1. Sobre Dp empareje con la respuesta correcta

   El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión. — Top down ✔

   Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado. — Bottom up ✔

2. El árbol de segmentos necesita como máximo n memoria adicional para su construcción (donde "n" es la cantidad de elementos a ser almacenados).

   **es verdadero**

3. La complejidad del algoritmo de Union Find con balanceo por ranking es O(logV) donde V es la cantidad de vértices.

   Seleccione una:
   - (•) Verdadero ✔
   - ( ) Falso

4. La respuesta correcta es 'Verdadero'
5. **Exponenciación binaria**

   La respuesta correcta es:
   Arrastre y rellene con el código correcto:

```
long long binary_exponentation( long long a, long long n ){
    long long result = 1;
    while( n != 0 ) {

        if( [n % 2] )
            [result = result * a;]

        [a = a * a;]

        [n >>= 1;]
    }
    return result;
}
```

6. El algoritmo BFS permite buscar un nodo específico dentro de cualquier grafo que sea conexo.

   **es verdadero**

7. 
```
int gcd(int a, int b){
    int r;
    while(b){
        [r = a % b;]
        [a = b;]
        [b = r;]
    }
    return a;
}
```

8. El algoritmo BFS permite realizar un ordenamiento topológico. **Si, tambien se puede**
9. **Complejidad de insertar en un heap binario: O(logn)**
10. **Complejidad de fibonacci con DP: O(n)**
11. El algoritmo BFS permite mostrar un árbol en amplitud, es decir por niveles antes que por profundidad. **es verdadero**
12. El principio de Optimalidad de Bellman dice "dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima" **es verdadero**

13. El algoritmo Prim para MST, necesita del Union Find para juntar los conjuntos disjuntos.

FALSO

14. El siguiente código para encontrar la potencia de un número a elevado a un numero n

```
double pow(double a, int n) {
 double ret = 1;
 while(n) {
  if(n%2 == 1) ret *= a;
   a *= a; n /= 2;
 }
 return ret;
}
```

tiene un error.

15.
```
int gcd(int a, int b) {
  while( b )){int r = a % b; a = b; b = r;}
  return a;
}
```

16. Take $v$ from $Q$
For each edge $v \to u$:
- Decrement $\deg(u)$ (essentially r
- If $\deg(u) = 0$, push $u$ to $Q$

'] y m es |E|. La complejidad de dicho algoritmo es: O(nm)

La complejidad del algoritmo es O(n+m)

17. El algoritmo quick hull tiene una complejidad esperada O(n log n), pero que puede degenerar a O(n^2) en el peor caso.

Es verdadero

18. Este algoritmo es un método iterativo, el cual, empieza con un flujo nulo y en cada iteración se va obteniendo un valor del flujo que va aumentando el camino, hasta que no se pueda aumentar más. Depende de tres puntos vitales:

Camino de la fuente al sumidero, donde cada una de las aristas tiene un flujo residual mayor que cero. Siendo el flujo residual, el flujo que se puede obtener en una arista una vez que haya pasado un flujo por ella. — Elegir...

Consiste simplemente en realizar una partición del conjunto de vértices en dos subconjuntos. — Elegir...

Se basa en ir aumentando el camino, hasta alcanzar el máximo. — Elegir...

19. El segment tree también conocido como árbol indexado es utilizado para resolver compresión de datos. Es una estructura muy eficiente para calcular sumas acumuladas .

Es el fenwick tree, no el segment tree

20. La complejidad de insertar en un heap binario es _____ O(logn)

21. Un árbol de segmento para un conjunto I de n intervalos usa O(n log n) de memoria de almacenamiento y puede construirse en un tiempo O(n log n). Los árboles de segmento soportan búsqueda para todos los intervalos que contienen un punto de consulta en O(log n + k), k el número de intervalos o segmentos recuperados.

es verdadero

22. Principio de optimalidad: Una política óptima tiene la propiedad de que cualquiera que sea el estado inicial y la decisión inicial son, las decisiones restantes deben constituir una política que puede ser óptima en relación con el estado resultante de la última decisión.

verdadero

23. Si, dada una subsecuencia de decisiones, siempre se conoce cuál es la decisión que debe tomarse a continuación para obtener la secuencia óptima, el problema es elemental y se resuelve trivialmente tomando una decisión detrás de otra, lo que se conoce como estrategia voraz o greedy.

