

AG1 – Diseño de algoritmos

03MIAR – Actividad Guiada 1(AG1)

Copiar el cuaderno(plantilla) de google colab:

https://colab.research.google.com/drive/1doATrEpzh3FsFYUyl5ErzvPC_SjGRHQL

Agenda

- 0.Entradas en el foro
- 1.Desarrollo de algoritmo con la técnica de **divide y vencerás**(Torres de Hanoi)
- 2.Desarrollo de algoritmo **voraz** para resolver problemas(devolución de cambio)
- 3.Desarrollo de algoritmo con la técnica de **vuelta atrás**(backtracking)(N-Reinas)
- 4.Desarrollo de algoritmo con **Programación dinámica**(paseo por el rio)

Aportaciones en el foro

Dijkstra y Floyd-Warshall para El descenso por el río

Table 3 Comparison Between Numerical and Experimental Results

	The Dijkstra algorithm	The Bellman-Ford algorithm	The Floyd-Warshall algorithm
space complexity	$O(M)$	$O(M)$	$O(N^2)$
time complexity	$O(N^2)$	$O(MN)$	$O(N^3)$
The edge weights are negative	×	✓	✓
M is the number of edges N is the number of nodes			



David Ángel Lozano García
Problema del viaje por el río (Dijkstra)



Karen Oliveros Félez
RE: Problema del viaje por el río (Dijkstra)

publicado hace 20 horas

En este caso, quizá es más eficiente utilizar el algoritmo de Floyd-Warshall, pues se ajusta un poco mas a nuestras necesidades.

Para todos los pares origen/destino

Aportaciones en el foro

Ejemplo de A* para encontrar un recorrido



Jorge Vergara Roa

Hace 22 horas

Ejemplo de A* para encontrar un recorrido

Para los que os quedasteis con la duda de como realizar una aplicación real sobre un algoritmo de búsqueda en Python del algoritmo A* aquí os adjunto un enlace donde podéis ver el desarrollo y poderlo ejecutar.

Antes de nada para crear esta ejecución, me he basado en el pseudocodigo que hay en la Wikipedia Pseudocodigo Wikipedia

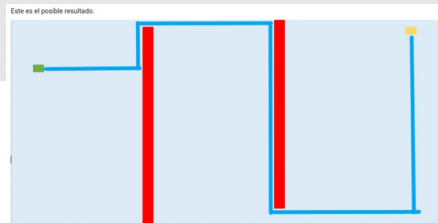
El funcionamiento parece enrevesado pero no lo es, vamos analizarlo.

En primer lugar tenemos que calcular para el punto de inicio la heurística de ese nodo, en este caso es la distancia de Manhattan, y añadir ese nodo a la lista de "openset", que indica los nodos que son propensos a ser visitados. En el caso de la lista "closeSet" son los nodos ya visitados. Con esas 2 listas son las que voy a ir jugando para coger o eliminar elementos.

Y luego lo que dice el algoritmo es lo siguiente:

1. Elegimos el nodo con la heurística mas baja.
2. Si el nodo seleccionado es igual al nodo destino, hemos llegado.
3. Se elimina el nodo seleccionado de la lista "openset" y se añade a la lista "closeSet"
4. Calculamos los vecinos que puede tener ese nodo, y para cada vecino:
 1. Comprobamos que no se halla evaluado, ni que sea pared.
 2. Calculamos su heurística
 3. Sumamos a G el valor de incremento en el paso, en mi caso 1
 4. Le asignamos el nodo padre, para luego seguir la traza.

- No tanto explicar el código sino la modelización



Aportaciones en el foro

8-puzzle con búsqueda en profundidad



Javier Jiménez de la Jara

Hace 1 día

8-puzzle con búsqueda en profundidad

Buenas tardes a todos,

En esta entrada al foro voy a resolver el 8-puzzle con el algoritmo de búsqueda en profundidad. Espero que os sea interesante. Primero explicaré cómo he modelado el problema y luego explicaré un poco la búsqueda.

La variable principal de la clase Puzzle es una matriz cuadrada de numpy que contiene el puzzle. En ella se guardarán los nombres de las casillas reprensatadas con números del 1 al 8 y una casilla vacía, marcada con un 0. Esta última será la que se vaya moviendo por la matriz siendo intercambiada con alguna de sus adyacentes. El constructor de la clase genera un puzzle moviendo las fichas de forma aleatoria a partir de la solución. De esta forma nos aseguramos que cada puzzle inicial siempre va a tener una solución. Luego hace falta determinar qué movimientos se pueden hacer y una operación que los realice. Finalmente será necesario una función que devuelva el valor heurístico del puzzle actual. En este caso simplemente se han contado el número de casillas que están en la posición correcta, aunque se podrían utilizar otras medidas como la distancia de Manhattan.

```
class Puzzle:
    def __init__(self):
        # El cero será la casilla vacía
        self.puzzle_sol = np.array([
            [1,2,3],
            [8,0,4],
            [7,6,5]
```

- ¿Gestión de bucles?

- ¿Todas las disposiciones tienen solución?


Para resolver el problema se ha utilizado el algoritmo de búsqueda por anchura. Es uno de los más sencillos y evalúa todos los posibles movimientos que se vayan a hacer. Lo bueno de utilizar este método es que para puzzles fáciles encontrará la mejor solución muy rápido, sin necesidad de aplicar ningún tipo de poda, pero para puzzles más complejos necesitará mucho tiempo y espacio.


