Universidad Nacional de Rosario
Instituto Politecnico Superior
Técnico en Informática Profesional y Personal
Algoritmos y Estructuras de Datos Avanzadas

Trabajo práctico

1 Introducción

El objetivo del siguiente trabajo es que comprendas las técnicas algorítmicas de programación dinámica y algoritmos Greedy. Para esto se te proporcionan distintos ejercicios de cada estrategia, que tendras que comprendidos, resolver y explicarlos minuciosamente. La mejor forma de entender un problema es poder explicárselo a otros. Es por ello que prepararas una clase, generando material de apoyo, donde explicaras los problemas que se te asignen. También elaborarás una informe donde se explique su resolución junto con el sustento teórico de esta.
La resolución del problema tiene que ser realizada en el lenguaje de programación que sugiera el doncente, por lo que deberás consultarlo antes de implementarla.
El trabajo se debe realizar en grupos de dos personas y la fecha límite de entrega es el martes 15 de Octubre, en donde se debe entregar:

- en papel, un informe con los ejercicios resueltos incluyendo todo el código que haya escrito;
- en forma electrónica (en un archivo comprimido) el informe, el material de apoyo para la clase y el código fuente de los programas, al correo `r20168@gmail.com` con el asunto `TP DINAMICA GREEDY: <Apellidos de los integrantes>`

Luego de esa fecha y a partir del Miércoles 16 de Octubre comensarán las presentaciónes de las clases. Se otorgará 25 minutos por grupo y el orden será dado por número de grupo.
Sobre el informe impreso: se pide que sea elaborado en hoja $A4$, interlineado simple, fuente `Computer Modern` de tamaño $10pt$. Los margenes: Izquierdo de 2 cm, derecho de 1.5 cm, superior de 3 cm e inferior de 1.5 cm.

2 Problemas de Programación Dinámica

A continuación se numeran los distintos problemas de programación dinamica que tendrán que ser presentados por los grupos:
1. Algunas normas tipográficas establecen que las líneas deben contener entre 40 y 60 caracteres. Supongamos que no queremos partir las palabras y que conocemos los tamaños (incluidos los signos de puntuación y espacios en blanco adyacentes) $s_1, s_1, ...s_N$ de las N palabras de un párrafo de M líneas. Si el objetivo es minimizar la longitud de la línea más larga del párrafo, escribe un algoritmo de programación dinámica que minimice la longitud de la línea más larga de un párrafo. Para esto:

a) completa la definición de la función $f(n, m)$ que devuelva la longitud mínima de la línea más larga cuando las n primeras palabras se distribuyen en m líneas.

$$f(n,m) = \begin{cases} 0 & n = 0 \\ \sum_{n=1}^{N} s_n & m = 1 \\ \ldots\ldots\ldots & \textit{en otro caso} \end{cases} \quad (1)$$

b) Utiliza programación dinámica para implementar eficientemente la función anterior.

2. Dada una secuencia finita de números enteros estrictamente positivos $S = \{a_1, a_2, \ldots, a_N\}$, se desea extraer una subsecuencia tan larga como sea posible que sea no decreciente. Por ejemplo, si

$$S = \{7, 2, 6, 5, 6\}$$

las subsecuencias $\{2, 5, 6\}$ y $\{2, 6, 6\}$ son no decrecientes y de longitud máxima. Puede suponer conocida M tal que $M \geq max\{a_1, a_2, \ldots, a_N\}$

a) Dar una definición por casos que sirva para resolver el problema por dinámica.

b) Escribe un algoritmo que utilice programación dinámica (bottom-up) para resolver este problema.

3. Se quiere encontrar una parentización apropiada para maximizar el valor de la siguiente expresión:

$$x_1/x_2/x_3 \ldots /x_{n-1}/x_n$$

donde $x_1, x_2, x_3, \ldots, x_n$ son números naturales positivos y / denota la división.

a) Realice la formulación recursiva para calcular el valor máximo de la expresión con una parentización adecuada.

b) Utilice programación dinámica (bottom-up) para diseñar un algoritmo que retorne el máximo valor de la expresión con una parentización apropiada.

c) Explique, no implemente, que cambios o agregados se le tendrían que hacer al algoritmo del apartado anterior para mostrar la parentización que maximiza el valor de la expresión.

