**ACTIVIDAD 15: ANÁLISIS EMPÍRICO**

SECCIÓN 1: ANALISIS TEORICO ESPACIO-TEMPORAL

Sección 1.1: Porqué Lista de Adyacencias y no Matriz de Adyacencias?

Una de las primeras decisiones que fue necesario tomar para afrontar esta actividad fue la selección de la implementación de la ED Grafo más eficiente para el problema, luego de deliberar sobre la cuestión llegamos a la conclusión que la mejor implementación en este caso sería el Grafo con listas de Adyacencias, veamos el porqué

Sabemos por lo visto en teoría que el espacio de ejecución de la implementación por matriz de Adyacencia es O(n^2), mientras que el espacio para la lista de adyacencias es O(n + a), por las restricciones del problema, en el peor caso (donde n = 500 y a = (500\*499)/2, el espacio para la matriz de adyacencias seria O(500^2) = 250000 mientras que el espacio en la lista de adyacencias seria O(500 + 124750) = 125250 , la diferencia de espacio es aproximadamente la mitad y entonces se justifica el uso de la Lista de Adyacencias

Y que ocurre con el tiempo de ejecución? Sabemos que, en este aspecto, la Matriz de Adyacencias es mejor ya que al ser una matriz, los accesos a la misma se pueden realizar en tiempo constante mientras que los accesos en la lista de adyacencias son O(n+a), pero en este aspecto, es posible reducir el tiempo de los accesos a las listas por medio del uso de punteros por ende se elimina la ventaja que poseía la matriz en cuanto al orden de acceso a la estructura. Por ende, nos fue más valioso ahorrar en espacio de ejecución del grafo

Sección 1.2: Análisis de Tiempo y Espacio de las EDs:

Analicemos los tiempos y espacios de Ejecución ED por ED:

Cola: Esta ED se comporta de la misma forma que la ED Cola vista en clase, por ende, los tiempos de las operaciones son:

Grafo: Esta ED implementa un Grafo según la implementación de Grafo con Listas de Adyacencias según la teoría, por ende, los tiempos de ejecución son:

Disjoint-Set: El Disjoint-Set esta presente de dos formas distintas, aunque ambas implementaciones respetan que la estructura interna de los cjtos es un árbol, las operaciones son implementadas de forma distinta, ya que una de las implementaciones del disjoint Set NO hace uso de las Heurísticas vistas en teoría, entonces, denominaremos al DIsjoint-Set SH a la implementación de disjoint set que NO hace uso de las heurísticas y a Disjoint-Set CH a la implementación de Disjoint-Set que HACE uso de las heurísticas vistas en teoría, entonces, los tiempos de ejecución son:

* DIsjoint-Set SH
  + Unión:
* DIsjoint-Set CH
  + Union

Heap: La ED Heap se comporta de la misma manera que la ED vista en teoría, entonces los timepos de ejecución son:

Sección 1.3: Análisis de Tiempo y Espacio de los Problemas:

Analicemos los tiempos y espacios de ejecución problema por problema:

Problema 1: Grafo Conexo:

Problema A: BFS

Problema B: Conjunto Disjunto

Problema 2: Árbol Minimal de Cubrimiento:

Problema 1A: Kruskal Lista Ordenada con Disjoint Set Con Heurística

Problema 1B: Kruskal Lista Ordenada con Disjoint Set Sin Heurística

Problema 1C: Kruskal Heap con Disjoint Set Con Heurística

Problema 1D: Kruskal Heap con Disjoint Set Sin Heurística

SECCIÓN 2: TABLAS DE RESULTADOS EMPÍRICOS

A continuación se muestran las tablas de resultados empíricos producto de ejecutar los algoritmos implementados para una variedad de grafos, se resaltará en amarillo el tiempo menor entre las variantes de implementación para los problemas resueltos. Los tiempos se midieron en Nanosegundos.

Tabla de resultados empíricos para el ejercicio 1:

|  |  |  |  |
| --- | --- | --- | --- |
| Grafo | | BFS (NanoSeg) | Disjoint-Set |
| N | A |  |  |
| 500 | 40000 | 3873061 | 9276409 |
| 5 | 10 | 38407 | 23838 |
| 500 | 124750 | 641316 | 8495705 |
| 200 | 15000 | 118529 | 494976 |
| 71 | 900 | 38076 | 49332 |
| 190 | 300 | 115550 | 28804 |
| 420 | 69870 | 216862 | 1130994 |
| 2 | 1 | 11588 | 2980 |
| 179 | 179 | 76481 | 20859 |
| 50 | 49 | 36750 | 5628 |

Tabla de resultados empíricos para el ejercicio 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Grafo | | Ordenado (NanoSeg) | | Heap | |
| N | A | C/heurística | S/heurística | C/heurística | S/heurística |
| 80 | 100 | 976377 | 969755 | 2882779 | 68893 |
| 500 | 124750 | 171285116 | 105070936 | 121618687 | 120227126 |
| 20 | 30 | 66880 | 39399 | 77474 | 47676 |
| 361 | 500 | 208254 | 263214 | 1223037 | 113563 |
| 211 | 210 | 77143 | 111908 | 73501 | 44697 |
| 124 | 6999 | 2803649 | 2632477 | 2105054 | 2139818 |
| 2 | 1 | 5959 | 35427 | 5297 | 2648 |
| 173 | 10000 | 4039267 | 3691625 | 2662937 | 3410863 |
| 10 | 10 | 10264 | 7615 | 3973 | 4636 |
| 300 | 41258 | 17191054 | 16624563 | 13602729 | 14041088 |

SECCIÓN 3: CONCLUSIONES

Observando las tablas de resultados empíricos, es posible observar que:

* Para el Problema 1:
  + Si A >> N, el BFS tendrá un tiempo de ejecución menor al Disjoint-Set ya que el tiempo del disjoint set es dependiente de la cantidad de arcos presentes en el grafo mientras que el BFS NO necesita recorrer todos los arcos del grafo para determinar si un grafo es conexo o no
  + Si el grafo es ralo (esto es, A se acerca a N-1), el tiempo del disjoint set será mucho menor que el tiempo del BFS
  + Si A = N, el DIsjoint Set parece ser más eficiente que el BFS, esto se debe a que el grafo en si será ralo y por la observación anterior, el disjoint set será más eficiente en tiempo
* Para el Problema 2:

SECCIÓN 4: CODIGOS FUENTE

El lenguaje elegido para implementar las soluciones y EDs fue Java, usando el IDE Eclipse