# Estructuras de Datos

DPTO INFORMATICA - U. CORDOBA

Grafos II
(caminos y recorridos)

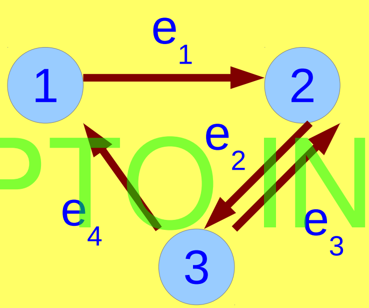
# Contenidos

- Introducción.
- Especificación.
- Implementaciones.
- Búsqueda de caminos.
- Recorridos.

DPTO. INFORMATICA - U. CORDOBA

# Buscando caminos

- Usando la matriz de adyacencia.
  - ¿Están dos nodos conectados? (fuerza bruta)




$$A=\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 1 & 0\end{bmatrix}$$

$$A^2=AxA=\begin{bmatrix}0 & 0 & 1\\ 1 & 1 & 0\\ 0 & 1 & 1\end{bmatrix}$$

$$A^3=A^2xA=\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$$B^3=\begin{bmatrix}1 & 2 & 1\\ 1 & 2 & 2\\ 2 & 3 & 2\end{bmatrix}$$

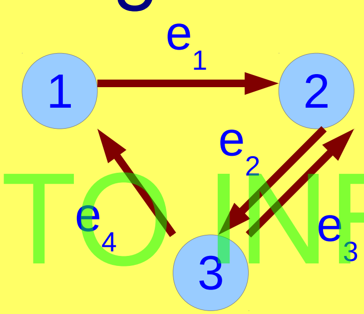
$$P=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

- $\mathbf{a^r_{ij}}$= número de caminos de longitud **r** que conectan el nodo **i** con el nodo **j**.
- $\mathbf{B^n=A+A^2+A^3+...A^n}$ = número de caminos de longitud <=N que conectan dos nodos.
- **P**, con $\mathbf{p_{ij}=1}$ si $\mathbf{b^n_{ij}>0}$, *matriz de conectividad* o *matriz de caminos*.
- Complejidad para obtener **P** en un grafo de N nodos: $A^r xA= O(N^3)$ x N → $\mathbf{O(N^4)}$

DPTO INFORMATICA - U. CORDOBA

# Buscando caminos

- Usando la matriz de adyacencia.
  - ¿Están dos nodos conectados? Alg. de Warshall.




$$A=\begin{bmatrix}0&1&0\\0&0&1\\1&1&0\end{bmatrix}$$

$$\underset{P^0}{\begin{bmatrix}0&1&0\\0&0&1\\1&1&0\end{bmatrix}}\quad \underset{P^1}{\begin{bmatrix}0&1&0\\0&0&1\\1&1&0\end{bmatrix}}\quad \underset{P^2}{\begin{bmatrix}0&1&\mathbf{1}\\0&0&1\\1&1&\mathbf{1}\end{bmatrix}}\quad \underset{P^3}{\begin{bmatrix}\mathbf{1}&1&1\\\mathbf{1}&\mathbf{1}&1\\1&1&\mathbf{1}\end{bmatrix}}$$

DPTO INFORMATICA - U. CORDOBA

**Claves:**

- Generar $A=P^0\rightarrow P^1\rightarrow\ldots P^n=P$
- $p^k_{ij}=1$ si existe camino directo entre **i,j** o un camino con nodos intermedios en el conjunto **$\{v_1,v_2,\ldots,v_k\}$**
- $p^1_{ij}=1$ si $a_{ij}=1$ o $a_{i1}=1$ y $a_{1j}=1$
- $P^1$ se calcula aplicando la regla:

```
Si p⁰ij=1 entonces
   P¹ij=1
Sino
   p¹ij=p⁰i1·p⁰1j
Fin-Si.
```

```
Warshall(A,N): //Time Analysis O(N³)
Begin
P ← A
For k ← 1 to N Do
  For i ← 1 to N Do
    For j ← 1 to N Do
      If P[i,j]=0 Then
        P[i,j]←P[i,k]*P[k,j]
      End-If
    End-For
  End-For
End-For
End.
```