4. Considere el problema de, dado un arreglo $A[a_1, \cdots, a_n]$ de enteros no negativos, encontrar una subsecuencia $A[a_i, \cdots, a_j]$ de tamaño máximo tal que: $i = j$ ó $i = j + 1$ ó para todo $r \geq 0$, valen las desigualdades $a_{i+r} \geq \sum_{k=i+r+1}^{j-r-1} a_k$ y $a_{j-r} \geq \sum_{k=i+r+1}^{j-r-1} a_k$.
Por ejemplo, en el arreglo $[1, 8, 2, 1, 3, 9, 10]$ la subsecuencia $[8, 2, 1, 3, 9]$ cumple la propiedad, ya que $8 \geq 2 + 1 + 3$ y $9 \geq 2 + 1 + 3$ $(r = 0)$ y además $2 \geq 1$ y $3 \geq 1$ $(r = 1)$.
Escriba un algoritmo de programación dinámica bottom-up para resolver el problema. La salida del algoritmo son dos índices $i \leq j$ tales que $A[a_i, \cdots, a_j]$ es una subsecuencia de tamaño máximo.

5. Dada una cadena $A = \langle a_1, a_2, \ldots, a_n \rangle$ decimos que $A_{i\ldots j} = \langle a_i, a_{i+1}, \ldots, a_j \rangle$ es un subpalindromo de A si $a_{i+h} = a_{j-h}$ para $0 \leq h \leq j - i$. Por ejemplo, si $A = accaba$ entonces $A_{1\ldots4} = acca$ y $A_{4\ldots6} = aba$ son subpalindromos de A.
Diseñar un algoritmo, usando la técnica de programación dinámica, que determine la longitud del subpalindromo de longitud máxima. **Ayuda:** Una cadena $a_1, a_2, \ldots, a_n$ (con $n > 2$) es un palíndromo si y sólo si $a_1 = a_n$ y $a_2 \ldots a_{n-1}$ es un palíndromo.

6. Considere el problema de, dado un arreglo $A[a_1, \cdots, a_n]$ de enteros no negativos, encontrar una subsecuencia $A[a_i, \cdots, a_j]$ de tamaño máximo tal que: $i = j$ ó $i = j + 1$ ó para todo $r \geq 0$, valen las desigualdades $a_{i+r} \geq \sum_{k=i+r+1}^{j-r-1} a_k$ y $a_{j-r} \geq \sum_{k=i+r+1}^{j-r-1} a_k$.
Por ejemplo, en el arreglo $[1, 8, 2, 1, 3, 9, 10]$ la subsecuencia $[8, 2, 1, 3, 9]$ cumple la propiedad, ya que $8 \geq 2 + 1 + 3$ y $9 \geq 2 + 1 + 3$ $(r = 0)$ y además $2 \geq 1$ y $3 \geq 1$ $(r = 1)$.
Escriba un algoritmo de programación dinámica bottom-up para resolver el problema. La salida del algoritmo son dos índices $i \leq j$ tales que $A[a_i, \cdots, a_j]$ es una subsecuencia de tamaño máximo.

