Comenzado el	lunes, 6 de diciembre de 2021, 10:08
Estado	Finalizado
Finalizado en	lunes, 6 de diciembre de 2021, 10:58
Tiempo empleado	50 minutos 2 segundos
Puntos	8.00/12.00
Calificación	**13.33** de 20.00 (**67**%)

Pregunta 1
Correcta
Puntúa 1.00 sobre 1.00

El siguiente algoritmo es el Heapsort ✔ :

```
def foo2(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[i] < arr[l]:
        largest = l
    if r < n and arr[largest] < arr[r]:
        largest = r
    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i]

        foo2(arr, n, largest)

def foo1(arr):
    n = len(arr)
    for i in range(n//2 - 1, -1, -1):
        foo2(arr, n, i)
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i] # swap
```

Shellsort

Balanaced Quicksort

Quicksort

Respuesta correcta

La respuesta correcta es:
El siguiente algoritmo es el [Heapsort]:

```
def foo2(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[i] < arr[l]:
        largest = l
    if r < n and arr[largest] < arr[r]:
        largest = r
    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i]

        foo2(arr, n, largest)

def foo1(arr):
    n = len(arr)
    for i in range(n//2 - 1, -1, -1):
        foo2(arr, n, i)
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i] # swap
        foo2(arr, i, 0)
```

Pregunta 2
Incorrecta
Puntúa 0.00 sobre 1.00

En el caso de codificar un grafo para un determinado problema, y no se va requerir ni hacer operaciones de insert ni delete, y el contenido de aristas es 0.8*n(n-1)/2, la mejor opción para repreentarlo es una matriz esparza de adyacencia.

Seleccione una:
- Verdadero ✖
- Falso

No necesita ser esparza al contrario eso incrementa el costo, basta con una matriz simple

La respuesta correcta es 'Falso'

Pregunta 3
Incorrecta
Puntúa 0.00 sobre 1.00

La complejidad del algoritmo mergesort es $\theta(nlogn)$ y de la función merge $\theta(logn)$.

Seleccione una:
- Verdadero ✖
- Falso

merge es O(n)

La respuesta correcta es 'Falso'

Pregunta 4
Correcta
Puntúa 1.00 sobre 1.00

El algoritmo mergesort es siempre más rápido que el insertion sort.

Seleccione una:
- Verdadero
- Falso ✔

En conjuntos pequeñps, ejemplo menos de 63 elementos, el insert es más rápido.

La respuesta correcta es 'Falso'

Pregunta 5
Incorrecta
Puntúa 0.00 sobre 1.00

Para realizar el ordenamiento topológico es necesario que la estructura sea un DAG (grafo dirigido acíclico)

Seleccione una:
- Verdadero ✖
- Falso

Puede ser cualquier grafo

La respuesta correcta es 'Falso'

Pregunta 6
Correcta
Puntúa 1.00 sobre 1.00

Determinar si un grafo puede ser coloreado con 2 colores, está en P, pero con tres colores es NP-completo, incluso cuando se restringe a los grafos planos.

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

La respuesta correcta es 'Verdadero'

Pregunta 7
Correcta
Puntúa 1.00 sobre 1.00

El siguiente código es para encontrar el número de Fibonacci,

* SIN UTILIZAR ESPACIOS, llene el pedazo de código que falta

```
def fib(n):
    F = [[1, 1],
         [1, 0]]
    if (n == 0):
        return 0
    power(F, n - 1)

    return F[0][0]

def multiply(F, M):

    x = (F[0][0] * M[0][0] +
         F[0][1] * M[1][0])
    y = (F[0][0] * M[0][1] +
         F[0][1] * M[1][1])
    z = (F[1][0] * M[0][0] +
         F[1][1] * M[1][0])
    w = (F[1][0] * M[0][1] +
         F[1][1] * M[1][1])

    F[0][0] = x
    F[0][1] = y
    F[1][0] = z
    F[1][1] = w

def power(F, n):

    M = [[1, 1],
         [1, 0]]

    # n - 1 times multiply the
    # matrix to {{1,0},{0,1}}
    for i in range( 2,n+1 ✔ ):
        multiply(F, M)
```

La respuesta correcta es: 2,n+1

Pregunta 8
Correcta
Puntúa 1.00 sobre 1.00

Si un grafo posee un circuito Euleriano, entonces cada vértice del grado tiene grado par, a excepción de un vértice.

Seleccione una:
- Verdadero
- Falso ✔

es par siempre

La respuesta correcta es 'Falso'

Pregunta 9
Finalizado
Puntúa 3.00 sobre 4.00

En la disciplina matemática de la teoría de grafos, una cobertura de vértices (en inglés, *vertex cover*) o simplemente cobertura de un grafo, es un conjunto de vértices tales que cada arista del grafo es incidente a al menos un vértice del conjunto.

El problema de encontrar la menor cobertura de vértices en un grafo se denomina problema de la cobertura de vértices. En teoría de la complejidad computacional se ha demostrado que este es un problema NP-completo.

