Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC1253 - Matemáticas Discretas

Tarea 6

23 de noviembre de 2020

2º semestre 2020 - Profesores G. Diéguez - F. Suárez

Requisitos

- La tarea es individual. Los casos de copia serán sancionados con la reprobación del curso con nota 1,1.
- **Entrega**: Hasta las 23:59:59 del 23 de noviembre a través del buzón habilitado en el sitio del curso (Canvas).
  - Esta tarea debe ser hecha completamente en LaTeX. Tareas hechas a mano o en otro procesador de texto **no serán corregidas**.
  - Debe usar el template LaTeX publicado en la página del curso.
  - Cada problema debe entregarse en un archivo independiente de las demas preguntas.
  - Los archivos que debe entregar son un archivo `PDF` por cada pregunta con su solución con nombre `numalumno-P1.pdf` y `numalumno-P2.pdf` , junto con un `zip` con nombre `numalumno.zip`, conteniendo los archivos `numalumno-P1.tex` y `numalumno-P2.tex` que compilan su tarea. Si su código hace referencia a otros archivos, debe incluirlos también.
- El no cumplimiento de alguna de las reglas se penalizará con un descuento de 0.5 en la nota final (acumulables).
- No se aceptarán tareas atrasadas.
- Si tiene alguna duda, el foro de Canvas es el lugar oficial para realizarla.

Problemas

Problema 1 - Algoritmos Recursivos

El algoritmo MergeSort tiene una función de recurrencia dada por

$$F(n) = \begin{cases} 1, & \text{if } n < 2 \\ \lceil F(\frac{n}{2}) \rceil + \lfloor F(\frac{n}{2}) \rfloor + n, & \text{if } n \geq 2 \end{cases}$$

Si reemplazamos $n = 2^k$ para $n \geq 2$:

$$\begin{aligned} F(2^k) &= F(2^{k-1}) + F(2^{k-1}) + 2^k && (1) \\ &= 2F(2^{k-1}) + 2^k && (2) \\ &= 2 \cdot 2F(2^{k-2}) + 2 \cdot 2^k && (3) \\ &= 2^2 \cdot F(2^{k-2}) + 2 \cdot 2^k && (4) \end{aligned}$$

si lo generalizamos, para un k cualquiera:

$$F(2^k) = 2^k \cdot F(1) + k \cdot 2^k \tag{5}$$

lo cual nos sirve pues no es una función recursiva, y podemos conocer su complejidad. Si volvemos a la variable n, como $n = 2^k$, $k = \log_2 n$ y por lo tanto

$$F(n) = n \cdot F(1) + \log_2 n \cdot n \tag{6}$$

y como vimos en clases, $n \log_2 n$ prevalece sobre n, por lo que mergeSort tiene una complejidad de $\mathcal{O}(n \log n)$

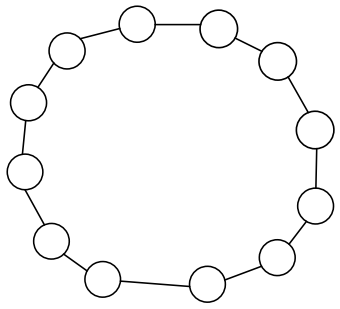
Problema 2 - Grafos

Para esta demostración ocuparemos inducción fuerte.

BI: Como el ciclo más pequeño es de largo 3 e impar, nuestro caso base se cumple, pues un grafo con un ciclo de largo 3 es isomorfo a C_3

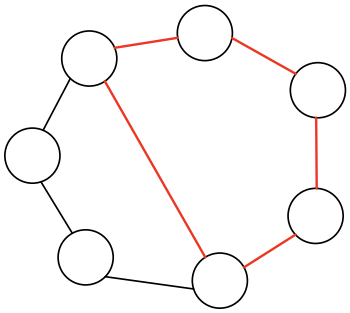
HI: Supongamos que para un grafo con un ciclo de largo impar de tamaño k ¡n, se cumple que este tiene un subgrafo inducido isomorfo a C_3 o P_4.

TI: Ahora pongamonos en dos casos.



si tenemos un grafo con un ciclo de largo impar de tamaño n, y sin diagonales, claramente este tiene un subgrafo inducido isomorfo a P_4, pues es cosa de escoger 4 vertices cualesquiera y estos formaran un camino entre sus aristas.

Si por el contrario nos encontramos con una diagonal



podemos separar el grafo en dos subgrafos, uno con número de aristas impares y otro con número de aristas pares. Resulta que el grafo con número aristas impares posee un ciclo de largo menor a n, y por hipótesis de inducción, cumple con tener un subgrafo inducido isomorfo a C_3 o P_4. Podemos aplicar esta lógica a cualquier grafo de manera recursiva, incluido un grafo completo, si este tiene un ciclo de largo impar, pues basta con ir dibujando las aristas una por una y fijandose en el grafo con número de aristas impares que se forma. De esta manera, queda demostrado que un grafo que posea un ciclo de largo impar, posee un subgrafo inducido isomorfo a C_3 o P_4.