Universidad de Concepción
Departamento de Ing. Informática y Cs. de la Computación

**TAREA 3: Análisis de Algoritmos**
*Profes. Cecilia Hernández*

### Algoritmos greedy y amortizados

La fecha de entrega es el Lunes 24 de Junio a las 23:59 horas.

1. En esta tarea se implementará y analizará un algoritmo greedy con una estructura de datos que permite análisis amortizado. El problema consiste en encontrar un árbol de cobertura mínima usando el algoritmo Kruskal.

   Considere un grafo no dirigido, conectado y con pesos $G = (V, E, w)$, el cual tiene $|V|$ vértices, $|E|$ aristas, y $w(u, v)$ es el peso en cada arista $(u, v) \in E$. El peso se define por la función $w : E \to \mathbb{N}$. El problema general considera la función para el peso definida con recorrido en los reales, pero en este proyecto se considerará en los naturales.

   Un árbol de cobertura mínima de $G$ se define como un árbol $T \subseteq E$ tal que:

   - $T$ conecta todos los vértices
   - $w(T) = \sum_{(u,v)\in T} w(u, v)$ es mínimo

   Un árbol de cobertura mínima tiene las siguientes propiedades:

   - Tiene $|V| - 1$ aristas
   - No tiene ciclos
   - Puede no ser único

2. Algoritmo de Kruskal

   El algoritmo de Kruskal es un algoritmo greedy, consiste en los siguientes pasos y se puede implementar usando la estructura de datos Union-Find.

   - Prepocesar las aristas de $G$ de menor a mayor peso.
   - Inicialmente cada vértice en $G$ se define como un componente aislado, es decir un conjunto compuesto sólo de un vértice.

- Repetidamente toma la siguiente arista de menor peso y busca si los vértices en la arista pertenecen al mismo componente. Si no es así selecciona la arista para formar $T$ y une los componentes.

Algoritmo 1 proporciona el pseudocódigo usando la estructura de datos Union-Find.

**Algorithm 1** MST-Kruskal(G)

```
A ← ∅
ES: aristas ordenadas de menor a mayor peso
for v ∈ V do
  MakeSet(v)
end for
for (u, v) ∈ ES do
  if Find(u) ≠ Find(v) then
    A ← A ∪ {(u, v)}
    Union(u,v)
  end if
end for
return A
```

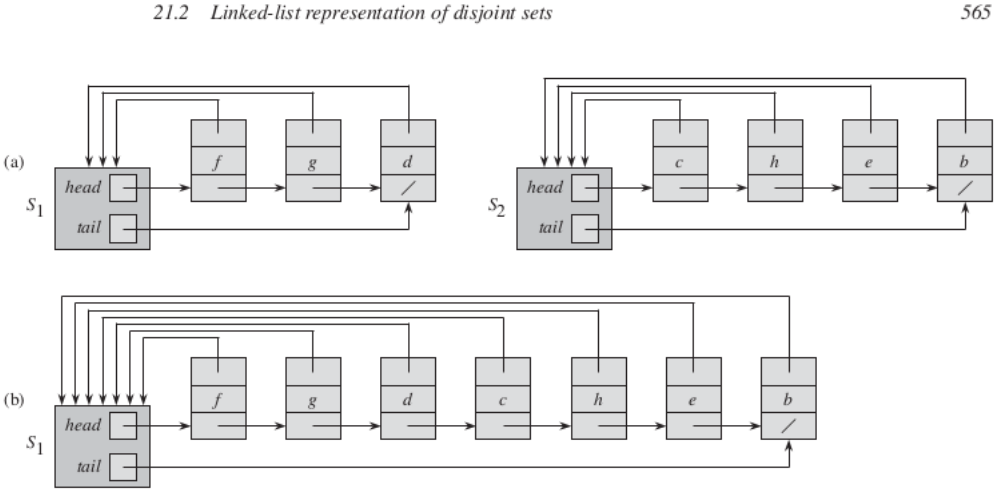
3. Estructura Union-Find

   Esta estructura maintiene una colección de conjuntos disjuntos dinámicamente $\mathbb{C} = \{S_1, ..., S_k\}$. Cada conjunto tiene un elemento denominado representante y la estructura soporta las siguientes operaciones:

   - MakeSet(x) crea un conjunto $S_x = \{x\}$ y lo agrega a la colección $\mathbb{C}$, y define $x$ como el representante de $S_x$.
   - Union(x,y): Si $x \in S_x$, $y \in S_y$, entonces $\mathbb{C} \leftarrow \mathbb{C} - S_x - S_y \cup \{S_x \cup S_y\}$. El representante del nuevo conjunto es cualquier miembro de $S_x \cup S_y$. Esto es, los conjuntos $S_x$ y $S_y$ son sustituidos por su unión.
   - Find-Set(x): retorna el representante del conjunto que contiene $x$.