7. Un alumno ingresante de Ciencias de la Computación se propone terminar la carrera en solo dos años. Para lograrlo, cursa simultáneamente materias correspondientes a años diferentes y en

muchos casos debe rendir un mismo día varios examenes. Supongamos que un día determinado debe rendir m examenes y solo dispone de n horas de estudio a repartir entre todas las materias. Sea $p_j(x)$ la probabilidad de reprobar el examen j-ésimo habiéndole dedicado x horas de estudio y $f_j(x)$ la probabilidad de reprobar el examen j-ésimo y todos los siguientes distribuyendo de manera óptima x horas de estudio.

a) Obtenga una fórmula recursiva para calcular $f_j(x)$. **Nota**: La probabilidad conjunta de reprobar dos examenes dedicándole x e y horas de estudio respectivamente se calcula como el producto de la probabilidad de reprobar cada uno independientemente dedicándole las mismas horas de estudio.

b) Escriba un algoritmo de programación dinámicaque encuentre una manera de distribuir las n horas de estudio disponibles entre los m examenes, minimizando la probabilidad de reprobar todos los examenes. El algoritmo debe también calcular esa probabilidad, que no es más que $f_1(n)$.

8. Considere un tablero con $n \times n$ cuadros y una función costo $c(i, j)$ que retorna un costo asociado a cada cuadro i, j. Se debe mover una ficha de dama desde la parte inferior del tablero hacia la parte superior del mismo. En cada paso se puede mover la ficha hacia uno de los siguientes cuadros:

- El cuadro inmediatamente superior.
- El cuadro que se encuentra uno arriba y uno a la izquierda (si la ficha no estaba en la columna izquierda).
- El cuadro que se encuentra uno arriba y uno a la derecha (si la ficha no estaba en la columna derecha).

a) Realice la formulación recursiva del problema, mostrando la subestructura optima.

b) Utilice programación dinámica (bottom-up) para diseñar un algoritmo que retorne el mínimo costo de mover una ficha desde cualquier cuadro de la parte inferior del tablero hacia cualquier cuadro de la parte superior del mismo. El costo del camino es la suma de los costos de los cuadros por los que pasa la ficha.

c) Adapte el algoritmo del ítem anterior para obtener la secuencia de cuadros del camino de mínimo costo.

9. Considere un polígono convexo $P = \langle v_0, v_1, ..., v_{n-1} \rangle$ con n lados $\overline{v_0v_1}, \overline{v_1v_2}, ..., \overline{v_{n-1}v_n}$, donde $v_n = v_0$. Dados dos vértices nos adyacentes v_i y v_j, el segmento $\overline{v_iv_j}$ es una **cuerda** del polígono. Una cuerda $\overline{v_iv_j}$ divide al polígono en dos polígonos: $\langle v_i, v_{i+1}, ..., v_j \rangle$ y $\langle v_j, v_{j+1}, ..., v_i \rangle$. Una **triangulación** de un polígono es un conjunto T de cuerdas de un polígono que lo divide en triángulos disjuntos. La figura **??** muestra dos maneras de triangular un polígono de 7 lados. En una triangulación, las cuerdas no se intersecan (excepto en los vértices) y el conjunto T de cuerdas es maximal: cada cuerda no perteneciente a T interseca alguna cuerda de T. Los lados de los triángulos producidos por la triangulación son o bien cuerdas pertenecientes a la triangulación o bien lados del polígono. Cada triangulación de un polígono convexo de n vértices posee $n - 3$ cuerdas y divide al polígono en $n - 2$ triángulos.
En el **problema de la triangulación óptima** de un polígono, tenemos dados un polígono $P = \langle v_0, v_1, ..., v_{n-1} \rangle$ y una función peso w definida sobre los triángulos formados por lados y cuerdas del polígono P. El problema es encontrar una triangulación que minimice la suma de los pesos de los triángulos pertenecientes a la triangulación. Un ejemplo natural de función peso sobre triángulos es

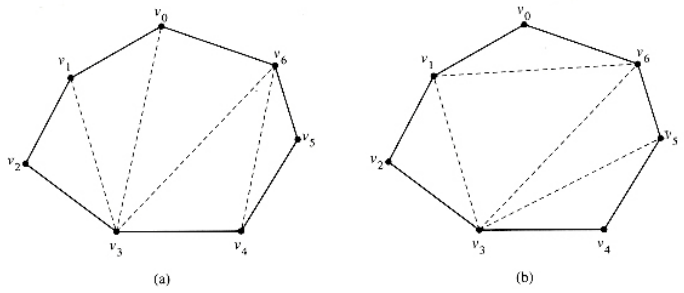


Figura 1: Dos maneras de triangular un polígono convexo. Cada triangulación de este polígono de 7 lados posee $7 - 3 = 4$ cuerdas y divide al polígono en $7 - 2 = 5$ triángulos.