24. Este algoritmo es un método iterativo, el cual, empieza con un flujo nulo y en cada iteración se va obteniendo un valor del flujo que va aumentando el camino, hasta que no se pueda aumentar más. Depende de tres puntos vitales:

Consiste simplemente en realizar una partición del conjunto de vértices en dos subconjuntos. — Elegir...

Se basa en ir aumentando el camino, hasta alcanzar el máximo. — Elegir...

Camino de la fuente al sumidero, donde cada una de las aristas tiene un flujo residual mayor que cero. Siendo el flujo residual, el flujo que se puede obtener en una arista una vez que haya pasado un flujo por ella. — Elegir...

La particion es **corte en redes de flujo**
aumentar el camino es **aumento de camino**
camino de la fuente al sumidero es **Red residual**

25. Contemplar un problema como una secuencia de decisiones equivale a dividirlo en problemas más pequeños y por lo tanto más fáciles de resolver como hacemos en Divide y Vencerás, técnica similar a la de programación dinámica.

verdadero

| | |
|---|---|
| **Comenzado el** | jueves, 9 de diciembre de 2021, 12:40 |
| **Estado** | Finalizado |
| **Finalizado en** | jueves, 9 de diciembre de 2021, 12:54 |
| **Tiempo empleado** | 14 minutos 5 segundos |
| **Puntos** | 4.00/7.00 |
| **Calificación** | **11.43** de 20.00 **(57%)** |

Pregunta **1**

Correcta

Puntúa 1.00 sobre 1.00

El Fenwick Tree es muy fácil para manejar un array de suma acumulativa y de este array de suma acumulativa es posible calcular la suma de las frecuencias en un cierto rango en el orden de O (n log (n)).

Seleccione una:

- Verdadero
- Falso ✔

Es O(logn)

La respuesta correcta es 'Falso'

Pregunta **2**

Incorrecta

Puntúa 0.00 sobre 1.00

Dada una cadena de tamaño m y un patrón de tamaño n. Un algoritmo de matching de cadenas que posee la siguiente lógica:

El algoritmo trata de localizar la posición de comienzo de una cadena dentro de otra. Previamente sobre el patrón a localizar se calcula una tabla de saltos (conocida como tabla de fallos) que después se utiliza (al examinar las cadenas) para hacer saltos cuando se localiza un fallo de coincidencia en la comparación de un carácter. Esta idea se fundamenta básicamente en la repetición de prefijos y sufijos en la sub-cadena.

Cuál es el nombre del algoritmo? boyer moore ✖

Cuál es su complejidad? O(m*n) ✖

Respuesta incorrecta.

La respuesta correcta es: Cuál es el nombre del algoritmo? → kmp, Cuál es su complejidad? → O(m+n)

Pregunta 3
Correcta
Puntúa 1.00 sobre 1.00

El par más cercano de puntos puede ser encontrado en tiempo proporcional a O(n^2) mediante una búsqueda por fuerza bruta. Para ello habría que computar las distancias entre las $\frac{n \cdot (n-1)}{2}$ combinaciones de pares de puntos, y elegir el par con la distancia más pequeña.

Seleccione una:

- Verdadero ✔
- Falso

La respuesta correcta es 'Verdadero'

Pregunta 4
Correcta
Puntúa 1.00 sobre 1.00

Podemos, mediante el Algoritmo de [Ford] ✔ - [Fulkerson] [Bellman] [Floyd] máximo de u[n] [Warshall]

Respuesta correcta

La respuesta correcta es:
Podemos, mediante el Algoritmo de [Ford]-[Fulkerson], encontrar el flujo máximo de una red.

Pregunta 5
Incorrecta
Puntúa 0.00 sobre 1.00

El siguiente código para encontrar la potencia de un número a elevado a un numero n

