Introducción

En este documento se presenta la explicación de la practica opcional de MAR1, apartado 8, realizada por Alejandro Barrachina Argudo.

1. Algoritmo de Prim

1.1. Estructura de archivos

La carpeta está estructurada de la siguiente manera:

- **bin:** Carpeta que contiene el archivo final de compilación.
- **graph:** Carpeta que contiene los archivos de la biblioteca de grafos y casos de prueba:
  - **graph.h y graph.cpp:** archivos con el código de implementación de grafos mediante listas de adyacencia y el propio algoritmo de Prim.
  - **dummygraphs.h:** cabecera que introduce los grafos al programa.
  - **test:** carpeta que contiene un generador de grafos aleatorio y los casos de prueba generados
- **obj:** carpeta para la compilación de la librería de grafos.
- **main.cpp**: archivo fuente que corre los casos de prueba.
- **makefile**: archivo make para facilitar la compilación de la práctica.

La implementación del grafo usada es una implementación similar a la vista en clase utilizando un vector de aristas para cada nodo, sustituyendo en esta lista el identificador del nodo por un par identificador-peso. Esto nos permitirá encontrar el camino menos costoso entre todos los nodos del grafo.

Los gráficos de las pruebas se pueden ver en la figura 4.4 (Distribución de los tiempos observados en las distintas muestras de Decrecer clave para distintas cargas de trabajo.). Estas pruebas incluyen la carga de un grafo desde fichero con todo lo que ello conlleva, pero no la generación del grafo como tal.

Los tiempos observados se pueden asimilar a las de una función $(a + v)log(v)$, siendo a las aristas y v los vértices, que es lo que se pedía en el enunciado. A mayor numero de aristas en el grafo, mayor será el coste.

2. Montículo de Williams con Decrecer clave

2.1. Estructura de archivos

Dentro de la librería *graph* encontramos esta librería La carpeta está estructurada de la siguiente manera:

- **bin:** Carpeta que contiene el archivo final de compilación.
- **williams-heap:** Carpeta que contiene los archivos de la biblioteca de montículos y casos de prueba:
  - **williams-heap.h y williams-heap.cpp:** archivos con el código de implementación de montículos mediante listas de adyacencia y el propio algoritmo de Prim.
  - **dummyheaps.h:** cabecera que introduce los montículos al programa.
  - **test:** carpeta que contiene un generador de montículos aleatorio y los casos de prueba generados
- **obj:** carpeta para la compilación de la librería de montículos.
- **main.cpp**: archivo fuente que corre los casos de prueba.
- **makefile**: archivo make para facilitar la compilación de la práctica.

La implementación del montículo usada es la vista en los apuntes de clase, traduciendo los métodos de Daphny a C++.

Los gráficos de las pruebas se pueden ver en la figura 4.3 (Gráfica de tiempo de Decrecer clave para varios tamaños de montículo.). Estas pruebas incluyen la carga de un grafo desde fichero con todo lo que ello conlleva, pero no la generación del grafo como tal.

Los tiempos observados se pueden asimilar a las de una función $log(n)$, siendo n el número de elementos del montículo, que es lo que se pedía en el enunciado. Podemos ver picos y valles en las gráficas dado que los grafos de un mismo caso pueden tener un número muy distinto de aristas dada la naturaleza aleatoria de su generación.

3. Ejecución

Distinguiremos **carpeta raíz** y **ejecutable** según cada apartado:

- Para el algoritmo de prim:
  - **Carpeta raíz:** src/prim-algo
  - **Ejecutable:** Prim-algo
- Para el montículo de Williams:
  - **Carpeta raíz:** src/graph/williams-heap
  - **Ejecutable:** Williams-decrease

Si queremos probar un pequeño (muy pequeño) caso de prueba, podemos ejecutar el siguiente comando sobre la carpeta raíz:

```
make clean all
prim-algorithm/bin/${ejecutable}.out

```

Si queremos ejecutar una prueba concreta, sobre la carpeta anteriormente mencionada ejecutaremos:

```
make clean prueba
./bin/${ejecutable}-prueba.out RUTA_AL_ARCHIVO_DE_PRUEBA

```

En ambos casos podemos añadir al final del comando make `CFLAGS:=$(CFLAGS)"-D __DISPLAY"` para mostrar el resultado de la operación.

Si queremos ejecutar la batería de pruebas proporcionadas junto a esta entrega solo hay que ir a la carpeta anteriormente ejecutada y lanzar `./runner.sh`.