   Existen diversas implementaciones de Union-Find unas mas eficientes que otras. En este proyecto considere dos alternativas, una basada en listas enlazadas y otra basada en conjunto de árboles. La Figura 1 muestra un ejemplo de Union usando listas enlazadas.

Figura 1: Union-Find con listas enlazadas




4. Implementación de Union-Find usando listas enlazadas

   - MakeSet(x): Crear una lista con solo un nodo conteniendo elemento $x$, puntero next a NULL y un puntero apuntando al representante. El representante se puede definir como otro objeto que contiene un puntero head apuntando al principio de la lista y un puntero tail apuntando al final.
   - Union(x,y): Hacer el merge de las dos listas, elegir el representante, puede ser cualquiera de los dos conjuntos. Además requiere actualizar los punteros a representantes de la lista que se une a la otra.
   - Find(x): Retornar el representante de la lista donde pertenece x.

   Se pide garantizar complejidad $O(1)$ para la operación Find(x), de manera que puede usar una estructura de datos adicional para ello.

   Considere esta implementación para una secuencia $m$ de operaciones consistiendo en $n$ $MakeSets$, uno para $n$ elementos seguida de $n-1$ operaciones de $Union$, esto es para $m = 2n-1$ operaciones en total. Analice cuál es el peor caso para las $m$ operaciones y cual es el costo amortizado para cada operación.

5. Implementación de Union-Find usando listas enlazadas con heurística

   Para este caso considere que la lista de menor largo siempre es la lista que se agrega al final de la lista mas larga al hacer Union. Agregue este cambio a su implementación y haga el análisis amortizado correspondiente.

6. Implementación de Union-Find usando bosque de árboles

   En esta implementación se simplifica la operación *Union(x,y)* en lugar de la operación *Find(x)*. En este caso, inicialmente cada elemento es un un árbol, donde sólo hay un nodo, y el representante del conjunto es el mismo nodo. Cada nodo en el árbol tiene un puntero al nodo padre, y el representante de un conjunto es la raíz en el árbol. Para este caso se define las operaciones Union y Find como siguen:

   - Find(x): Desde $x$ recorrer el árbol por sus padres hasta llegar a la raíz, la cual es el representante.
   - Union(x,y): Se hace apuntar la raíz de $x$ a la raíz de $y$ o viceversa.

   Dado que esta implementación es equivalente a la de listas enlazadas se usan dos heurísticas que permiten mejorar los tiempos de Union y Find.

   La primera heurística consiste en Union usando ranks. Para ello se define una operación auxiliar denominada *rank(x)*, la cual corresponde al máximo número de saltos desde una hoja en el árbol hasta el nodo $x$.

   La idea es usar la función rank de manera parecida al tamaño de la lista en la implementación en base a listas enlazadas para ser usada en la operación *Union(x,y)*. En este caso en la operación Union, la raíz con menor rank se hace apuntar a la raíz con mayor rank. Los ranks de los nodos no se modifican en Union a menos que los ranks sean iguales, en ese caso uno de las raíces se elije como padre y se incrementa su rank correspondiente.

   La segunda heurística se llama compresión de ruta y consiste en hacer que el nodo $x$ apunte directamente a su raíz una vez que la encuentra en una operación *Find(x)*.

   Esta implementación requiere dos arreglos, uno para proporcionar los padres de los nodos y otro para almacenar los ranks de los nodos. Los pseudocódigos para las operaciones de MakeSet, Find y Union se presentan a continuación:

```
MakeSet(x)
  p[x] = x
  rank[x] = 0

Union(x,y)
  Link(Find(x),Find(y)

Link(x,y)
  if rank[x] > rank[y]
     p[y] = x
  else
     p[x] = y
     if rank[x] = rank[y]
        rank[y] = rank[x] + 1

Find(x)
   if x != p[x]
      p[x] = Find(p[x])
   return p[x]
```

7. Para la implementación, puede elegir entre C, C++, Java y Python.

8. Contenidos mínimos del informe:

   *a*) Breve descripción de las estructuras de datos implementadas.

   *b*) Pseudocódigo de todas las operaciones implementadas.

   *c*) Análisis de complejidad y amortizado (excepto para compresión de rutas).

   *d*) Demuestre que incluir aristas de menor a mayor peso es una opción greedy que permite encontrar un MST mínimo.

   *e*) Evaluación experimental considerando diversos grafos de entrada

   *f*) Discusión de los resultados obtenidos.

La entrega consiste en un informe de no más de 5 páginas y el código fuente documentado.

Observaciones

- Tanto el algoritmo de Kruskal como la estructura Union-Find están descritos en el Cormen.