Repaso I3

Clase 27

IIC 2133 - Sección 2

Prof. Mario Droguett

Sumario

Interrogación 3

Objetivos a evaluar en la I3

- ☐ Identificar dominios de aplicación de algoritmos en grafos
- ☐ Aplicar algoritmos canónicos de grafos
- ☐ Modificar algoritmos de grafos

Lo esencial de esta interrogación es decidir qué algoritmo usar y cómo modificarlo/usarlo de inspiración

Interrogación 3

Formato de la prueba

- 2 horas de tiempo
- Pool de 4 preguntas para elegir 3
- Cada pregunta incluye un título que describe sus temas
- ¡**SOLO** se entregan 3 preguntas respondidas!

Nota de la I3: promedio de las 3 preguntas entregadas

Interrogación 3

Material adicional

- Pueden usar un formulario/apuntes durante la prueba
- Debe estar escrito a mano (puede ser impreso de tablet)
- Una hoja (por ambos lados)
- Sugerencia: incluyan los pseudocódigos vistos

No se aceptarán diapositivas impresas

les recuperativas

Formato de la prueba recuperativa

- Solo aquellos con justificación
- 1 hora de tiempo
- 2 preguntas
- Con formulario tal como la prueba no rendida
- Nota: promedio de las dos preguntas

Esta nota reemplaza la interrogación a la que se faltó

Objetivos de la clase

- ☐ Recordar elementos esenciales de los contenidos a evaluar
- ☐ Identificar diferencias de contenidos que aplican a un mismo escenario
- ☐ Aplicar algunos de los contenidos a ejemplos concretos
- ☐ Conocer enfoque esperado en las respuestas de algunos ejemplos

Sumario

Algoritmos en grafos

Cada algoritmo tiene un contexto de aplicación

- ¿Qué problema resuelve?
- ¿Qué queda guardado en sus variables/EDD's?

Además, el grafo estudiado debe tener ciertas características

- Dirección de aristas: dirigido o no dirigido
- Pesos en aristas: sin pesos, pesos no negativos, pesos libres

Es vital conocer los requisitos y contextos de aplicación de cada uno

Algoritmos en grafos

Familias de problemas que estudiamos

1. Recorrido en amplitud
   1.1 BFS: Grafos dirigidos (y también no dirigidos)
2. Rutas más baratas
   2.1 Dijkstra: costos no negativos
   2.2 Bellman-Ford: costos libres
3. Árboles de cobertura mínimos (MST)
   3.1 Kruskal y Prim: grafo no dirigido con costos libres

Recordar que en "este mundo", las complejidades dependen de E y V

Algoritmos en grafos

Aspectos de implementación

- BFS es un caso particular de Dijkstra
  - Costos iguales para toda arista
  - No requiere estructuras más poderosas que listas
- Dijkstra y Prim son algoritmos codiciosos
  - Su decisión codiciosa se basa en elegir menor costo actual
  - Esto se mantiene en una **cola de prioridad**
- Kruskal basa su funcionamiento en determinar comunidades
  - Implementación de **conjuntos disjuntos**
  - Permiten saber de forma eficiente si dos elementos están en el mismo subárbol

No olvidar las estructuras que se usan por debajo en cada algoritmo

Algoritmos en grafos

Orientaciones para el estudio

- ☐ Comprender qué problema resuelve cada algoritmo
- ☐ Comprender su funcionamiento/pseudocódigo
- ☐ Aplicarlos a situaciones diferentes a los problemas originales: modificarlos!

Incluyan en el formulario los pseudocódigos!

Sumario

Ejemplo: BFS

Ejercicio (I3 P4(b) - 2022-2)

El diámetro de un árbol no dirigido conexo T se define como el largo del camino más largo en T. Suponga que existe un único camino de largo máximo en T con extremos u y v. Si x es un nodo cualquiera, se puede demostrar que el nodo más lejano a x es u o v. Usando este resultado, proponga un algoritmo que determine el diámetro de un árbol T.