$$w(\triangle v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|.$$

donde $|v_i v_j|$ es la distancia euclídea desde v_i hasta v_j.

a) Proponga una función costo $t[i, j]$, para $1 \leq i < j \leq n$, como el peso de la triangulación óptima del polígono $\langle v_{i-1}, v_i, ..., v_j \rangle$. Suponga una función peso w conocida para cada posible triángulo.

b) Diseñe un algoritmo utilizando programación dinámica para obtener la tiangulación óptima del polígono.

c) Calcule el orden de complejidad temporal y espacial del algoritmo propuesto.

10. El **problema de la mochila 0-1** consiste en encontrar la manera óptima de cargar una mochila con objetos. Existen n objetos disponibles. El i-ésimo objeto cuesta v_i pesos y pesa w_i kilos, donde v_i y w_i son enteros. Se pretende cargar el mayor valor en pesos posible pero con un máximo de W kilos, para algún entero W. Se debe decidir qué objetos tomar. Cada objeto puede ser elegido o no. No se puede tomar una parte de un objeto ni se puede elegir un objeto más de una vez.

a) Dar una fórmula recursiva para $c(i, w)$, donde $c(i, w)$ denota el valor de la solución para los items $1, \ldots, n$ y peso máximo w.

b) Utilizar la recursión del item anterior para desarrollar un algoritmo con la técnica de programación dinámica que devuelva el valor máximo en pesos que pueda cargar la mochila.

c) Modificar el algoritmo del item anterior para que retorne qué items deben cargarse para alcanzar el valor óptimo.

d) Dar la complejidad del algoritmo propuesto.

11. Sobre el río Paraná hay n embarcaderos. En cada uno de ellos se puede alquilar un bote que permite ir a cualquier otro embarcadero río abajo (es imposible ir río arriba). Existe una tabla de tarifas que indica el coste del viaje del embarcadero i al j para cualquier embarcadero de partida i y cualquier embarcadero de llegada j más abajo en el río ($i < j$). Puede suceder que un viaje de i a j sea más caro que una sucesión de viajes más cortos, en cuyo caso se tomaría un primer bote hasta un embarcadero k y un segundo bote para continuar a partir de k. No hay coste adicional por cambiar de bote.

a) Define la función de recursiva para calcular el costo óptimo para ir del embarcadero i al j.

b) Diseñar un algoritmo de programación dinámica que determine el coste mínimo para cada par de puntos i,j $(i < j)$

12. Las palabras *pelota* y *paleta* son muy similares. En efecto, una puede transformarse en otra cambiando solo dos letras. La palabra *palta* también es similar a *paleta* porque puede transformarse *palta* en *paleta* insertando la letra *e* entre la *l* y la *t*, o equivalentemente puede transformarse *paleta* en *palta* eliminando la *e* entre la *l* y la *t*.
La distancia de edición de dos palabras s_1 y s_2, $\mathrm{d}(s_1, s_2)$, se define como el mínimo número de operaciones requeridas para transformar s_1 en s_2, donde una operación puede ser

- Sustituir una letra por otra
- Insertar una letra
- Borrar una letra

a) Completar las siguientes relaciones de recurrencia, que son suficientes para definir la distancia de edición:

$$\begin{aligned} \mathrm{d}([\,],[\,]) &= \ldots \\ \mathrm{d}(s_1,[\,]) &= \ldots \\ \mathrm{d}([\,],s_2) &= \ldots \\ \mathrm{d}(s_1{}^\frown c_1, s_2{}^\frown c_2) &= \min(\ldots) \end{aligned}$$

donde $[\,]$ es la cadena vacía, s_1, s_2 son cadenas cualesquiera y $s^\frown c$ es la cadena obtenida a partir de s agregando el carácter c al final.

