PROBLEMAS NP Y P

TAREA 3

DANIELA NICOLAS
FACULTAD DE INGENIERIA | ESTRUCTURA DE DATOS Y ALGORITMOS 1

INTRODUCCION. -

En la programación un algoritmo no solamente debe realizar lo especificado, si no que debe de realizarlo también realizando el mayor número de instrucciones elementales en el menor tiempo de ejecución posible y utilizando poca memoria de almacenamiento.

Estas son medidas de complejidad, las cuales son estudiadas conforme el peor caso de cada algoritmo, el objetivo principal de estas medidas se basa en clasificar los problemas dependiendo su eficiencia para resolverlos.

Ahora que sabemos esto podemos definir de mejor manera lo que es la tratabilidad y con ello lo que son los problemas P y NP, las medidas de complejidad se encargan de ver la tratabilidad de un algoritmo, es decir en otras palabras que tan eficiente es, la eficiencia dependerá completamente del problema, un ejemplo de ello es tener un problema con una sola solución, ya que, aunque pueda ser larga, es la única que cumple con el objetivo.

Debido a lo dicho anteriormente, se han creado distintas clasificaciones de las complejidades esto es con base a los tiempos de ejecucion y a la cantidad de memoria que ocupan los algoritmos que se han creado para resonder a problemas de distintos tipos. Dentro de estas clasificaciones nos centraremos en las P y NP.

CLASE DE COMPLEJIDAD P. –

Podemos decir que un problema que se encuentra en esta clase de complejidad debe ser un problema de desición y se puede resolver en tiempo polinomial. Tiempo polinomial. Al decir que un problema corre en teimpo polinomial hacemos referencia a que la entrada n corra en cierto tiempo, siendo estrictamente usada esta notación:

D= O(n^k)

Podriamos decir que un problema que corre en tiempo polinomial es “eficiente”, pero la realidad de esto depende del grado del polinomio, ya que si es muy alto no será muy posible considerar a este como “eficiente” o “rapido”.

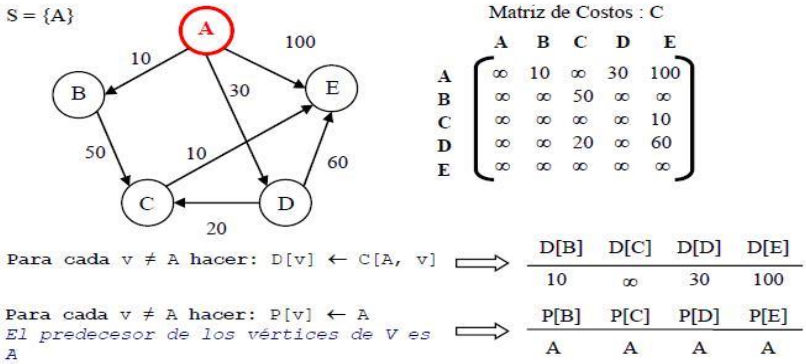
Por esto es que P contiene a la mayoría de los problemas naturale, algoritmos de programación lineal y funciones simples, entre otros.

EJEMPLOS

1.- ALGORITMO DIJKSTRA.

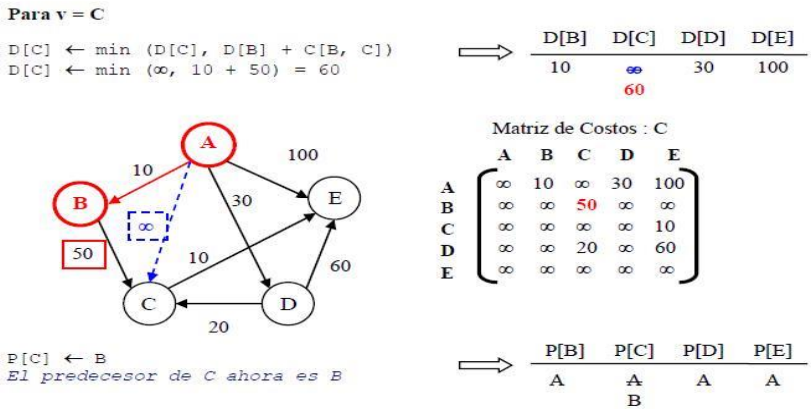
Este algoritmo también es llamado algoritmo de caminos mínimos, es un algoritmo para determinar el camino más corto desde un vertice al resto de las vertientes.

