Diseño y Análisis de Algoritmos

Indice

Fecha: 15/05/2017

La literatura (libro) que se usará es Algoritmos de E. Commer (3era edición)

Evaluación

Tipo	**Valor (%)**
Examen 1 (19 junio)	20%
Examen 2 (5 julio)	20%
Trabajos	30%
Conferencias	10%
Examen final	20%

Repaso de estructuras de datos:

Búsqueda binaria en datos ordenados:

Explicación del código a continuación ⇒ Primero se comienza en la mitad del arreglo, diciendo que si el número está en la mitad del arreglo se devolverá true, es decir que si existe en el arreglo. De lo contrario, si limite izquierdo es mayor al limite derecho esto significa que no están ordenados los números por lo tanto no sería posible encontrar un número con este algoritmo. Pero como están ordenados vemos que si el numero es mayor al numero en la mitad del arreglo significa que está a la derecha y se toma el limite izquierdo como medio + 1, lo cual indica que se comenzará a buscar el numero desde la mitad + 1, ya que es mayor a la mitad, e igualmente si es menor a la mitad, el limite derecho pasará a ser la mitad - 1 y este proceso se repetirá recursivamente para así ver si el número está en el arreglo A.

```
Bool bbinaria(A[], x, izq, der)
 {
                med = (izq + der)/2;
                if(x == A[med]) return true;
        if(izq > der) return false;
                if (x < A[med]) return bbinaria(A, x, izq, med -1);
                return bbinaria(A, x, med + 1, der); }
```

Hallar el máximo conjunto creciente monotómico: Esto se refiere a hallar el máximo entre un conjunto de número que puede o no ir creciendo.

Explicación del código a continuación ⇒ Primero se comienza por la mitad del arreglo y se dice que si diz(diferencia izquierda) que sería en este ejemplo 17 - 11

si tomamos al 17 como referencia, se dice que si diz es mayor a 0 y dder que sería 21 - 17 es menor a 0 se retorna ese valor significando que se encontró el máximo, pero en caso de que los resultados varían se hará igual que en búsqueda binaria acortado el límite derecho y el límite izquierdo para encontrar el valor máximo en el arreglo. En otras palabra, si en el punto a evaluar (o med) el conjunto es creciente, el medio de mueve hacia la izquierda, pero si es decreciente el medio se mueve a la izquierda de lo contrario es el punto máximo y lo hemos acabado.

Un ejemplo de esto sería: 3 7 9 11 17 21 16 12 2, donde 21 sería el máximo.

```
int bmax(A[], izq, der)
{
   med = (izq + der)/2;
   diz = A[med] - A[med - 1];
   dder = A[med + 1] - A[med];
   if((diz > 0) && (dder < 0)) return A[med];
   if((diz > 0) && (dder > 0)) return bmax(A, med + 1, der);
   if((diz < 0) && (dder < 0)) return bmax(A, izq, med -1);
}
```

Dado un conjunto de números no positivos y un número t

a) **Hallar x1, x2 tal que x1 + x2 = t.**
b) **Hallar x1, x2, x3 tal que x1 + x2 + x3 = t**
c) **Hallar x1, x2...xn tal que** $\Sigma Xi = t$

Solución al problema a)

Explicación del código a continuación ⇒ Primero se ordenan los números para que se pueda aplicar a este problema el algoritmo de búsqueda binaria. Después se recorren todos los 89números del arreglo A y se va igualando x2 a (t-x) que vendría siendo el número que sumado con x da t. Y al final buscamos si x2 existe en el arreglo.

Este problema se hace en O(nlog n)
[x1, x2] busqueda(A[], n, t)
Pasos para hacer este problema:
a) Ordenar
b) Recorrer → O(n)

```
x = A[i];
x2 = t - x;
busquedabinaria(A, x2, izq, der) → O(log n)
return [x1, x2]
```

Solución al problema b)

Explicación del código a continuación ⇒ Es una modificación del problema anterior (a). Primero se ordena el arreglo -con el objetivo de aplicar búsqueda binaria posteriormente- , luego se itera desde la primera posición hasta n, en cada iteración se hace otro recorrido pero ahora se inicia en el siguiente elemento

(i + 1) ahora con los valores del arreglo A[i] y A[j] se calcula el posible valor de C, (esto es w = t - x- y **) entonces con búsqueda binaria se determina si existe dicho valor de C.**

```
Este problema se hace en O(n²logn)
[x1, x2, x3] busqueda(A[], n, t){
Ordenar(A[], n);
Recorrer(A[], 1,n) → O(n)
        x = A[i];
        Recorrer(A[], i + 1, n); → O(n)
                y = A[j];
                w = t - x- y;
                if(busquedabinaria(A, j + 1, der, w) return [x, y, w];
                else return [];
}
```

Cosas que vimos en esta clase:

- Vimos los temas del programa que vamos a tratar.
- Vimos la diapositiva "isc405_intro.pdf".
- Vimos el documento de introducción a la clase llamado "introduccion_405.pdf.

Todos estos documentos se encuentran en la PVA.

Cosas que veremos el miércoles:

- Problema c) con fuerza bruta.
- Fibonacci con recursividad.
- La fecha de exámenes y de conferencias.

Fecha: 17/05/2017

1.- a) Suma de conjuntos (S) ¿Cuántas o cuáles sumas se pueden obtener a partir del arreglo?

Explicación del código a continuación ⇒ "n" es igual a la longitud o tamaño del arreglo, luego se itera el arreglo completo y por cada elemento de agregan todas las sumas inexistentes que pueda resultar, es evidente que este es un problema NP.

```
n = |S|

L0 = {0}   //la respuesta inicia con un conjunto vacío

for i = 1 to n

Li = Merge(Li - 1, Li -1 + Xi)
```

S + X = { s + x: s E S }
Ejemplo: { 1, 4, 5 } L0 = {0} L1 = {0, 1} L2 = {0, 1, 4, 5} L3 = {0, 1, 4, 5, 6, 9, 10}

1.- b) Aproximar (S)

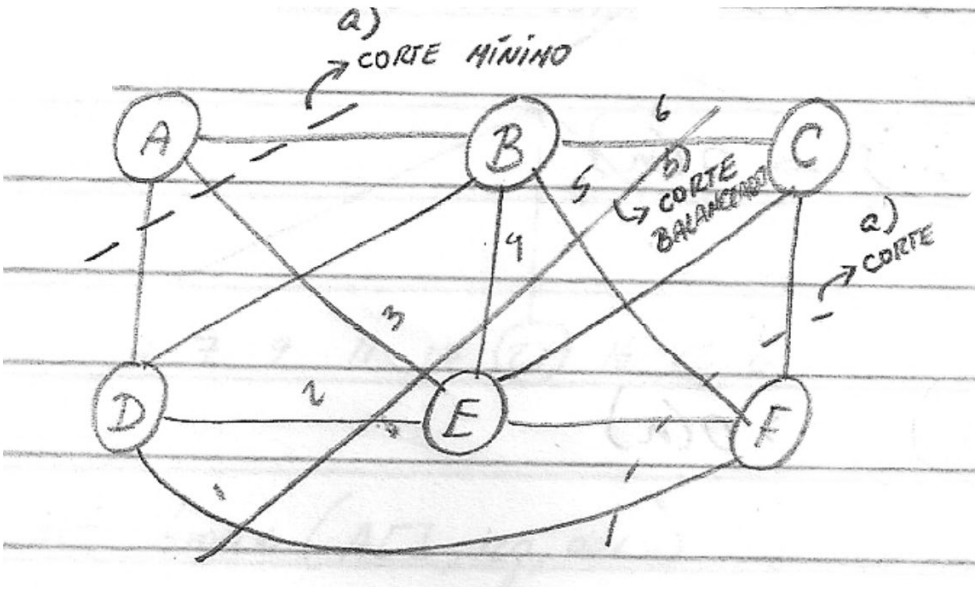
Explicación del código a continuación ⇒ El objetivo es verificar si una suma t existe, este problema puede resolverse con un valor aproximado. se recorre de 1 hasta n y se agregan las sumas que no existan a partir del valor del arreglo en cada iteración, luego se eliminan los valores cercanos ya que es una solución aproximada.

```
n = | S |
L0 = {0}
for n = 1 to n
Li = Merge(Li, Li - 1 + Xi)
Li = TRIM(Li) → Se eliminan los valores mayores a t (no se ha puesto t).
z = Max(Ln)
return z;
```

2.- Tenemos n chicos, n chicas, cuántas posibles parejas?

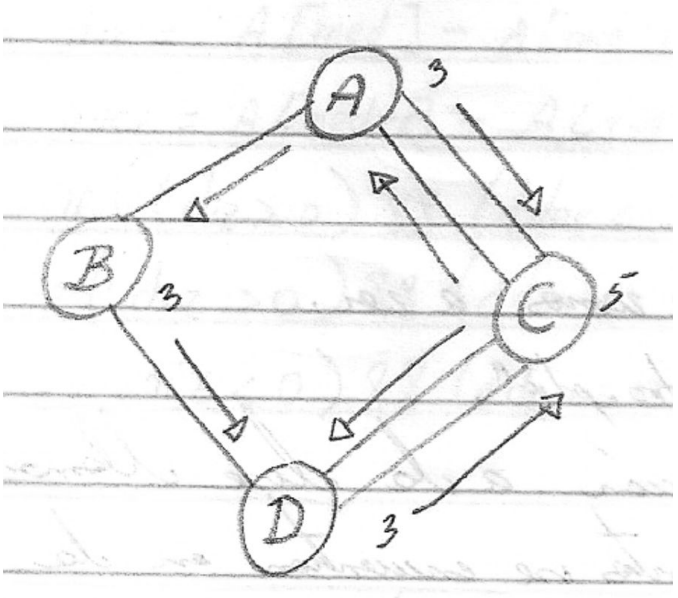
Se hace con conmutaciones.

3) G(V, E), donde V es n y E es m.



Cuántos árboles de expansión? - Se podría hacer viendo el número de vértices que conecten y cuales aristas se conectan y mientras todas las aristas se conecten de alguna manera se puede ver la cantidad haciéndolo uno a uno.

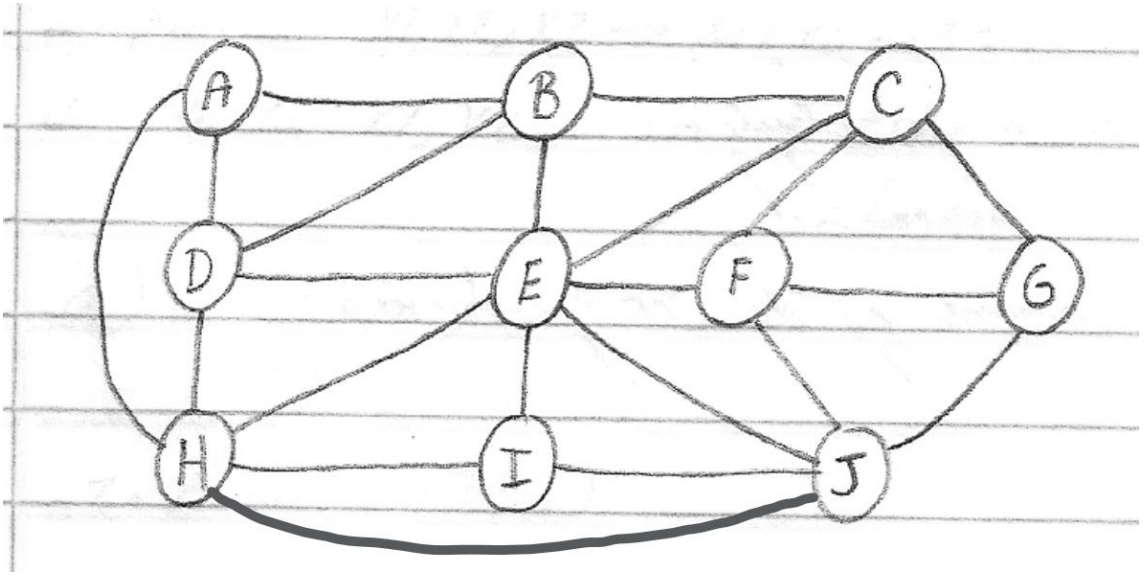
4) Camino de Euler: es un camino que pasa por cada arista solo una vez.



Nota: Los grafos se representarán de la siguiente manera en los exámenes:
A → B D E B → C D E F C → F E D → E E → F F

Donde lo que está después de la flecha es a donde apuntan.

5) Conjunto independiente: Hallar la suma máxima de conjunto independiente, es decir el valor máximo del conjunto de no vecinos.



3, 5, 2, 7, 7, 15, 13, 11, 6, 8

G = (V, E) → conjunto independiente

V → u, v donde pertenece a u.v que pertenece a E

{D, G}

{A, I, F}

6) Cobertura de vértices: Cantidad mínima de nodos que tocan todas las aristas *(con el grafo de arriba)*

V = {E, A, G}, también puede ser V = {A, J}

7) Clique: Todos los nodos que están dentro de una clique están conectados.

Una clique (del grafo anterior) puede ser {A, B, D}

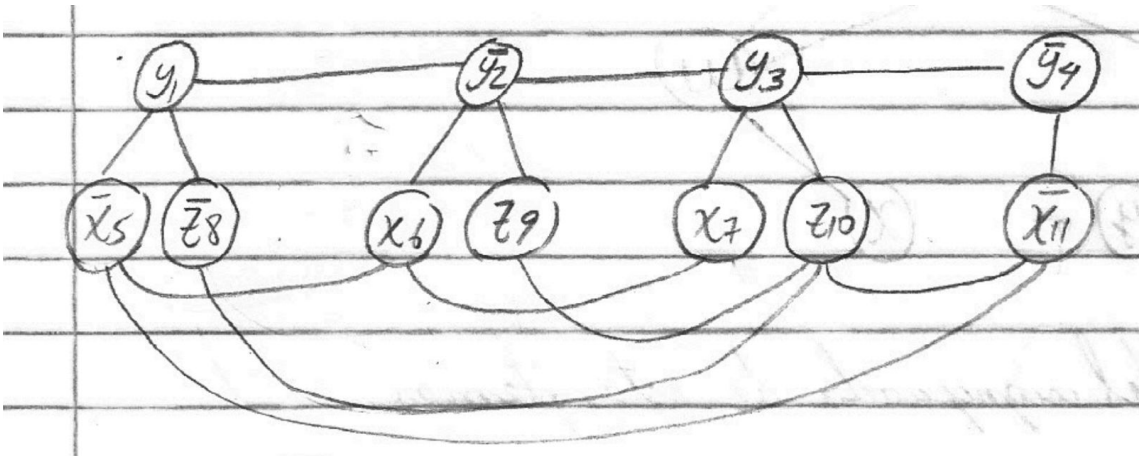
Otra clique es: {E, H, I, J}

8) Conjunctive Normal Form - 2 SAT, Conjunctive Normal Form - 3 SAT

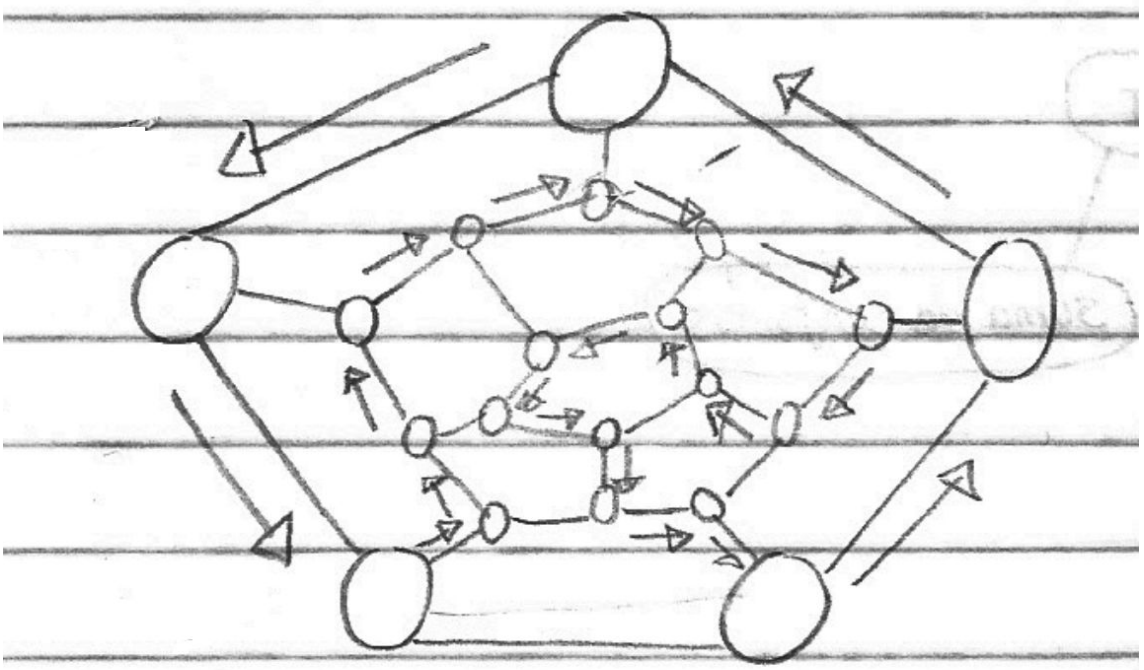
Problema 3SAT: encontrar valores de X, Y y Z que hagan que esta proposición sea verdadera.

