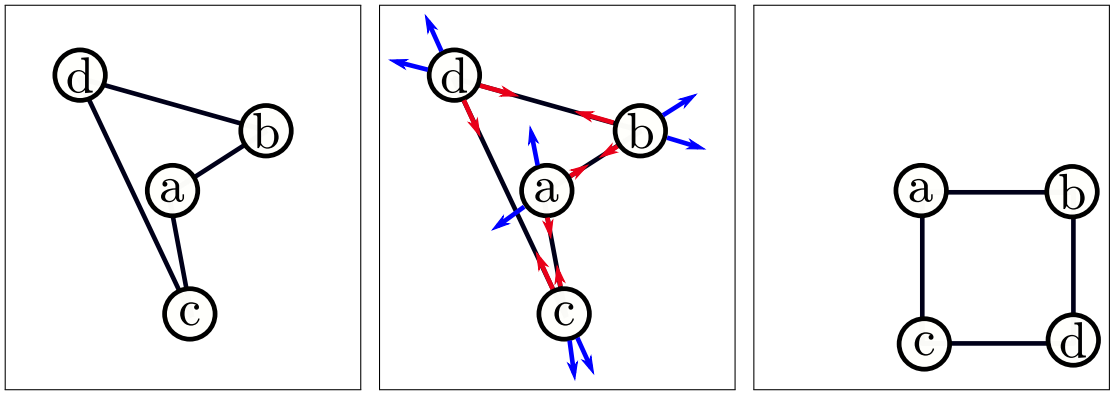
Apunte de Clases 6
Trabajo Final

1. Condiciones generales del Trabajo Práctico

- Entregar y aprobar el trabajo práctico de laboratorio es condición necesaria para la aprobación de la materia, y debe hacerse antes de rendir el examen final. Una vez establecidas las fechas de las mesas de examen, se publicará el día límite en el que el trabajo puede ser entregado, según la mesa en que se rinda.
- El trabajo solo puede estar Aprobado o Desaprobado.
- El trabajo debe realizarse en grupos de dos personas.
- En caso de ser necesario, además de subir el trabajo a la plataforma de comunidades, se pactará una fecha para la defensa virtual o presencial.

2. Objetivo

Desarrollar un programa para la visualización de grafos. El mismo debe buscar una distribución de los vértices de forma pareja, es decir, que las aristas tengan un largo uniforme y reflejen simetría. Para llevar a cabo esto se pide implementar el algoritmo de Fruchterman-Reingold [1] que se basa en simular fuerzas de repulsión entre los vértices del grafo y fuerzas de atracción entre los vértices que se encuentran unidos por aristas. Una posible interpretación es pensar a las aristas como resortes y a la ejecución del algoritmo como la búsqueda del equilibrio del sistema mecánico que conforma el grafo. En la Figura 1 se muestra un ejemplo de aplicación del algoritmo de Fruchterman-Reingold.



(a) Grafo inicial.

(b) Fuerzas de atracción en rojo y de repulsión en azul.

(c) Grafo de salida.

Figura 1: Aplicación del algoritmo de Fruchterman-Reingold.

2.1. Gestor de paquetes

`pip`[1] es un manejador de paquetes que puede ser utilizado a través de su interfaz de línea de comando para instalar paquetes de software de python fácilmente. Su modo de uso más simple es `pip install <paquete>`. Se recomienda su uso para instalar los paquetes necesarios para la realización del trabajo práctico.

2.2. Formato obligatorio de parámetros de entrada

- El grafo debe ser leído desde un archivo a través de línea de comando. Ejemplo: `python practica6_esqueleto.py <archivo_contiene_grafo>`
- Los parámetros *iteraciones* y *refresh* deben ser opcionales e ingresados por línea de comando.
- Se debe implementar la opción de modo *verbose* (`-v`, `--verbose`) para proveer detalles útiles de lo que el programa está haciendo y sus resultados.
- Pueden agregarse los parámetros opcionales que se consideren útiles.