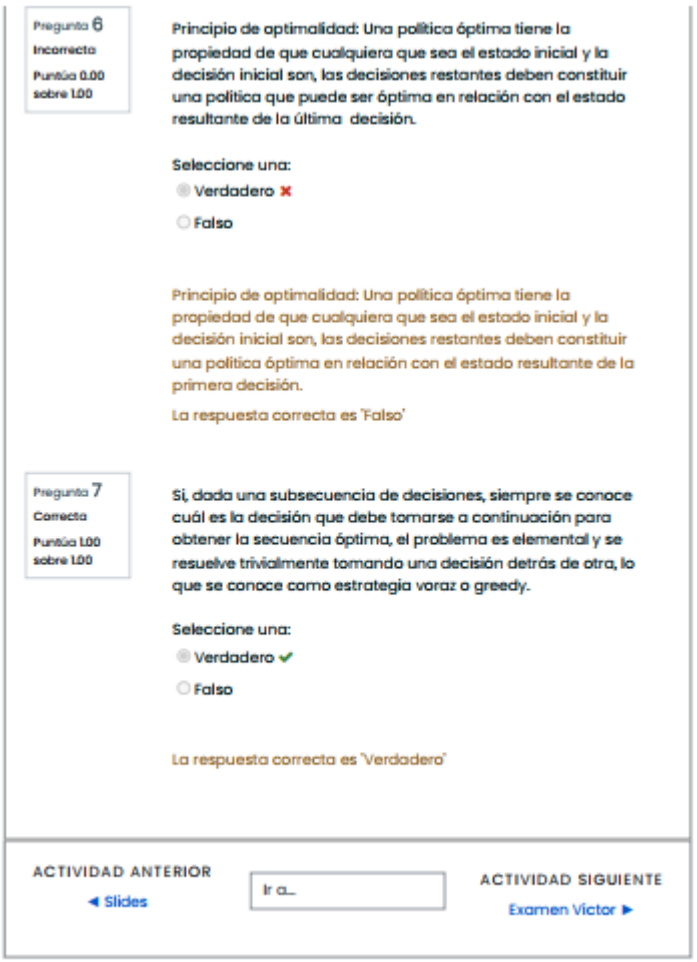
```
double pow(double a, int n) {
 double ret = 1;
 while(n) {
 if(n%2 == 1) ret *= a;
   a *= a; n /= 2;
 }
 return ret;
}
```

tiene un error.

Seleccione una:

- Verdadero ✖
- Falso

La respuesta correcta es 'Falso'



Pregunta 6
Incorrecta
Puntúa 0.00 sobre 1.00

Principio de optimalidad: Una política óptima tiene la propiedad de que cualquiera que sea el estado inicial y la decisión inicial son, las decisiones restantes deben constituir una política que puede ser óptima en relación con el estado resultante de la última decisión.

Seleccione una:
Verdadero ✖
Falso

Principio de optimalidad: Una política óptima tiene la propiedad de que cualquiera que sea el estado inicial y la decisión inicial son, las decisiones restantes deben constituir una política óptima en relación con el estado resultante de la primera decisión.

La respuesta correcta es 'Falso'

Pregunta 7
Correcta
Puntúa 1.00 sobre 1.00

Si, dada una subsecuencia de decisiones, siempre se conoce cuál es la decisión que debe tomarse a continuación para obtener la secuencia óptima, el problema es elemental y se resuelve trivialmente tomando una decisión detrás de otra, lo que se conoce como estrategia voraz o greedy.

Seleccione una:
Verdadero ✔
Falso

La respuesta correcta es 'Verdadero'

ACTIVIDAD ANTERIOR
◄ Slides

Ir a...

ACTIVIDAD SIGUIENTE
Examen Víctor ►

El algoritmo BFS permite buscar un nodo específico dentro de cualquier grafo que sea conexo.

Seleccione una:
Verdadero ✔
Falso

La respuesta correcta es 'Verdadero'

Arrastre sobre los espacios en blanco el código correcto.

```
int gcd(int a, int b){
    int r;
    while(b){
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

b = a;  a = r;  r = a << b;

El principio de Optimalidad de Bellman dice

"dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima"

Seleccione una:

- Verdadero
- Falso

Arrastre sobre el código la respuesta correcta.

```
function Union(x, y)
   xRoot := Find(x)
   yRoot := Find(y)

   // x and y are already in the same set
   if xRoot == yRoot
       return

   // x and y are not in same set, so we merge them
    if xRoot.rank < yRoot.rank
      xRoot.parent := yRoot
    else if xRoot.rank > yRoot.rank
      yRoot.parent := xRoot
   else
     xRoot.parent := yRoot
     yRoot.rank   := yRoot.rank + 1


function Find(x)
   if x.parent != x
     x.parent := [Find(x.parent)] ✔
   return x.parent
```

| x.parent | Find(x) | x.parent-1 | parent(y) |
|---|---|---|---|

La programación recursiva de la función de Fibonacci

```
int fib(int n){
    if(n < 2)
        return n;

    return fib(n-1) + fib(n-2);
}
```

tiene una complejidad, como mínimo, exponencial.

Seleccione una:

- (●) Verdadero ✔
- ( ) Falso

La complejidad para realizar un "update" en el árbol de Fenwick es O(logn).

Seleccione una:

- Verdadero ✔
- Falso

Sobre Dp empareje con la respuesta corecta

Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado. — Bottom up ✔

El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión. — Top down ✔

La respuesta correcta es: Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado. → Bottom up, El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión. → Top down

La programación recursiva de la función de Fibonacci