(~X v Y v ~Z) ^ (X v ~Y v Z) ^ (X v Y v Z) ^ (~X v ~Z)
Solución: X = Falso Y = Verdadero Z = Falso

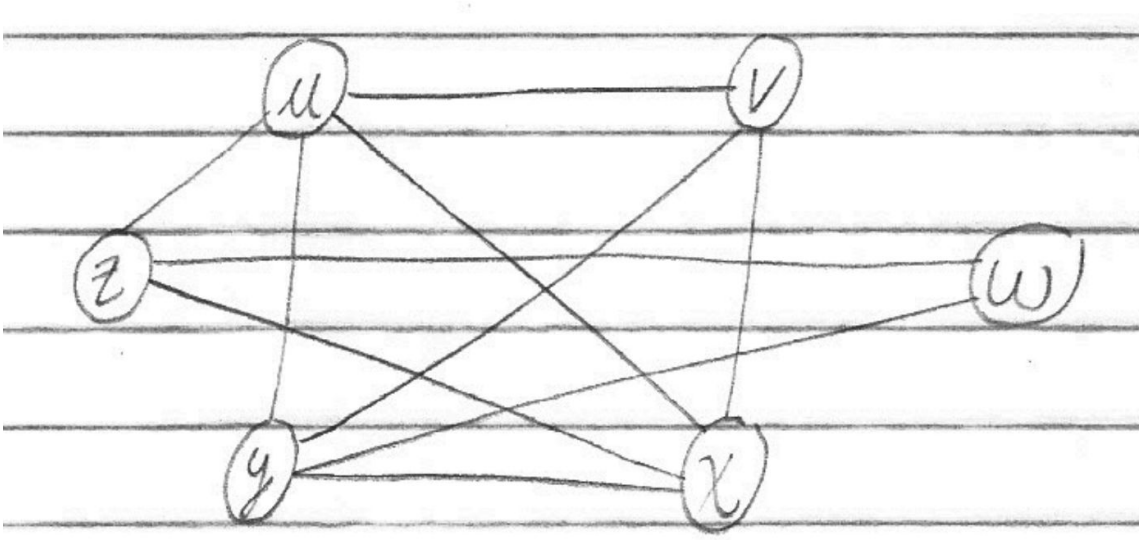
Hacer un grafo a través de esta proposición



9) Trayecto de Hamilton: Es una ruta que toca todos los nodos sin repetir vértices, es una ruta cerrada.



10) Clique y cobertura de vértices juntos



Clique: {u, v, y, x}

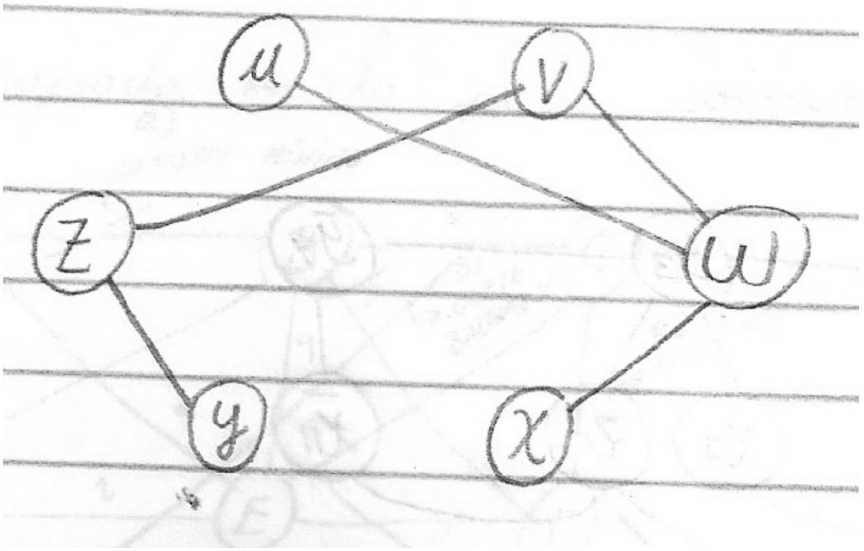
G = (V, E)

V' = {u, v, y, x}

~G = (~V, ~E)

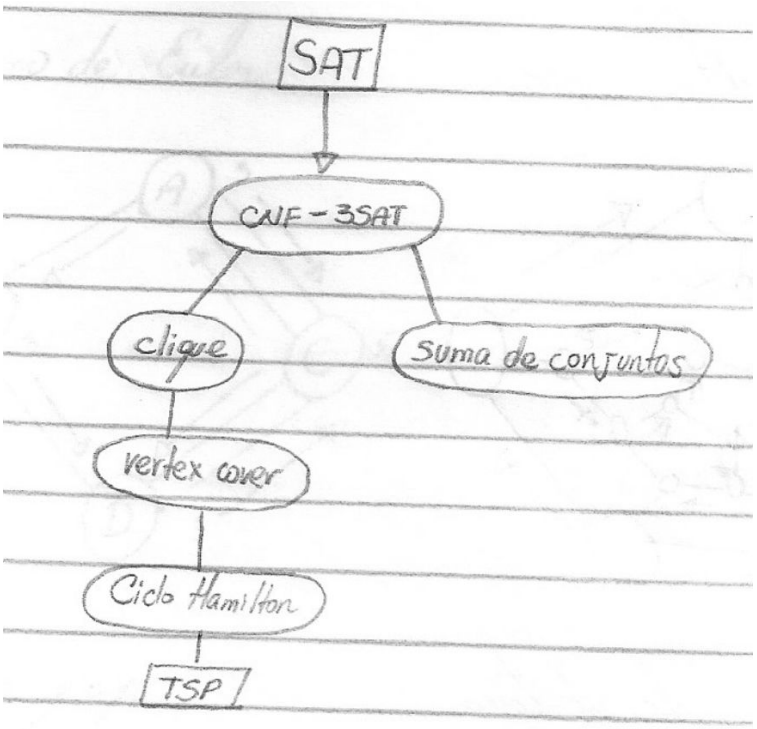
~V = v - v'

Cobertura de vértices:



Nota: *El gráfico presentado expresa la cobertura de los vértices del grafo anterior.*

Resumen del significado de estos temas:



IMPORTANTE: En la primera tarea no se quiere más de cuatro colores para colorear el mapa de Europa.

Fecha: 22/05/2017

IMPORTANTE: Se cambió la fecha del primer parcial al 19 de junio.

La primera conferencia será el Sábado 27 a las 6:00 P.M

Algunas deducciones:

1.- P != NP → Falso

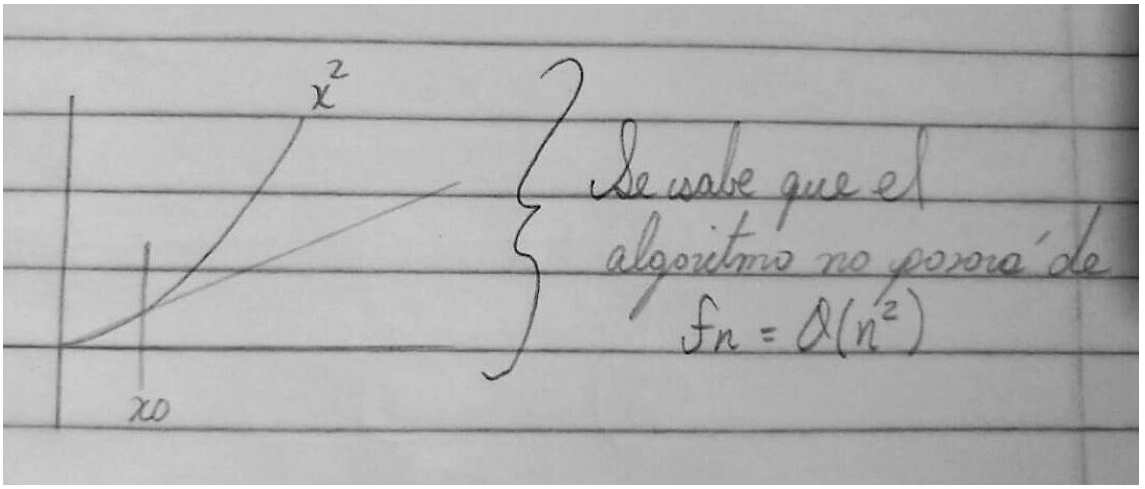
2.- P = NP → Todavía no se sabe.

NPC → NP Completos

Complemento de NP → Lo veremos luego

Análisis amortizado

Si utilizamos la forma vieja de ver el O(n) de los algoritmos, podemos ver que pasa lo siguiente:



T(n) → costo total de n operaciones en el peor caso.

Fórmula → Costo amortizado (n) = T(n)/n

Ejemplo:

Tenemos una pila con S con tres operaciones: pop, push y multipop(S, k), donde multipop saca k elementos de la pila.

Explicación del código a continuación ⇒ Multipop es una función que a través de la función pop saca los último k elementos de la pila.

```
multipop(S, k)
        while S != 0 && k > 0
        pop(s);
        k = k - 1;
```

Por lo tanto multipop es → **O(n)**

Operaciones en pila → O(n^2), ya que si se realizan dos operaciones, en este caso n push y un multipop, cada una iría hasta n y eso causaría un n^2.

NOTA: Para hacer multipop debe haber más de un elemento en la pila

Si hago n push y n pop daría como resultado 2n y siguiendo la fórmula de costo amortizado (T(n)/n), nos daría como resultado 2n/n = 2. En este caso 2 es el costo amortizado.

OTRO EJEMPLO: *Contador binario*

Tenemos 4 bits en 9 números:

0 → 0000

1 → 0001 → Costo: 1

2 → 0010 → Costo: 2

3 → 0011 → Costo: 1

4 → 0100 → Costo: 3

5 → 0101

6 → 0110

7 → 0111

8 → 1000

9 → 1001

NOTA: El costo es la cantidad de bits que se cambian.

Si quisiéramos ver cómo es que cambian cada uno de los bits el algoritmo se escribiría de la siguiente manera:

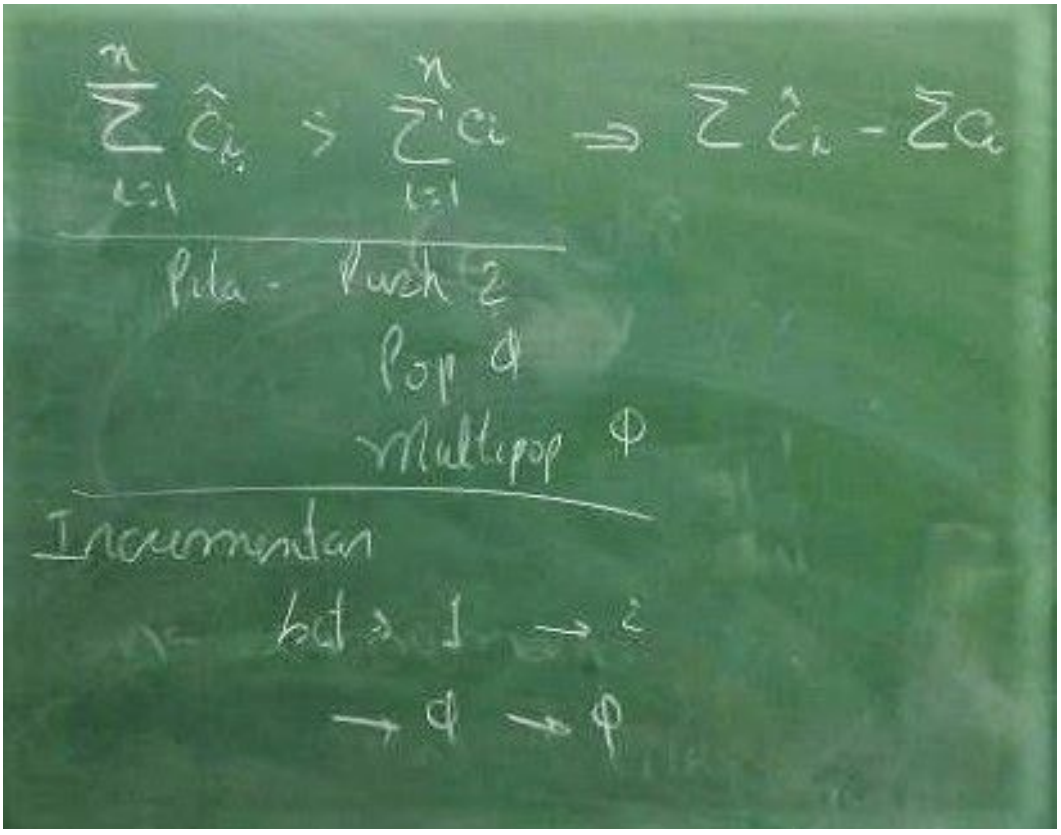
Explicación del código a continuación ⇒ Este algoritmo va poniendo en "1" los valores del arreglo mientras i sea menor que k, i simplemente es un contador y k es la cantidad de bits. De esta manera se cambian todos los bits a 1 correspondiente al número que se desee incrementar, es decir si se desea incrementar la cadena de bits del 4 a la cadena de bits del 5 vemos que solo cambia la primera posición a 1 ya que i es igual a 4 y cuando llega a 5, ya no entra al if de si i es menor que k.

```
Incrementar(A) → A es un arreglo de bits.
i = 0;
while i != k and A[i] = i → k es la cantidad de bits.
{
        A[i] = 0;
        i = i + 1;
}
if i < k
{
        A[i] = 1;
}
```

NOTA: Los bits se recorren de derecha a izquierda. Este algoritmo es O(k)

Si este algoritmo se recorre n veces, la complejidad sería O(nk), donde k es un número.

