Tarea 2

Dijkstra y análisis amortizado

Profesores: Benjamín Bustos y Gonzalo Navarro
Auxiliares: Sergio Rojas y Diego Salas

Fecha de entrega: jueves 30 de mayo a las 23:59

El objetivo principal de esta tarea es que podamos evidenciar empíricamente una diferencia en las complejidades globales entre algoritmos tradicionales y aquellos que son modificados para tener mejores tiempos amortizados. Para ello, usaremos el problema de las distancias más cortas en un grafo no dirigido con pesos estrictamente positivos: Dados un grafo $G = (V, E)$ con $|V|$ vértices y $|E|$ aristas que las unen, cada una de ellas con un costo o peso w_i asociado, definimos como el camino más corto entre dos vértices como el grupo de aristas que los conectan con el menor peso acumulado posible. De esta forma, el problema de la distancias más cortas en un grafo es el de encontrar el camino más corto desde una raíz al resto de nodos.

1. Descripción de la Tarea

Para esta tarea, el trabajo será analizar, implementar y evaluar experimentalmente los algoritmos que serán descritos para computar el camino más corto de un grafo generado. Se deberá entregar un informe que describa la experimentación y resultados de este análisis.

2. Algoritmo de Dijkstra

Dado un nodo raíz para un grafo con pesos positivos en sus aristas, encuentra el árbol de caminos más cortos para ese nodo raíz, de modo que para cualquier otro nodo en el grafo, la distancia entre la raíz y ese nodo es mínima para las aristas pertenecientes al árbol que los conectan. Este algoritmo se describe de la siguiente manera:

1. Definimos dos arreglos de tamaño $|V|$, *distancias* y *previos*.
2. Definimos una estructura Q permita:
   - Almacenar pares de la forma $(distancia, nodo)$.
   - Obtener y eliminar el par con menor distancia.
   - Tener acceso directo desde cada nodo (por medio de un puntero) al par que lo representa, además de poder modificar su distancia con la función *decreaseKey* (en el contexto del algoritmo, no existe la necesidad de aumentar la distancia, solo de reducirla).
3. Definimos la distancia del nodo raíz como 0, su nodo previo como -1, y agregamos el par $(distancia = 0, nodo = raíz)$ a Q.

4. Por cada nodo v distinto de la raíz en el grafo:
   - Definimos $distancias[v]$ como infinita y $previos[v]$ como indefinido.
   - Agregamos el par $(distancia = \infty, nodo = v)$ a Q.
5. Se espera que la creación de Q se resuelva por medio de un *Heapify*, que transforme un array con las distancias en una cola en tiempo lineal ($O(n)$).
6. Mientras Q no se encuentre vacío, repetimos:
   a) Obtenemos el par (d, v) con menor distancia en Q y lo eliminamos.
   b) Por cada vecino u del nodo v:
      i. Si la distancia guardada para u ($distancias[u]$) es mayor a la distancia guardada para v ($distancias[v]$) más el peso de la arista (u, v), actualizamos el valor de la distancia de u, guardamos v como el nodo previo de u y actualizamos la distancia del par que representa al nodo u en Q utilizando $decreaseKey$.
7. Retornamos el arreglo de previos y distancias.

2.1. Algoritmo 1: Dijkstra con Heap

Como estructura Q, se utiliza un Heap como cola de prioridad, este debe soportar las funcionalidades mencionadas en la descripción del algoritmo, y que se construya en tiempo lineal. En este primer algoritmo, para la función *decreaseKey* se espera un tiempo logarítmico, en el que actualizar un elemento de Q reordena la estructura del árbol.

2.2. Algoritmo 2: Dijkstra con Colas de Fibonacci

Como estructura Q, se utiliza una cola de Fibonacci como cola de prioridad, que permita las mismas operaciones, pero implemente *decreaseKey* en un tiempo constante. En el libro "Introduction to Algorithms" de Cormen et al. (ver la bibliografía del curso) hay una descripción detallada de la implementación de las colas de Fibonacci.

3. Objetivos

Para esta tarea, se deberá:

1. Implementar el algoritmo de Dijkstra utilizando un Heap como estructura de datos para la cola de prioridad. La complejidad de este algoritmo es de $O(e \log v)$, en el que e es la cantidad de aristas y v la cantidad de nodos.
2. Implementar el algoritmo de Dijkstra utilizando una cola de Fibonacci como estructura de datos para la cola de prioridad. La complejidad de este algoritmo es de $O(e + v \log v)$.
3. Evaluar el rendimiento de cada implementación en términos de tiempo de ejecución, y mostrar su cota teórica por medio de un fitting sobre los resultados, los cuales deberán reflejar esta complejidad.