Ejemplo: BFS

La propiedad descrita permite plantear el siguiente algoritmo

`Diameter`(V, E)**:**

$x \leftarrow$ cualquier nodo de V

$d \leftarrow \texttt{BFS}(x)$

$u \leftarrow$ nodo que tiene máximo $d[\cdot]$

$d \leftarrow \texttt{BFS}(u)$

$D \leftarrow \max\{d[v] \mid v \in V\}$

return D

donde se usa una versión modificada de `BFS` para que retorne el arreglo con distancias desde la fuente, así como el nodo asociado a cada distancia.

Ejemplo: Dijkstra

Ejercicio (I3 P4(a) - 2022-2)

Considere un grafo dirigido $G = (V, E)$ que en lugar de aristas con pesos tiene nodos con pesos. El problema de rutas más cortas desde una fuente se define análogamente, donde el costo de un camino es la suma de los costos de sus nodos. Proponga cómo modificar el grafo para poder utilizar el algoritmo de Dijkstra para resolver dicho problema con la misma complejidad vista en clase.

Ejemplo: Dijkstra

Ejercicio (I3 P4(a) - 2022-2)

Al iniciar el recorrido Dijkstra desde una fuente s, el costo de este nodo no se considera pues está presente en todos los posibles caminos que salen de él. Nos interesamos en los costos para llegar a los vecinos en un paso.

De acuerdo con esto, dado un grafo con pesos en sus nodos, podemos modificar el grafo incluyendo pesos en sus aristas de acuerdo al nodo de llegada:

```
CreateWeights(V, E):
    for v ∈ V :
        for (x, y) ∈ E :
            if v = y :
                cost(x, y) ← cost(v)
```

Al ejecutar Dijkstra, el costo resultando de cada camino debe ser aumentado en $cost(s)$ para incorporar el costo de la fuente.

Ejemplo: BFS

Ejercicio (I3 P4 - 2023-1)

En un laboratorio interesa almacenar una gran cantidad de residuos en la menor cantidad de recintos posible. Sabemos que ciertos residuos son incompatibles entre sí y no pueden almacenarse en el mismo recinto por su riesgo de explosiones o reacciones químicas.

Ejemplo: BFS

Ejercicio (I3 P4 - 2023-1)

(i) [1 pto.] Describa cómo modelar con grafos el escenario de incompatibilidades de residuos, de tal forma que sirva para resolver el inciso (ii). Específicamente, indique qué tipo de grafo utilizará, qué representan los nodos y las aristas y en qué estructura de datos almacenará el grafo.

(ii) [2 pto.] Proponga el pseudocódigo de un algoritmo que opere sobre su grafo propuesto en (i) y que retorne un booleano indicando si es posible o no almacenar todos los residuos en 2 recintos. Su algoritmo debe ejecutarse a lo más en tiempo lineal respecto al tamaño del grafo.

Ejemplo: BFS

Ejercicio (I3 P4 - 2023-1)

(i) Se puede usar un grafo no dirigido, sin costos, de forma que los nodos son residuos y las aristas indican que los nodos conectados son incompatibles. El grafo se puede almacenar como listas de adyacencias o matriz de adyacencias indistintamente.

Idea: intentaremos 2-colorear el grafo (cada color representa un recinto)

Ejemplo: BFS

```
2Col(s):
    for u ∈ V − {s} :
        u.color ← blanco;  u.δ ← ∞;  π[u] ← ∅;  u.flag ← 0
    s.color ← gris;  s.δ ← 0;  π[s] ← ∅;  u.flag ← 0
    Q ← cola vacía
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco :
                v.color ← gris;  v.δ ← u.δ + 1;  v.flag ← (1 − u.flag)
                π[v] ← u
                Insert(Q, v)
            else:
                if v.flag = u.flag :
                    return false
        u.color ← negro
    return true
```

Sumario

Recomendaciones finales

Para los ejemplos vistos en clase

- Replicarlos comprendiendo los pasos de su resolución
- Asegurarse de poder motivar las decisiones

Pautas anteriores

- Hay harto material resuelto en el repo!
- No se aprendan pautas... seleccionen y aprovéchenlas
- Planifiquen su solución antes de verla, y luego consulten la pauta

Objetivos de la clase

- ☐ Recordar elementos esenciales de los contenidos a evaluar
- ☐ Identificar diferencias de contenidos que aplican a un mismo escenario
- ☐ Aplicar algunos de los contenidos a ejemplos concretos
- ☐ Conocer enfoque esperado en las respuestas de algunos ejemplos