# Buscando caminos mínimos

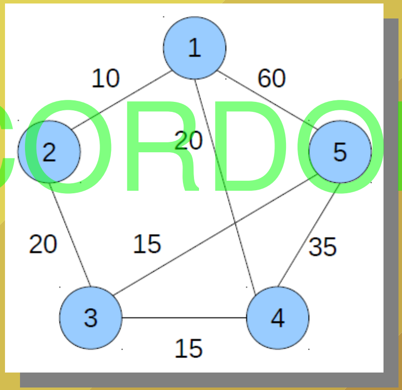
- Alg. de Dijkstra.

**Claves:**

- Busca el camino mínimo desde un nodo al resto.
- Los pesos deben ser >=0.
- Es un algoritmo voraz: en cada iteración busca la mejor solución local.
- Usa tres vectores:
  - **S**: que indica los nodos visitados.
  - **D**: distancia mínima al nodo origen.
  - **P**: predecesor en el camino al nodo origen.
- Inicialmente S sólo marca el nodo inicial.
- En cada iteración:
  - Es seleccionado el nodo j no visitado (S[i]=0) cuya distancia D[j] al origen sea la menor.
  - Actualizar las distancias D y predecesores P en función del nuevo nodo j introducido en S.
- Terminar cuando todos los nodos estén en S.
- Complejidad: $O(N^2)$

DPTO INFORMATICA - U. CORDOBA

Ejemplo: Todos los caminos de desde el nodo 1.




| | Inic. | | | Iter1 | | | Iter2 | | | Iter3 | | | Iter4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S | D | P | S | D | P | S | D | P | S | D | P | S | D | P |
| 1 | **1** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 10 | 1 | **1** | 10 | 1 | 1 | 10 | 1 | 1 | 10 | 1 | 1 | 10 | 1 |
| 3 | 0 | ∞ | 0 | 0 | **30** | **2** | 0 | 30 | 2 | **1** | 30 | 2 | 1 | 30 | 2 |
| 4 | 0 | 20 | 1 | 0 | 20 | 1 | **1** | 20 | 1 | 1 | 20 | 1 | 1 | 20 | 1 |
| 5 | 0 | 60 | 1 | 0 | 60 | 1 | 0 | **55** | **4** | 0 | **45** | **3** | **1** | 45 | 3 |

¿Cómo obtengo el camino del nodo 1 al 5?

P[5]=3 -> P[3]=2 -> P[2] -> 1

# Buscando caminos mínimos

- Alg. de Dijkstra (algoritmo)

```
Dijkstra(IN: W[N,N], start; OUT: D[N],P[N])
LOCAL: S[N],i,j,x,minD
Begin:
   S[start] <- 1
   For i From 1  To N Do
      Dij] = W[start,i]
      Pij] = start
   End-For.
   For i From 1 To N-1 Do
      minD <- inf
      For j From 1 To N Do
         If S[j]=0 And D[J]<=minD Then
            minD <- D[j]
            x <- j
         End-If
      End-For
      S[x] <- 1
      For j From 1 To N Do
         If S[j]=0 And D[j] > D[x]+W[x,j] Then
            D[j]=D[x]+W[x,j]
            P[j]=x
         End-If
      End-For
   End-For
End.
```

Inicializar distancias y predecesores.

Buscar siguiente nodo a visitar.

Actualizar distancias y predecesores.