```
int fib(int n){
    if(n < 2)
        return n;

    return fib(n-1) + fib(n-2);
}
```

tiene una complejidad, como mínimo, exponencial.

Seleccione una:
- Verdadero ✔
- Falso

La respuesta correcta es 'Verdadero'

La complejidad del algoritmo de Fibonacci basado en matrices es O(log n) igual a la complejidad que se puede alcanzar utilizando la fórmula cerrada

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Seleccione una:
- Verdadero
- Falso ✖

La respuesta correcta es 'Verdadero'

El algoritmo BFS permite mostrar un árbol en amplitud, es decir por niveles antes que por profundidad.

Seleccione una:
- Verdadero ✔
- Falso

La respuesta correcta es 'Verdadero'

El algoritmo de Kruskal basa su complejidad en el Union Find.

Seleccione una:
- Verdadero ✖
- Falso

La complejidad esta dada por el sort u creación del heap que organiza las aristas.

La respuesta correcta es 'Falso'

La siguiente función consulta la suma acumulada en un árbol de Fenwick, donde tree es el array donde se guardan los valores cumulados, ind es la pocisión de consulta. Rellene con el código que falta, no utilice ningún espacio en su respuesta, ponga el código todo junto: ejem:var1=var2+3; sin espacio ni adelante ni atrás.

```
functionConsulta(ind)
    suma :=0
    while ind >0
        suma :=suma+tree[ind]
        ind :=ind-( ind&&-ind ✖ )
    returnsuma
```

La respuesta correcta es: ind&-ind

Como se sabe cada entero puede ser representado como suma de potencias de 2. De la misma manera las frecuencias acumulativas (suma de las posiciones del principio hasta una posición i) pueden ser representadas como suma de sub-frecuencias. Esta es la idea básica del árbol de Fenwick. Sólo se necesitaría un array (tree) de la misma longitud del array original.

Seleccione una:

- Verdadero ✔
- Falso

## Pregunta 1

**Incorrecta**
**Se puntúa 0.00 sobre 1.00**

La programación recursiva de la función de Fibonacci

```
int fib(int n){ if(n < 2) return n; return fib(n-1) + fib(n-2); }
```

tiene una complejidad, como mínimo, exponencial.

Seleccione una:

- Verdadero
- Falso ✗

La respuesta correcta es 'Verdadero'

## Pregunta 2

**Incorrecta**
**Se puntúa 0.00 sobre 1.00**

El algoritmo para recorrido de grafos DFS permite encontrar el camino más cercano entre dos nodos en un grafo con aristas no valoradas.
Seleccione una:

- Verdadero ✗
- Falso

es el BFS

La respuesta correcta es 'Falso'

## Pregunta 3

**Parcialmente correcta**