El costo amortizado de este problema sería 2, ya que sería 2n/2 = 2. Este 2n se determinó de la fórmula de la sumatoria de 0 hasta n de 1/2^i , y esta sumatoria será multiplicada por n.



Explicación de la foto → En esta foto podemos ver un pequeño resumen de la fórmula para el costo amortizado, cuanto cuesta hacer push, pop y multipop y como más o menos se va incrementando los bits.

Divide y vencerás
Si tengo problemas de tamaño n, los dividiremos en 2 problemas de n/2 o tres problemas de n/3, es decir que se dividen en partes iguales, a estas partes las llamamos subproblemas y las soluciones a estas serán recursivas.

PREGUNTA: Si tenemos dos números en un arreglo, cuantas veces se comparan entre sí al ordenarlo con quicksort? → 0 veces si ya se encuentran en las posiciones que van y 1 vez sí uno de estos se convierte en el pivote.

Ejemplo:

Tenemos un arreglo → [7, 5, 2, 3, 4, 8, 1, 9, 0]

Si queremos el número con orden estadístico Xi, se debe aplicar quicksort modificado hasta que el pivote caiga en la posición Xi.

Si ponemos el 4 como pivote pasa lo siguiente:

[2, 3, 1, 0, 4, 7, 5, 8, 9] → Como queremos la posición 5 (4 si lo vemos como un arreglo), podemos ver que el quinto más pequeño es el 4.

Se repasará el miércoles:

- Multiplicación de Karatsuba (en la PVA se encuentra el contenido sobre esto).
- Transformada de Fourier

Fecha: 24/05/2017

1a .- Problema 3sum → A → colección # de flotantes
Hallar a, b, c → a + b + c = 0

Si viéramos este problema solo con dos números, nos dan un arreglo A, y un valor t, tal que x + y = t. Podemos hacerlo de la siguiente forma:

1.- A → t → O(n)
2.- Recorro *Tabla Hash*→ O(n)
si i == x o i == y
v = t - i
Hago hash para encontrar el valor → O(1)

Te damos los valores A = [15, 5, 7, 12, 3], t = 8
Tabla hash

#	**Valor**
0	15
1	
2	5
3	7
4	
5	
6	12
7	3
8	

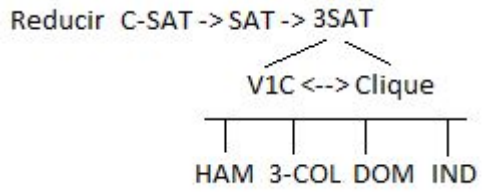
Encontramos un valor que sea menor que 8, que es 5. Entonces después decimos 8 - 5 = 3, y 3 es el valor que está en el arreglo y listo. Encontramos **x + y = t**, que

sería **5 + 3 = 8.**

1b.- Problema colineal → recta
iff (sí y solo sí) (a, a^3), (b, b^3), (c, c^3) → los paréntesis representan (abscisa, ordenada).
Explicación del código a continuación ⇒ **Esta es la solución que prueba que a través de la formula de la pendiente de una recta podemos probar que a + b + c es igual a 0, lo cual es el problema de 3sum que queríamos hacer anteriormente.**

Pendiente 1 = Pendiente 2
(b^3 - a^3)/(b - a) = (c^3 - b^3)/(c - b)
(b - a)(b^2 + ab + a^2)/**(b - a)** = **(c-b)**(c^2 + cb + b^2)/**(c - b)**
b^2 + ab + a^2 = c^2 + cb + b^2
a^2 + ab - cb - c^2 = 0
(a^2 - c^2) + ab - cb = 0
(a + c)(a - c) + b(a - c) = 0
(a - c)(a + b + c) = 0 → De aquí podemos concluir que (a + b + c) = 0. Y así resolvemos el problema #1 de 3sum.

2.-



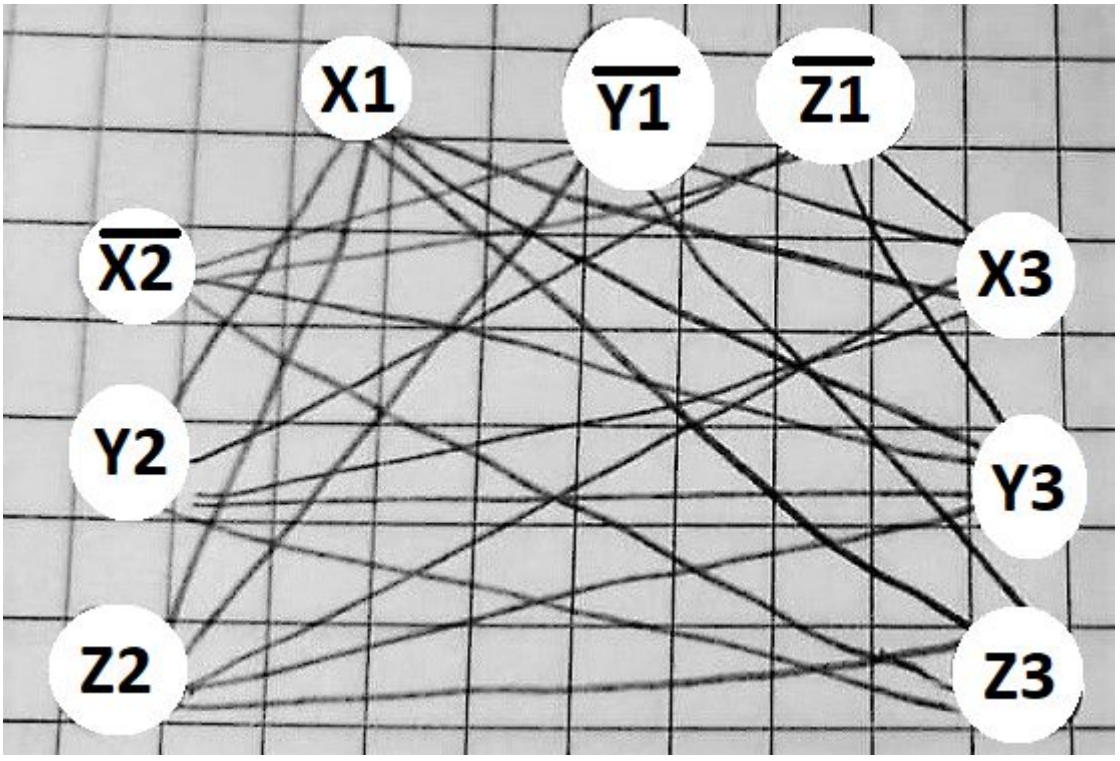
a.- 3-SAT → Clique

(x + ~y + ~z)(~x + y + z)(x + y + z) → La solución sería si todos son verdaderos, o si cualquiera de ellos es verdadero.

1 < x < 3 → Representa la cantidad de variables

1 < r < k → Representa la cantidad de proposiciones

Construir un grafo cada li^r será un nodo será un nodo, una arista entre li^r y li^s r != s y li^r no negativo li^s.



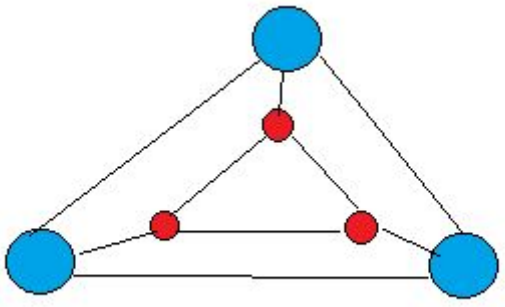
Clique: ~X, ~Y, Z.

b.- 3-SAT → Vertex Cover

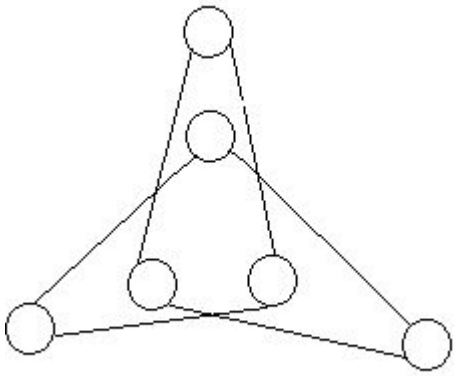
No se hizo en clase.

c.- Vertex Cover → Clique

Si c es el vertex cover de G = (V, E) entonces V - V' ⇒ Clique de ~G



Grafo inverso al anterior:



d.- Vertex Cover → DOM (Conjunto dominante)

u = {1, 2, 3, 4, 5, 6, 7, 8} → Elementos del conjunto u.

s1 = {1, 3} → si

s2 = {1, 2, 4, 5} → si

s3 = {4, 7, 8} → si

s4 = {6, 1, 2} → si

s5 = {1, 4, 8}

Las S's que cubran a la u seran los que conformen el **conjunto dominante.**

Tablas Hash

$$\varphi = m/n$$

Tablas Hash → lista enlazada → abierta

Si no se usan listas enlazadas sería cerrada.

---

si ⇒ tamaño de la tabla hash

c1 ⇒ costo

i → 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

si → 1, 2, 4, 4, 8, 8, 8, 8, 16, 16

ci → 1, 2, 3, 1, 5, 1, 1, 1, 9, 1 → cada vez que se incrementa el tamaño de la tabla (si) se cambia el peso, y el peso tiene que ver con la posición i del elemento que se insertó.

Corrida calculada usando el método asintótico ⇒ O(n^2)

$$ci = i \text{ si } (i - 1) \text{ es } 2^k = 1$$

$$\sum_{i=1}^{n} Ci <= n + \sum_{j=0}^{m} 2^{j-1} \qquad 2^m - 1 \Rightarrow m = \log(n - 1)$$

Karatsuba

x = 1789171345

y = 2583531122

x = a^n/2 + b

y = c^n/2 + d

z = x * y = ac + bc + ad + bd

4 subproblemas de tamaño n/2:

$$a > b^d$$

$$O(n) = n^{\log_b a} = n^{\log_2 4} = n^2$$

3 problemas de tamaño n/2:

$$O(n) = n^{\log_2 3} = n^{1.59}$$

Fecha: 29/05/2017

1.- Recordando - Método maestro

$T(n) = a\,T(n/b) + O(n^d)$

$T(n) = O(n^d \log_b n)$, $a = b^d$

$T(n) = O(n^d)$, $a < b^d$

$T(n) = O(n^{\log_b a})$, $a > b^d$

Ejemplo:

d → La parte no recursiva, el tiempo que se tardó en resolverse.

O(n^3logn)

d = 3

a = b^3

2.- Pseudocódigo
int karatsuba(int x, int g, int n) → a = 3, b = 2, d = 1 ← Método maestro

Explicación del código a continuación ⇒ El siguiente código muestra como se divide los dos números (x, y) en partes iguales para que después pueden ser multiplicados por medio del metodo de karatsuba que es a través de ciertas fórmulas.

```
x = a*10^n/2 + b
y = c*10^n/2 + d
u = a * c
v = b * d
z = (a + b)(c + d)
R = u*10^n + v + (z - u - v)*10^n/2
```

```
if n == 1 return x * y
masc → No se hizo en clase
a = x >> n/2 & masc
b = x & masc
u = karatsuba(a, c, n/2)
v = karatsuba(b, d, n/2)
z = karatsuba(a+b, c+d, n/2)
R = u << n + (z - u - v) << n/2
return R;

int masc(uint n)
int m = ~((~0) << n)
return m;
```

3.- Exponenciación m^emod(n) → La exponenciación rápida nos va a servir más adelante para ver si un número es primo.

Este algoritmo se basa en la relación vista por Euler entre la exponenciación y la representación binaria de un número, el algoritmo consiste en consiste en elevar dev al cuadrado, dev*=dev siempre y luego multiplicarlo por a si es impar (si se suma esa potencia de dos en binario). Recordar que este algoritmo sigue el patrón descubierto por Euler.

```
ll exp(ll a, ll b, ll c){
   if(b==0)return 1;
   ull dev=exp(a,b>>1,c);
   dev=(dev*dev)%c;
   if(b&1)dev=(dev*a)%c;
   return dev;
}
```

Ejemplo:
R = m^e, m = 3, e = 25
e = 11001 → 4 = 1, 3 = 1, 2 = 0, 1 = 0, 0 = 1
4 - c = m → 3
3 - c = c * c. c = c * m → (9, 27) → (3^2, 3^3)
2 - c = c* c → 3^6
1 - c = c* c → 3^12
0 - c = c * c, c = c * m → 3^24, 3^26

Algoritmo de Fermat

Si *p* es un número primo, entonces, para cada número natural *a*, con *a*>0 , coprimo con *p* , $a^{p-1} \equiv 1 \pmod{p}$, es lo que enuncia el pequeño teorema de Fermat

$m^e \bmod n = 1$ si e = n - 1
m = 2, n = 7, e = 7-1 = 6
2^6 = 64 (mod 7) = 1
m = 3, n = 7
3^6 mod 7 = (3^4 * 3^2) mod 7 = 4*9 mod 7 → 36mod7 = 1
m = 5, e = 10, n = 11
m^2mod11 ⇒ 3
m^4mod11 ⇒ 9
m^10mod11 ⇒ 1 ← Por fermat

4.- int Rand_Select(A, p, q, i) → Método aleatorio para hallar el i-esimo elemto en orden ascendente
Este algoritmo es una variante de quicksort empleada solo para saber a cual elemento le corresponde la posición "i" del arreglo.
int Rand_Select(A, p, q, i)
if p == q return A[p]
r = Rand_Partition(A, p, q)
k = r - p + 1 → Cantidad de elementos que van a quedar a la izquierda.
k = rank(A(r)) → Rango de posiciones del pivote.
if i == k return A[r]
if i < k return Rand_Select(A, p, r - 1, i)
else return Rand_Select(A, r + 1, q, i)

En el mejor de los casos es O(1).
El peor de los casos es $\log_2(n)$.

int Rand_Partition(A, p, q) → Echar todo lo que este menor que el pivote para la izquierda, y todo lo que sea mayor que el pivote para la derecha.
piv = A[p], j = p, k = q
while j <= k
while(A[j ++] < p < 0) → j es la izquierda, mientras sea mejor que el pivote lo ponemos a la izquierda.
while(A[--k] > p < 0) → k es la derecha, mientras sea mayor que el pivote lo

ponemos a la derecha.
swap(A, j, k)
return j

5.- Algoritmos voraces (greedy)
Mochila con fracciones → Problema NP

Item	**Valor v**	**Peso w**	**W** (limite de peso)	**v/w**
A	100	2	4	50
B	10	2		5
C	120	3		40

Solución: 2 * A + 2 * C = 2*50 + 2*40 = 180

Item	**Valor v**	**Peso w**	**W**	**v/w**
A	200	1	5	200
B	240	3		80
C	140	2		20
D	150	3		50

Este algoritmo se toma O(nlogn) → Ya que tenemos n items y se ordenan.

Con fracción: 1*A + 3*B + 1*C = 200 + 240 + 170 = 510
Sin fracción → Se finalizará el miércoles.