b) Las relaciones de recurrencia del punto anterior indican un procedimiento trivial para calcular la distancia de edición de dos palabras en un tiempo de orden exponencial. Utilice la técnica de programación dinámica para construir un algoritmo que mejore este orden (ver Fig. **??**).

		p	a	l	e	t	a
	0	1	2	3	4	5	6
p	1	0	1	2	3	4	5
e	2	1	1	2	2	3	4
l	3	2	2	1	2	3	4
o	4	3	3	2	2	3	4
t	5	4	4	3	3	2	3
a	6	5	4	4	4	3	2

Figura 2: Ejemplo de cómo pueden tabularse los resultados de los subproblemas para calcular la distancia de edición entre *pelota* y *paleta*.

c) Sugiera cómo puede mejorarse el **orden** de complejidad espacial de su algoritmo (no lo implemente, es suficiente con una justificación informal).

13. Un edificio de N pisos tiene con un único ascensor que tarda un tiempo extremadamente largo en comenzar a moverse luego de cada detención. Los vecinos de los últimos pisos protestan porque cada vez que toman el ascensor en planta baja junto con vecinos de pisos inferiores, deben soportar la lenta reacción del ascensor luego de cada detención.

En una reunión de consorcio se decide limitar a un máximo de M las detenciones en cualquier viaje desde planta baja. Con esta restricción, es posible que el ascensor no pueda detenerse en todos los pisos donde desean descender los pasajeros, por lo que algunos tendrían que bajar o subir escaleras para llegar a su piso. Se decide entonces contratar un ascensorista que programe las detenciones de forma tal que resulte mínima la cantidad **total** de pisos que tengan que subir o bajar los P pasajeros usando las escaleras.

a) Sea $m_{i,j}$ la cantidad total mínima de pisos que tendrán que subir o bajar usando las escaleras los pasajeros que quieran ir al piso i, o a pisos superiores cuando el ascensor se ha detenido en el piso i y aún se puede detener j veces más. Obtenga una fórmula recursiva para calcular $m_{0,M}$.
Ayuda: observe que una vez que el ascensor se detiene en un piso, todos los pasajeros que deseaban descender en ese piso o en pisos inferiores ya habrán descendido.

b) Escriba un algoritmo eficiente que programe las detenciones en un viaje que comienza en planta baja, minimizando la cantidad de pisos que deben bajar o subir los pasajeros usando las escaleras. El algoritmo debe dar la secuencia de pisos donde se detiene el ascensor y el costo minimizado. En caso de existir más de una programación posible se debe elegir aquella que termine en el piso más bajo (para ahorrar electricidad). Suponga que los pasajeros declaran su destino al ascensorista al comenzar el viaje y que el ascensorista les indica donde deben descender.
Ayuda: ¡es posible que el ascensorista le indique a un pasajero que debe descender al comienzo del viaje!

c) Calcule la complejidad temporal y espacial del algoritmo propuesto.

3 Problemas Greedy

A continuación se numeran los distintos problemas de Greedy que tendrán que ser presentados por los grupos:

1. Un hombre maneja un automóvil desde Rosario a Río Gallegos. Cuando su auto tiene el tanque lleno, puede viajar n kilómetros, y en sumapa tiene las distancias entre las estaciones de servicio que se encuentran en su ruta. Como el hombre está apurado por llegar, quiere hacer la menor cantidad de paradas posibles.

a) Dar una estrategia Greedy óptima, para resolver el problema anterior.

b) Probar que la estrategia es óptima.