Una idea de poda sería llevar una solución global y que todos los nodos que no la mejoren se eliminen. Pero para resolver este puzzle, muchas veces hay que pasar por estados que empeoran la heurística para llegar a la solución final. Esto ocurre de forma similar con otros puzzles, como el cubo de Rubick. Por esta razón la única "poda" que se ha aplicado, consiste en no permitir que se realice el mismo movimiento dos veces seguidas. Esto podría generalizarse para eliminar bucles que generen el mismo resultado, pero recorriendo el árbol de decisión por anchura puede ser muy complejo. Para plantear podas y heurísticas que tengan en cuenta movimientos anteriores sería recomendable utilizar otras formas de recorrer el árbol en profundidad.

Finalmente se ha determinado un máximo de nodos, porque llegada a cierta profundidad tarda mucho tiempo y necesita mucha memoria.


Aportaciones en el foro

Curiosidades fósiles: A* en Pascal...




Manuel Esteban-Infantes
Curiosidades fósiles: A* en Pascal... 


Para los que os guste la arqueología, os dejo una implantación de TurboPascal, pero con pocos retoques ahora funciona en Free Pascal. El programa va pintando en pantalla las soluciones que va probando que ni se ve. Si no marcáis una salida (G goal - está en inglés) el laberinto que resultan en búsquedas muy ineficientes. La estimación puedo mandar el ejecutable.

 496HW5.PAS (27,507 KB)


Responder **CITAR** **EDITAR** **ELIMINAR**

- Aunque no es propósito de la asignatura, ¿cómo ejecutar Pascal, Prolog...?



Manuel Esteban-Infantes
... y Prolog 


Y esto es otro A* en Prolog. Estaba escrito para Quintus Prolog, pero todo se andará. No hace falta saber Prolog para intuir el funcionamiento de otros estudiantes, no A* -- y la estimación de esfuerzo es un test que preocuparse de las fruslerías de pintar laberintos ni soluciones.

 ASTAR.PL (2,748 KB)

Aportaciones en el foro

MergeSort Recursivo

☐


**David Calvente Nomdedeu**
MergeSort Recursivo

Hace 2 días

He encontrado una web eb la que está el mismo caso que se explica en los apuntes (ver: <https://www.educative.io/edpresso/merge-sort-in-python>)

Yo lo he hecho algo diferente jugando con el pop de la lista y condicionando a que las listas queden vacías.

Además he implementado el merge en un método a parte, pero ha sido necesario hacer un clear de la lista y pasarla como parámetro para que el algoritmo recursivo funcione correctamente

**José Ignacio Hernández Velasco**
RE: MergeSort Recursivo

Hace 2 días

Buena solución, yo he encontrado otra aproximación que de la misma forma divide el algoritmo en dos funciones separadas.
<https://pythondiario.com/2018/08/ordenamiento-por-mezcla-merge-sort.html>

En primer lugar comprueba la longitud de la lista y la divide trozo a trozo, y de cada trozo la funcion merge intercala sus valores.

```
def merge_sort(lista):
```

Aportaciones en el foro

Otros



Jessica Costoso Martín
Problema devolver cambio monedas

Buenos días, en la última clase se planteó el problema de



Marc Badosa Samsó
Problema de las N reinas

Buenos días, después de investigar y probar, he conseguido resolver
pequeño resumen de que trata el problema así como también una e



Iván Pazo Pérez
Algoritmo mejorado de elementos comunes de dos listas (Clase VC1)

```
def comunes(A, B):  
    A_sorted = A.copy()  
    B_sorted = B.copy()
```



Jessica Costoso Martín
Solución fibonacci

Buenas, he estado trabajando sobre el problema de fibonacci que hemos visto en la clase de hoy.
Adjunto un pantallazo con mis soluciones (no soy capaz de insertarlas en el comentario).

La primera función 'fibonacci' realiza el cálculo de forma recursiva. La cual tiene un coste de $O(2^n)$ como he



Álvaro Chávarri Ruiz
Algoritmo Bubble Sort usando lista random de numpy



María Taboada Pena
Algoritmo de la burbuja recursivo



Cristina Reyes Daneri
Posible código para algoritmo de burbuja

Es muy posible que existan formas más eficientes de implementar este algoritmo de

- Alberto Albadelejo
- Jéssica Costoso
- Ramón Touza
- Daniel Molina
- José Ignacio Hernández
- Fabricio Tipantocta
- Manuel Esteban-Infantes
- David Calvente

Preparar la actividad en Google Colaboratory.

- Abrir un notebook en Google Colaboratory
- Abrir la plantilla: https://colab.research.google.com/drive/1doATrEpzh3FsYUyI5ErzvPC_SjGRHQL

```
Actividad Guiada 1 de Algoritmos de Optimización
Nombre: XXXXX XXXXXXXX XXXXXX
https://colab.research.google.com/drive/xxxxxxxxxxxxxxxxxxxxx
https://github.com/mi_usuario/03MAIR--Algoritmos-de-Optimizacion

[ ] #Torres de Hanoi - Divide y venceras
#####
#....

[ ] #Cambio de monedas - Técnica voraz
#####
#....

[ ] #N Reinas - Vuelta Atrás.
#####

#Verifica que en la solución parcial no hay amenazas entre reinas
def es_prometedora(SOLUCION,etapa):
    #print(SOLUCION)
    #Si la solución tiene dos valores iguales no es valida => Dos reinas en la misma fila
    for i in range(etapa+1):
        #Verifica si el valor i + 1 es igual al valor i + 1 en la misma fila
```

Preparar la actividad en Google Colaboratory.