DPTO. INFORMATICA - U. CORDOBA

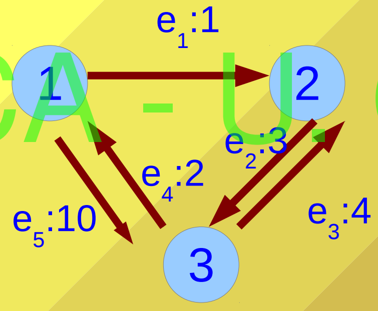
# Buscando caminos mínimos

- Alg. de Floyd.

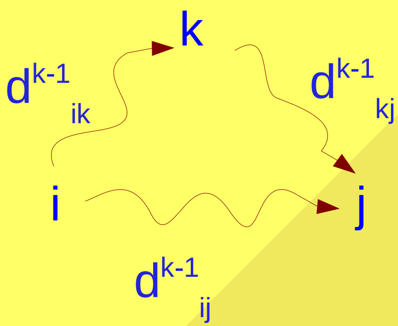
**Claves:**

- Camino mímino para cualquier par de nodos.
- Generar **$W=D^0 \rightarrow D^1 \rightarrow \ldots D^n$= D matriz distancia mínimas.**
- $\mathbf{d^k_{ij}}$=long. Camino más corto entre **i,j** con nodos intermedios sólo en el conjunto $\{v_1, v_2, \ldots, v_k\}$
- $\mathbf{D^1}$ se calcula a partir de $\mathbf{D^0}$ aplicando la regla: $d^1_{ij} \leftarrow \min\{d^0_{ij},\ d^0_{i1}+d^0_{1j}\}$
- Complejidad: **$O(N^3)$**

DPTO INFORMATICA - U. CORDOBA




$$W=\begin{bmatrix}0 & 1 & 10\\ \infty & 0 & 3\\ 2 & 4 & 0\end{bmatrix}$$




$$\underset{D^0}{\begin{bmatrix}0 & 1 & 10\\ \infty & 0 & 3\\ 2 & 4 & 0\end{bmatrix}}\ \underset{I^0}{\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}}\qquad \underset{D^1}{\begin{bmatrix}0 & 1 & 10\\ \infty & 0 & 3\\ 2 & \mathbf{3} & 0\end{bmatrix}}\ \underset{I^1}{\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & \mathbf{1} & 0\end{bmatrix}}$$

$$\underset{D^2}{\begin{bmatrix}0 & 1 & \mathbf{4}\\ \infty & 0 & 3\\ 2 & 3 & 0\end{bmatrix}}\ \underset{I^2}{\begin{bmatrix}0 & 0 & \mathbf{2}\\ 0 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}}\qquad \underset{D^3}{\begin{bmatrix}0 & 1 & 4\\ \mathbf{5} & 0 & 3\\ 2 & 3 & 0\end{bmatrix}}\ \underset{I^3}{\begin{bmatrix}0 & 0 & 2\\ \mathbf{3} & 0 & 0\\ 0 & 1 & 0\end{bmatrix}}$$

¿Cómo obtengo el camino entre dos nodos?

# Buscando caminos mínimos

- Alg. de Floyd.

```
Floyd(IN: W; OUT: D,I)
Begin
D ← W
P ← 0
For k From 1 To N Do
  For i From 1 To N Do
    For j From 1 To N Do
      If D[i,k]+D[k,j]<D[i,j] Then
        D[i,j] ← D[i,k]+D[k,j]
        I[i,j] ← K
      End-If
    End-For
  End-For
End-For
End.
```

DPTO. INFORMATICA - U. CORDOBA

# Recorridos

- Recorrido en profundidad.

```
DepthFirst(G,F)
Begin
  G.resetNodes()
  For each node v not visited Do
    G.scan(v, F)
  End-For
End.
```

**Claves**:

- Los nodos tienen un campo “visitado”.
- Partimos de un nodo s.
- Recorremos de forma recursiva cada nodo conectado a s no visitado.
- En el recorrido forma un árbol abarcador.
- Podemos tener más de un árbol=bosque.