2. Un plomero necesita hacer n reparaciones urgentes, y sabe de antemano el tiempo que le va a llevar cada una de ellas: en la tarea $i-$ésima tardará t_i minutos. Como en su empresa le pagan dependiendo de la satisfacción del cliente, necesita decidir el orden en el que atenderá los avisos para minimizar el tiempo medio de espera de los clientes. En otras palabras, si llamamos E_i a lo que espera el cliente $i-$ésimo hasta ver reparada su avería por completo, necesita minimizar la expresión:

$$E(n) = \sum_{i=1}^{n} E_i$$

a) Dar una estrategia greedy que resuelva el problema.

b) Probar que esta estrategia nos da efectivamente un óptimo.

3. En el campo de la teoría de grafos un matching o conjunto independiente de arcos en un grafo es un conjunto de arcos sin vértices en común. En un grafo con pesos, un matching de peso máximo es aquel que maximiza el peso total de sus arcos.

a) Describir un algoritmo greedy que dado un grafo (E, V) intente calcular un matching de peso máximo.

b) ¿Es óptimo el algoritmo anterior? En caso afirmativo, dé una prueba de ello, razonando por el absurdo. En caso negativo, dé un contraejemplo.

4. Dado un conjunto de intervalos $C = \{(a_1, b_1); \ldots; (a_n, b_n)\}$ sobre la recta de los números reales, se desea encontrar un subconjunto de C, de máxima cardinalidad, formado por intervalos disjuntos.

a) Diseñar una estrategia de selección greedy que resuelva este problema. Analice si la estrategia dada es optimal. Justique adecuadamente.

b) Dar un algoritmo en base a la estrategia anterior determinando claramente las estructuras de datos utilizadas. Analizar el tiempo de ejecución.

5. Sea $a[1..n]$ un vector de enteros no ordenado. Un elemento x se denomina elemento mayoritario de a si x aparece en el vector más de $n/2$ veces, es decir, $\#\{i|a[i] = x\} > n/2$. Necesitamos implementar un algoritmo capaz de decidir si un vector dado contiene un elemento mayoritario (no puede haber más de uno) y calcular su tiempo de ejecución. Una forma de plantear la solución a este problema consiste en encontrar primero un posible candidato, es decir, el único elemento que podría ser mayoritario, y luego comprobar si realmente lo es. De no serlo, el vector no admitiría elemento mayoritario, como le ocurre al vector $[1, 1, 2, 2, 3, 3, 3, 3, 4]$.

- Para n=2 existe elemento mayoritario si los elementos del vector son iguales.
- Para n par vamos a ir comparándolos por pares (a_1 con a_2, a_3 con a_4, etc.). Si para algún k = 1,3,5,7,... se tiene que $a_k = a_k + 1$ entonces copiaremos a_k en un segundo vector auxiliar b. Una vez recorrido todo el vector a, buscaremos recursivamente un candidato para el vector b, y así sucesivamente.
- Para n impar y mayor que 2, aplicaremos la idea anterior para el subvector compuesto por sus primeros $n - 1$ elementos. Como resultado puede que obtengamos que dicho subvector contenga un candidato a elemento mayoritario, con lo cual éste lo será también para el vector completo. Pero si la búsqueda de candidato para el subvector de $n - 1$ elementos no encuentra ninguno, escogeremos como candidato el n-ésimo elemento.

a) Escriba un procedimiento BUSCARCANDIDATO que utilizando la descripción anterior devuelva el candidato a ser elemento mayoritario de un vector.

b) ¿Qué técnica de diseño de algoritmo sigue el procedimiento? Justifica tu respuesta.

c) Calcular la complejidad temporal en función de n del procedimiento BUSCARCANDIDATO.

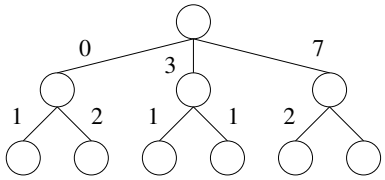
d) Como se podría eliminar el vector auxiliar b. Explica una posible solución.