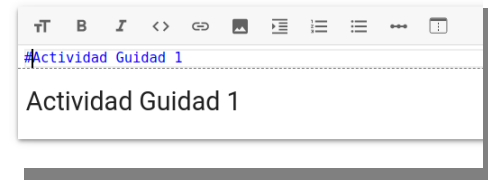
- Abrir un notebook en Google Colaboratory
- Renombra el documento python : **Algoritmos - <nombre apellido> - AG1**
- Crear un texto con:
 - * AG1- Actividad Guiada 1
 - * Nombre Apellidos
 - * Url a la carpeta AG1 de GitHub



Ayuda para texto en Google Colab(Jupyter)

- Markdown:

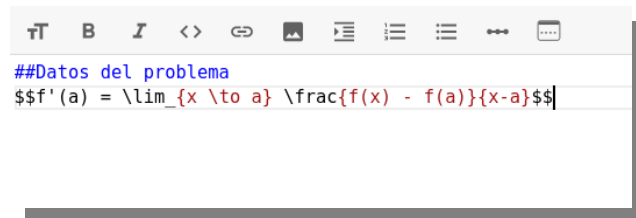
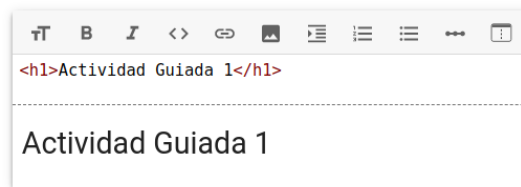
<https://www.math.ubc.ca/~pwalls/math-python/jupyter/markdown/>



- Latex:

<https://www.math.ubc.ca/~pwalls/math-python/jupyter/latex/>

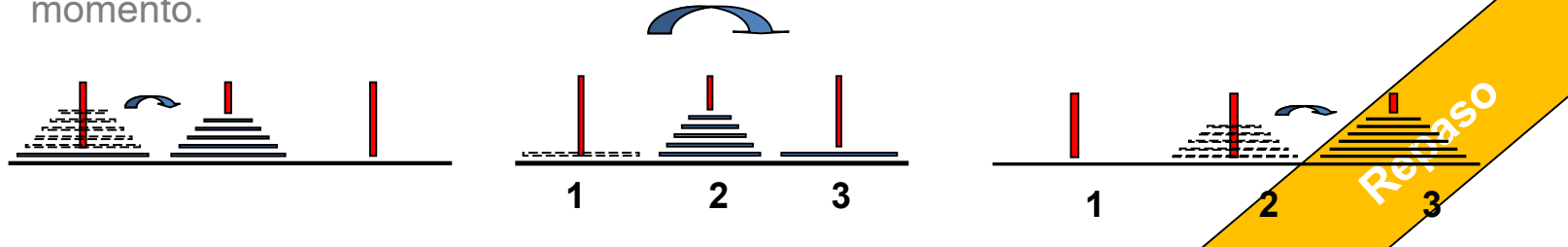
- HTML



$$f'(a) = \lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a}$$

Divide y vencerás (I)

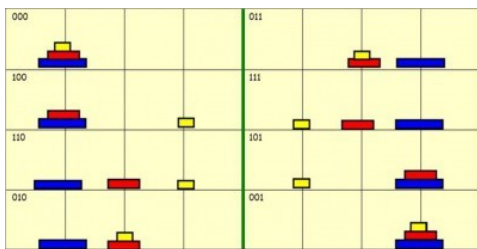
- Características que permiten identificar problemas aplicables:
 - ✓ El problema puede ser **dividido** en problemas mas pequeños pero de la misma naturaleza que el principal.
 - ✓ Es posible resolver estos sub-problemas de manera recursiva o de otra manera sencilla (**caso simple**).
 - ✓ Es posible **combinar** las soluciones de los sub-problemas para componer la solución al problema principal.
 - ✓ Los sub-problemas son **disjuntos** entre si. No hay solapamiento entre los sub-problemas.
 - ✓ Debemos asegurar que el proceso de divisiones recursivas **finaliza** en algún momento.



Divide y vencerás (II)

Problema: Torres de Hanoy. Código Python

- Solución

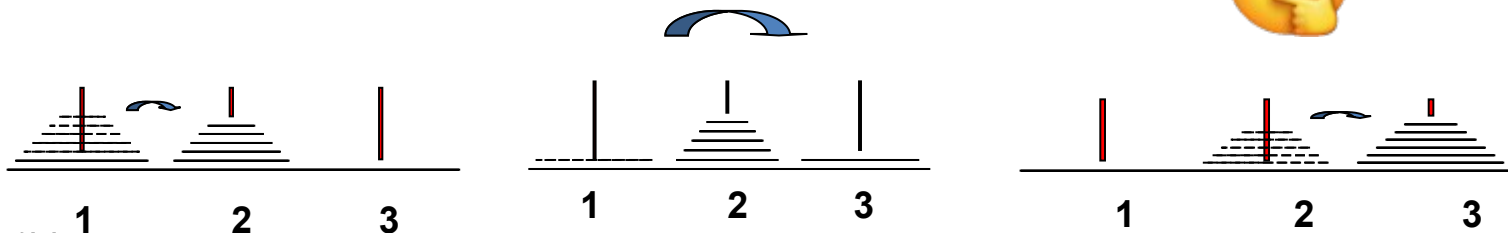


Fuente: <https://innovacioneducativa.upm.es/pensamientomatematico/node/76>

```
#Torres de Hanoi
def torres_hanoi(N, desde, hasta):
    if N==1:
        print("Llevar desde " + str(desde) + " hasta " + str(hasta) )
    else:
        torres_hanoi(N-1,desde,6-desde-hasta )
        print("Llevar desde " + str(desde) + " hasta " + str(hasta) )
        torres_hanoi(N-1,6-desde-hasta , hasta )

torres_hanoi(4,1,3)
```

```
Llevar desde 1 hasta 2
Llevar desde 1 hasta 3
Llevar desde 2 hasta 3
Llevar desde 1 hasta 2
Llevar desde 3 hasta 1
Llevar desde 3 hasta 2
Llevar desde 1 hasta 2
Llevar desde 1 hasta 3
Llevar desde 2 hasta 3
Llevar desde 2 hasta 1
Llevar desde 3 hasta 1
Llevar desde 2 hasta 3
Llevar desde 1 hasta 2
Llevar desde 1 hasta 3
Llevar desde 2 hasta 3
```