Ejecución de algoritmo

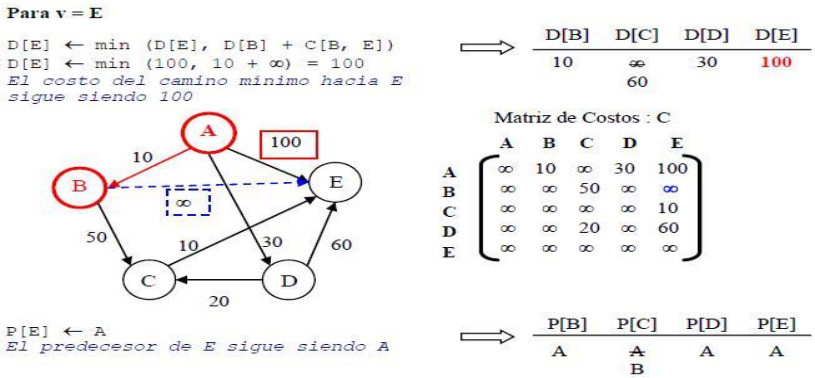


- **Paso 1: Inicialización**

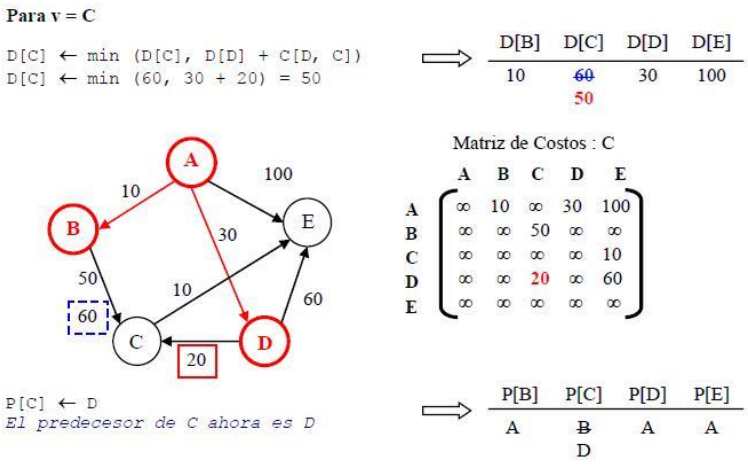
- **Paso 2: Elegir un vértice w ∈ V - {A} tal que D[w] sea mínimo, y agregar w al conjunto solución S**
- **Paso 3: cada v ∈ {C, D, E} hacer D[v] ← min(D[v], D[w]+C[w, v])**



- **Paso 4: Elegir un vértice w ∈ V - {A, B} tal que D[w] sea mínimo, y agregar w al conjunto solución S.**

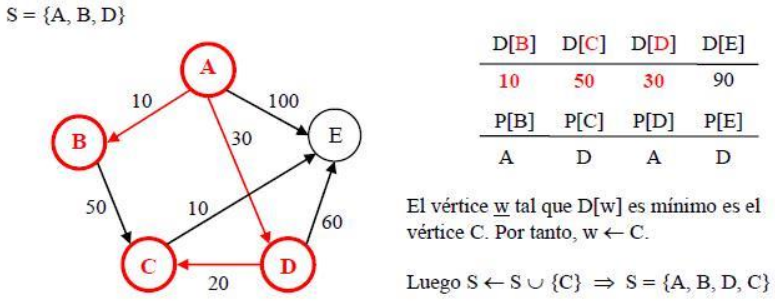


- **Paso 5: Para cada v ∈ {C, E} hacer D[v] ← min(D[v], D[w]+C[w, v]**



Paso 6: Elegir un vértice w ∈ V - {A, B, D} tal que D[w] sea mínimo, y agregar w al conjunto solución S.

- **Paso 7: Para cada v ∈ { E} hacer D[v] ← min(D[v], D[w]+C[w, v]**



- **Paso 8: Elegir un vértice w ∈ V - {A, B, D, C} tal que D[w] sea mínimo, y agregar w al conjunto solución S.**

Luego del paso 8 tenemos la siguiente situación:

- El conjunto de vértices V = {A, B, C, D, E}
- El conjunto de vértices para los que ya se determinó el camino mínimo S = {A, B, D, C, E}

Por lo que como V-S=Ø se termina el algoritmo, pues para cada nodo del grafo se ha determinado cuál es su camino mínimo. Como resultado de la ejecución del algoritmo tenemos:

D[B]	D[C]	D[D]	D[E]	P[B]	P[C]	P[D]	P[E]
10	50	30	60	A	D	A	C

Al finalizar la ejecución del algoritmo Dijkstra, en el arreglo D quedan almacenados los costos de los caminos mínimos que parten del vértice origen al resto de los vértices. Por

tanto, en nuestro ejemplo, el camino mínimo de A hacia B tiene costo 10, el camino mínimo de A hacia C tiene costo 50, el camino mínimo de A hacia D tiene costo 30 y el camino mínimo de A hacia E tiene costo 60.