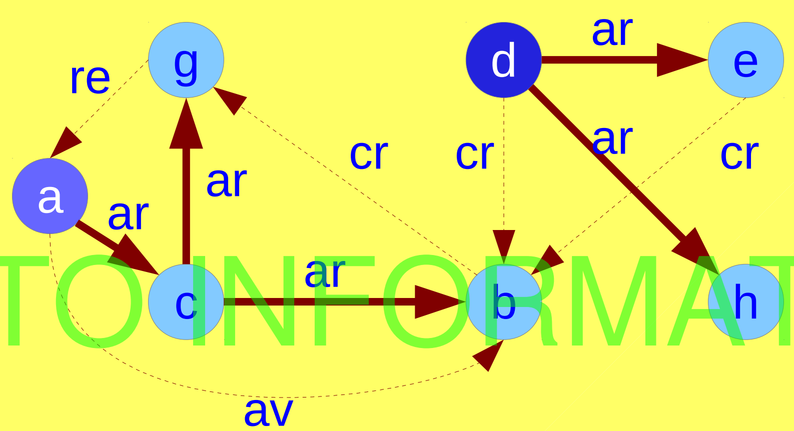
DPTO INFORMATICA - U. CORDOBA

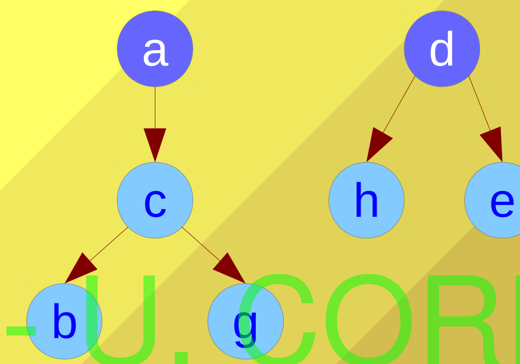
```
G::scan(v, F) //Versión iterativa
Begin
  Stack S
  S.insert(v)
  While not S.empty() Do
    v ← S.top()
    If v.isVisited() Then
      S.remove()
    Else
      v.setVisited()
      F(v)
      For each node u adjacent to v Do
        If not u is visited then
         S.insert(u)
        End-If
      End-For
    End-If
  End-While
End.
```

```
G::scan(v, F) //Versión recursiva
Begin
  mark v as visited.
  F(v)
  For each node u adjacent to v Do
    If not u is visited then
      scan(u, F)
    End-If
  End-For
End.
```

# Recorridos

- Recorrido en profundidad.




DPTO INFORMATICA - U. CORDOBA

Bosque abarcador en profundidad.

Nota importante: en el caso de que haya más de una elección posible, **no** hay un orden predefinido.

Orden de recorrido:
1:a 2:c 3:g 4:b 5:d 6:h 7:e

Los nodos n descendientes de n1 cumplen:
o(n1)<o(n)<=o(n1) + d(n1)

**Tipos de lados para grafos dirigidos:**
- Del árbol abarcador.
- (n1,n2) es de *avance* si: o(n1)<o(n2)<=o(n1)+d(n1)
- (n2,n1) es de *retroceso* si: o(n1)<o(n2)<=o(n1)+d(n1)
- (n1,n2) es *cruzado* si: o(n1)>o(n2)+d(n2).

**Tipos de lados para grafos no dirigidos:**
- Del árbol.
- De avance/retroceso.

# Recorridos

- Recorrido en amplitud.