6. Angueto, el perro de Carlitos sufre de incontinencia y no puede caminar mas de k metros sin orinar. En el camino por el cual Carlitos pasea su perro desde su casa a la de su tía hay n árboles que se encuentran a distancias $d_1, d_2, \ldots, d_n$ de su casa. Afortunadamente, $\forall i \in \{1, \ldots, n-1\}, ||d_{i+1} - d_i|| < k$, y por lo tanto Carlitos puede llevar su perro hasta la casa de su tía sin meterse

en problemas con los vecinos porque su perro orina puertas, bicicletas y transeúntes. Para llegar temprano a lo de su tía Carlitos, quiere hacer la menor cantidad posible de paradas en árboles. Ayude a Carlitos a encontrar la estrategia óptima y demuestre que lo es.

7. Dado un árbol con pesos no negativos en los lados, se quiere resolver el problema de encontrar el mínimo camino desde la raíz hasta una hoja. Considere el algoritmo greedy que, partiendo desde la raíz como nodo actual, elige siempre el lado l con menor peso. El nuevo nodo actual es el apuntado por l. El algoritmo termina cuando el nodo actual es una hoja.

- Encuentre un ejemplo en que el camino encontrado por el algoritmo no es óptimo
- Encuentre una condición C sobre árboles de manera tal que, si un árbol cumple con C, entonces el camino encontrado por el algoritmo en ese árbol es óptimo. El árbol que se muestra en la figura debe cumplir la condición. Sugerencia: piense en árboles con pesos muy grandes cerca de la raíz y pesos muy chicos cerca de las hojas.



- Pruebe que la condición encontrada en el punto anterior asegura que el algoritmo encuentra el óptimo

8. En el campo de la teoría de grafos un matching o conjunto independiente de arcos en un grafo es un conjunto de arcos sin vértices en común. En un grafo con pesos, un matching de peso máximo es aquel que maximiza el peso total de sus arcos.

a) Describir un algoritmo greedy que dado un grafo (E, V) intente calcular un matching de peso máximo.

b) ¿Es óptimo el algoritmo anterior? En caso afirmativo, dé una prueba de ello, razonando por el absurdo. En caso negativo, dé un contraejemplo.

9. Mauricio quiere armar un CD con publicaciones para la cátedra de ED. Para ello ha recopilado durante muchos años distintos textos y le ha asignado un puntaje según la importancia de cada uno. Sólo quiere armar un único CD y no caben todos los papers por lo que para crear la mejor recopilación debe maximizar la puntuación total de los archivos que contenga, teniendo en cuenta que estos archivos no se pueden dividir.

a) Explica por que no existe una estrategia greedy optima para este problema.

b) Dar una estrategia Greedy para encontrar una solución adecuada para este problema.

c) En que casos la solución sub-optima del item anterior daría una solución optima.

10. Supongamos que disponemos de n trabajadores y n tareas. Sea $b_{ij} > 0$ el coste de asignarle el trabajo j al trabajador i. Una asignación de tareas puede ser expresada como una asignación de los valores 0 ó 1 a las variables x_{ij}, donde $x_{ij} = 0$ significa que al trabajador i no le han asignado la tarea j, y $x_{ij} = 1$ indica que sí. Una asignación válida es aquella en la que a cada trabajador sólo le corresponde una tarea y cada tarea está asignada a un trabajador. Dada una asignación válida, definimos el coste de dicha asignación como:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}b_{ij}$$

Diremos que una asignación es óptima si es de mínimo coste. Para diseñar un algoritmo greedy para resolver este problema podemos pensar en dos estrategias distintas: asignar a cada trabajador la mejor tarea posible, o bien asignar cada tarea al mejor trabajador disponible. Sin embargo, ninguna de las dos estrategias tiene por qué encontrar siempre soluciones óptimas.

a) Resolver el problema planteado en el enunciado anterior utilizando la técnica de Greedy, para cada una de las las dos estrategias planteadas.

b) ¿Es óptima algunas de las soluciones planteadas en el item anterior? En caso afirmativo demostrarlo, de lo contrario dar un contra ejemplo.