Veamos cómo se pueden obtener los caminos mínimos a partir del arreglo de predecesores. Los caminos se reconstruyen partiendo del vértice destino hasta llegar al vértice origen.

CaminoMínimo (A, B) = AB ya que P[B]=A

CaminoMínimo (A, C) = ADC ya que P[C]=D y P[D]=A

CaminoMínimo (A, D) = AD ya que P[D]=A

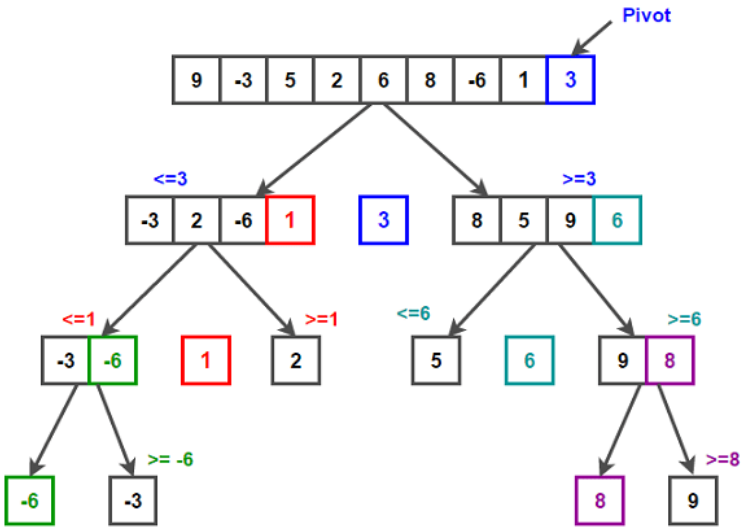
CaminoMínimo (A, E) = ADCE ya que P[E]=C, P[C]=D y P[D]=A

Finalmente, el camino mínimo que incluye todos los vértices del grafo es ADCE.

2. QUICKSORT

Es un algortimo de ordenacion funciona escogiendo un elemento del array al que denominaremos *pivote* de modo que podamos dividir el array inicial en los que son mayores y en los que son menores a el.

A continuación, para cada uno de esos dos *arrays* volveremos a escoger un pivote y repetiremos la misma operación. Llegará un momento en que los *arrays* resultantes tengan o bien un elemento o bien ninguno si ya no existe un elemento con el que compararlos. El resto de los elementos habrán sido separados como pivotes en algún punto del algoritmo y ya se encontrarán en su posición correcta



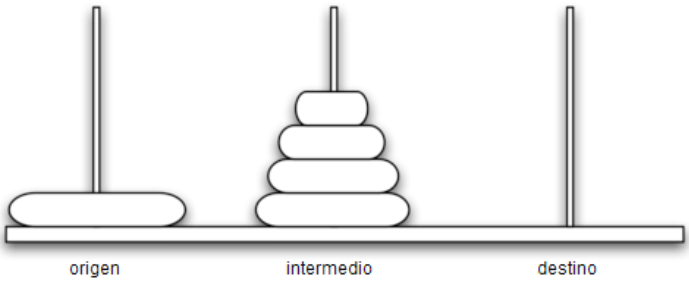
CLASE DE COMPLEJIDAD NP. –

La clase de complejidad NP contiene problemas que son de decisión, es decir, se aplica a problemas que tienen como respuesta SI o NO y que son verificables en tiempo polinomial, al decir verificar, nos referimos a comprobar si el certificado emitido por el algoritmo cumple con los requerimientos del problema.

Cuando se dice que un algoritmo no tiene una solución en tiempo polinomico, siempre se intenta encontrar otra solución que lo haga mejor.

EJEMPLOS

1.TORRES DE HANOI



El siguiente es un esquema de alto nivel de cómo mover una torre desde el poste de origen, hasta el poste destino, utilizando un poste intermedio:

1. Mover una torre de altura-1 a un poste intermedio, utilizando el poste destino.
2. Mover el disco restante al poste destino.
3. Mover la torre de altura-1 desde el poste intermedio hasta el poste destino usando el poste de origen.

Siempre y cuando obedezcamos la regla de que los discos más grandes deben permanecer en la parte inferior de la pila, podemos usar los tres pasos anteriores recursivamente, tratando cualquier disco más grande como si ni siquiera estuviera allí. Lo único que falta en el esquema anterior es la identificación de un caso base. El problema de la torre de Hanoi más simple es una torre de un disco. En ese caso, sólo necesitamos mover un solo disco a su destino final. Una torre de un disco será nuestro caso base.