Para lograr esto puede ser de ayuda la librería *argparse*. Este módulo hace muy sencillo escribir una interfaz por línea de comando amigable con el usuario. Existen funciones para indicar qué parámetros son necesarios, y el módulo se encarga de parsearlos adecuadamente.[2] Se puede installar sencillamente a través del siguiente comando:

```
pip install argparse
```

En el archivo *practica6_esqueleto.py* se encuentra un ejemplo de cómo utilizar *argparse*.

3. Librería de graficado

Para el graficado del grafo se recomienda utilizar la herramienta *matplotlib*[3]. Esta puede ser instalada a través del gestor de paquetes *pip* con los siguientes comandos:

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

En la carpeta *plots* se encuentran dos ejemplos de gráficos con matplotlib, a fin de corroborar que la librería está correctamente instalada.

4. Otras librerías de Python útiles

Se listan a continuación algunas librerías que pueden ser útiles. Sus usos no son obligatorios.

- Entorno virtual: `virtualenv`.
- Debugger de Python: `pdb`. Modo de uso: colocar `import pdb; pdb.set_trace()` desde la línea del archivo fuente que se desee inspeccionar. Luego existen varios comandos útiles entre los que podemos comentar:
  - `l` (imprime código)
  - `n` (siguiente línea de código)
  - `s` (step)
  - `c` (continue)

[1] `https://pypi.org/project/pip/`
[2] Se puede encontrar un tutorial en el siguiente link: `https://docs.python.org/3/howto/argparse.html`
[3] `https://matplotlib.org/`

5. Primera parte

1. Proveer funciones para:
   - Leer grafos desde archivos (similar a las funciones vistas en la práctica 1).
   - Asignar coordenadas aleatorias (en el plano) a cada vértice de un grafo. En esta instancia es importante pensar cómo es conveniente representar dichas coordenadas.
   - Dibujar grafos en las coordenadas correspondientes. Pueden ser útiles los métodos `matplotlib.pyplot.scatter()` y `matplotlib.pyplot.plot()` para graficar los vértices y las aristas.
     Posteriormente, los métodos `matplotlib.pyplot.clf()` y `matplotlib.pyplot.pause()` pueden servir para hacer gráficos consecutivos y generar efecto de movimiento.
2. Implementar el algoritmo siguiendo el pseudo-código propuesto en Algoritmo 1.[4] Las ocurrencias de `accum` en dicho pseudo-código hacen referencia a las fuerzas de atracción y repulsión acumuladas. Antes de empezar a escribir código es conveniente pensar cómo se representarán dichas fuerzas. Las funciones `f_attraction(v0, v1)` y `f_repultion(v0, v1)` pueden ser definidas de diferentes formas. Por ahora, definirlas simplemente como la distancia euclidiana entre los vértices.

6. Segunda parte

Realizar una implementación básica del algoritmo de Fruchterman-Reingold más detallado en el Algoritmo 2.

En la ecuación 1 se muestra la estimación del valor k referente a la dispersión de los nodos del grafo. Si el área del layout es muy grande, entonces los nodos estarán muy dispersos. En cambio, si el área es muy chica entonces el grafo estará muy concentrado. c es una constante para modificar el esparcimiento.

$$k = c\sqrt{\frac{area}{\#nodes}} \tag{1}$$

En las ecuaciones 2 y 3 se muestran las funciones que calculan las fuerzas de atracción y repulsión respectivamente. Observar que si la distancia d se incrementa, la fuerza de atracción será mayor. Del mismo modo, cuando la distancia d es decrementada, la fuerza de respulsión será mayor.

$$f_{attraction}(d) = \frac{d^2}{k} \tag{2}$$

$$f_{repulsion}(d) = \frac{k^2}{d} \tag{3}$$

[4] En todos los casos el pseudo-código es ilustrativo sobre el comportamiento del programa. Es buena idea tomar decisiones propias sobre su implementación, utilizando buenas prácticas de programación vistas a lo largo de la carrera.