4. Figuras

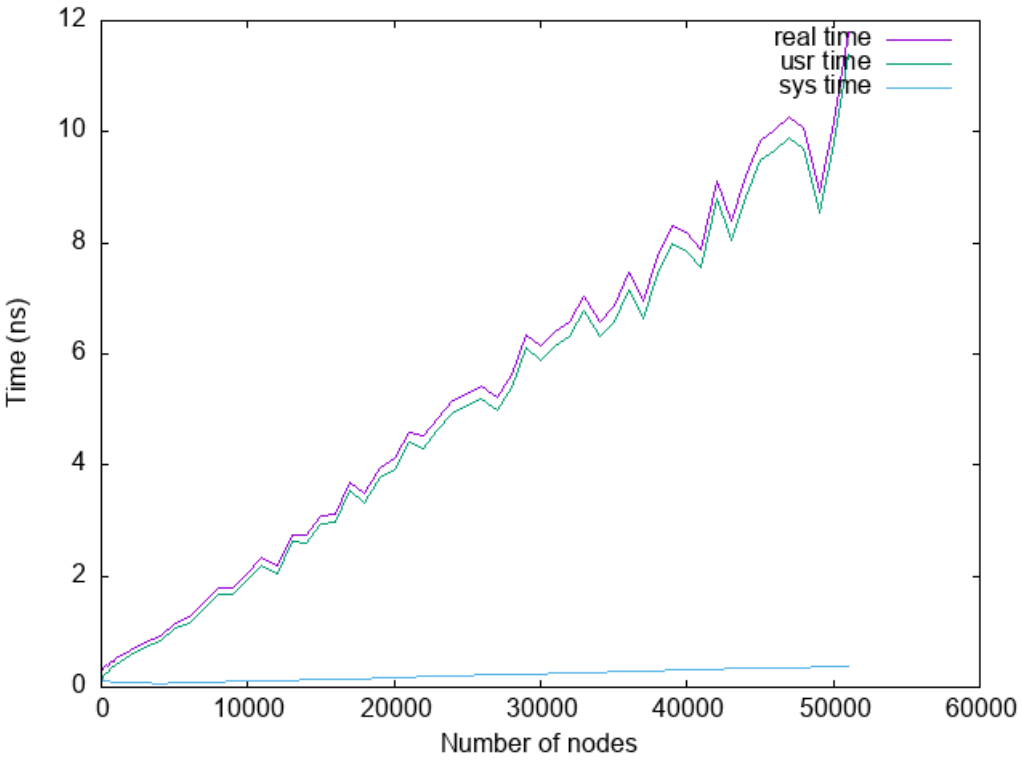

Figura 4.1: Gráfica de tiempo del algoritmo de Prim para distintas cargas de trabajo.

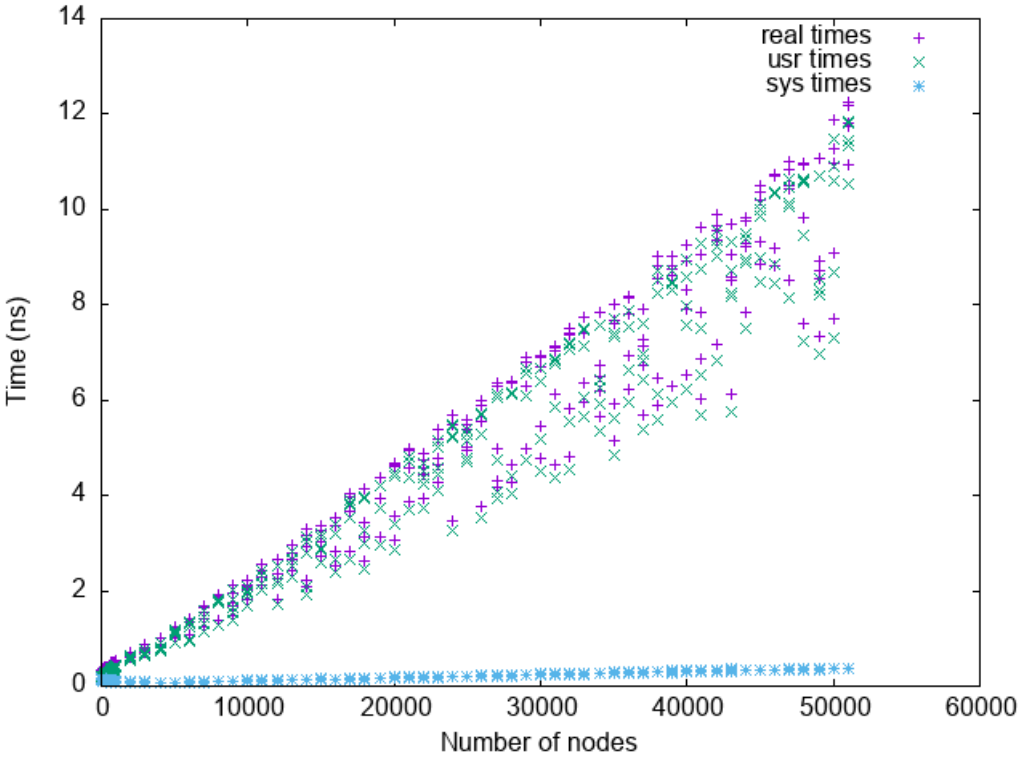

Figura 4.2: Distribución de los tiempos observados en las distintas muestras del algoritmo de Prim para distintas cargas de trabajo.