11. Dado un tablero de ajedrez y una casilla inicial, queremos decidir si es posible que un caballo recorra todos y cada uno de los escaques sin duplicar ninguno. No es necesario en este problema que el caballo vuelva al escaque de partida. Un posible algoritmo greedy decide, en cada iteración, colocar el caballo en la casilla desde la cual domina el menor número posible de casillas aún no visitadas.

a) Implementar dicho algoritmo a partir de un tamaño de tablero $n \times n$ y una casilla inicial (x_0, y_0).

b) Buscar, utilizando el algoritmo realizado en el apartado anterior, todas las casillas iniciales para los que el algoritmo encuentra solución.

c) Basándose en los resultados del apartado anterior, encontrar el patrón general de las soluciones del recorrido del caballo.

12. Suponiendo que el sistema monetario de un país está formado por monedas de valores $v_1, v_2, \ldots, v_n$, el problema del cambio de dinero consiste en descomponer cualquier cantidad dada M en monedas de ese país utilizando el menor número posible de monedas. En primer lugar, es fácil implementar un algoritmo ávido para resolver este problema, que es el que sigue el proceso que usualmente utilizamos en nuestra vida diaria. Sin embargo, tal algoritmo va a depender del sistema monetario utilizado y por ello vamos a plantearnos dos situaciones para las cuales deseamos conocer si el algoritmo greedy encuentra siempre la solución óptima:

a) Suponiendo que cada moneda del sistema monetario del país vale al menos el doble que la moneda de valor inferior, que existe una moneda de valor unitario, y que disponemos de un número ilimitado de monedas de cada valor.

b) Suponiendo que el sistema monetario está compuesto por monedas de valores $1, p_1, p_2, \ldots, p_n$, donde $p > 1$ y $n > 0$, y que también disponemos de un número ilimitado de monedas de cada valor.

13. Dados n elementos $e_1, e_2, \ldots, e_n$ con pesos $p_1, p_2, \ldots, p_n$ y beneficios $b_1, b_2, \ldots, b_n$, y dada una mochila capaz de albergar hasta un máximo de peso M (capacidad de la mochila), queremos encontrar las proporciones de los n elementos $x_1, x_2, ..., x_n$ $(0 \leq x_i \leq 1)$ que tenemos que introducir en la mochila de forma que la suma de los beneficios de los elementos escogidos sea máxima, siguiendo una estrategia Greedy y minimizando el costo algorítmico. Esto es, hay que encontrar valores $(x_1, x_2, ..., x_n)$ de forma que se maximice la cantidad $\sum_{i=1}^{n} b_i\ x_i$, sujeta a la restricción $\sum_{i=1}^{n} p_i x_i \leq M$

4 Distribución de los ejercicios por Grupo

Apellido	Nombre	**Grupo**	Eje. Dinámica	Eje. Greedy
Canut	Iker	1	3	11
Fernandez	Agustin	1	3	11
Barleta	Luciano	2	9	10
Formichella	Bruno	2	9	10
Mondino	Georgina	2	9	10
Chol	Alejo Martin	3	8	1
Peinado	Victoria	3	8	1
Aseguinolaza	Luis	4	2	2
Gonzalez	Alejo	4	2	2
Nilsen	Maximiliano	5	4	7
Uliassi	Manuel	5	4	7
Antunez	Joaquin	6	1	3
Quinteros	Iago	6	1	3
Pitinari	Tomás	7	10	13
Rodriguez	Sebastian	7	10	13
Rodriguez	Nerina	8	12	6
Roggeroni	Victoria	8	12	6
Antonelli	Tomás	9	13	12
Mendez	Amadeo	9	13	12
Coronel	Santiago	10	7	5
Mestre	Sebastian	10	7	5
Caporale	Damian	11	6	4
Ferreira	Tomás	11	6	4
Ferreyra	Camila Anahí	12	5	8
Martinez Castro	Martin	12	5	8
Aguirre	Dante	13	11	9
Roma	Martin	13	11	9