Algorithm 1 Idea del Algoritmo de Fruchterman-Reingold.

```
procedure LAYOUT
    N, E = G
    randomize_positions(N)
    for k = 1 to NumIterations do
        # Inicializar los acumuladores en cero
        initialize_accumulators()
        # Calcular fuerzas de atracción
        for each e ∈ E do
            f = f_attraction(e)
            accum[e.origin] += f
            accum[e.dst] -= f
        end for
        # Calcular fuerzas de repulsión
        for each n_i ∈ range(N) do
            for each n_j ∈ range(N) do
                if n_i ≠ n_j then
                    f = f_repultion(n_i, n_j)
                    accum[n_i] += f
                    accum[n_j] -= f
                end if
            end for
        end for
        # Actualizar posiciones. Aún no implementado.
    end for
end procedure
procedure INITIALIZE_ACCUMULATORS( )
    acum_x = {node: 0 for node in N }
    acum_y = {node: 0 for node in N }
end procedure
```

Algorithm 2 Implementación básica del algoritmo de Fruchterman-Reingold.

procedure LAYOUT(G)
  $N, E = G$
  randomize_positions(N)
  for k = 1 to NumIterations **do**
    step()
  end for
end procedure

procedure STEP( )
  initialize_accumulators()
  compute_attraction_forces()
  compute_repulsion_forces()
  update_positions()
end procedure

procedure COMPUTE_ATTRACTION_FORCES( )
  for each $n_i, n_j \in E$ **do**
    $distance = \sqrt{(pos\_x[n_i] - pos\_x[n_j])^2 + (pos\_y[n_i] - pos\_y[n_j])^2}$
    $mod\_fa = f_{attraction}(distance)$
    $f_x = \frac{mod\_fa\,(pos\_x[n_j] - pos\_x[n_i])}{distance}$
    $f_y = \frac{mod\_fa\,(pos\_y[n_j] - pos\_y[n_i])}{distance}$
    $accum\_x[n_i] += f_x$
    $accum\_y[n_i] += f_y$
    $accum\_x[n_j] -= f_x$
    $accum\_y[n_j] -= f_y$
  end for
end procedure

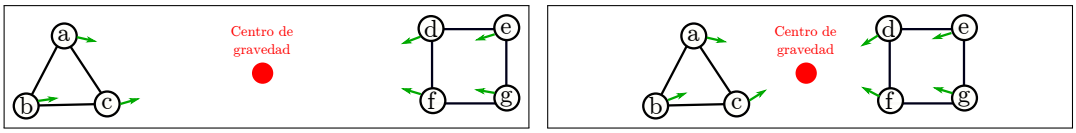
procedure COMPUTE_REPULSION_FORCES( )
  # Similar a compute_attraction_forces() pero aplicándose a todos los nodos
end procedure

procedure UPDATE_POSITIONS( )
  for each $node \in N$ **do**
    # cuidado con los bordes de la ventana
    $pos\_x[node] = pos\_x[node] + accum\_x[node]$
    $pos\_y[node] = pos\_y[node] + accum\_y[node]$
  end for
end procedure

7. Tercera parte

7.1. Fuerza de gravedad

La gravedad es una fuerza que se aplica de igual manera a todos los vértices del grafo (independientemente de dónde estén ubicados). La Figura 2 ilustra el efecto de la inclusión de la fuerza de gravedad en el algoritmo de Fruchterman-Reingold. El vector gravedad de cada vértice apunta al centro de la imagen de visualización. En la práctica, la gravedad debe tener un orden de magnitud menor al resto de las fuerzas resultantes del grafo para evitar la deformación del grafo. Por ejemplo, si las fuerzas resultantes del grafo tienen un orden de las centenas, la gravedad tendrá un orden de las decenas.



(a) (b)

Figura 2: Uso de gravedad en el algoritmo de Fruchterman-Reingold.

El Algoritmo 3 ilustra cómo la gravedad puede ser incorporada a la actual implementación básica del algoritmo de Fruchterman-Reingold.

