Informe TP2

Facundo Gonzalez Barnator, Joaquin Polonuer

Presentación y descripción del problema

Dada una cantidad de oficinas N y una cantidad de módems W, queremos minimizar el costo de conectar las oficinas usando dos tipos de cables: UTP y fibra óptica. El precio por centímetro de cable UTP es menor o igual al de fibra óptica, pero sólo lo podemos utilizar si la distancia entre dos oficinas es menor o igual a R.
Sabiendo esto, podemos observar que las oficinas que están a distancia mayor que R deben usar cable de fibra óptica y las de distancia menor o igual, UTP.
Vamos a resolver este problema modelandolo como un grafo en el que las oficinas son vértices y las aristas los posibles cables que las unen.

Explicación del algoritmo desarrollado y justificación.

El grafo que modela el problema es un grafo denso, donde cada oficina está conectada a las N-1 restantes con el cable adecuado (UTP o fibra óptica, dependiendo de la distancia). Ya sabemos de antemano con qué tipo de cable vamos a conectar cada par de oficinas, dado que elegimos uno u otro en función de la distancia, eligiendo UTP siempre que podemos.

Tenemos entonces un grafo de N vértices y $O(N^2)$ aristas. Poder colocar W módems es equivalente a tener W componentes conexas, cada una con 1 modem. Notar que, como el costo está dado por las conexiones que se realizan entre las oficinas, no es relevante en qué vértice de la componente colocamos el modem.
Luego, para minimizar el costo total, queremos llegar a un bosque generador que tenga W árboles y sea el mínimo entre todos los posibles.

Por cada arista que agregamos disminuimos en 1 la cantidad de árboles del bosque. De está manera si agregamos $N - W$ aristas vamos a quedarnos con $N - (N - W) = W$ componentes y eso es lo que queremos. Además, queremos que sea el bosque generador de costo mínimo entre los bosques de $N - W$ aristas.

Para esto es muy útil pensar en el invariante de algoritmo Kruskal:

- En la iteración i se tiene un bosque generador de i aristas que es mínimo entre los bosques de i aristas.

La idea es correr Kruskal sobre este grafo, hasta que formemos W componentes conexas. Luego el invariante nos asegurará que es mínimo entre los bosques de W árboles (o N-W aristas).

Pseudocódigo:

```
construir grafo denso a partir de oficinas()
kruskal():
    componentes = N
    UTP, F = 0
    while(componentes>W):
        e = arista más liviana que no genera ciclos
        agrego(e)
        if (usaUTP(e)):
            UTP += peso(e)
        else:
            F += peso(e)
        componentes--
    return UTP, F
```

Elegimos utilizar la implementación de Kruskal con complejidad $O(N^2)$. Es la más óptima en este caso, ya que al ser un grafo completo, $M \in O(N^2)$, por lo que utilizar la implementación $O(M * log(N))$ resultaría en un algoritmo de complejidad $O(N^2 * log(N))$.
Utilizamos una implementación de DSU particular para obtener la complejidad deseada. La idea es mantener un vector de componentes donde inicialmente cada nodo pertenece a su componente, y un vector de componente más cercana, en la que guardamos, para cada componente conexa, cual es la arista de costo mínimo que sale de esa componente. Luego el DSU tiene un método próxima_arista que devuelve la arista más chica que no genera ciclos, es decir, que une dos componentes distintas.
Luego, ésta información se debe actualizar en la función `unite`, que se encarga de unir las componentes de los nodo que se están uniendo y actualizar `más_cercano` con las nuevas aristas que salen de la componente.

Una vez hecho esto, las complejidades en el DSU son:

1. $O(N^2)$ para inicializar,
2. $O(N)$ para unir
3. $O(N)$ obtener la próxima arista

Donde 2. y 3. deben hacerse $O(N)$ veces, de manera que la complejidad resultante es $O(N^2 + (N-1) * N) = O(N^2)$.

Comparación de tiempos de cómputo entre implementaciones

Vamos a comprar:

- Kruskal $O(N)$
- Kruskal $O(M * log(N))$, con path compression y union by rank
- Kruskal $O(M * log(N))$, sin path compression

Para cada implementación creamos casos de test aleatorios con N entre 2 y 2000, luego ajustamos curvas del tipo $A * N^2$ y $A * N^2 * log(N)$ para verificar que la complejidad obtenida efectivamente correspondiera a la complejidad teórica del algoritmo (Figs 1, 2 y 3). El resultado fue satisfactorio y nos permitió comprobar empíricamente el tiempo de cómputo de cada implementación.

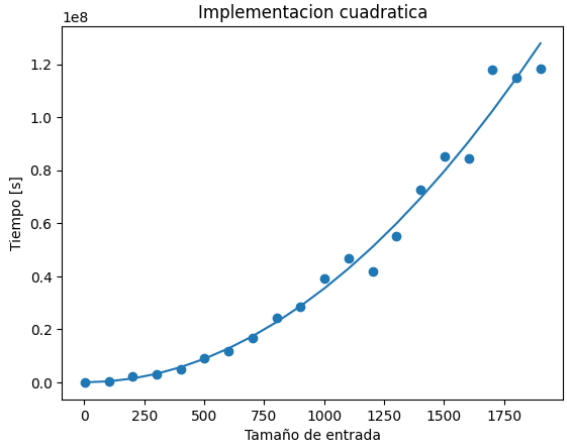

Fig 1: Ajuste realizado a los tiempos de cómputo obtenidos para la implementación $O(N^2)$. El valor obtenido para la constante fue de ~ 35