Algoritmo de Horner:

Explicación del código a continuación ⇒ El siguiente código evalua un polinomio dado, es decir que si tenemos x + 2x, esto evaluará cada posición del arreglo A que sería cada parte del polinomio.

```
int horner(A, n) → A es un arreglo, n es la longitud del arreglo.
s = 0
for(i hasta n)
s = s * x + A[i]
```

```
return s
```

Para el miércoles:

Completar el algoritmo voraz de la mochila

Fecha: 31/05/2017

int ChoosePivot(int A[], int n) → Sirve para escoger el pivote.

```
Divide A en m grupos con m = [n/s]
for i = 1 hasta m
pi = mediana(grupo m)
p = Select([p0, pm], m/2)
return p
```

Rand_Select(A[], p, q, n) → Sirve para hallar el valor estadístico de orden i. p son los valores a la derecha y q son los valores a la izquierda, A es un arreglo cualquiera, n es el número de elementos.

```
if p = q then return A[p]
r = Rand_Partitin(A, p, q)
R = r - p + 1 → La cantidad de elementos entre p y r
i = rank(A(r))
if i < k then return
else return
```

Rand_Partition(A, p, q) → Nos dá la posición del pivote.

```
piv = A[p]
i = p
k = q
while i <= k
while A[i++] < piv
while A[--j] > piv
swap(A, i, j)
```

A = 5, 3, 7, 8, 1, 6, 2, 9, 4
Hallar rango(3):
Quedaría así:
[1, 3, 4, 2, **5**, 6, 8, 9, 7] → **5 es el pivote** → Todos los valores menores al 5 quedan del lado izquierdo y todos los valores mayores al 5 quedan del lado derecho.

2.- FFT → Transformada de Fourier rápida
a)

$$A(x) = \sum_{j}^{n-1} aj\, x^{j}$$

P(x) = -2x^3 + 4x - 5
[-2, 0, 4, -5]

Pseudocódigo de la transformada de fourier

fft(n, a0, a1.....an) → n representa el grado + 1 del polinomio
if(n == 1) return a0
(e0, e1... en/2 -1) ← fft(n/2, a1, a2....an-2)
(d0, d1....dn/2 - 1) ← fft(n/2, a1, a3...an-1)
for k = 0 hasta n/2 - 1 {
w^k ∈ e^2pi i k/n
yk ∈ ek + w^kdk
yk + n2 }
return (y0, y1... yn-1)

b) Valor puntual
xk = 0, 1, 2, 3
Yk = A[xk] = 1, 0, 5, 22

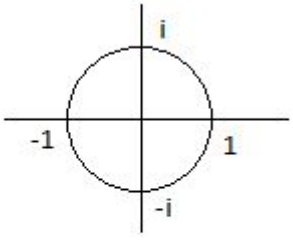
$$\text{Lagrange} = \sum_{k=0}^{n-1} yk \; \frac{\prod(X - Xj)}{j\text{-}k(Xk - Xj)}$$

Si tenemos este conjunto de puntos, podemos hacer la siguiente representación de polinomios con Lagrange:

Puntos = [(0, 1), (1, 0), (2, 3), (3, 22)]

P(x) = 1(x-1)(x-2)(x-3)/1*2*3

<u>Raíz unitaria</u>



$$Wn^{k} = e^{2pi\ ik/n} = \cos(2pik/n) + i\ \sin(2pi\ k/n)$$

Representación	**Multiplicar**	**Evaluar (Horner)**
Coeficiente	O(n^2)	O(n)
Valor puntual	O(n)	O(n^2)

PREGUNTA: Cuál es la mejor decisión para flojar o no la presa de Santiago, es decir, en que condiciones se podria predecir a través de un algoritmo cuando soltarla?

Mochila fraccionaria

Item	**v**	**w (disponibles)**	**w/k**	**W**
A	200	1	200	5
B	240	3	80	
C	140	2	70	
D	150	3	50	

Sacamos → 1 de A, 3 de B y 1 de D.

Mochila sin fracciones

Item	**w**	**v**	**W**
A	8	20	16
B	10	25	
C	2	8	
D	4	12	
E	1	5	
F	4	6	
G	1	8	

Tomamos A, C, D, E, G → 20 + 8 + 12 + 5 + 8

A = 8, C = 2, D = 4, E = 1, G = 1

Peso: 8 + 2 + 4 + 1 + 1 → 16 que es el peso máximo que es W.

PROBLEMA: De un arreglo de monedas tenemos un precio gastado de 8795 y el arreglo de monedas es = [2000, 1000, 500, 200, 100, 50, 1]. La cantidad de monedas que cada uno que debemos usar es?

4 de 200, 1 de 500, 1 de 200, 1 de 50 y 45 de 1.

200*4 + 1*500 + 1*200 + 1*50 + 1*45 = 8795

PROBLEMA: Tenemos una cantidad N de actividades, tenemos que buscar un algoritmo que seleccione la mayor cantidad de actividades que ocupen un mayor tiempo de uso.

# de actividad	**tiempo inicio**	**tiempo fin**
0	0	7
1	1	4
2	3	5
3	3	8
4	4	7
5	5	9
6	6	10
7	8	11

Si lo ordenamos por tiempo de finalización, nos daría lo siguiente:

1 1 4

2 3 5

0 0 7

4 4 7

3 3 8

5 5 9

6 6 10

7 8 11

Tenemos que tomar las partes que sean compatibles entre si, es decir, que la hora de fin no sobrelape la hora de inicio de la próxima actividad.

1 → 1, 4

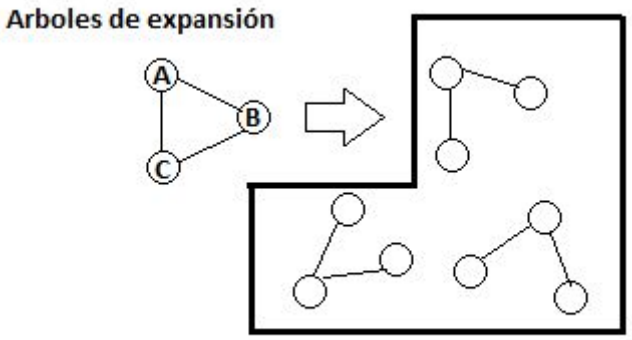
4 → 4, 7

7 → 8, 11

Para hacer esto:

Ordenar segun tiempo de fin → O(nlogn) Recorremos Escoger actividad compatible i. Eliminar actividad incompatible.

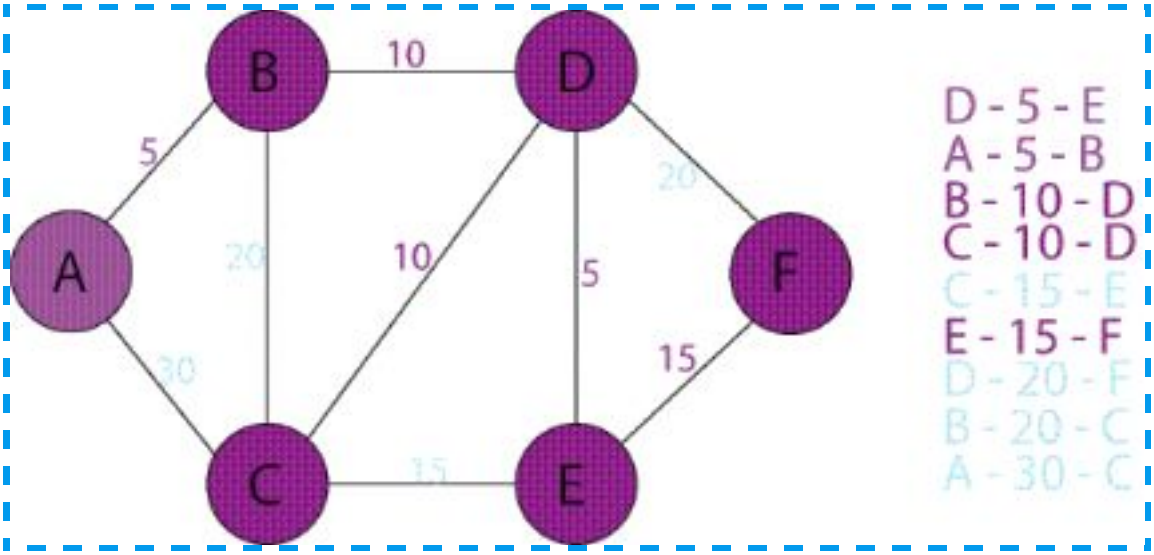
Arboles de expansión que obtenemos de un grafo simple.



Algoritmo de Kruskal:

Explicación del código a continuación ⇒ El siguiente algoritmo nos muestra como el algoritmo de Kruskal elige primero los caminos que tienen menos peso y va aumentando tomando en consideración los caminos con menos peso primero, y así va conectando todas las aristas hasta que hay un camino que las conecta a todas y esos caminos son aquellos que tienen menos peso en todo el grafo.

Gif explanatorio:
https://jariasf.files.wordpress.com/2012/04/kruskals-algorithm.gif



Fecha: 5/06/2017

TAREAS

T1 - 3COLORES

T2 - 2SAT

T3 - TSP

T4 - CLUSTER

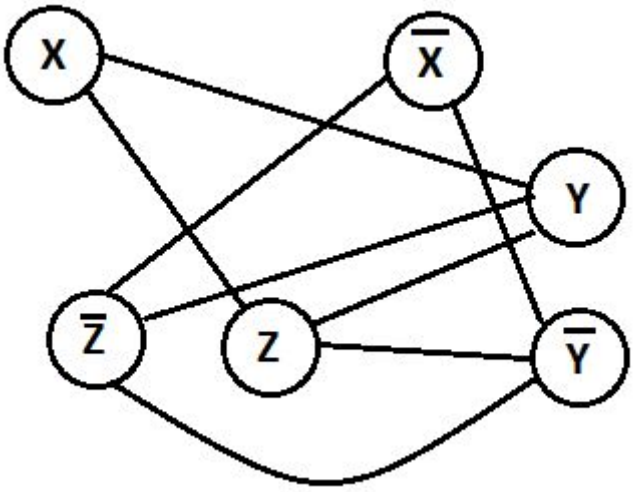
(2) → 2SAT

N variables y m proposiciones

a) (~x + y)(~y + z)(x + ~z)(z + y)

Solución → x = verdadero, y = verdadero, z = verdadera, como por lo menos hay una verdadera en cada una, esta es la solución.

Grafo hecho a través de esta proposición



De esto podemos sacar que los elementos fuertemente conectados son:

x y z ~x ~y ~ z

Lo que se debe evitar en esto es que no aparezca un elemento y su inverso, es decir, x y z ~x ← Esto no debe pasar al intentar sacar los elementos fuertemente conectados, si pasa esto significa que no hay solución.

(3) → (Alpha, Beta) ⇒ (~Alpha, ~Beta)

1.- Probamos en (Alpha, Beta) → Tomo Alpha como → true

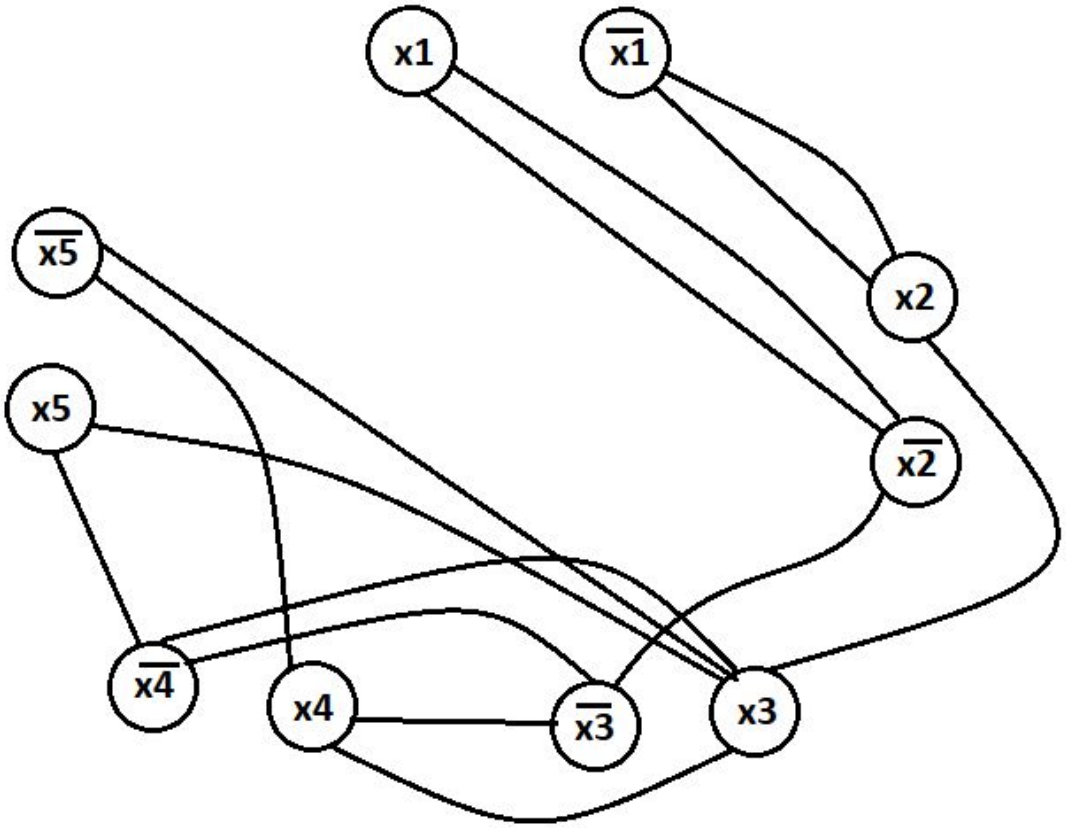
2.- Si (~Alpha, Beta) → Tomo Beta como → true

3.- Repetir (1, 2)

(4) →

a) (x1 + x2)(~x2 + x3)(~x1 + ~x2)(x3 + x4)(x3 + x5)(~x4 + ~x5)(~x3 + x4)

Grafo a través de esta proposición



Solución → X1 = 1, X2 = 0, X3 = 1, X4 = 1, X5 = 0 → Se demuestra que el algoritmo de 2SAT es P.

b) (x + y)(z + ~y)(x)(~x + ~y)(w + z)(~z + w)(~u + ~w)(~z + ~u)

Clusterizar

Dados n puntos, clasificar k grupos coherentes

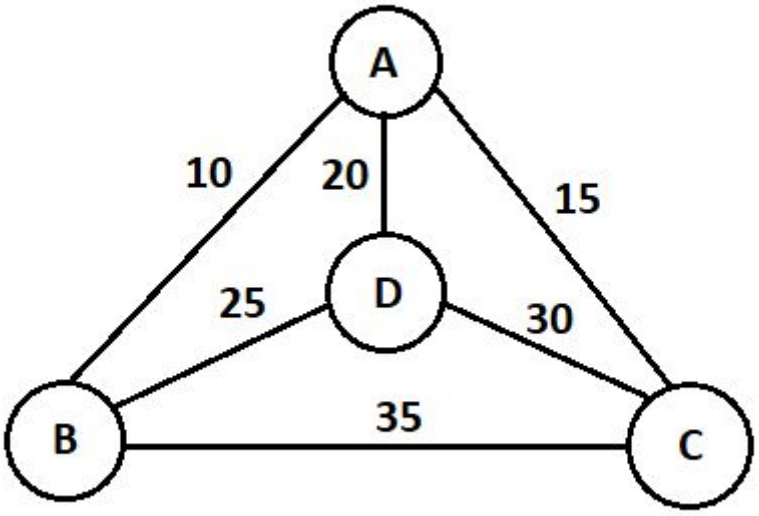
a. Sean p, q dos elementos

   d(p, q) <--> d(q, p) → Distribución euclidiana

Algoritmo

1. Inicio = 1 elemento → 1 cluster
2. Repetir hasta k cluster
   sea p, q → elementos más cerrcanos
   juntar p,q en 1 cluster

TSP→ Tree Spanning Problem



La ruta más corta partiendo desde A es → **A B D C → 10 + 25 + 30 + 65**

Partiendo desde C → **C A B D → 15 + 10 + 25 = 50 ← Es más corto**

Orden topológico de este grafo si recorremos por A → **A(0,7), B(1, 6), D(2, 5), C(3, 4).**

Algoritmo de Prim → Minimum Spanning Tree

Se tiene un grafo G = (V, E) , V = vertices, E = aristas

T = 0 → T es un árbol, no habrán ciclos

1.- Se parte del nodo inicial s. T va a tener a s → T = {s}

1a.- Los vecinos de elementos en T hago heap_min (montículo)

2.- Si (u, v) pertence a E y u toca un nodo en T y la distancia entre (u, v) es la menor.