**Claves**:

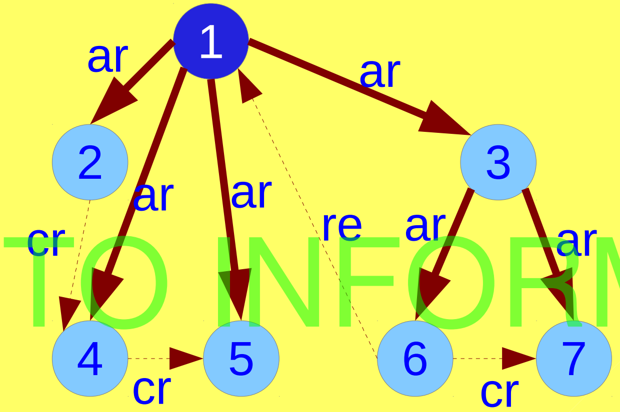
- Partimos de un nodo s.
- Recorremos el grafo aumentando en cada iteración en 1 la distancia de los nuevos nodos visitados.
- Se utiliza una cola FIFO para guiar el recorrido.
- El recorrido forma un árbol abarcador.
- Podemos tener más de un árbol=bosque.

```
BreadthFirst(G,F)
Begin
  G.resetNodes()
  s <- G.firstNode()
  F(s)
  s.setVisited()
  Queue.insert(s)
  While not Queue.isEmpty() Do
    s <- queue.front()
    queue.removeFront()
    For each n adjacent to s Do
      If not n.isVisited() Then
        F(n)
        n.setVisited()
        queue.insert(n)
      End-If
    End-For
  End-While
End.
```

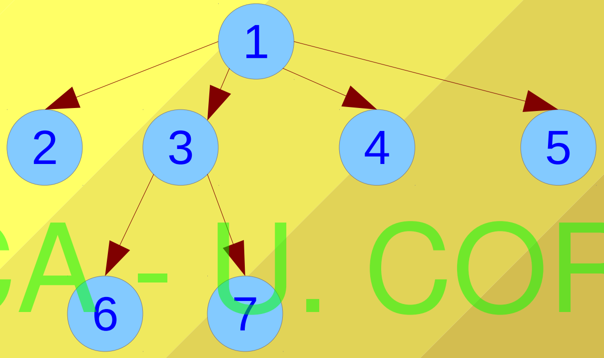
DPTO INFORMATICA - U. CORDOBA

# Recorridos

- Recorrido en amplitud.




Orden de recorrido:
1 2 3 4 5 6 7




Árbol abarcador en amplitud.

DPTO INFORMATICA - U. CORDOBA

**Tipos de lados para grafos dirigidos:**

- Del árbol abarcador.
- (n2,n1) es de retroceso si: $o(n1)<o(n2)<=o(n1)+d(n1)$
- (n1,n2) es cruzado si: $o(n1)>o(n2)+d(n2)$.

**Tipos de lados para grafos no dirigidos:**

- Del árbol.
- Cruzados.

# Recorridos

- Ordenación topológica.

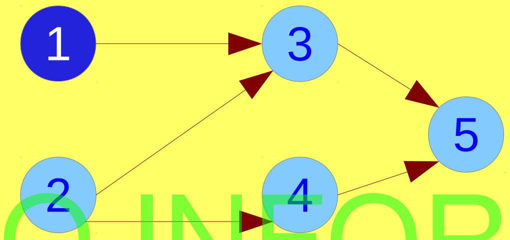
**Claves**:
- Se aplica a GDA.
- **Ejemplo**: grafo de proyecto (PERT):
  - Cada nodo es una tarea.
  - Un lado (n1,n2) indica que la tarea n1 es un **prerrequisito** para la tarea n2.
- La ordenación topológica es un recorrido en orden de los nodos respetando los prerrequisitos.
- Se implementa con una modificación del recorrido en profundidad, procesando el nodo después de la llamada recursiva.
- Complejidad $O(N^2)$

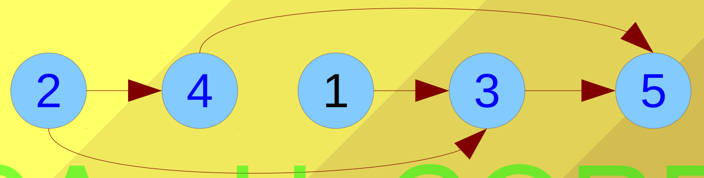
DPTO INFORMATICA - U. CORDOBA

```
TopologicalSorting(G,s,F)
Begin
  G.resetNodes()
  For each node s not visited Do
    G.scan(s, F)
  End-For
End.

G::Scan(s, F)
Begin
  mark s as visited.
  For each node w adjacent to s Do
    If not w.isVisited() then
      Scan(w, F)
    End-If
  End-For
  F(s)
End.

Functor::operator (n)
Begin
   list.insertFront(n)
End.
```

# Recorridos

- Ordenación topológica.