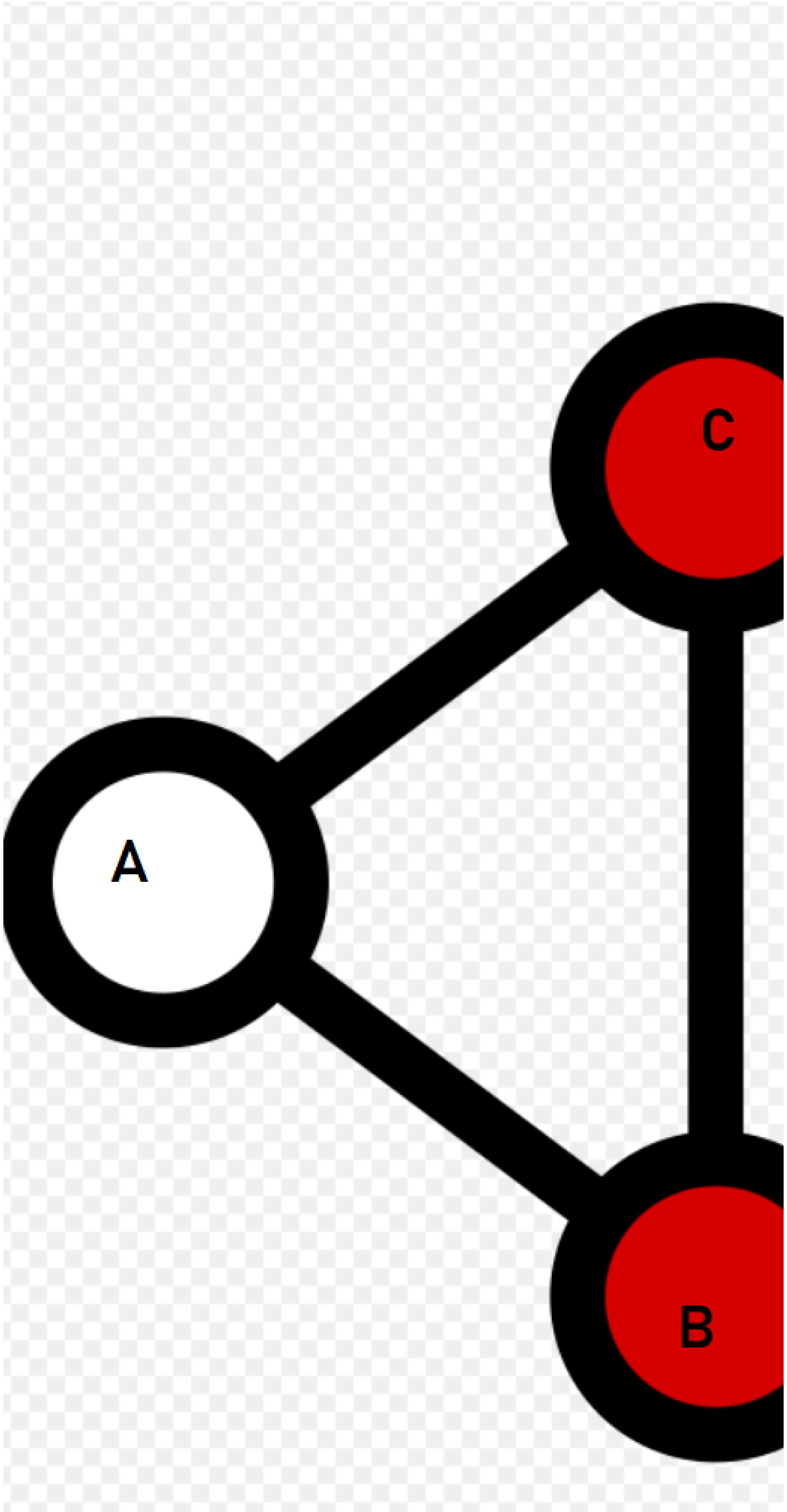
- Demuestre que el Problema de Cobertura de Vértices es NP

 * Algoritmo ND polinomial

 * Reducción polinómica, en este punto debe de determinar la complejidad de la reducción no basta con describir el algoritmo

- Código fuerza bruta que resuelva el problema para el siguiente grafo

* Suba todo en un archivo .pdf con el screenshot de la salida del código en c++ y la demostración



Para probar que el problema de la cobertura de verticee esta en la clase NP, necesitamos probar que el problema es verificable en tiempo polinomial.

Si tomamos un certificado para este problema, tendriamos un subconjunto V' de V. entonces se puede comprobar si el conjunto V' es una cobertura de vertices de tamaño t utilizando un algoritmo, es claro que esto se puede hacer en un tiempo polinomial. Por lo tanto el problema definido pertenece a la clase NP.

Ahora vamos a reducir el problema de Clique a el problema de la cobertura de vertices.

ADA Final.pdf

Comentario:
La prueba no estuvo informal,

ACTIVIDAD ANTERIOR

◀ Slides

Ir a...

Mantente en contacto

Universidad Católica San Pablo

https://ucsp.edu.pe

+51 54 605630 | +51 54 605600

institucional@ucsp.edu.pe

Descargar la app para dispositivos móviles