Divide y vencerás(III)

Recursividad y calculo del numero de operaciones

```
#Torres de Hanoi

def torres_hanoi(N, desde, hasta):
    if N==1:
        print("Llevar desde " + str(desde) + " hasta " + str(hasta) )
    else:
        torres_hanoi(N-1,desde, 6-desde-hasta )
        print("Llevar desde " + str(desde) + " hasta " + str(hasta) )
        torres_hanoi(N-1, 6-desde-hasta , hasta )
    torres_hanoi(4,1,3)
```

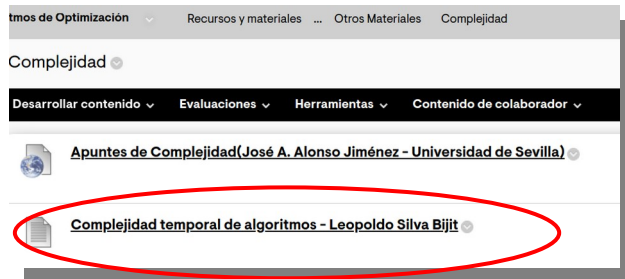
Operaciones

$$T(1) = 1$$

$$T(n) = 1 + 2 \cdot T(N-1) = 1 + 2 \cdot (1 + 2 \cdot T(n-2)) = 1 + 2 + 2^2 + \dots + 2 \cdot T(1) = 2^n - 1$$

Complejidad

$$O(2^n)$$



Técnica Voraz. Algoritmos voraces(I) - (*Greedy algorithms*)

- **Definición:** Basada en la división del problema por etapas, eligiendo en cada etapa una decisión para construir la solución que resulte la más adecuada o ambiciosa sin considerar las consecuencias. Las decisiones descartadas será descartadas para siempre. **Resumen: elegir en cada etapa la decisión óptima.**
- Características que permiten identificar problemas aplicables:
 - ✓ Conjunto de candidatos (elementos seleccionables por etapas)
 - ✓ Solución parcial
 - ✓ **Función de selección** para determinar el mejor candidato en cada etapa.
 - ✓ Función objetivo
 - ✓ Función de factibilidad que asegure que una selección parcial es “prometedora”
 - ✓ Criterio o función que compruebe que una solución parcial ya es una solución final.

Técnica Voraz. Algoritmos voraces(II) - (*Greedy algorithms*)

- Problema: Cambio de monedas

Buscar las monedas para completar la cantidad con el sistema [25, 10, 5, 1]

- Definimos la función : **cambio_monedas** con dos parámetros: **Cantidad a calcular** y **sistema monetario**
- Inicializamos la variable(lista) **SOLUCION** a cero con tantos valores como tipos de monedas.
- Inicializamos la variable **VALOR_ACUMULADO** para contener el valor acumulado actual
- Recorremos todas las monedas en orden decreciente en valor (voracidad)
 - Calculamos el máximo de monedas posibles en cada iteración:
monedas = int((CANTIDAD-VALOR_ACULULADO)/SISTEMA[i])
 - Actualizamos: **SOLUCION** y **VALOR_ACUMULADO**
 - Si llegamos a la cantidad devolvemos la solución:
if VALOR_ACULULADO == CANTIDAD: return SOLUCION



Técnica Voraz. Algoritmos voraces(III) - (*Greedy algorithms*)

```
#Cambio de monedas
#####

def cambio_monedas(CANTIDAD,SISTEMA):

    print("SISTEMA:")
    print(SISTEMA)

    SOLUCION = [0 for i in range(len(SISTEMA)) ]           #Inicializamos el array que contendrá la cantidad de monedas de cada valor
    VALOR_ACULULADO = 0                                     #Inicializamos el valor acumulado

    for i in range(len(SISTEMA)):                           #Recorremos el sistema monetario (Conjunto de candidatos)
        monedas = int( (CANTIDAD-VALOR_ACULULADO)/SISTEMA[i]) #Calcula la cantidad de monedas de valor SISTEMA[i] (Función de selección)
        SOLUCION[i] = monedas                                #Añade el numero de monedas a la solución
        VALOR_ACULULADO += monedas * SISTEMA[i]              #Incrementa el valor acumulado (Función de factibilidad)
        if VALOR_ACULULADO == CANTIDAD: return SOLUCION      #finalizamos si ya hemos llegado a la solución(Criterio de solución final)

    return SOLUCION

SISTEMA = [25, 10, 5, 1]
cambio_monedas(27, SISTEMA)

SISTEMA:
[25, 10, 5, 1]
[1, 0, 0, 2]
```

Técnica Voraz. Algoritmos voraces(IV) - (*Greedy algorithms*)

```
#Cambio de monedas
#####

def cambio_monedas(CANTIDAD,SISTEMA):

    print("SISTEMA:")
    print(SISTEMA)

    SOLUCION = [0 for i in range(len(SISTEMA)) ]
    VALOR_ACULULADO = 0

    for i in range(len(SISTEMA)):
        monedas = int( (CANTIDAD-VALOR_ACULULADO)/SISTEMA[i])
        SOLUCION[i] = monedas
        VALOR_ACULULADO += monedas * SISTEMA[i]
        if VALOR_ACULULADO == CANTIDAD: return SOLUCION

    return SOLUCION
```

- ¿Qué ocurre con otros sistemas monetarios?

```
Sistema_Monetario = [ 11, 5, 1]
```

```
print(cambio_monedas(N=15, SM=Sistema_Monetario))
```

```
[1, 0, 4]
```

- ¡Ojo! No siempre es funciona
- ¿Cuando funciona bien y cuando no?