```
if x.parent != x
  x.parent := [Find(x.parent)]
return x.parent
```

Pregunta 6

**Incorrecta**
**Se puntúa 0.00 sobre 1.00**

Sobre Dp empareje con la respuesta corecta

El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión.

Recursividad sin memoria ✗

Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado.

Tabulación ✗

La respuesta correcta es: El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión. → Top down, Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado. → Bottom up

http://elvex.ugr.es/decsai/algorithms/slides/6%20dynamic%20programming.pdf
https://www.geeksforgeeks.org/segment-tree-efficient-implementation/

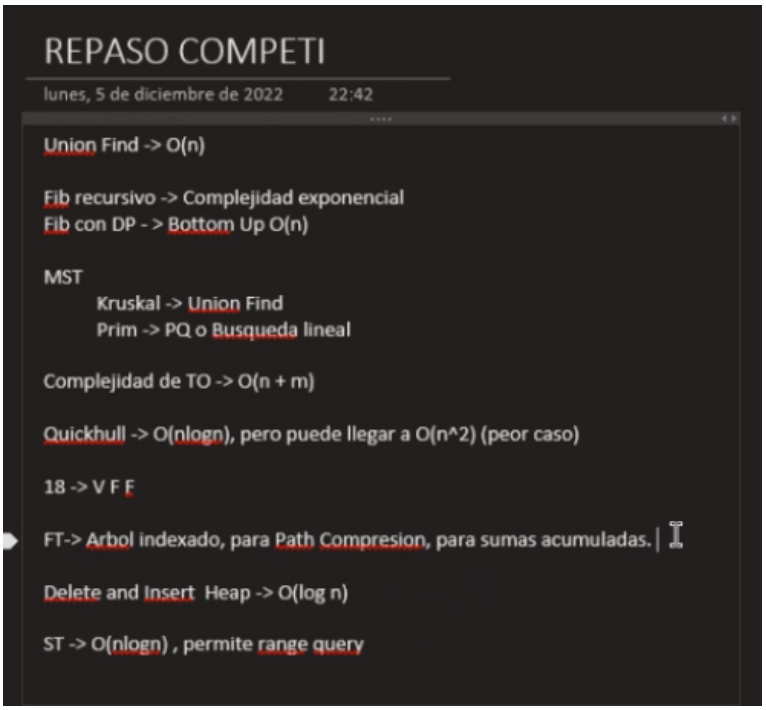
## Principio de Optimalidad de Bellman

[Bellman, R.E.: "Dynamic Programming". Princeton University Press, 1957]

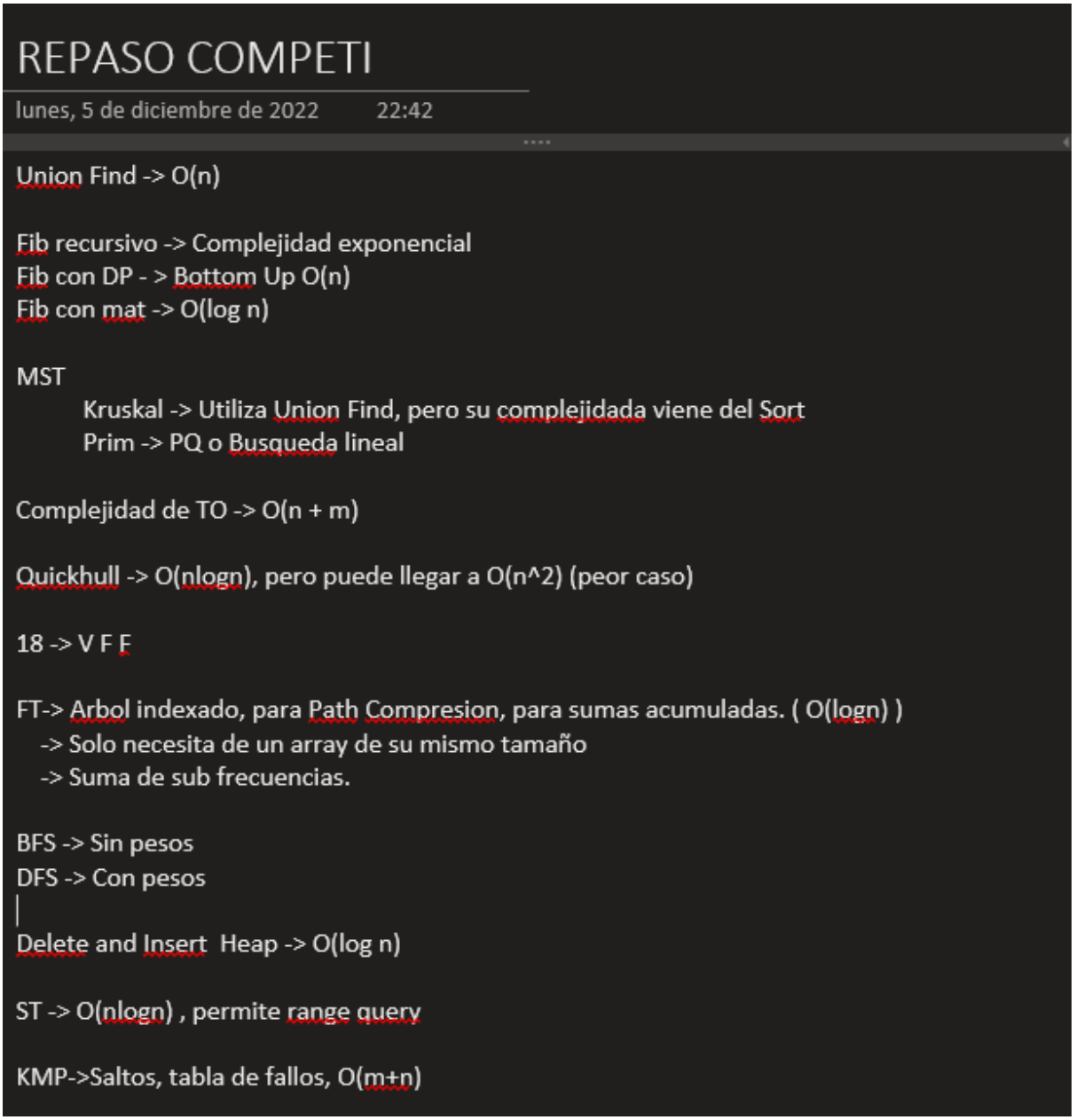
"Una política óptima tiene la propiedad de que, sean cuales sea el estado inicial y la decisión inicial, las decisiones restantes deben constituir una solucíón óptima con respecto al estado resultante de la primera decisión".

**Time Complexities:**

- Tree Construction: O( n )
- Query in Range: O( Log n )
- Updating an element: O( Log n ).



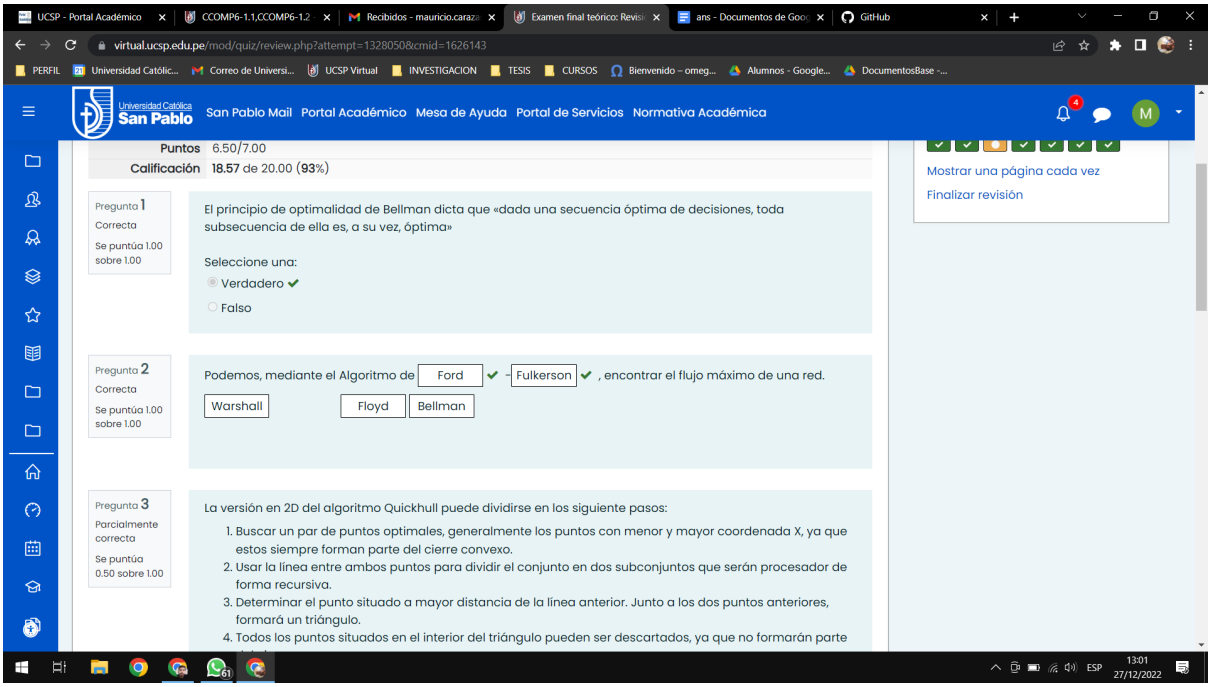

ya fue voy a ponerme una corbata roja y pararme en el parque del avión a las 1am