2. KNAPSACK: Imagina que tenemos una mochila con capacidad para 10 kg. Tenemos 400 elementos para meter en la bolsa, cada uno de ellos con un peso y un valor. ¿Cuál es el valor máximo que podemos llevar en la bolsa?

Este problema parece muy complicado de resolver. Quizá lo único que se nos ocurra es probar todas las combinaciones posibles de elementos, descartar las combinaciones que exceden el peso que puede llevar la mochila y de las restantes quedarnos con el máximo.

Funcionaría, pero sería extremadamente lento.

Como cada elemento puede estar o no estar en la mochila, y son 400, las combinaciones son

2^{400}=258224987808690858965591917200301187432970579282922351283065935654064762201684119462964535328013783143590317197274749337 6.

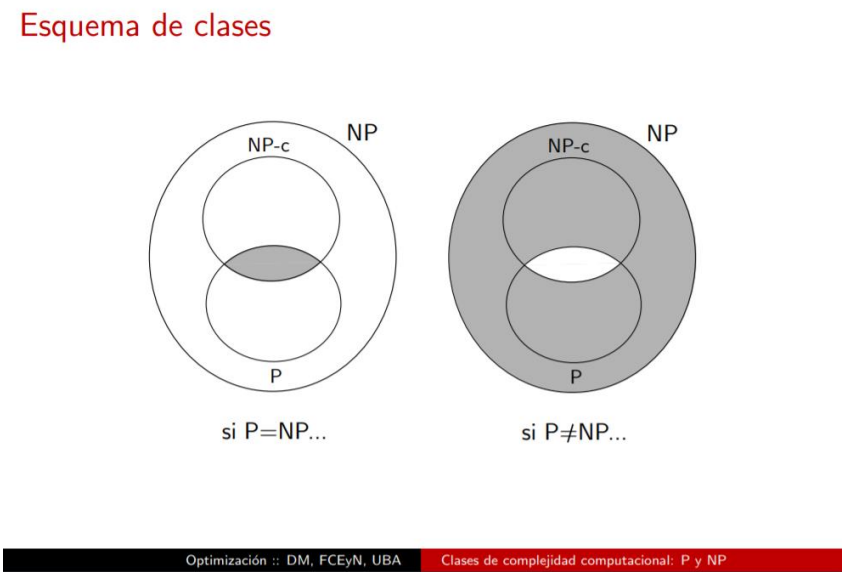
CLASE DE COMPLEJIDAD NP (completo). –

Un problema de clasificacion NP completo pertenece a la clase de complejidad NP y cuando cualquier problema NP se puede reducir polinomialmente. Se dice que son los peores problemas de la clase NP, debido a que son de extrema complejidad y a que se caracterizan por ser todas iguales, pero por eso mismo este tipo de problemas es muy importante.

El punto clave aquí es que para cualquier problema dentro de NP existe una reduccion polinomial en un problema NP − Completo, esto quiere decir que siempre podemos tomar un problema arbitrario dentro de NP y tratarlo como un problema NP − Completo. Luego, si es que podemos encontrar un algoritmo que corra en tiempo polinomial y resuelva un problema NP − Completo, entonces, por reduccion polinomial, estaríamos diciendo que cualquier problema dentro de NP puede resolverse en tiempo polinomial. Así, es ahora mucho más claro la manera en que se orienta la demostracion del problema P vs NP.

Si existe un problema en NP-Completo ∩ P, entonces P=NP.

Hasta el momento no se conoce ningún problema en NP-Completo ∩ P, así como tampoco se ha demostrado que un problema esté en NP \ P. En ese caso, obviamente, se probaría que P ≠ NP.



INTERNETGRAFÍA

http://www.cs.uns.edu.ar/~prf/teaching/AyC17/downloads/Teoria/Complejidad-1x1.pdf

http://www.amarun.net/phocadownload/userupload/Diego_Chamorro/Rev_Div_Amarun_1_2014_PvsNP.pdf

https://www.dm.uba.ar/materias/optimizacion/2006/1/fbonomo_clase_npc.pdf

https://runestone.academy/runestone/static/pythoned/Recursion/LasTorresDeHanoi.html

https://www.youtube.com/watch?v=lilBGvaOSy8#:~:text=Las%20Torres%20de%20Hanoi%20es,otro%20m%C3%A1s%20peque%C3%B1o%20que%20%C3%A9l.

https://www.genbeta.com/desarrollo/implementando-el-algoritmo-quicksort

https://slideplayer.es/slide/9561130/