En el Foro



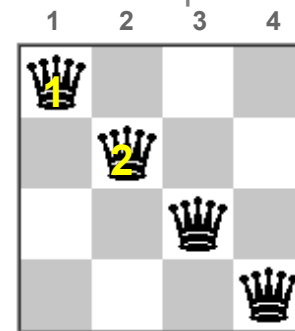
Algoritmo con la técnica vuelta atrás con Python(I). Backtracking

- **Definición:** Construcción sistemática y por etapas de todas las soluciones posibles que pueden representarse mediante una tupla (x_1, x_2, \dots, x_n) en la que cada componente x_i puede explorarse en la etapa i -ésima. A través de un **árbol de expansión** se modela todo el espacio de soluciones donde cada nodo es un valor diferente para cada elemento x_i .
- Características que permiten identificar problemas aplicables:
 - ✓ Problemas discretos en los que las soluciones se componen de elementos que pueden ser relacionados para expresarlos en un árbol de expansión.
 - ✓ Es posible encontrar un criterio para descartar determinadas ramas (ramificación y poda[*]) y evitar un análisis exhaustivo (fuerza bruta)

(*) La veremos más adelante asociada a la técnica general de búsqueda en árboles

Algoritmo con la técnica vuelta atrás con Python(II). Backtracking

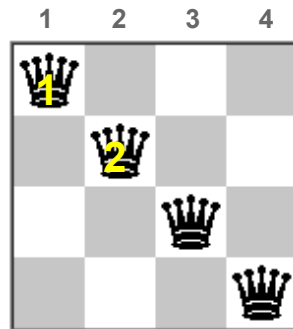
- **Problema: Problema de las 4 reinas**
 - Solución : 4-tuplas (x_1, x_2, x_3, x_4) donde el valor de cada elemento es la posición de una reina en la columna i -ésima. P.ej la del dibujo es $(1, 2, 3, 4)$
 - El árbol de expansión recorrerá todas las posibilidades.
 - Con este modelo, es posible determinar si una solución parcial (rama del árbol) es “prometedora”
 - No puede haber dos reinas en la misma columna. Esta restricción se verifica por el modelo que hemos adoptado
 - Dos reinas estarán en la misma fila si hay dos valores iguales para una solución parcial.
- P.Ej: $(1, 2, *, *)$ representa las dos primeras reinas de la imagen (2ª etapa)
- Dos reinas estará en la misma diagonal si $|x_i - x_j| = |i - j|$



Repaso

Algoritmo con la técnica vuelta atrás con Python(III). Backtracking

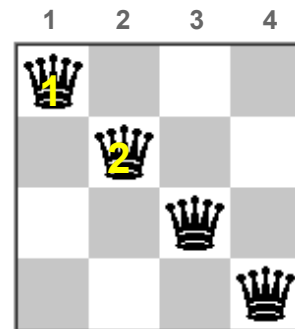
- **Problema: Problema de las 4 reinas**
 - Usaremos la recursividad (habitual también en la técnica de vuelta atrás) para ir construyendo en cada iteración una etapa de la solución (una nueva reina en la siguiente columna)
 - Definimos una función: **reinas(N, solución , etapa)**
 - N = n° de reinas del tablero NxN
 - solución = [0,0....0]
 - Etapa = 0
 - ¿Qué hace la función **reinas()** ?
 - 1º añade una etapa : **SOLUCION[etapa] = i** ; para i desde 1 hasta N
 - 2º comprueba que es prometedor con la función: **es_prometedora(solucion,etapa)**
 - 3º si la solución es prometedor y es la última etapa entonces es solución final



Algoritmo con la técnica vuelta atrás con Python(IV). Backtracking

- **Problema: Problema de las 4 reinas**

- ¿Qué realiza la función `es_prometedora(solucion,etapa)` ?
 - Comprueba que no hay reinas en la misma fila o lo que es lo mismo que no hay dos elementos iguales en `SOLUCION`:
`if SOLUCION.count(SOLUCION[i]) > 1: return False para i=0 hasta etapa`
 - Comprueba que no hay reinas en la misma diagonal o lo que es lo mismo que no se da $|i - j| = |X_i - X_j|$ para todos los i, j de `SOLUCION`:
`if abs(i-j) == abs(SOLUCION[i]-SOLUCION[j]) : return False`



Algoritmo con la técnica vuelta atrás con Python(V). Backtracking

- Problema: Problema de las 4 reinas con recursividad

```
#Proceso principal de N-Reinas
def reinas(N,solucion=[],etapa=0):
    # N      - Tamaño del tablero
    # solucion - Solucion parcial
    # etapa   - nº de reinas colocadas en la solución parcial(

    #Inicializa la solución: una lista con ceros
    if len(solucion) == 0:
        solucion=[0 for i in range(N)]

    #Recorremos todas las reinas
    for i in range(1,N+1):
        solucion[etapa] = i

    #print(solucion)
    if es_prometedora(solucion,etapa):
        if etapa == N-1 :
            print("\n\nLa solución es:")
            print(solucion)
            escribe_solucion(solucion)
        else:
            #print("Es prometedor\n#####")
            reinas(N,solucion,etapa+1)
    else:
        #print("NO PROMETEDORA\n#####")
        None

    solucion[etapa] = 0
```