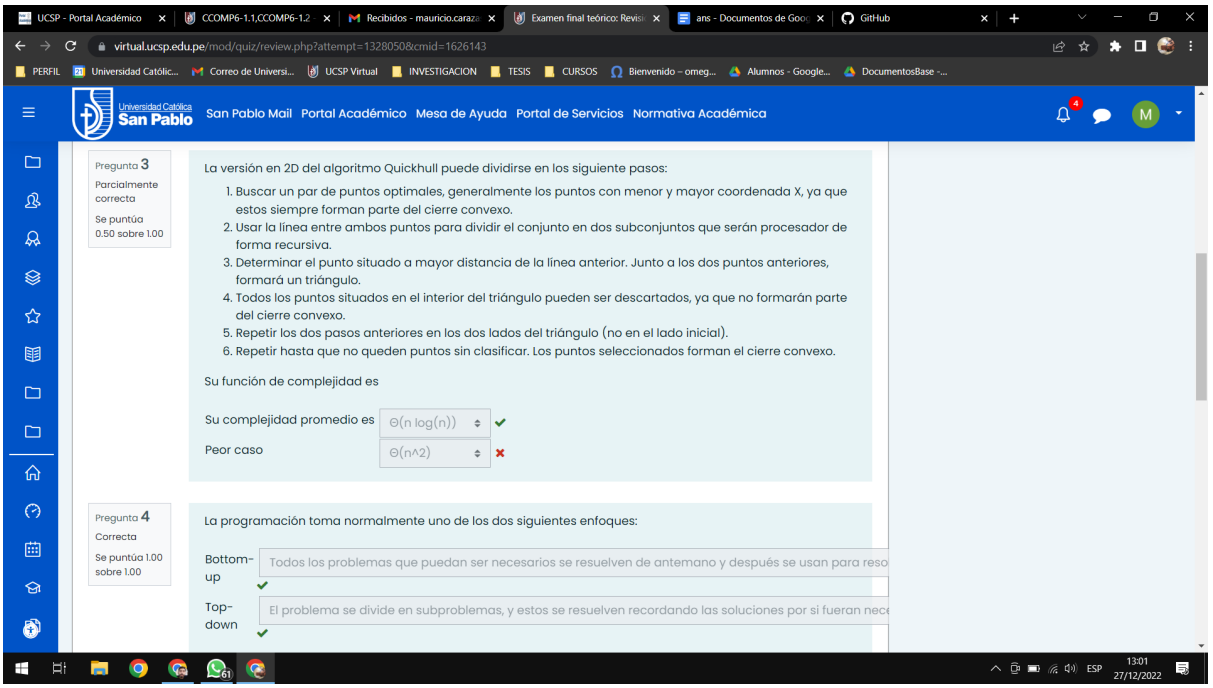


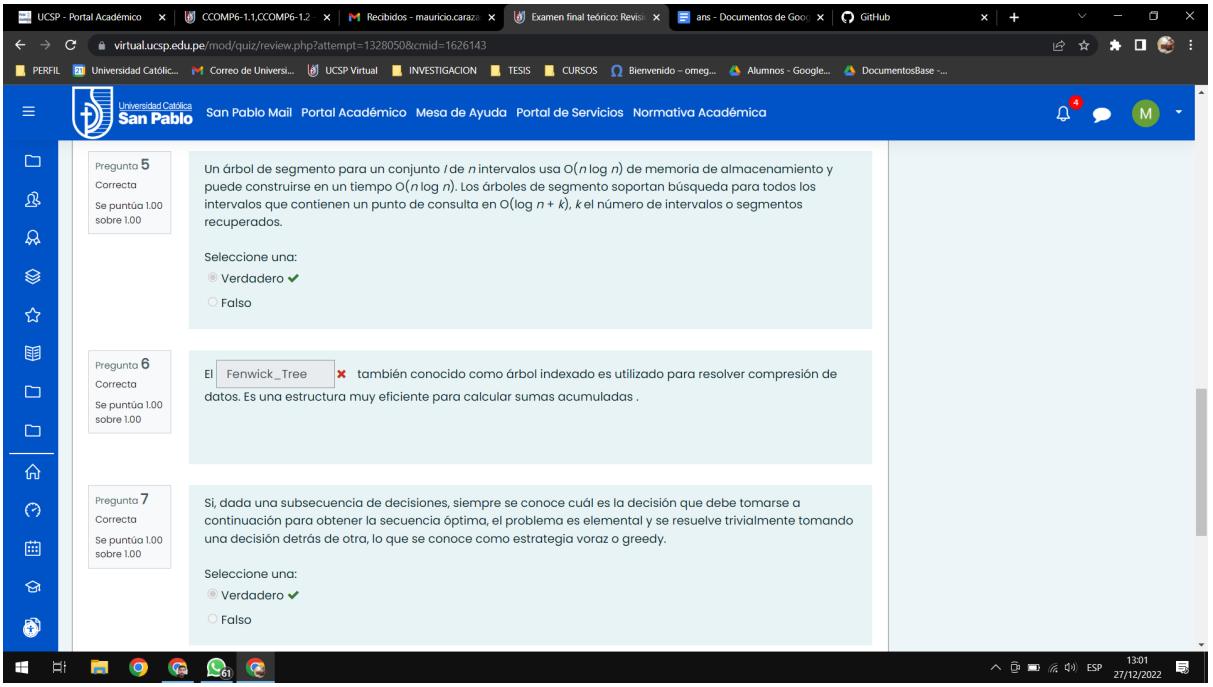

problema E del contest anterior Cheap dinner

https://codeforces.com/contest/1487/submission/116078483

Problema Bad prices
https://github.com/Waqar-107/Codeforces/blob/master/B-set/1213B.%20Bad%20Prices.cpp
+
Problema rising bacteria
https://github.com/AsifurRahman/Codeforces-Solutions/blob/master/579A%20-%20Raising%20Bacteria.cpp
varios soluciones entre ella apic transformation
https://github.com/samarthraj11/CODEFORCES
con respecto a lo anterior la soluciones tienen palabras arabes asi que cuidado con eso.


UCSP - Portal Académico
CCOMP6-1.1,CCOMP6-1.2
Recibidos - mauricio.caraza
Examen final teórico: Revisi
ans - Documentos de Goo
GitHub
virtual.ucsp.edu.pe/mod/quiz/review.php?attempt=1328050&cmid=1626143
PERFIL
Universidad Católic...
Correo de Universi...
UCSP Virtual
INVESTIGACION
TESIS
CURSOS
Bienvenido – omeg...
Alumnos - Google...
DocumentosBase -...
Universidad Católica San Pablo
San Pablo Mail
Portal Académico
Mesa de Ayuda
Portal de Servicios
Normativa Académica
Puntos 6.50/7.00
Calificación 18.57 de 20.00 (93%)
Mostrar una página cada vez
Finalizar revisión
Pregunta 1
Correcta
Se puntúa 1.00 sobre 1.00
El principio de optimalidad de Bellman dicta que «dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima»