Se añade T a V sin formar ciclos.

3.- La salida es T.

Se toma O(n*m)

Kruskal

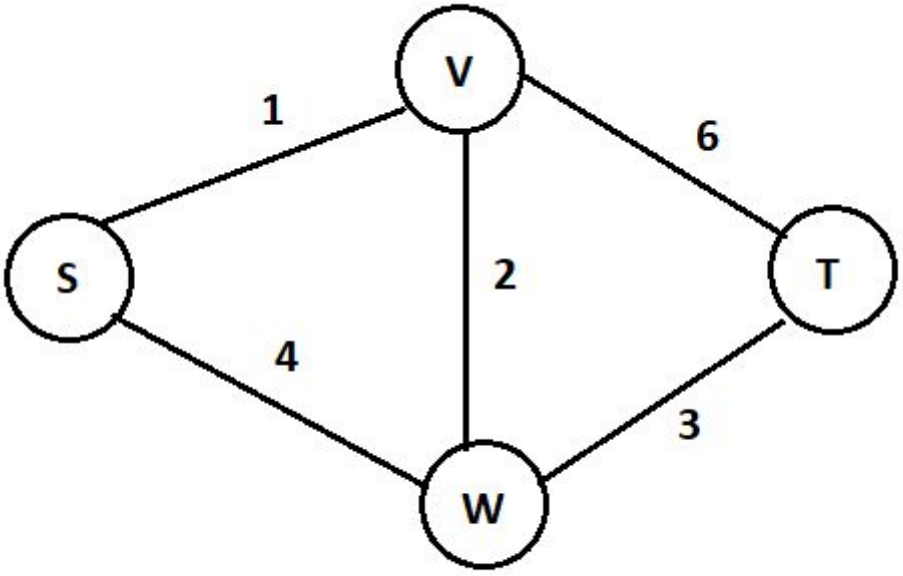
Se tiene un grafo G = (V, E)

Ordenamos el peso de las aristas y lo llamamos le → lo ordenamos de menor a mayor → O(mlogm)

T = 0

Seleccionar la menor arista siempre que no se forme ciclo agregamos la arista a T. → add(T, e).

Algoritmo de Dijkstra



Solución → Distancia a → S = 0, V = 1 W = 3, T= 6

Algoritmo

Entrada G = (V, E) con longitud de aristas le, s pertence a V→ V son los vertices y E las aristas.

X = [s], B[s] = 0

Mientras que x != v

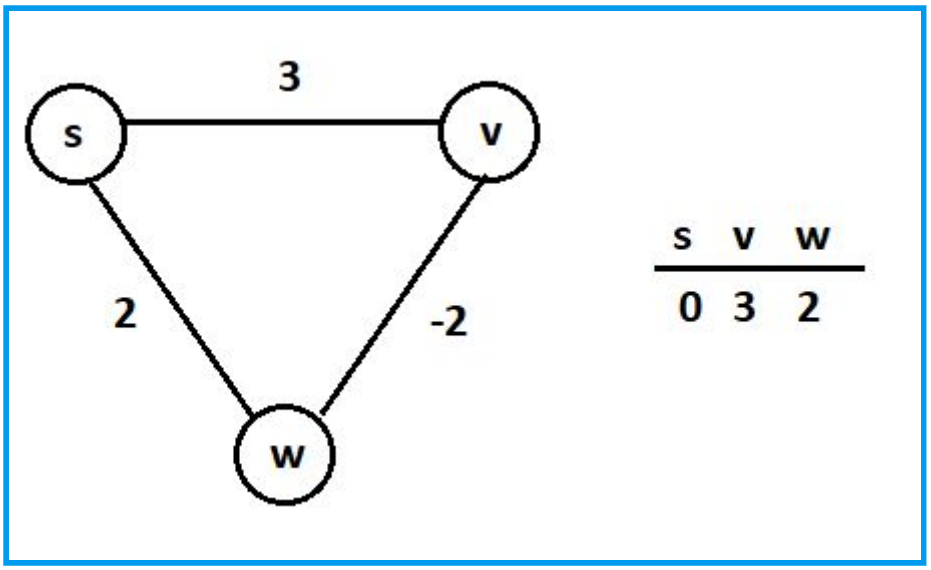
(v, w) pertence a E con v pertence a X, w != x

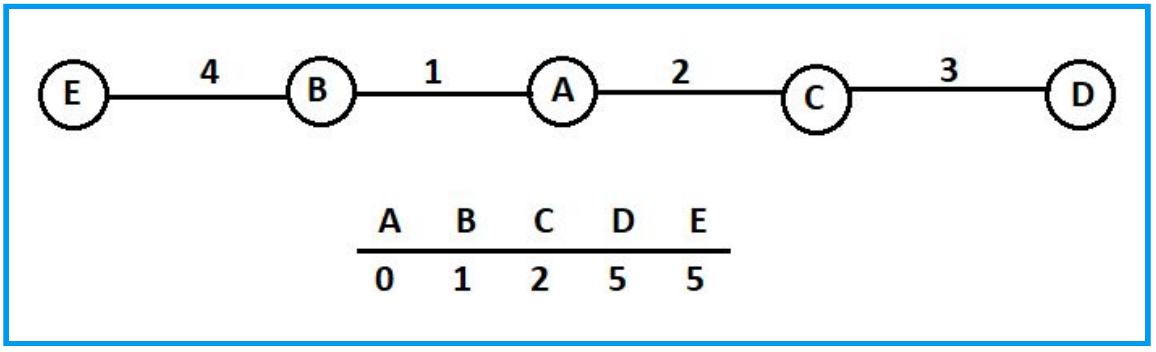
Tomar mínimo A[v] + lvw

A[w] = A[v] + lvw

x[w] = x[v] u(v, w)

Algunos grafos con sus pesos según Dijsktra al lado





Planeamiento para minimizar terminación

Tenemos n actividades, cada actividad tiene un peso y un tiempo de duración. wi = peso. li → tiempo de duración.

c = sumatorio de wi*li

i	wi	li	li (2do caso)
1	3	1	1
2	2	1	2
3	1	1	3

c = 3 * 1 + 2 * 2 + 1 * 3 = 3 + 4 + 3 = 10

c = 1*1 + 2*2 + 3*3 = 1 + 4 + 9 = 14

---------- **2DO CASO** ----------

c = 3 * 1 + 2 * 3 + 1 * 6 = 3 + 6 + 6 = 15

c = 1 * 3 + 2 * 5 + 3 * 6 = 3 + 10 + 18 = 31

En este problema hay dos algoritmos voraces:

a) Ordena (wi - li) en forma decreciente → funciona aveces
b) Ordenar (wi/li) → funciona siempre

Fecha: 7/06/2017

1.- Recursión de cola

a.- def recsum(x) → **Este algoritmo suma todos los números de 1 hasta X.**
if x == 1 return X
return x + recsum(x - 1)

Explicación
x = 3 → 3 + recsum(2)
x = 2 → 2 + recsum(1)
x = 1 → 1 ← Me devuelvo

b.- def tailrecsum(x, ac)
if x == 1 return ac
return tailrecsum(x - 1, ac + x)

Explicación
x ac
3 1 → tailrecsum(2, 1 + 3)
después → tailrecsum(1, 4 + 2)
1, 6 → **Resultado** = 6

2a.- def fact(X)

if x< 2 return 1;

return x * fact(x - 1)

2b.- def tailfact(x, ac)

if x < 2 return ac

return tailfact(x-1, x * ac)

Memorización → Se van guardando resultados por cada operación.

a.- fib(n)

a[0...n] ← null → arreglo.

a[0] = 0, a[1] = 1

return fibaux(n, A)

b.- fibaux(n, A[])

if(A[n] != null) return A[n]

A[n] = fibaux(n-1, A) + fibaux(n-2, A)

return A[n]

Programación Dinámica

1.- Dado n

Salida → número mínimo de operaciones hasta llegar a 1

Operación n - 1,

n / 2 si n % 2 == 0

n / 3 si n % 3 == 0

Algoritmo:

minoper(n)

if n == 1 return 0

min = 1 + minimum{

minoper(n -1),

minoper(n/2) //→ si n%2 == 0

minoper(n/3) //→ si n%3 == 0 };

Explicación

n	minoper
23	1 + min{22}
22	1 + min{21, 11}
21	1 + min{20, 7}

1	2	3	4	5	6	7	8	9	10	11	12
0	1	1	2	3	2	3	3	2	3	4	3

Podemos ver que el 3 es n/3, pero el 4 es 1 + minimo de 3 que sería 1 + 2 = 3. Y así sucesivamente para el 5 que sería 1 + minimo de 4 = 1 + 2 = 3. Pero si probamos con el 12 tendriamos que probar n -1 que es 11, y como 12 se divide entre 2 y entre 3 daría 6 y 4 y el minimo entre 11, 6 y 4 es 2 y 2 + 1 es 3.

13	14	15	16	17	18	19	20	21	22	23
4	4	4	4	5	3	4	4	4	5	6

2.- Suma de secuencia creciente máxima → **Encontrar la suma máxima que hay entre n numeros que uno sea mayor al otro, osea no puede haber: 3 + 2 + 4, ya que el 2 es menor que el 3 y cada número siguiente debe ser mayor al anterior.**

3, 2, 5, 4, 7, 1, 6, 8, 5, 9

```
S(n)
S(1) = A[i]
S(n) = A[n] + max{S(j), j pertence a {1...n-1} y A[j] < A[n]}
```

3.- Máximizar venta de tubos cortados → i es la longitud en pies y pi es el precio.

i	1	2	3	4	5	6	7	8	9	10
pi	1	5	8	9	10	17	17	20	24	30

¿Para un n dado, cuál es el precio máximo que se puede obtener? → Con fuerza bruta simplemente lo haríamos viendo todos los precios de cada partición y todas las combinaciones posibles.

```
int preciomaximo(A[1..n], n)

A[0] = 0

if n == 0 return 0;

q = - infinito

for(i = 1 hasta n)

q = max{q, preciomaximo([An-i] + A[i])}
```

```
return q;
```

Algoritmo de Dijkstra

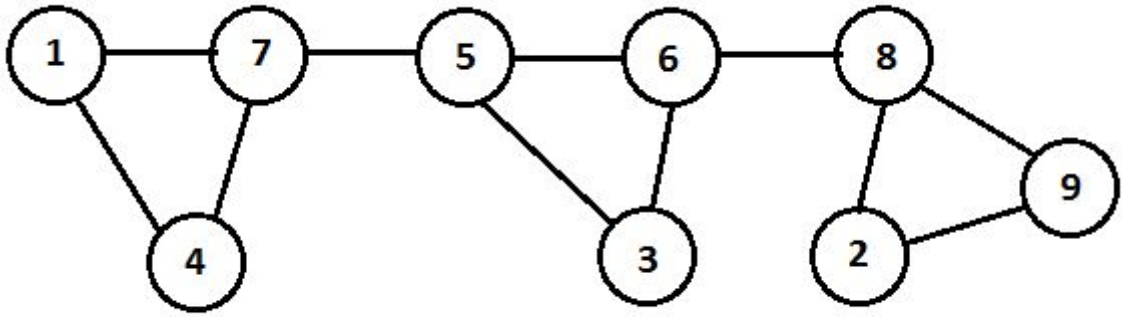
```
Dado G = (V, E) y s pertenece a E
inicio = X[s]. A[s] = 0
while X != V
Si v, w pertence a E, v pertenece a X, w no pertenece a X
Seleccionar min A[v] + lvw
add(x, w)
A[w] = A[v] + lvw
B[w] = B[v]  u (v, w)
```

Hablamos del corte mínimo → ver al principio del documento.

Tenemos un grafo direccionado → G(E, V)

Algoritmo de Kosaraju → SCC (Strongly Connected Component) → Entre cada nodo se forma un ciclo.

u, v pertence a E, u ⇒ v y u ⇒ v → Esto significa que tenemos un ciclo.



Cuáles están fuertemente conectados?

```
1, 7, 4 y 5, 6, 3 y 8, 2, 9
```

Algoritmo de Kosaraju

```
Grev ← G → Grafo

DFS_LOOP (Grev)

f(v) = Ls

DFS_LOOP(G)
```

```
SCC ← nodos con el mismo lider


DFS_LOOP(G)
{
        t = 0, s = null
        nodos - 1 hasta n
        for i = n - 1
        si i no está explorado
        s = i
        DFS(G, i)
}


DFS (G, i)
{
        i ← explorado
        lider(i) = s
        foreach arista (i, j) in
                if j no esta explorado entonces DFS(G, j)
        t + i
        f(i) = t
}
```

Hallar la suma máxima de valores no vecinos

[3, 5, 4, 7, 6, 8, 11, 9, 13]

```
int S(A, n)
{
        if n = 1 ⇒ return A[1]
        if n = 2 ⇒ max[A[0], A[1]]
        S(n) = max{S(n-2) + A[n], S(n-1)}
}
```

Fecha: 12/06/2017

Para el examen, estudiar los siguientes temas:

- Complejidad computacional → Preguntas teóricas sobre problemas NP y problemas 2SAT, 3SAT.
- Técnicas de análisis de algoritmos → Análisis amortizado
- Divide y venceras → algunas aplicaciones como: estadística de orden, multiplicación de enteros y transformada rapida de fourier.
- Algoritmos voraces → mochila fraccionaria y otros.

1.- Maximizar corte de un tubo

PREGUNTA: De qué manera se podría maximizar los precios de venta de los tubos?

i	1	2	3	4	5	6	7	8	9	10
pi	1	5	8	9	10	17	17	20	24	30

Algoritmo → O(n^2) → j es el tamaño total del cubo, r es una lista con los valores máximo por cada posición
r = [ ]
for(j =1 hasta n)
q = -infinito
for i hasta j
q = max(q, pi + r[j -i]) → i + j - i = j.
r[j] = q

Para el primer valor el máximo sería 1.
Para el segundo valor sería 5 + 1 = 6.
Para el tercer valor sería 8.
Para el cuarto valor es 10.

<u>Explicación</u>

j	i	j - i	q
3	1	2	1 + 5 = 6
	2	1	5 + 1 = 6
	3	0	**8 + 0 = 8 (GANA)**

Aquí lo que se hace es que i y j son las diferentes formas de formar un 3, es decir 1 + 2, 2 + 1 o 3 + 0, y la mayor q gana, q es la suma de los precios de i y j - i.

j	i	j - i	q
4	1	3	1 + 8 = 9
	2	2	**5 + 5 = 10 (GANA)**
	3	1	8 + 1 = 9
	4	0	9 + 0 = 9

Al igual aquí se toma las diferentes formas de formar un 4 y gana la suma mayor.