Complejidad:

Cálculos complejos(con ecuaciones en recursividad)
pero en general Vuelta atrás es Exponencial

```
def es_prometedora(SOLUCION,etapa):
    #print(SOLUCION)
    #Si la solución tiene dos valores iguales no es valida => Dos reinas en la misma fila
    for i in range(etapa+1):
        #print("El valor " + str(SOLUCION[i]) + " está " + str(SOLUCION.count(SOLUCION[i])) + " veces")
        if SOLUCION.count(SOLUCION[i]) > 1:
            return False

    #Verifica las diagonales
    for j in range(i+1, etapa +1 ):
        #print("Comprobando diagonal de " + str(i) + " y " + str(j))
        if abs(i-j) == abs(SOLUCION[i]-SOLUCION[j]):
            return False
    return True
```

$O(n^2)$

Algoritmo con la técnica vuelta atrás con Python(VI). Backtracking

- Problema: Problema de las N reinas.

LAVANGUARDIA | Otros Deportes RESULTADOS

Al Minuto Internacional Política Opinión Vida Deportes Economía Local Gente Cultura Sucesos Temas

Directo Negociación in extremis entre PSOE y Unidas Podemos: todos los detalles de última hora

HASTA AHORA INDESCIFRABLE

Ofrecen un millón de dólares a quien resuelva este problema de ajedrez

• Unos investigadores proponen resolver el 'enigma de las ocho reinas' para tableros de 1000X1000

REDACCIÓN
06/09/2017 14:44
Actualizado a
06/09/2017 15:17



Fuente: <https://www.lavanguardia.com/deportes/otros-deportes/20170906/431089708625/ocho-reinas-ajedrez-millon-dolares.html>

Programación dinámica (I)

- **Definición:** Es posible dividir el problema en subproblemas más pequeños, guardando las soluciones para ser utilizadas más adelante.
- Características que permiten identificar problemas aplicables:
 - ✓ Es posible almacenar soluciones de los subproblemas para ser reutilizadas.
 - ✓ Debe verificar el **principio de optimalidad** de Bellman: “*en una secuencia optima de decisiones, toda sub-secuencia también es óptima*” (*)
 - ✓ La necesidad de guardar la información acerca de las soluciones parciales unido a la recursividad provoca la necesidad de preocuparnos por la complejidad espacial (cuantos recursos de espacio usaremos)



importante



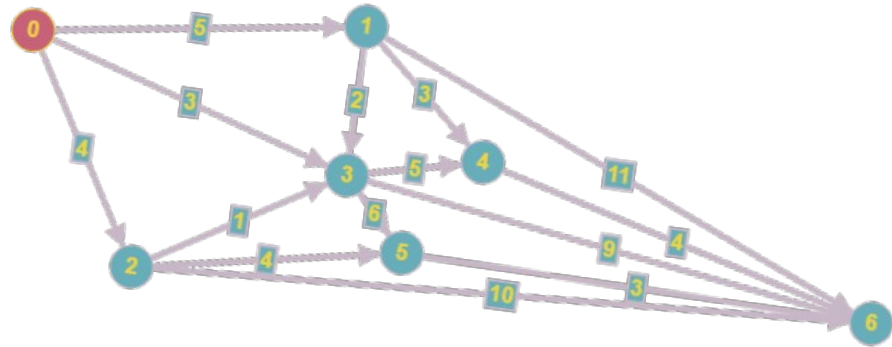
Repaso

Programación dinámica (II)

Problema: Viaje por el río

- Consideramos una tabla $T(i,j)$ para almacenar todos los precios que nos ofrecen los embarcaderos
- Si no es posible ir desde i a j daremos un valor alto para garantizar que ese trayecto no se va a elegir en la ruta óptima(modelado habitual para restricciones)
- Establecer una tabla intermedia($P(i,j)$) para guardar soluciones óptimas parciales para ir desde i a j .

$$P(i,j) = \min \{T(i,j) , P(i,k)+T(k,j) \text{ para todo } i < k \leq j \}$$



Programación dinámica (III)

Problema: Viaje por el río

- Establecemos las tarifas:

```
#Viaje por el río - Programación dinámica
```

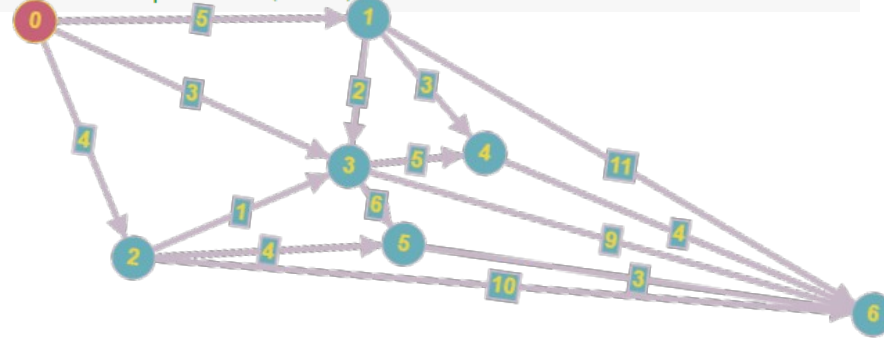
```
#####
```

```
TARIFAS = [  
[0,5,4,3,999,999,999],  
[999,0,999,2,3,999,11],  
[999,999, 0,1,999,4,10],  
[999,999,999, 0,5,6,9],  
[999,999, 999,999,0,999,4],  
[999,999, 999,999,999,0,3],  
[999,999,999,999,999,999,0]  
]
```

