias da?
- Tomar la más barata?

Estrategia para empresas que dan ganancias



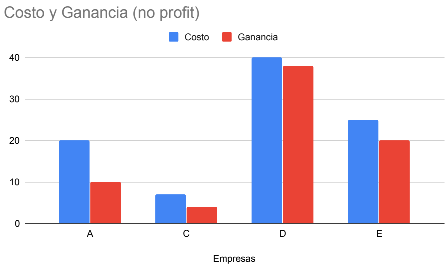
- Tomar cualquiera?
- Tomar la que más ganancias da?
- Tomar la más barata?
- Otra?

Estrategia para empresas que dan perdidas



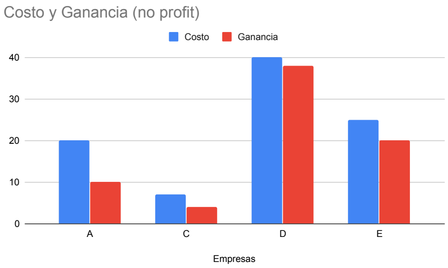
- Tomar la que tiene menor $costo - ganancia$?

Estrategia para empresas que dan perdidas



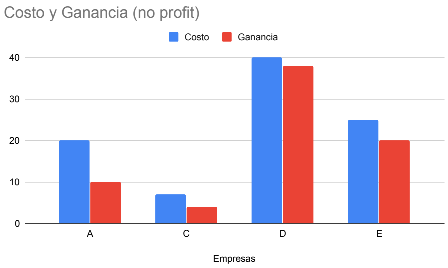
- Tomar la que tiene menor $costo - ganancia$?
- Tomar la mas barata?

Estrategia para empresas que dan perdidas



- Tomar la que tiene menor $costo - ganancia$?
- Tomar la mas barata?
- Tomar la mas cara?

Estrategia para empresas que dan perdidas



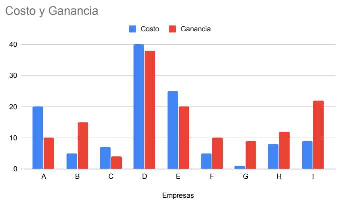
- Tomar la que tiene menor *costo* − *ganancia*?
- Tomar la mas barata?
- Tomar la mas cara?
- Otra?

Solución

La solcuión es: comprar la más barata en el conjunto de las que dan ganancias.

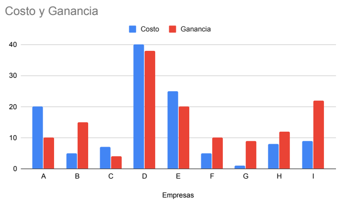
Comprar la que tiene menor $costo - ganancia$ en las que dan perdidas.

Solución idea de correctitud



Comprar las más baratas, de las que dan ganancias, siempre me deja mejor parado para comprar las demas empresas que dan ganancias.

Solución idea de correctitud



Comprar las más baratas, de las que dan ganancias, siempre me deja mejor parado para comprar las demas empresas que dan ganancias.
Comprar la que mejor $costo - ganancia$ tiene, me deja mejor parado, para comprar las demás empresas ya que el dinero que perdí es menor que el que perdido comprando otra.

Preguntas

La Rana Fred

La Rana Fred está en la orilla izquierda de un río y hay N rocas que están en línea recta desde la orilla izquierda a la orilla derecha. La distancia entre la orilla izquierda y la derecha es D metros. Hay rocas de dos tamaños. Las más grandes pueden soportar cualquier peso, pero los más pequeñas comienzan a undirse tan pronto como se coloca cualquier cosa. Fred tiene que ir a la orilla derecha, donde tiene que recoger un regalo y regresar a la orilla izquierda donde se encuentra su casa. Puede aterrizar en cada roca pequeña una vez como máximo, pero puede usar las más grandes tantas veces como quiera. Nunca puede tocar el agua, pues hay animales peligrosos. Se puede planificar los saltos de modo que se minimice la distancia máxima de un solo salto?

La Rana Fred

Sweep Line

Imagina que tienes una lńea vertical que "barre" el plano de izquierda a derecha. Esa es la idea principal detrás de la línea de barrido.
Es posible que esté pensando "espere, no es muy ineficaz seguir la línea de barrido en todas las posiciones posibles?" Y estarías en lo correcto. Sin embargo, no necesitamos realizar un seguimiento de la línea de barrido en todas las posiciones posibles, solo en las posiciones "críticas" (por ejemplo, puntos e intersecciones).

Un pequeño problema

Juan quiere perdirle dinero a su padre. Pero su padre está dispuesto a prestarle si no tiene muchas tareas en proceso. Dado N tareas que el padre de juan debe realizar, su tiempo de inicio I_i y su tiempo de de finalización F_i decir la mínima cantidad de tareas que hace en paralelo el padre de juan en un determinado momento. Por otro lado el dinero lo necesita con urgencia, por lo tanto no puede esperar hasta que termine el día para pedirselo. Supongamos que el dia termina en T y empieza en E.

Recuerden

- Un algoritmo greedy es una estrategia de búsqueda.

Recuerden

- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.

Recuerden

- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.
- El algoritmo debe conducir a una solución óptima.

Recuerden

- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.
- El algoritmo debe conducir a una solución óptima.
- Para saber que el algoritmo conduce a una solución óptima hay que demostrarlo. Y por lo general se hace un demostración formal.

Recuerden

- Un algoritmo greedy es una estrategia de búsqueda.
- En la cual se usa una heurística consistente en elegir la opción óptima en cada paso local.
- El algoritmo debe conducir a una solución óptima.
- Para saber que el algoritmo conduce a una solución óptima hay que demostrarlo. Y por lo general se hace un demostración formal.
- Hay problemas greedy que son conocidos: Dijkstra, Prim, Kruskal, entre otros.

Referencias

Tim Roughgarden, Alexa Sharp, and Tom Wexler
Guide to Greedy Algorithms
https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/handouts/120%20Guide

Dynamic Frog.
https://vjudge.net/problem/UVA-11157.