Seleccione una:
Verdadero
Falso
Pregunta 2
Correcta
Se puntúa 1.00 sobre 1.00
Podemos, mediante el Algoritmo de Ford - Fulkerson , encontrar el flujo máximo de una red.
Warshall
Floyd
Bellman
Pregunta 3
Parcialmente correcta
Se puntúa 0.50 sobre 1.00
La versión en 2D del algoritmo Quickhull puede dividirse en los siguiente pasos:
1. Buscar un par de puntos optimales, generalmente los puntos con menor y mayor coordenada X, ya que estos siempre forman parte del cierre convexo.
2. Usar la línea entre ambos puntos para dividir el conjunto en dos subconjuntos que serán procesador de forma recursiva.
3. Determinar el punto situado a mayor distancia de la línea anterior. Junto a los dos puntos anteriores, formará un triángulo.
4. Todos los puntos situados en el interior del triángulo pueden ser descartados, ya que no formarán parte
ESP
13:01
27/12/2022


UCSP - Portal Académico
CCOMP6-1.1,CCOMP6-1.2
Recibidos - mauricio.caraza
Examen final teórico: Revisi
ans - Documentos de Goo
GitHub
virtual.ucsp.edu.pe/mod/quiz/review.php?attempt=1328050&cmid=1626143
PERFIL
Universidad Católic...
Correo de Universi...
UCSP Virtual
INVESTIGACION
TESIS
CURSOS
Bienvenido – omeg...
Alumnos - Google...
DocumentosBase -...
Universidad Católica San Pablo
San Pablo Mail
Portal Académico
Mesa de Ayuda