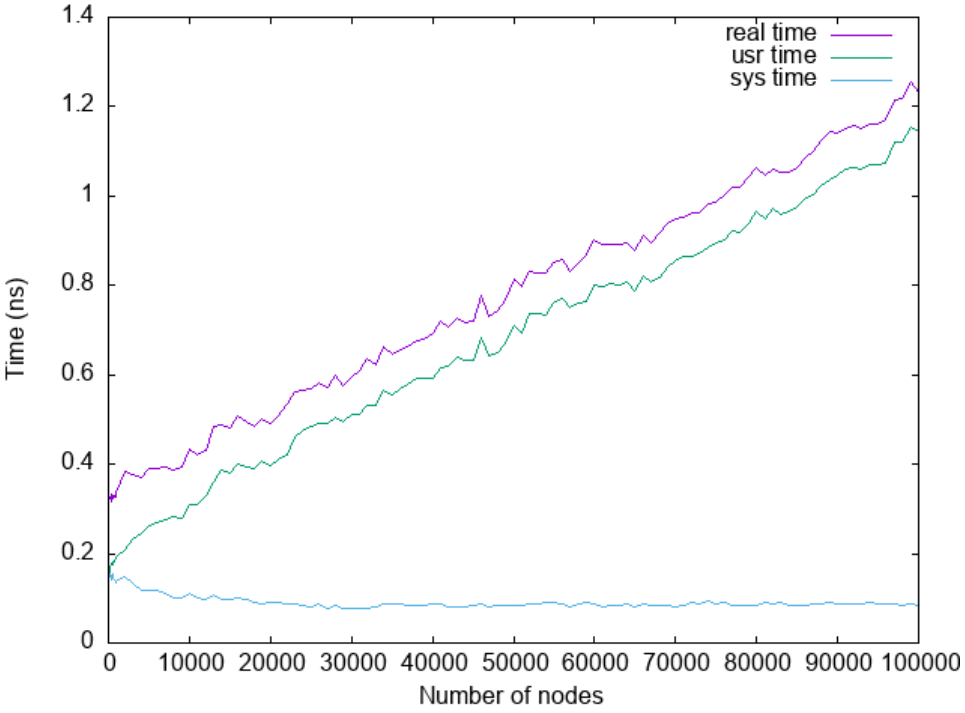

Figura 4.3: Gráfica de tiempo de Decrecer clave para varios tamaños de montículo.

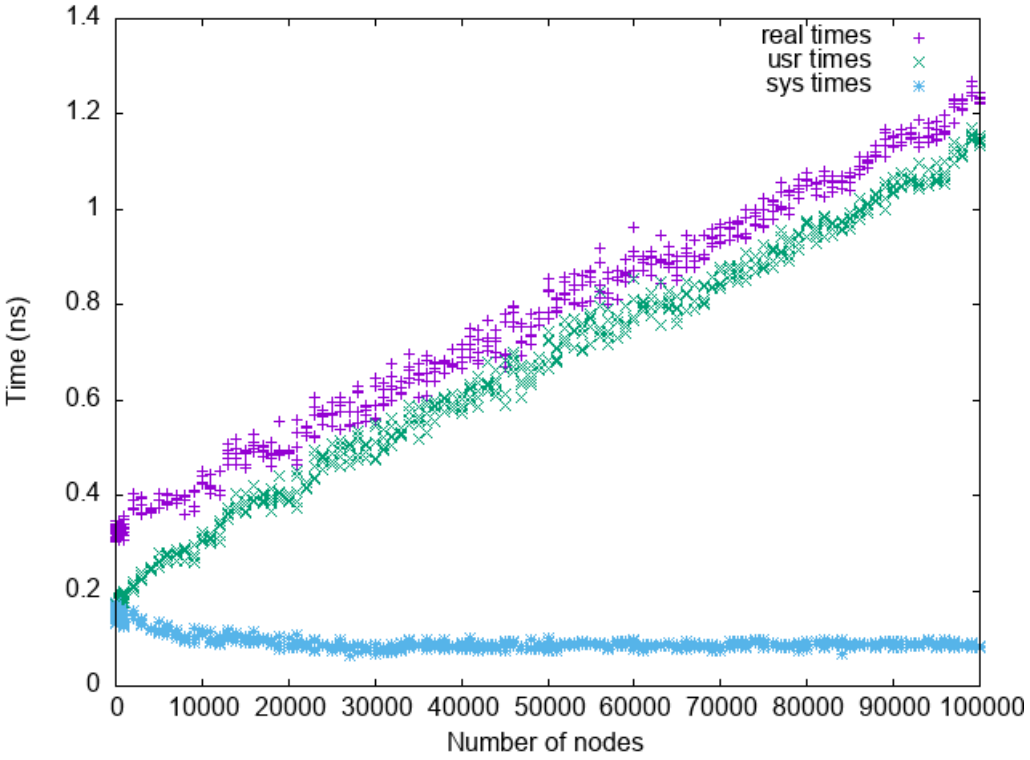

Figura 4.4: Distribución de los tiempos observados en las distintas muestras de Decrecer clave para distintas cargas de trabajo.