Multiplicación de matrices

Tenemos dos matrices, C = A*B, donde A y B son matrices.

```
for i hasta p
        for j hasta r
                for k hasta q
                s += A[i, k] * B[k, j]
C[i, j] = s;
```

A = p * q

B = q * r

C = r * s

((A * B) * C) → p*q*r + p*r*s

(A*(B*C)) → p*q*s + q*r*s

Ejemplo:

A = 10 * 30

B = 30 * 5

C = 5 * 60

((A*B)*C) = 10*30*5 + 10*5*60= 1500 + 3000 = 4500.

(A*(B*C)) = 10*30*60 + 30*5*60 = 18000 + 9000 = 27000.

Multiplicación de cuatro matrices

Hay 4 formas para multiplicar 4 matrices, y estas son:

A1*A2*A3*A4 = (A1((A2*A3)*A4)

A1*A2*A3*A4 = (A1((A2(A3*A4)))

A1*A2*A3*A4 = ((A1*A2)*A3)*A4)

A1*A2*A3*A4 = (((A1*A2)*A3)*A4)

m[i, n] → Solución óptima

p(n) =

{

1 si n = 1

ó sumatoria de P_i desde k = 1 hasta n - 1 de P_{n-k} si n >= 2

}

p(n) = C(n-1)

C(n) = 1/(n+1) (2n n) = $4^n/n^{3/2}$

m[i, j] → 1 <= i < j <= n

m[i, j] =

{

0 si i = j

min{m[i, k] + m[k + 1, j] + $p_{i-1}P_kP_j$ → solo si i <= k < j

}

P_{i-1} → Filas de A_i

P_k → Columnas de A_{k-1} y Filas de A_j

<u>Notación</u>

A1 A2 A3 A4 A_{N-1} A_N → Son las matrices.

P0 P1 P2 P3 → Son las dimensiones de cada matriz

A1, A2, A3, A4, A5

30, 35, 15, 5, 10, 2 → P0, P1, P2, P3, P4, P5

i/j	1	2	3	4	5	6
1	0	15750	7875			
2		0	2625			
3			0			
4				0		
5					0	
6						0

i	j	k	m[i, k] + m[k+1, j] + Pi-1*Pk*Pj
1	2	1	0 + 0 + 30*35*15 = 15750
2	3	2	0 + 0 + 35*15*5 = 2625
1	3	1	0 + 2625 + 30*35*5 = 2625 + 5250 = 7875
1	3	2	15750 + 2625 + 30*15*5 = 20625

Maximización del problema de la mochila

W = peso total = 11

Nos dan unos items, con Vi valor y debemos tomar tantos itemes se puedan maximizando el valor de los mismos.

n	1	2	3	4	5
Wi	1	2	3	6	7
Vi	1	6	18	22	28

Sn → Solución

Sn = S1 si n = 1

Sn = Sn-1 + Wn → si n es parte de la solución, W debes ser W - wn.

Sn = Sn-1 → si n no es parte de la solución.

Se tomará el máximo de las últimas dos Sn.

i/W	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	7	7	7	7	7	7	7	7	7	7
3	1	1										
4												
5												

Fecha: 14/06/2017

Mochila 0 - 1

W = 11

i	1	2	3	4	5
vi	1	6	18	22	28
wi	1	2	5	6	7

i/w	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	18	18	25	25	25	25
4	0	0	1	6	7	7	18	18	25			
5	0	0	1	6	7	7						

i	i - 1	X	X - wi	a	b
3	2	5	0	7	0 + 18 = 18

3	2	11	11 - 5 = 6	7	7 + 18 = 25
3	2	8	8 - 5 = 3	7	7 + 18 = 25
4	3	7	1	18	1 + 22 = 23
4	3	11	5	25	40

NOTA: El valor que se obtiene en la ultima posición de la ultima fila de la tabla es la suma de los valores máximos.

A[i,x] = max{A[i-1, X], A[i-1, X - W1] + VI}

```
int a = 35, b;
char ar[10];
char *ptc;
printf(ar, “%x”, a);
ptc = ar, sccarf(ar, %0, &b),
b? →
int ncif(int n, int b, int d)
{
        int res, cont = 0;
        while(n >= 0)
        {
                res = n % b;
                n = n / b;
                if(res == d) cont + 1;
        }
        return cont;
}
```

Secuencia 4, 7, 2, 9, 1, 3, 5, 8, 6

Crear heap_min en arreglo A[10].

1	2	3	4	5	6	7	8	9	
4	7	2	9	1	5	8	6		
1	2	3	6	7	4	9	8		

hijos → 2i, 2i + 1

4)

$A(x) = x^3 - 2x + 1$ → [1, 0, -2, 1]

$B(x) = x^2 + 2x + 1$ → [0, 1, 2, 1]

C(x) = A(x) + B(x) → [1, 1, 0, 2]

Pares ordenados → A(X) → [(0, 1), (1, 0), (2, 5), (3, 22)]

Para el examen:

- Decir sin los problemas son P o NP.
- Distinguir entre recorrido Hamiltoniano, y el recorrido de Euler y decir que si son P o NP.
- Reducciones, tengo un problema A y un problema B, cuando A se reduce a B? - Reducción es cuando se transforma un problema en otro, saber hacer reducciones.
- Un algoritmo para resolver el problema 2SAT.
- Probar que el problema X es NP duro, pregunta de seleccion multiple. a) X ⇒ 3SAT, b) 3SAT ⇒ X, c) ambas d) niguna. → **La respuesta es la A.**
- Tenemos un arreglo de 4 elementos, ordenar ese arreglo con solo 4 comparaciones. → **No se puede con ningún algoritmo.**
- Análisis amortizado → Tengo una secuencia dde operaciones, los costos son:

  i → 1, 2, 3, 4, 5, 6, 7, 8
  O(n) → 1, 2, 1, 4, 1, 1, 1, 8
  Cuál es el costo amortizado por operación? → **O(1)**
  Cuál es el tiempo en análisis asintotico? → **O(n)**

Como ponerle zapatos a n niños que les sirvan a cada uno, con n zapatos?
Ordenamos los tamaños de los zapatos con los tamaños de los pies de los niños y después hacemos búsqueda binaria para buscar el tamaño de zapato de cada niño.

Buscar el segundo mayor en un arreglo, que tiempo tiene, lo más eficiente posible?
Tengo un grafo no dirigido ver si hay un ciclo?

Un camion llega a su destino con un tanque de gasolina que solo puede recargar en la parada, cual es la para mas lejos que el camion puede pararse sin gastar su galón de gasolina.
Planeamiento para minimizar terminación → planeamiento de actividades en un horario → **ver arriba.**

Queremos minimizar el tiempo de finalización

i	1	2	3
li	1	2	3
wi	1	1	1

Se puede comenzar desde cualquiera ya que tienen el mismo peso.

Fecha: 19/06/2017

Día de examen - No se dio clase.

Fecha: 21/06/2017

Corregimos el examen que hicimos en línea y el examen escrito.

Cantidad de formas de dar un cambio

Si tenemos las monedas [1, 5, 10, 25, 50], de cuantas maneras diferentes puedo cambiar:

1 a 4 → 1 manera

5 a 9 → 2 manera

10 a 14 → 4 maneras

15 a 19 → 6 maneras

Pseudocódigo:

```
int cc(monedas[], n, monto)
{
        if monto < 0 return 0
        if monto == 0 return 1
        return cc(monedas[], n, monto - monedas[n]) + cc(monedas[], n-1, monto)
}
```

Monedas	1	5	10	25	50
Cantidad	1	2	3	4	5
0	1	1	1	1	1
1					
2					
3					
4	1	1			
5	2	2	2	2	2
6					

```
for i = 1 hasta cant + 1
for j = 4 hasta n
x = (i - monedas[j]) >= 0 ? table[i - monedas[j], j] : 0
y = (j >= 1) ? table[i][j - 1] : 0
table[i, j] = x + y
```

Buscar la subsequencia creciente mayor

A[1...n] = 2, 7, 3, 5, 9, 11, 8, 12, 15

```
int sub_mayor(A[], n)
        if n == 1 return 1
        Si no entra → sub_mayor(A, n - 1)
        Si entra → sub_mayor[A, n -1] con A[n]
```

Ahora hacer lo mismo pero que el último valor sea menor que X.

```
int sub_mayor_condicion(A[], n, x)
```

```
{
        if n == 1 return 1
        if x < A[n] return sub_mayor_condicion(A, n-1, x)
        else return sub_mayor_condicion(A, n, x)
}
```

Otra forma de hacer esto es:

```
int sub_mayor_condicion(A[], n, x)
{
        if n == 0 return 0
        m = sub_mayor_condicion(A, n-1, x)
        if A[n] < x
                m = max(m, 1 + sub_mayor_condicion(A, n - 1, A[n])
        return m
}
```

Fecha: 26/06/2017

IMPORTANTE: La tarea #3 será abierta desde el próximo viernes hasta el domingo en la mañana. Se puede hacer con cualquiera de los algoritmos de aproximación, cual sea. Hay que analizar que tan errada está la aproximación.

1.- Tengo 8 monedas iguales, solo que 1 pesa más, solo tengo una balanza, ¿Cuál es la cantidad mínima de balances?

Solución

A.- Dividirlo en grupos de 3:

A → A1, A2, A3

B → B1, B2, B3

C → C1, C2

B.- Balancear A y B

a) Si está en A (porque Bal A1 y A2), si es equal → A3.
b) else balanceo C1 y C2

2.- Tengo 25 caballos, quiero saber cuales son los tres mejores, solo puedo echar carreras de máximo 5 caballos. ¿Cuál es la menor cantidad de carreras?

A → A1 A2 A3

B → B1 B2 B3

C → C1 C2 C3

D → D1 D2 D3

E → E1 E2 E3

3.- Sequencia de longitud m. ¿Cuántas subsecuencias hay?

s = “ABC” ← ejemplo

Solución

La cantida de subsequencias es 2^n, ya que si le sacamos las subsecuencias al anterior sería: ABC, AB, AC, BC, A, B, C, “ “ ← Las cuales son 8 subsecuencias y 8 es $2^3 = 8$.

4.- LCS (Least Common Subsecuencia)

LCS(X, Y, m, n) =

m, n = 0 → 0

Si x[m - 1] = y [n-1] → 1 + LCS(x, y, m-1, n-1)

x[m-1] = bl → LCS (x, y, m-1, n)

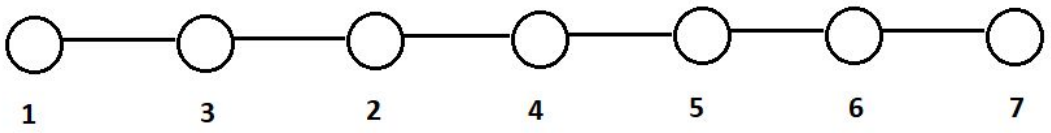
y[n] = bl → LCS(x, y, m, n-1)

Lo que debemos ver es si hay coincidencias entre la fila de arriba, a la izquierda y la de arriba a la izquierda. Si son iguales se le saca el máximo, sino se le suma uno.

		G	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2
G	0	1	2	2	2	2	3	3	3
C	0	1	2	3	3	3	3	4	4
A	0	1	2	3	3	4	4	4	4
A	0	1	2	3	3	4	4	4	4
T	0	1	2	3	4	4	4	4	4
G	0	1	2	3	4	5	5	5	5

Podemos ver que cuando la fila de la derecha es igual al de la derecha osea que si G es igual a G se toma el numero de esas dos columnas de coincidencia como candidato y al final se imprimen las letras de los números candidatos.

GCCAG ⇐ Sería un candidato, ya que son iguales arriba y a la izquierda y podemos ver que van incrementando los numeros. G = 1, C = 2, C = 3, A = 4, G = 5.



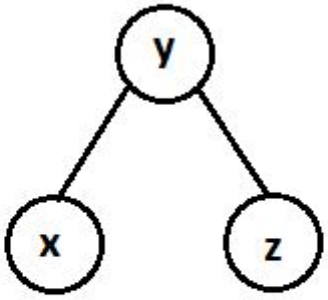
Sn = si n pertenece a S ⇒ Sn-2 + Wn

si n no pertence a S ⇒ Sn-1

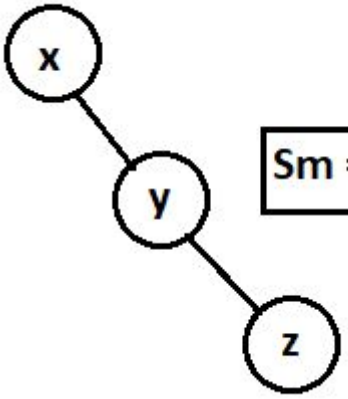
i	0	1	2	3	4	5	6	7
wi	0	1	3	2	4	5	6	7
si	0	1	3	3	7	8	13	15