4. Experimentación

Tanto las estructuras para el grafo como para las colas de prioridad descritas deben ser implementadas en su totalidad por ustedes. Los experimentos se realizarán utilizando datos sintéticos. A partir de un grafo con $v = 2^i$ nodos, genere aleatoriamente $e = 2^j$ aristas con pesos aleatorios y uniformes dentro del rango (0..1], con $i \in \{10, 12, 14\} \wedge j \in [16...22]$. El grafo resultante debe ser conexo y no dirigido, para esto se recomienda utilizar el siguiente método:

- Agregar $v - 1$ aristas al grafo, de tal forma que generen un árbol cobertor. Una forma es que para cada nodo i, se lo conecta con un nodo aleatorio elegido en $[1..i - 1]$.
- Una vez garantizada la conectividad del grafo, añadir las $2^j - (v - 1)$ aristas restantes.

Para cada par (i, j), ejecute ambos algoritmos y evalúe sus desempeños; repita este proceso al menos 50 veces (con grafos distintos) para poder obtener resultados estadísticos significativos, graficándolos.

Para realizar el fitting, considere los valores sucesivos de e para cada valor fijo de v, haciendo regresión lineal de los tiempos en términos de la variable e y obteniendo la pendiente de la recta. Interprete esa pendiente en términos de la complejidad que debiera tener cada tipo de cola de prioridad.

Documente las características del hardware utilizado (CPU, RAM, etc.), el sistema operativo, así como el lenguaje utilizado y la versión del compilador. Tenga cuidado con el tamaño de los grafos, es posible que no disponga de la suficiente memoria para alguno de los algoritmos. Calcule e indique si esto puede ocurrir (o ocurre) en su máquina.

5. Entregables

Se deberá entregar el código y un informe donde se explique el experimento en estudio. Con esto se obtendrá una nota de código ($NCod$) y nota de informe ($NInf$). La nota de la tarea será:

$$NT_1 = 0.5 \cdot NCod + 0.5 \cdot NInf$$

5.1. Código

La entrega de código debe ser hecha en C, C++ o Java. Tiene que contener:

- (**0.5 pts**) README: Archivo con las instrucciones para ejecutar el código, debe ser lo suficientemente explicativo para que cualquier persona solo leyendo el README pueda ejecutar la totalidad de su código.
- (**0.5 pts**) Experimento: Creación de los nodos y aristas a utilizarse en los experimentos, esto basándose en la cantidad pedida en este mismo informe.
- (**1.5 pts**) Dijkstra con Heap.
- (**1.5 pts**) Dijkstra con colas de Fibonacci.
- (**1.5 pts**) Obtención de resultados: La forma en el que se obtienen los resultados pedidos es correcta, es decir, se cuentan los tiempos de ejecución adecuados.
- (**0.5 pts**) Main: Un archivo o parte del código (función main) que permita ejecutar los distintos métodos de construcción y experimentos.

5.2. Informe

El informe debe ser claro y conciso. Se recomienda hacerlo en LaTeX. Debe contener:

- (**0.8 pts**) Introducción: Presentación del tema en estudio, resumir lo que se dirá en el informe y presentar la hipótesis mencionada en el enunciado.
- (**0.8 pts**) Desarrollo: Presentación de algoritmos, estructuras de datos, en lo que se diferencian los algoritmos/estructuras y cómo funcionan y por qué. Recordar que los métodos ya son conocidos por el equipo docente, lo que importa son sus propias implementaciones.
- (**2.4 pts**) Resultados: Especificación de los datos que se utilizaron para los experimentos, la cantidad de veces que se realizaron los tests, con qué inputs, que tamaño, etc. Se debe mencionar en que sistema operativo y los tamaños de sus cachés y RAM con los que se ejecutaron los experimentos. Se deben mostrar gráficos/tablas y mencionar solo lo que se puede observar de estos, se deben mostrar los valores y parámetros que se están usando.
- (**1.2 pts**) Análisis: Comentar y concluir sus resultados. Se hacen las inferencias de sus resultados.
- (**0.8 pts**) Conclusión: Recapitulación de lo que se hizo, se concluye lo que se puede decir con respecto a sus resultados. También ven si su hipótesis se cumplió o no y analizan la razón. Por último, se menciona qué se podría mejorar en su desarrollo en una versión futura, qué falta en su documento, qué no se ha resuelto y cómo se podrían extender.

Todo lo mencionado debe estar en sus informes en las secciones en las que se señalan, la falta de algún aspecto o la presencia de algún aspecto en una sección equivocada hará que no se tenga la totalidad del puntaje.