Se puede usar

```
import math  
float("inf")
```

```
#999 se puede sustituir por float("inf")
```



Programación dinámica (IV)

```
#####
def Precios(TARIFAS):
#####
    #Total de Nodos
    N = len(TARIFAS[0])

    #Iniciación de la tabla de precios
    PRECIOS = [ [9999]*N for i in [9999]*N]
    RUTA = [ [""]*N for i in [""]*N]

    for i in range(N-1):
        for j in range(i+1, N):
            MIN = TARIFAS[i][j]
            RUTA[i][j] = i

            for k in range(i, j):
                if PRECIOS[i][k] + TARIFAS[k][j] < MIN:
                    MIN = min(MIN, PRECIOS[i][k] + TARIFAS[k][j])
                    RUTA[i][j] = k
                    PRECIOS[i][j] = MIN

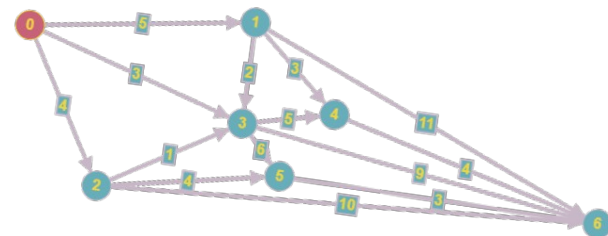
    return PRECIOS, RUTA
#####
```

Operaciones

n^1

n

$3 \cdot n^3$



```
[ ] TARIFAS = [
    [0,5,4,3,999,999,999],
    [999,0,999,2,3,999,11],
    [999,999, 0,1,999,4,10],
    [999,999,999, 0,5,6,9],
    [999,999, 999,999,0,999,4],
    [999,999, 999,999,999,0,3],
    [999,999,999,999,999,999,0]
]
```

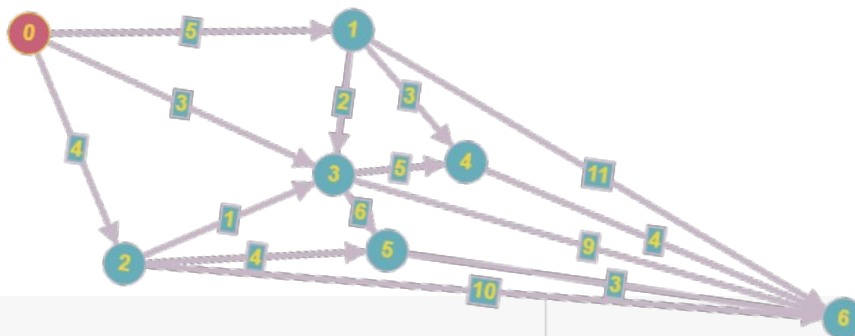
$$P(i,j) = \min \{ T(i,j) , P(i,k)+T(k,j) \text{ para todo } i < k \leq j \}$$

Programación dinámica (V)

- RUTA contiene la mejor opción intermedia para ir de un nodo a otro

```

RUTA
['', 0, 0, 0, 1, 2, 5]
['', '', 1, 1, 1, 3, 4]
['', '', '', 2, 3, 2, 5]
['', '', '', '', 3, 3, 3]
['', '', '', '', '', 4, 4]
['', '', '', '', '', '', 5]
['', '', '', '', '', '', '']
  
```



```

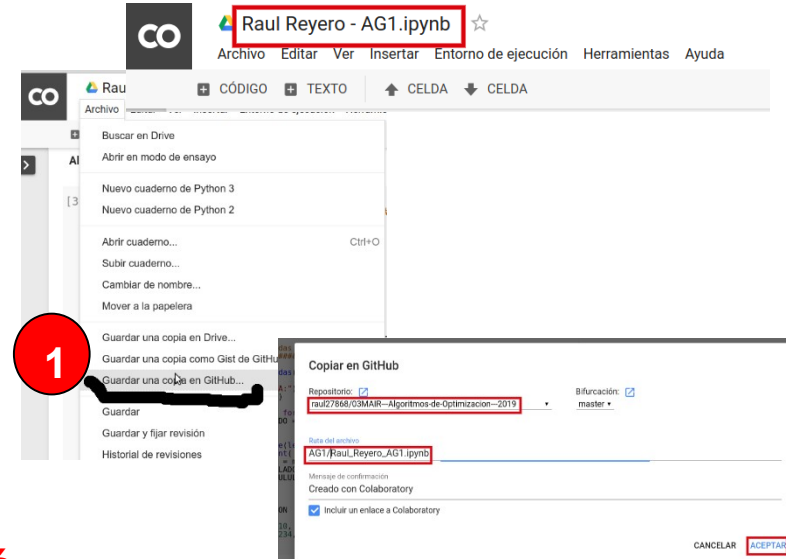
def calcular_ruta(RUTA, desde, hasta):
    if desde == hasta:
        #print("Ir a :" + str(desde))
        return desde
    else:
        return str(calcular_ruta(RUTA, desde, RUTA[desde][hasta]) ) + ',' + str(RUTA[desde][hasta])

print("\nLa ruta es:")
calcular_ruta(RUTA, 0,6)
  
```

Recursividad

Finalizar la actividad. Grabar, subir a GitHub, Generar pdf (I)

- Guardar en GitHub
Repositorio: 03MIAR ---Algoritmos de Optimizacion
Ruta de Archivo con AG1



Ojo! Si el repositorio es **privado** no se podrá.

