Ayudantía 12: Grafos

Objetivo Principal: Conocer cómo analizar e implementar algoritmos de grafos.
Autor: Kevin Pizarro Aguirre

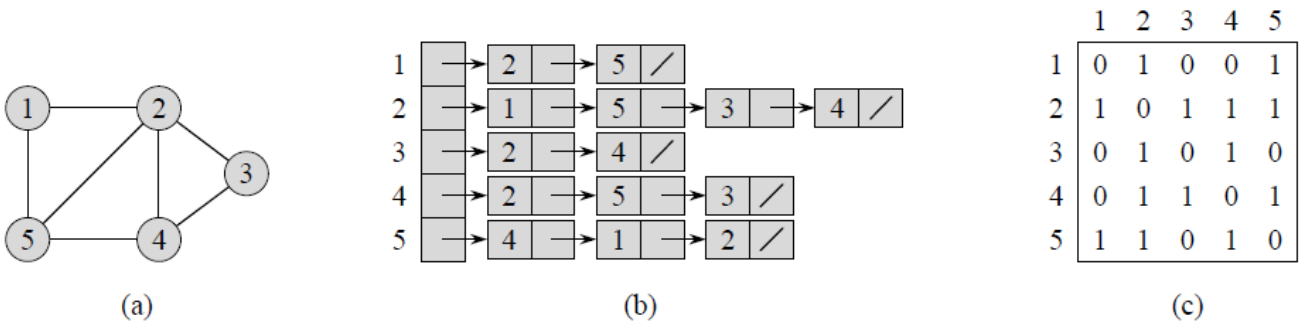
1. Grafos

Los grafos son una estructura de datos que son ampliamente usados en *computer science*, aplicaciones de navegación (Google Maps), redes sociales como Facebook, entre otras por su eficiencia a la hora de utilizar algoritmos que busquen el camino de menor costo o potenciales redes de amigos como sugerencia. Están conformados por vértices (vertex) y enlaces (edge), de esta forma podemos conectar los vértices a través de los enlaces y entablar relaciones con o sin pesos asociados.

1.1. Clasificaciones y representaciones

Una de las clasificaciones que se le pueden dar a los grafos es si son o no dirigidos. Si los grafos son **no dirigidos** entonces son **bidireccionales**, mientras que para grafos **dirigidos** son **unidireccionales**.

Por otro lado, podemos representar los grafos a través de sus vértices y enlaces como es usual, pero también se puede representar a través de la *lista de adyacencia* y la *matriz de adyacencia*. Todas son equivalentes, pero no todas son almacenadas de la misma forma en la memoria. Para la lista de adyacencia se tienen tantas listas simplemente enlazadas como vértices se tengan, los elementos enlazados serán con los que se tenga efectivamente el enlace. Para la matriz de adyacencia se determina de tamaño $|V|^2$, siendo $|V|$ la cantidad de vértices, de esta manera el par (i,j) nos dirá si entre i y j existe o no un enlace.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Figura 1: Grafo no dirigido: (a) representación usual (b) representación de lista de adyacencia y (c) representación de matriz de adyacencia.

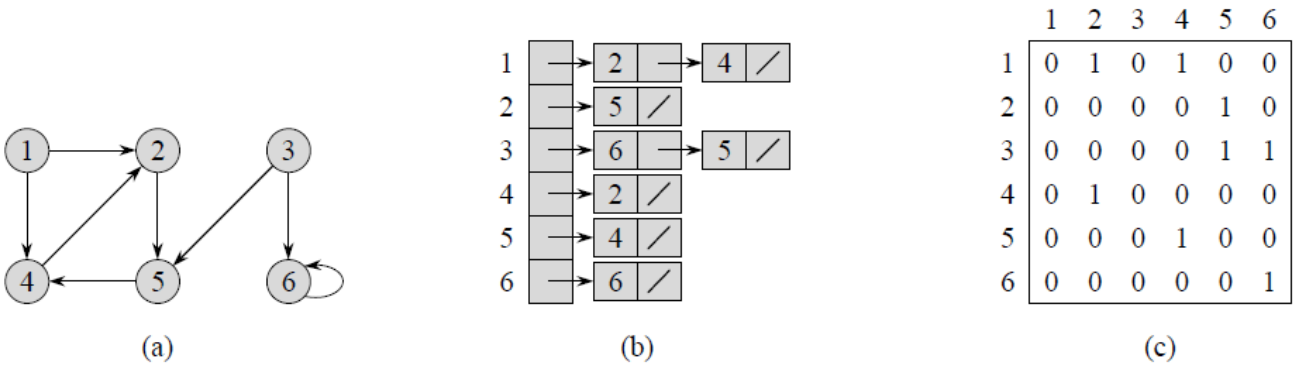


Figura 2: Grafo dirigido: (a) representación usual (b) representación de lista de adyacencia y (c) representación de matriz de adyacencia.

Notar que la cantidad de enlaces $|E|$ es equivalente a la cantidad de elementos distintos de 0 en la matriz de adyacencia o la suma de todos los largos de las listas de adyacencia. También, al analizar las representaciones y potenciales operaciones se encuentran las siguientes ventajas y desventajas de usar una u otra.

1. La matriz de adyacencia es fácil de implementar.
2. Las operaciones de eliminar y consultar el peso de un enlace, en la matriz de adyacencia, toma tiempo constante.
3. La matriz de adyacencia siempre consume un espacio de $|V|^2$, hayan o no enlaces de todos con todos.
4. Para una matriz de adyacencia añadir un nuevo vértice toma $\Theta(|V|^2)$.
5. Para la lista de adyacencia se tendrá un espacio de $|V| + |E|$, en el peor caso se iguala a la matriz de adyacencia.
6. Añadir un nuevo vértice a la lista de adyacencia es más simple.
7. Las consultas en la lista de adyacencia serán realizadas en tiempo $\Theta(|V|)$.

2. Algoritmos de grafos

2.1. Breadth-first search (BFS)

Para el primer algoritmo tenemos a Breadth-first search, el cual asume una implementación del grafo mediante listas de adyacencia. Además de la memoria para el grafo en sí, se utiliza memoria para almacenar el color de los vértices, su predecesor, la distancia fuente (source) - demás vértices y una fila para saber qué vértice visitar en la siguiente iteración. Se dice que es Breadth-first search porque por cada vértice se visitará 1 vez a todos los vértices adyacentes a él.

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

Figura 3: Pseudocódigo para el algoritmo BFS.

Al analizar el algoritmo se llega que su complejidad algorítmica está dada por $O(|V| + |E|)$.

2.2. Dijkstra

Por otro lado, se tiene uno de los grandes algoritmos que encuentran el camino más corto para una fuente, hablamos del algoritmo de Dijkstra. Dijkstra funciona bajo las condiciones de que se trata de un grafo dirigido y pesos no negativos. El algoritmo visitará los vértices y sus adyacencias, comparando los caminos trazados y relajando al camino más corto.

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Figura 4: Pseudocódigo para el algoritmo de Dijkstra.