COMPLEJIDAD COMPUTACIONAL

ALGORITMOS NO-DETERMINISTAS

MARCO SILVA HUERTA

Semestre: 2024-2

1. Ruta más corta

1.1. Dar su forma canónica

Problema de decisión: Dada una gráfica no dirigida $G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de aristas, y dados dos vértices u y v, y un entero positivo k, ¿existe una trayectoria entre u y v cuyo peso total sea menor que k?

1.2. Diseñar un algoritmo no-determinístico polinomial

- Partimos que trabajaremos con:
  - Una grafica
  - Aristas
  - Vértices
  - Dos vértices preseleccionados
  - un entero
- Vamos a comenzar con que distancias entre vértices son desconocidas.
- Creamos un arreglo para guardar esas distancias (distancias)
- Esas distancias se van a inicializar con valores infinitos
- Excepto para la distancia del vertice de inicio y final la cual será 0.
- Usaremos una cola de prioridad para explorar todo los vértices.
- El vertice de inicio será el primero en agregarse a esta cola con distancia 0 (elegido aleatoriamente)
- Para poder explorar toda la gráfica vamos a seleccionar los vértices adyacentes al vértice que tengamos evaluando en la cola
- Una vez alcanzado vamos a actualizar las distancias a sus vecinos
- La actualización debe ser si es que encontramos un camino más corto
- Una vez que terminamos de revisar toda la grafica verificamos si la distancia entre los vértices de inicio y final es menor a la distancia k
- Si lo es, hemos encontrado la ruta

- Si no, no hay una ruta menor.
- Nuestro proceso termina cuando todos los vértices fueron explorados o cuando la cola de prioridad está vacía.

```
    Función principal rutaMasCorta(grafica, inicio, final, k):
    // Paso 1: Inicialización
    distancias = inicializarDistancias(grafica, inicio)
    colaPrioridad = inicializarColaPrioridad(inicio)

    // Paso 2: Exploración
    mientras colaPrioridad no esté vacía:
        verticeActual = extraerVerticeMinimo(colaPrioridad)
        para cada vecino en grafica[verticeActual]:
            actualizarDistancia(distancias, verticeActual, vecino)

    // Paso 3: Verificación
    si distancia entre inicio y final < k:
        devolver "Sí, hay una ruta de peso menor que k
        entre inicio y final"
    sino:
        devolver "No, no hay una ruta de peso menor que k
        entre inicio y final"

Función inicializarDistancias(grafica, inicio):
    distancias = arreglo de tamaño igual al número de
    vértices en la gráfica
    para cada vértice en la gráfica:
        si vértice es igual a inicio:
            distancias[vértice] = 0
        sino:
            distancias[vértice] = infinito
    devolver distancias

Función inicializarColaPrioridad(inicio):
    colaPrioridad = cola de prioridad
    agregar inicio a colaPrioridad con prioridad 0
    devolver colaPrioridad

Función extraerVerticeMinimo(colaPrioridad):
    extraer y devolver vértice con menor prioridad
    de colaPrioridad

Función actualizarDistancia(distancias, verticeActual, vecino):
    si distancia desde inicio hasta vecino pasando por verticeActual
    es menor que distancias[vecino]:

        distancias[vecino] = distancia desde inicio hasta vecino
        pasando por verticeActual

        agregar vecino a colaPrioridad con prioridad
        distancias[vecino]
```

2. 3-SAT

2.1. Dar su forma canónica

Problema de decisión 3-SAT: Dada una fórmula booleana en forma conjuntiva normal (CNF) con exactamente tres literales por cláusula, ¿existe una asignación de valores de verdad a las variables que haga que la fórmula sea verdadera?

2.2. Diseñar un algoritmo no-determinístico polinomial

```
    Función principal rutaMasCorta(grafica, inicio, final, k):
    // Paso 1: Inicialización
    distancias = inicializarDistancias(grafica, inicio)
    colaPrioridad = inicializarColaPrioridad(inicio)

    // Paso 2: Exploración
    mientras colaPrioridad no esté vacía:
        verticeActual = extraerVerticeMinimo(colaPrioridad)
        para cada vecino en grafica[verticeActual]:
            actualizarDistancia(distancias, verticeActual, vecino)

    // Paso 3: Verificación
    si distancia entre inicio y final < k:
        devolver "Sí, hay una ruta de peso menor que k entre inicio y final
    sino:
        devolver "No, no hay una ruta de peso menor que k entre inicio y fi

Función inicializarDistancias(grafica, inicio):
    distancias = arreglo de tamaño igual al número de vértices en la gráfic
    para cada vértice en la gráfica:
        si vértice es igual a inicio:
            distancias[vértice] = 0
        sino:
            distancias[vértice] = infinito
    devolver distancias

Función inicializarColaPrioridad(inicio):
    colaPrioridad = cola de prioridad
    agregar inicio a colaPrioridad con prioridad 0
    devolver colaPrioridad

Función extraerVerticeMinimo(colaPrioridad):
    extraer y devolver vértice con menor prioridad de colaPrioridad

Función actualizarDistancia(distancias, verticeActual, vecino):
    si distancia desde inicio hasta vecino pasando por verticeActual es men
        distancias[vecino] = distancia desde inicio hasta vecino pasando po
        agregar vecino a colaPrioridad con prioridad distancias[vecino]
```

Referencias

[1] C. M. Andreas y S. Guido, *Introduction to Machine Learning with Python*, 1th. O'Reilly, 2016.

[2] S. Russell y P. Norvig, *Inteligencia Artificial Un Enfoque moderno*, 2nd. Pearson Prentice Hall, 2016.