Portal de Servicios
Normativa Académica
Pregunta 3
Parcialmente correcta
Se puntúa 0.50 sobre 1.00
La versión en 2D del algoritmo Quickhull puede dividirse en los siguiente pasos:
1. Buscar un par de puntos optimales, generalmente los puntos con menor y mayor coordenada X, ya que estos siempre forman parte del cierre convexo.
2. Usar la línea entre ambos puntos para dividir el conjunto en dos subconjuntos que serán procesador de forma recursiva.
3. Determinar el punto situado a mayor distancia de la línea anterior. Junto a los dos puntos anteriores, formará un triángulo.
4. Todos los puntos situados en el interior del triángulo pueden ser descartados, ya que no formarán parte del cierre convexo.
5. Repetir los dos pasos anteriores en los dos lados del triángulo (no en el lado inicial).
6. Repetir hasta que no queden puntos sin clasificar. Los puntos seleccionados forman el cierre convexo.
Su función de complejidad es
Su complejidad promedio es Θ(n log(n))
Peor caso Θ(n^2)
Pregunta 4
Correcta
Se puntúa 1.00 sobre 1.00
La programación toma normalmente uno de los dos siguientes enfoques:
Bottom-up Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para reso
Top-down El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran nece
ESP
13:01
27/12/2022




https://codeforces.com/gym/414610