Arboles de expansión de búsqueda propia

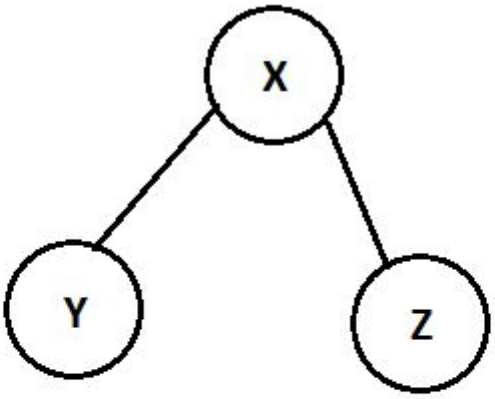
	x	y	z
Pi	8	1	1



Sm = 2 * 0.8 + 1*0.1 + 2*0.1 = 1.9



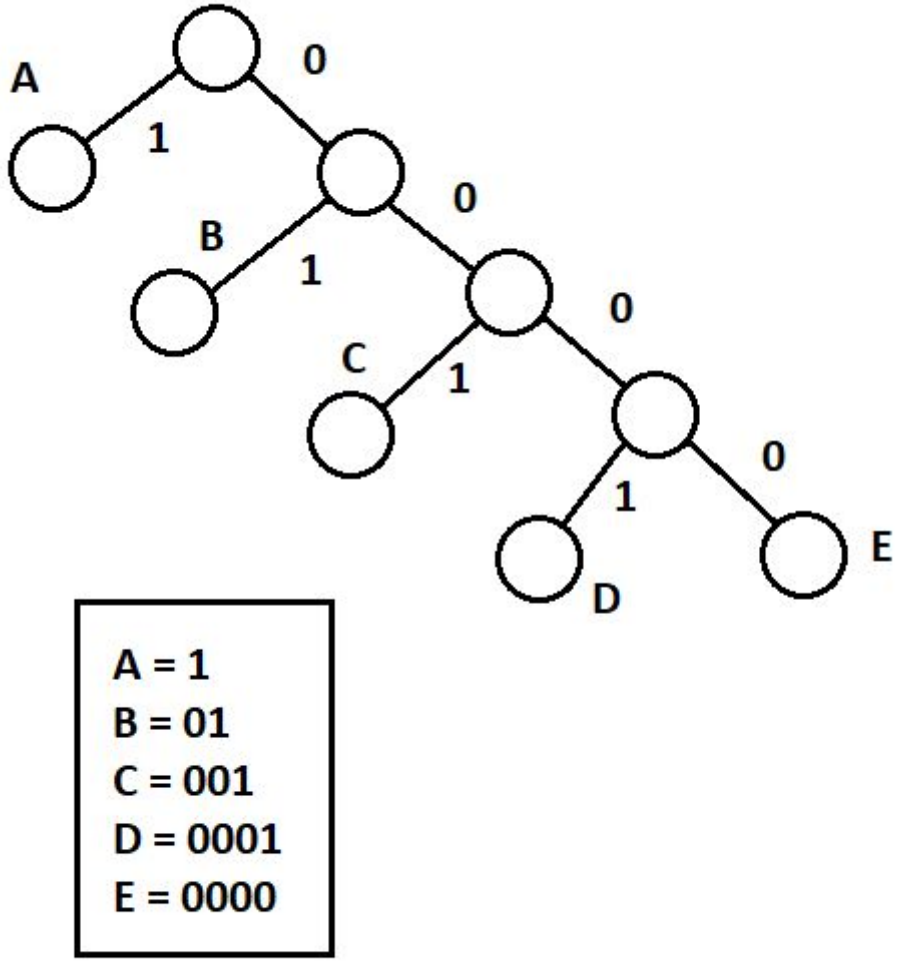
Sm = 1 * 0.8 + 2 * 0.1 + 3 * 0.1 = 1.3



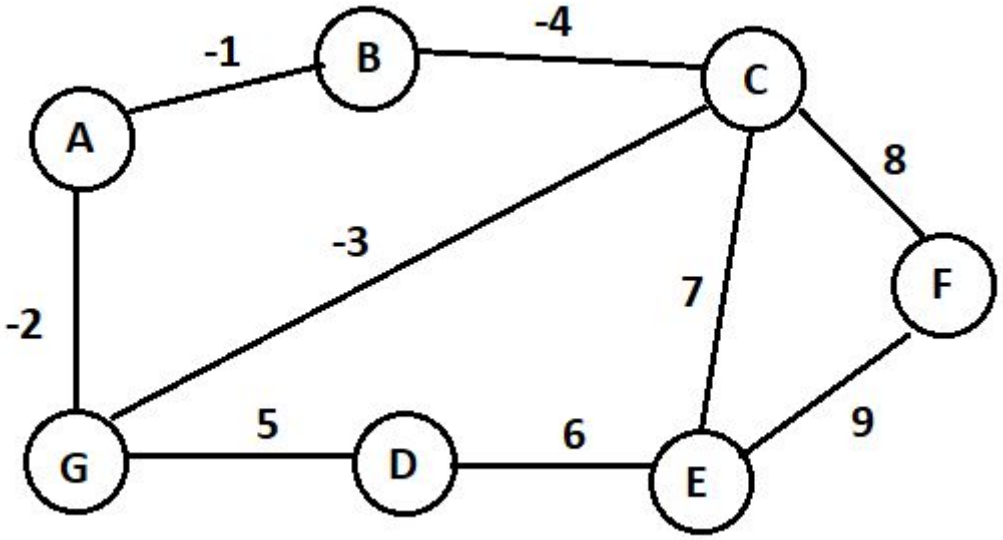
$$Sm = 1 * 0.8 + 1 * 0.1 + 1 * 0.1 = 1.0$$

Algoritmo de Huffman

Si	A	B	C	D	E
p	0.4	0.3	0.2	0.05	0.05



Algoritmo de Bellman-Ford:



G = (V, E)

| V | = n

| E | = m

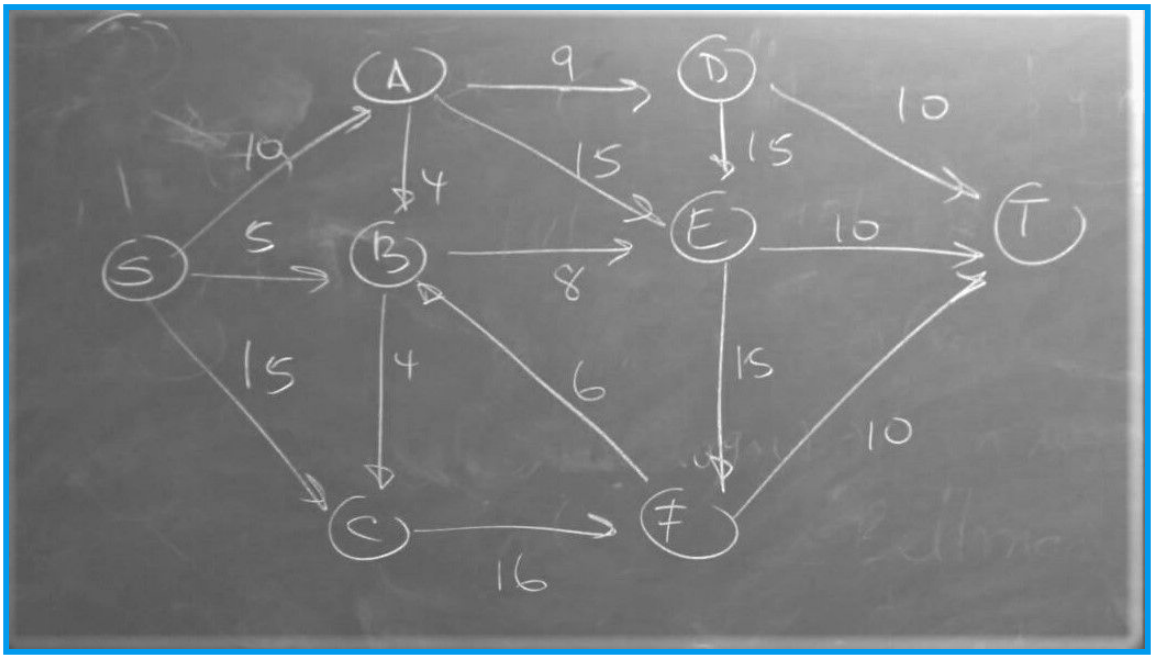
O(m*n) ó O(mn^2)

O(mlogn) ó O(m*nlogn)

Cuándo se dice que m = O(n) o m = O(n^2). Si es n^2 decimos que es un grafo muy denso y si es n es un grafo poco denso. Este algortimo consiste en ver si existe un ciclo negativo y hallar la distancia de una fuente a todos los demás nodos y permite determinar ciclos.

Floyd_Warshall → Hallar la distancia de todos los pares de nodos, osea de A a B, de B a C, entre otros. Es prácticamente como tomar Bellman Ford y repetirlo n veces.

Flujo de redes



Se basa en cuantos unidades se pueden enviar desde el source (S) hasta el destino o sumidero (T). El flujo máximo es aquel en el que es posible pasar a través del corte mínimo.

Investigar el **algoritmo de máximo flujo** y grafo residual.

Fecha: 28/06/2017

Temas para el examen

Fecha - Miércoles 5 de Julio del 2017

Programación Dinámica - 2 problemas

Flujo de redes (en especial flujo máximo) y backtracking - 2 problemas

Para practicar flujo máximo → https://visualgo.net/en/maxflow

Backtracking → https://www.hackerearth.com/es/practice/basic-programming/recursion/recursion-and-backtracking/tutorial/

Programación dinámica → http://elvex.ugr.es/decsai/algorithms/slides/6%20Dynamic%20Programming.pdf

Heurística: Son metodos que pueden parecer algoritmos pero no hay regla fija.

1.-

19.95
29.97
39.98
50.02
60.05

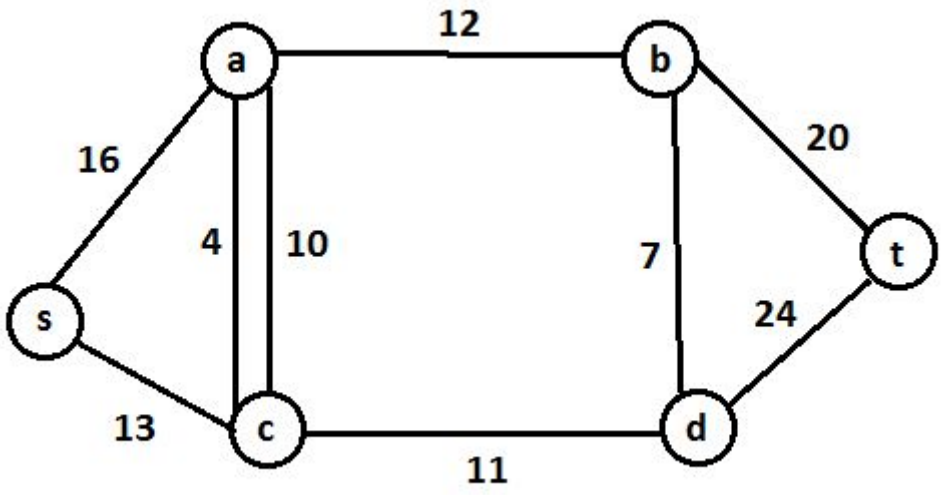
199.97

2.-

1000
40
1000
30
1000
20
1000
10

3.- Ford Fulkerson: Hay una fuente (s) y un destino o sumidero (t) y queremos saber cuál es el flujo máximo entre la fuente y el sumidero. El algoritmo consiste en tratar de tomar todos los caminos y llenar el flujo es decir subirlo.

Se elige el minimo camino hasta el sumidero, osea entre 16, 12 y 20, se elige el 12. Mientras haya camino se toma otro y se va cojiendo el mínimo flujo, por ejemplo 7 si es por abajo. Una vez que tenemos el flujo maximo el corte minimo es facil de determinar.

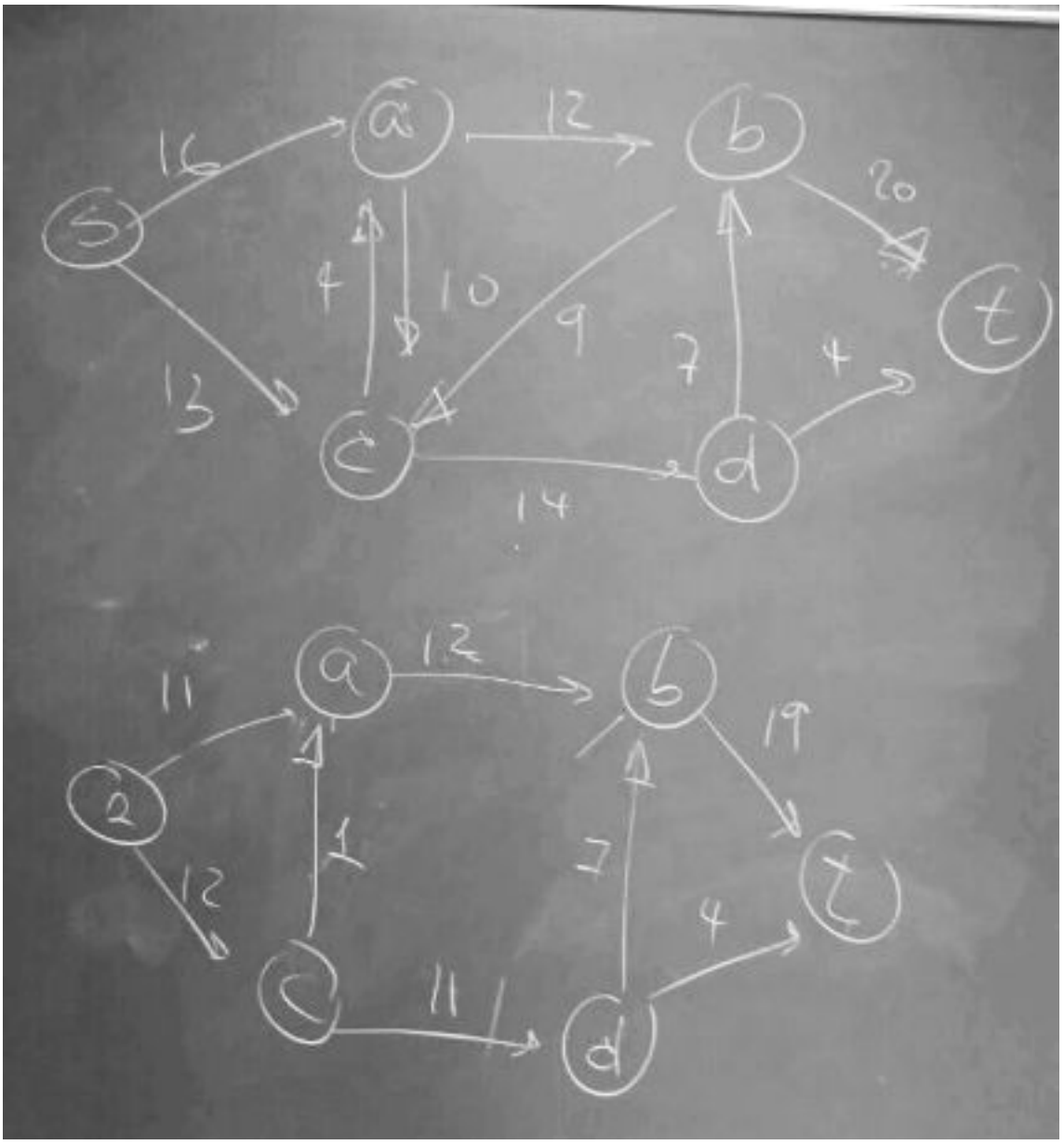


Lista de adyacencia del grafo de arriba (peso para llegar de un nodo a otro).