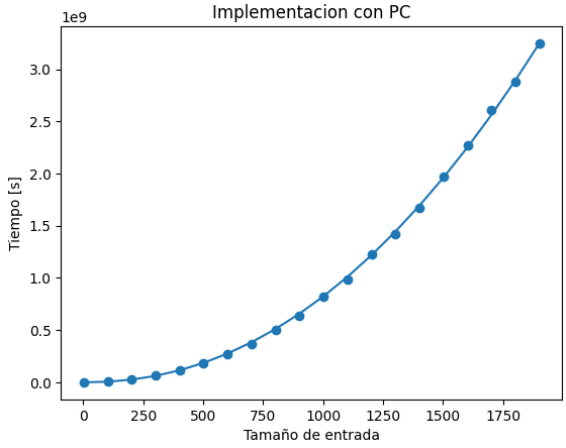

Fig 2: Ajuste realizado a los tiempos de cómputo obtenidos para la implementación $O(M * log(N))$ con path compression. El valor obtenido para la constante fue de ~ 118

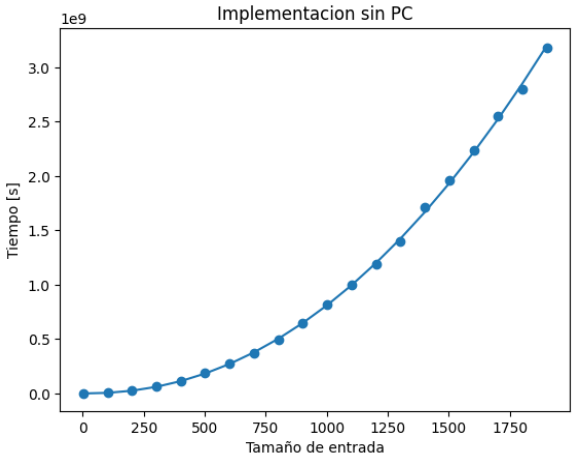

Fig 3: Ajuste realizado a los tiempos de cómputo obtenidos para la implementación sin path compression. El valor obtenido para la constante fue de ~ 117

Una vez que verificamos las complejidades de cada implementación, realizamos un gráfico para comparar los tiempos entre cada una de ellas. Esperamos que la implementación en $O(N^2)$ arroja tiempos más rápidos que las otras, y que la implementación de path compresión sea levemente mejor que aquella que solo utiliza union by rank.

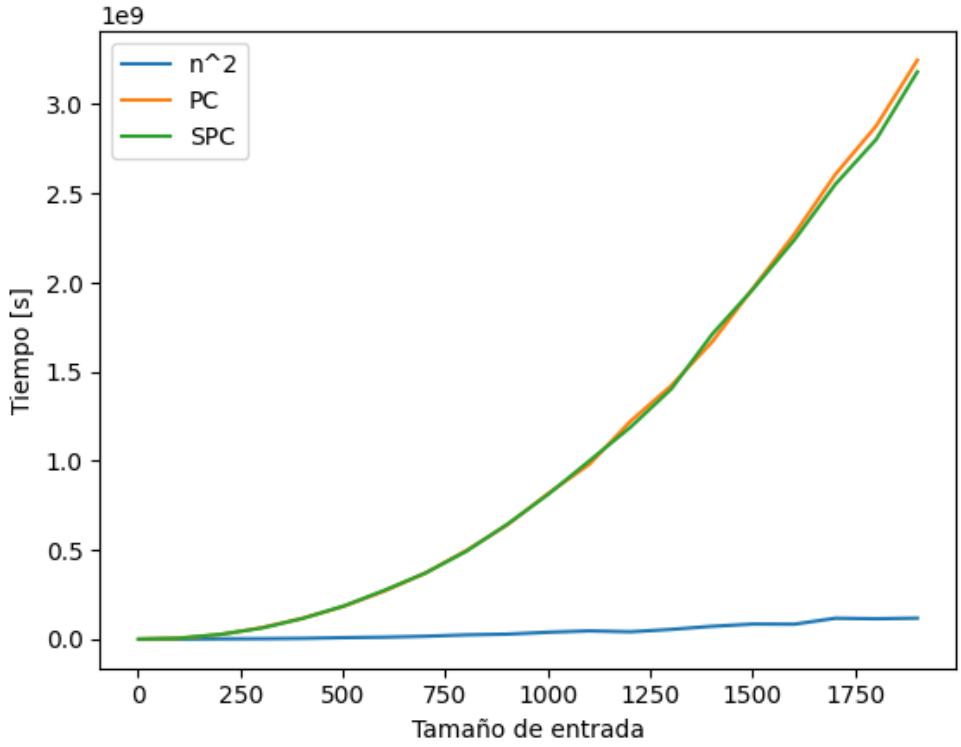

Fig 5: Comparación entre los tiempos de cómputo de las tres implementaciones realizadas. Se observa una gran diferencia entre la implementación elegida y las otras, que entre sí resultan prácticamente idénticas.

En los gráficos no se observa diferencia entre la implementación con path compresión y sin. Esto puede ser porque el tamaño de las instancias no llegan a números altos como para que la optimización de path compresión mejore el tiempo de cómputo.
Además, en ambas implementaciones primero se debe ordenar, de manera que la complejidad teórica esperada sigue siendo $O(M * log(N))$ para ambos.

Por otro lado, comprobamos que la implementación en $O(N^2)$ es significativamente mejor para el grafo que modela el problema, dado que con ella se obtuvieron tiempos sustancialmente menores.

Especificación de la computadora utilizada para correr los tests:

```
-memory: 12GiB System memory
-processor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
```