Algorithm 3 Agregando gravedad al algoritmo de Fruchterman-Reingold.

```
procedure STEP( )
    initialize_accumulators()
    compute_attraction_forces()
    compute_repulsion_forces()
    compute_gravity_forces()
    update_positions()
end procedure
```

7.2. Temperatura

La temperatura del sistema permite que el mismo pueda converger rápidamente, sin contraerse y dilatarse repetidamente como se muestra en la Figura 3. Una mayor temperatura implica mayores variaciones en el sistema. Por el contrario, una menor temperatura, implica menores variaciones. Intuitivamente, en un comienzo el sistema requerirá variaciones significativas, y a medida que pasan las iteraciones, el grafo estará cada vez más ordenado y por lo tanto las variaciones requeridas serán menores. Esto se logrará con un descenso de la temperatura. El Algoritmo 4 muestra cómo el algoritmo de Fruchterman-Reingold se ve modificado al considerar el parámetro de temperatura. La temperatura del sistema se verá reducida de manera constante luego de cada iteración.

7.3. Caso borde

Si dos vértices caen en el mismo lugar puede producirse una división por cero. Esto se puede resolver haciendo que cuando dos nodos se encuentran a una distancia menor a ϵ (por ejemplo, $\epsilon = 0{,}05$), entonces se les aplica una fuerza de repulsión constante aleatoria. Puede calcularse

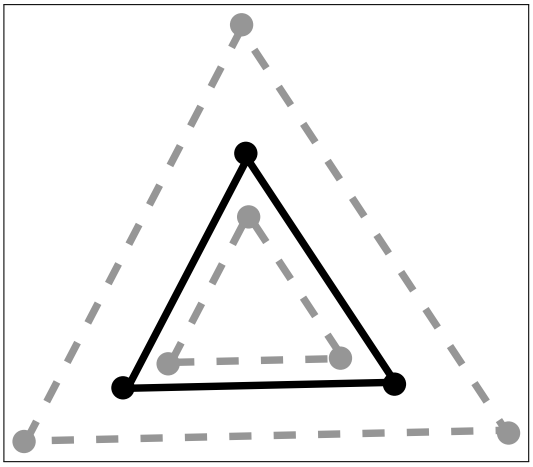


Figura 3: Oscilación del algoritmo de Fruchterman-Reingold por falta del uso del parámetro de temperatura.

aleatoriamente un vector dirección y se le aplica una pequeña fuerza en esa dirección al punto. El otro punto debe sufrir la misma fuerza de repulsión en sentido contrario.

Algorithm 4 Agregando temperatura al algoritmo de Fruchterman-Reingold.

procedure STEP( )
  initialize_temperature()
  initialize_accumulators()
  compute_attraction_forces()
  compute_repulsion_forces()
  compute_gravity_forces()
  update_positions()
  update_temperature()
end procedure

procedure INITIALIZE_TEMPERATURE( )
  $t = t_0$
end procedure

procedure UPDATE_TEMPERATURE( )
  $t = \mathrm{c}\, t$ ▷ c es una constante. Ejemplo: c = 0,95
end procedure

procedure UPDATE_POSITIONS( )
  for each $node \in N$ **do**
    $\vec{f} = (accum\_x[node], accum\_y[node])$
    if $\|\vec{f}\| > t$ **then**
      $\vec{f} = \dfrac{\vec{f}}{\|\vec{f}\|}\, t$
      $(accum\_x[node], accum\_y[node]) = \vec{f}$
    end if
    # cuidado con los bordes de la ventana
    $pos\_x[node] = pos\_x[node] + accum\_x[node]$
    $pos\_y[node] = pos\_y[node] + accum\_y[node]$
  end for
end procedure

Referencias

[1] THOMAS M. J. FRUCHTERMAN, EDWARD M. REINGOLD. *Graph Drawing by Force–Directed Placement*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801-2987, U.S.A.