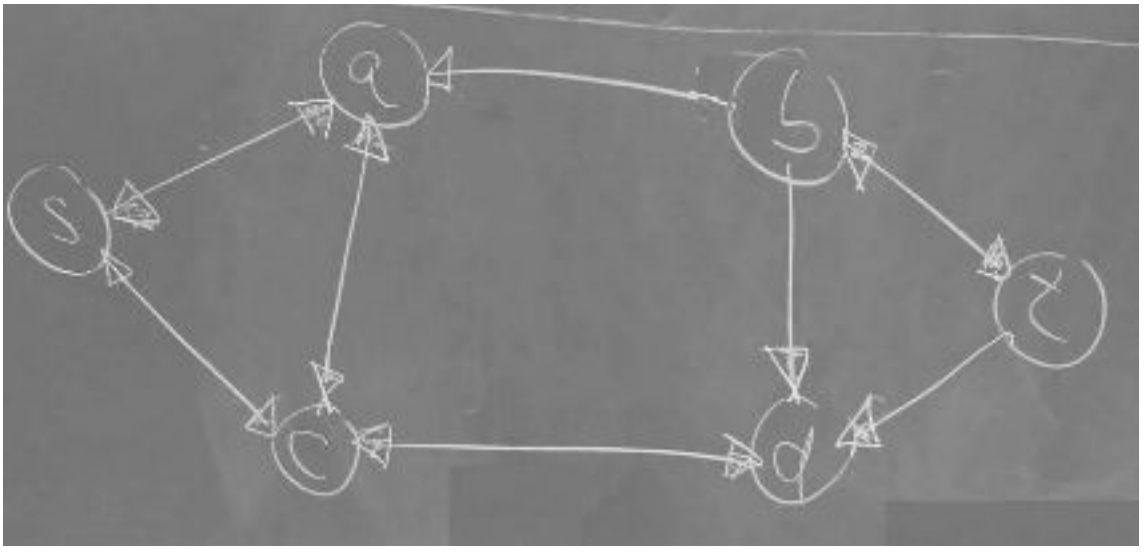
0	16	13	**0**	**0**	**0**

0	0	10	12	0	0
0	4	0	0	14	0
0	0	9	0	0	20
0	0	0	0	0	0

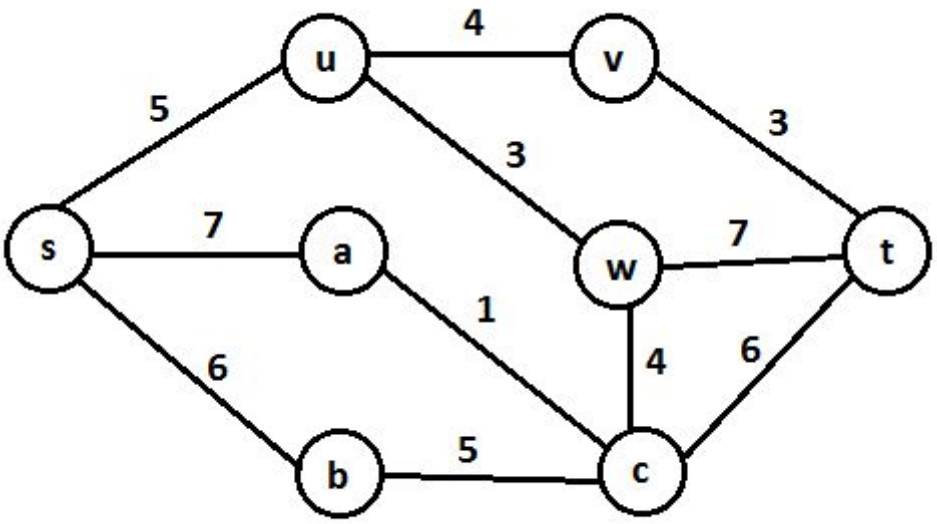
Grafo de origen y grafo de flujo máximo



Grafo residual

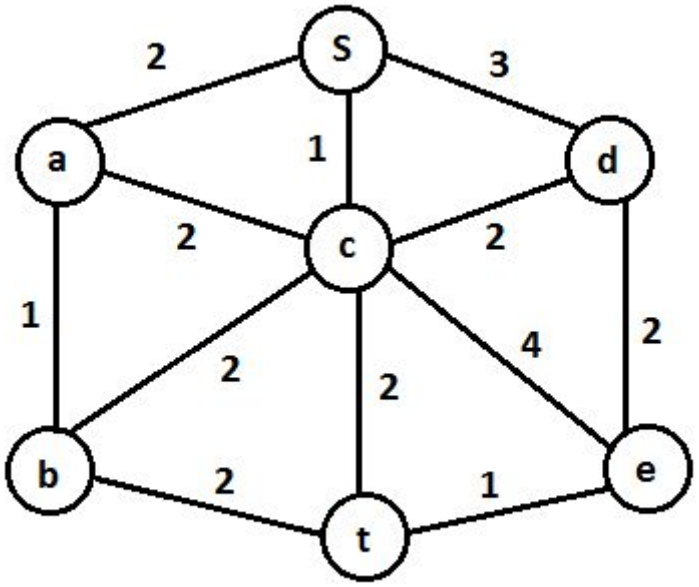


4.- Kardane



Semi-grafo → S1 = {S, U, V, W}

S2 = {A, B, C, T}



Flujo máximo

sabt → Primer camino → 1 es el minimo

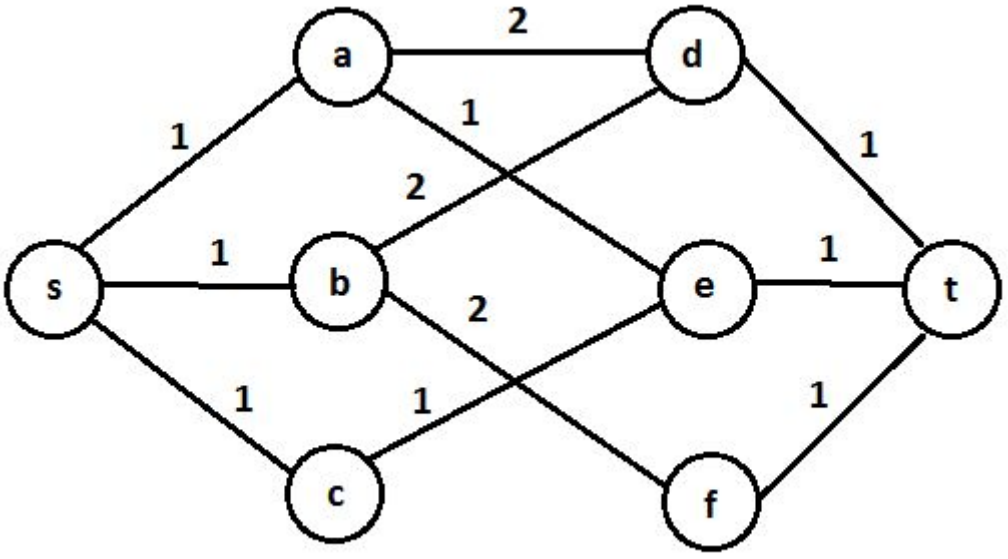
sct → Segundo camino → 1 es el minimo

sdet → Tercer camino → 1 es el minimo.

sact → Cuarto camino → 1 es el minimo

Todos los caminos están llenos así que ya sacamos el **flujo máximo.**

Grafo bipartito: Muchos fuentes van a muchos sumideros y estos se dirigen a un sumidero destino.



Pareo = aristas sin vertices comunes.

Pareo perfecto = maximo pareo.

Investigar problema de las 8 reinas. ← Backtracking

Las reinas no pueden estar en el mismo x, o en el mismo y, o el valor absoluto de la resta del x1 con x2 y y1 con y2 sean diferentes.

Angulos:

1.- Tenemos dos vectores (P0P1, P0P2) con una parte comun y queremos saber el angulo contra la manecillas del reloj. P0P1, P0P2, cual es el angulo que forman?

2.- P0P1, P1P2, queremos saber si siguiendo el trayecto yo doblo a la izquierda o la derecha.

3.- Tengo P1P2, P3P4, como saber si intersectan?

Polígono concavo y convexo:

Diremos que un polígono es **convexo** si todos sus ángulos interiores son menores de 180º.

Diremos que un polígono es **cóncavo** si alguno de sus ángulos interiores es mayor de 180º.

Fecha: 3/07/2017

Si tenemos a = 973, b = 301, se puede expresar como 973 = 3 * 301 + 70.

Por lo tanto 70 es igual a 1 * 973 - 3 * 301, por lo tanto el resto de la división de estos dos es 21 = 1 * 301 - 4 * 70 = 1 * 301 - 4 (1 * 973 - 3 * 301) = - 4 * 973 + 13 * 301.

7 = 1 * 70 - 3 * 21 = 1 * 973 - 3 * 301 - 3 * (-4 * 973 + 13 * 301) = 13 * 973 - 42 * 301.

Pasa lo **siguiente:**

a	b
973	301
301	70
90	21
7	0

Dos numeros son congruentes % n si la diferencia % n es 0.

n = 7, 3 y 10 son congruentes ya que ambos % 7 es 0.

1093028 * 190301 mod 100

28 * 1 = 28.

Un **anillo** lo definimos con dos operaciones, la cualidad del anillo es el cierre, el cierre quiere decir que si tenemos dos elementos en el anillo para dos operaciones este nos da un numero entre el mismo conjunto. Un anillo es un conjunto mas dos operaciones que cumplan las condiciones.

Las propiedades del anillo son asociatividad, el elemento neutro es 0, ya que si sumamos o restamos 0 da el mismo numero, 1 es la unidad ya que al multiplicar por 1 da el mismo número y puede tener un inverso.

En el **grupo** tenemos cierre, identidad, inverso y asociatividad, y si tiene commutatividad es un **abeliano, es decir que el orden de los factores no altera el producto.**

Tenemos el grupo de numeros reales

R = {0, 1, 2... n - 1}.

Se cumple que que (R, +) es un grupo es decir que los numeros reales que se suman es un grupo ya que se cumplen todas las unidades.

Z7+ = {0, 1, 2, 3, 4, 5, 6} → Dos numeros sumados módulo 7 que den 0 son (5 + 2)%7 = 0.

Z7* = {1, 2, 3, 4, 5, 6} → Dos números que multiplicados módulo 7 de 1 son (5 * 3)%7 = 1.

Z8* = {1, 2, 3, 4, 5, 6, 7} → Los números que sean primos relativos es decir que 8 modulo este numero de 1 son 3, 5 y 7. El inverso de 3 es 3 ya que 3 * 3 mod 8 = 1.

Tenemos el grupo:

R7+ = {0, 1, 2, 3, 4, 5, 6}, queremos elevar 3^4 es sumar 3 cuatro veces es decir 3 + 3 + 3 + 3 mod 7 = 12 % 5 = 7.

Z7* = {1, 2, 3, 4, 5, 6}, buscar 3^5 mod 7 = 3^2 mod 7 * 3^2 mod 7 * 3 mod 7 = 2 * 2 * 3 mod 7 = 12 % 7 = 5.

Z10* = {1, 2, 3, 4, 5, 6, 7, 8, 9}.

3^5 = 3^2 mod 10 + 3^2 mod 10 + 3 mod 10 = 9mod 10 * 9 mod 10 + 3 mod 10 = 81 mod 10 * 3 mod 10 = 1 * 3 mod 10 = 3.

n = 240, Z240* ⇒ **Grupo abeliano**, es un grupo abeliano ya que todos tienen inverso y tiene commutatividad.

Z12* = {1, 5, 7, 11}

Z6* = {1, 5}

Z15* = {1, 2, 4, 7, 8, 11, 13, 14}.

| Z15* | = 8 → También se escribe O(15) = 8.

Para saber cuanto seria el O(240) sería O(n) = pi(pi^ci - pi^ci -1)*pi → El primer pi es el 3.1416... y los otros son los numeros primos.

$240 = 3 * 5 * 2^4$

$O(240) = (3^1 \times 3^{1-1}) (5^1 * 5^{1-1}) (2^4 - 2^3)$.

O(240) = 2 * 4 * 8 = 64

Si lo aplicamos con 15 → 15 = 3 * 5 → 3^1 - 1 * 5^1 - 1= 2 * 4 = 8 y ese es el resultado que da.

$12 = 2^2 * 3 = O(12) = (2^2 - 2^1)(3 - 1) = 2 * 2 = 4$.

N = p * q → N es igual al producto de dos primos relativos.

Zn* = Zp* Z*q

n = 15, p = 5, q = 3

Z15* = {1, 2, 4, 7, 8, 11, 13, 14}

Z5* = {1, 2, 3, 4}

Z3* = {1, 2}

RESIDUO CHINO:

Esto se puede expresar como un conjunto:

1 → (1, 1) → 1 % 5 y 1 % 3.

2 → (2, 2) → 2 % 5 y 2 % 3.

4 → (4, 1) → Se toma 4 % 5 y 4 % 3.

8 → (3, 2) → Se toma 8 % 5 y 8 % 3.

11 → (1, 2)

13 → (3, 1)

14 → (4, 2)

14 * 13 mod 15 = 2

14 = (4, 2)

13 = (3, 1)

(4 * 3 mod 5, 2 * 1 mod 3)

= (2, 2) y (2, 2) = 2 en el ejemplo anterior.

$11 \text{ ^ } 3 \bmod 15 = (1^3 \bmod 5, 2^3 \bmod 3) = (1, 2) = 11$

11 = (1, 2)

29^100 mod 35 =

35 = 7 * 5

29 mod 7 = 1, 29 mod 5 = 4 ⇒ (1, 4)

(1^{100}mod7, 4^{100}mod5)

(1, 1) → 1

$3^{2/5}$mod 7 = 3 * 2 * 5^{-1} mod 7 = El inverso de 5 es 3 ya que 5 * 3 mod 7 = 1 entonces → 3 * 2 * 3 mod 7 = 4.

7^{100}mod13 = 7^9 mod 13 = 7^3 mod 13 * 7^3 mod 13 * 7^3 mod 13 = 125 mod 13 = 5.

7^{X} = 11 mod 13 =

Fecha: 5/07/2017

Fecha de examen, no hubo clases.

Fecha: 10/07/2017

Clase no copiada.

Fecha: 12/07/2017

1.- Algoritmo de Euclidex extendido:

- Congruencia
- Anillo
- Grupo
- Grupo abeliano
- Campo
- Campo galors
- Isomorfismo
- Teorema del residuo chino
- Grupo cíclico

Generar primos grandes

Teorema de Lagrange, si H es un subgrupo de G entonces || H || | || G ||.

Encriptación RSA (Ravin, Shamum y Adler).

Clave pública W, e → e → ganador de Z * N.

x → texto, cifrafo y = x^emodN

a. Escoger primos (grandes) P y q → N = p * q.

b. O(N) = (p-1)(q-1)
c. e E {1, 2, O(N) - 1}
d. Hallar d= e^-1 mod O(N).

x → texto, y = x^e.

Diffie-Hellman

a. p - primo
b. g E {2, 3, p-2}
c. (p, g)

Luego x → texto

k = $(a^x)^y = (a^y)^x$

Elgamal

Exponente de 1024 bits == 300 cifras decimales.

a) **Fuerza bruta $2^{1024} = 10^{300}$**
b) **Exponenciación rápida → 1024*15**

```
si = 1
mientras e > 0
s = n * n
if n es impar
s = s * n
return s.
```

Encryptar RSA. x = 4, p = 3, q = 11, c = 5, N = p*q

(N, e) = (3, 5), O(n) = (3-1) * (11-1) = 20.

$y = x^5 \bmod 20 = 4^5 \bmod 20 = 4^3 * 4^2 \bmod 20 = 4*16 \bmod 64 = 4$.

d = e^1

e = 5

d = 20

Z33* = {1, 2, 4, 5, 7, 8, 9, 10, 13, 14, 16, 17, 19, 20, 23, 25, 26, 28, 29, 31, 32}.

RSA

(2) P = 11, q =13, e = 7, dando cifra y = 15.

hallar X.

Y = 15 = X^e

$15 = X^7$

X = raiz septima de 15.

N = p*q → O(N) = (p-1)(q-1) = 10 * 12 = 120.

N = p = 11, q = 13

N P = 11 Q = 13

7 (7 , 7)

7 (7^-1, 7^-1)

(8, 8)

8, 9, 30, **41**, 52 , 63, 74, 85

2, 15, 28, **41.**

4) g = 1131, (N, e) (26, 23, 2111)

desencyptar

Sacar los últimos tres digitos de 3^1000.

5.- Ultimos digitos de 3^{1000},

101^4, 800, 000, 23 mod 23

3) $3^{1000 mod 100}$ mod 100 = 3^{0}mod100 = 0.

7) 29^100 mod 35 = 29^30 mod 35

PARA EL EXAMEN: Problemas de selección múltiple y problemas de residuo chino y todos los temas que el profesor diga el lunes.

Temas que van:

P vs NP → 2SAT, 3SAT, REDUCCIONES, PROBLEMAS TIPICOS.
Divide y vencerás

Programación Dinámica

Huffman, Dijkstra, Conjuntos disjuntos

Union Find

Hallar los dos puntos mas distantes de un plano.

N = p * q = 3 * 5 = 15

7^14 mod15

Z15* Z3* Z5*

7 → (1 , 2)

7^{-1} → (1^{14}, 2^{14})

1 → (1, 1)

N = p * q = 7 * 11 = 77

Hallar raiz quinta de 24 = $24^{1/5}$ = 24^{5-1}

Z77* (Z7* Z11*)

24 (3, 2)

5 (5, 5)

24^{31} = (2^{3}, 2^{9})

5^1 = 31 (3, 9) → 3, 10, 17, 24, 31, 38, 45, 9, 20, 31, 42

(5, 4) → 5, 12, 19, 26, 33, 40, 47, 54, 61, 4, 17, 30, 43, 56, 69, 82, 95, 108

Fecha: 19/07/2017

Ultimo dia de clases:

Para el examen:

- Problemas de maximum flow
- Problemas de P y NP → Si un problema es NP y se puede reducir a NP Duro.
- Problemas de reducción