Opciones:

- Hacerlo publico + guardar + hacerlo privado
- Guardar el .ipynb manualmente

Finalizar la actividad. Grabar, subir a GitHub, Generar pdf (I)

- Descargar pdf y adjuntar el documento generado a la actividad en la plataforma
 - Adjuntar .pdf en la actividad
 - URL GitHub en el texto del mensaje de la actividad



Practica individual



- Problema: Encontrar los dos puntos más cercanos
 - Dado un conjunto de puntos se trata de encontrar los dos puntos más cercanos
 - Guía para aprendizaje:
 - ✓ Suponer en 1D, o sea, una lista de números: [3403, 4537, 9089, 9746, 7259,
 - ✓ Primer intento: Fuerza bruta
 - ✓ Calcular la complejidad. ¿Se puede mejorar?
 - ✓ Segundo intento. Aplicar Divide y Vencerás
 - ✓ Calcular la complejidad. ¿Se puede mejorar?
 - ✓ Extender el algoritmo a 2D: [(1122, 6175), (135, 4076), (7296, 2741)...
 - ✓ Extender el algoritmo a 3D.

Practica individual



- Problema: Encontrar los dos puntos más cercanos
 - Para generar conjuntos de datos aleatorios

Práctica Individual. Dos puntos más cercanos

```
[ ] import random  
  
LISTA_1D = [random.randrange(1,10000) for x in range(1000)]  
  
LISTA_2D = [(random.randrange(1,10000),random.randrange(1,10000)) for x in range(1000)]
```

- Buscar documentación sobre el problema

Próxima clase VC4 – Descenso del gradiente

- Descenso del gradiente
- Repaso de Análisis de complejidad.
- Trabajo práctico. Enunciado de los problemas (30% nota)

Sistema de Evaluación	Ponderación
Portafolio	60 %
Trabajo Práctico(*) : 30% Actividades Guiadas(*) : 10% Estudio y análisis de un artículo científico(*) : 10% Participación en Foro(Evaluable): 10%	
Sistema de Evaluación	Ponderación
Prueba final*	40 %
10 preguntas tipo test con una sola respuesta válida. Cada respuesta válida suma 1 punto y cada respuesta fallida resta 0.33 puntos	

Próxima Actividad Guiada AG2

- Práctica: Búsqueda en grafos, ramificación y poda(problema Asignación de tareas).
- Practica: Descenso del gradiente.

Foro.

Armas de destrucción matemática – Cathy O’Neil



Raul Reyero Diez ★

Hace 3 minutos

Algoritmos. Armas de destrucción matemática. Cathy O’Neil

Dado que hemos empezado ya la asignatura quiero plantearos un debate para ser tratado no tanto desde el punto de vista técnico sino ético o filosófico.

¿Estamos haciendo lo correcto, desde otros puntos de vista que no son los puramente técnicos, con "delegar" en los algoritmos los análisis y decisiones importantes?

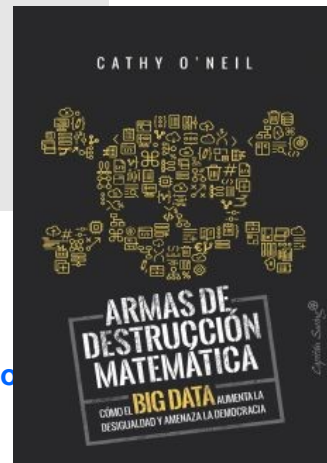
¿Debemos preocuparnos como una sociedad justa e igualitaria a la que deberíamos aspirar por la puesta en práctica en situaciones que nos afectan en primera persona como la conducción autónoma, diagnósticos médicos, concesión de créditos, selección de personal y otros tantos?

En el libro "Armas de destrucción matemática", Cathy O’Neil nos cuenta algunas situaciones que nos van a hacer, al menos reflexionar, inevitablemente. Podéis encontrarlo en algunas bibliotecas populares.
Hay un buen resumen en:

<https://www.revistadelibros.com/resenas/armas-de-destruccion-matematica-cathy-oneil>

Reflexiona sobre este asunto, **busca** información sobre algunas opiniones de diferentes perfiles (políticos, técnicos, filosóficos, afectados,...) y **comparte** tu reflexión.

Responder



Manual de la asignatura

01. Materiales docentes

Desarrollar contenido Evaluaciones Herramientas Contenido

Manual de la asignatura

Archivos adjuntos: O3MIAR_RReyero_nueva_imagen.pdf (5,02 MB)

O3MIAR | ALGORITMOS DE OPTIMIZACIÓN

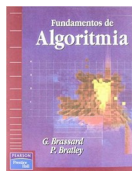
Habilitado: Seguimiento de estadísticas
Explicación y práctica de las técnicas y métodos para diseñar y analizar algoritmos orientados a la optimización.



MÁSTER UNIVERSITARIO EN INTELIGENCIA
ARTIFICIAL
Módulo de Matemáticas



Bibliografía



Fundamentos de algoritmia: Una perspectiva de la ciencia de los computadores

Paul Bratley , Gilles Brassard

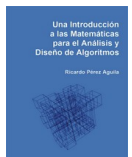
ISBN 13: 9788489660007



Introducción al diseño y análisis de algoritmos

R.C.T. Lee,...

ISBN 13: 9789701061244



Una introducción a las matemáticas para el análisis y diseño de algoritmos(*)

Pérez Aguila, R.

ISBN 13: 9781413576474

<https://tinyurl.com/yzlt5oed>



Técnicas de diseño de algoritmos

Guerequeta, R., y Vallecillo, A. (2000).

<http://www.lcc.uma.es/~av/Libro>

¿Preguntas?





Feliz Navidad

Gracias

raul.reyero@campusviu.es