DPTO INFORMATICA - U. CORDOBA

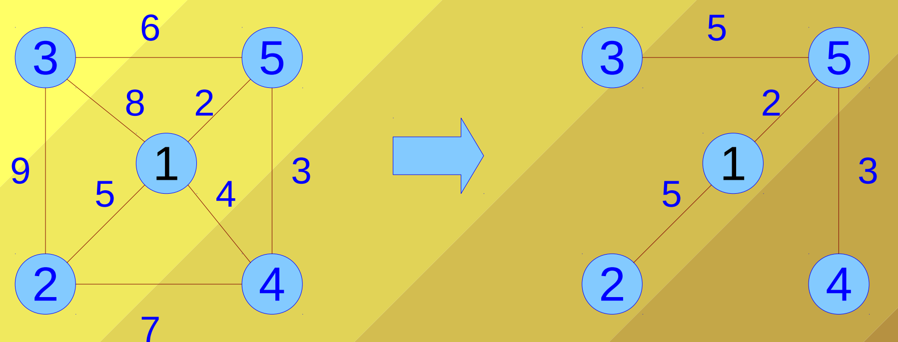
Orden de procesamiento:
5 3 1 4 2
Ordenación topológica:
2 4 1 3 5

# Árbol abarcador mínimo

- Conceptos:

  - Se aplica a **grafos conexos no dirigidos**.
  - **Problema**: encontrar el subgrafo conexo que enlaza todos los nodos con coste mínimo.
  - **Propiedad explotada**: Si un grafo G se puede dividir en en dos subgrafos U y G-U, el lado de coste mínimo que une estos subgrafos pertenece al AAM.
  - Un AAM con N nodos tendrá N-1 lados (no tiene ciclos).
  - Puede existir más de una solución.

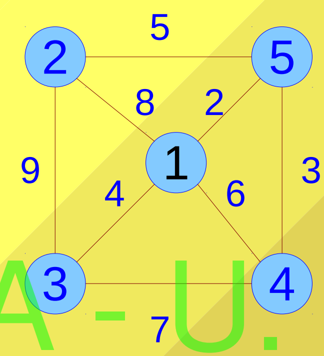
DPTO INFORMATICA - U. CORDOBA

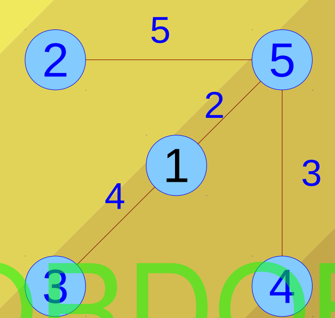


# Árbol abarcador mínimo

- Algoritmo de Prim:
  - **Inicializa** los conjuntos U:{1} y N-U: {2,...,N}
  - **Repetir** N-1 veces:
    - Aplicar propiedad AAM: **buscar** lado (i,j) con i∈U, j∈N-U con menor coste.
    - Introduce nodo j en U.
  - Para reducir la complejidad de la búsqueda del lado se usan dos vectores:
    - **V[i]=j:** nodo j∈U más cercano al nodo i∈N-U.
    - **C[i]:** coste del lado (i,V[i]).
  - Complejidad: $O(N^2)$

DPTO INFORMATICA - U. CORDOBA




| | U | V | C |
|---|---|---|---|
| 1 | 1 | - | ∞ |
| 2 | 0 | 1 | 8 |
| 3 | 0 | 1 | 4 |
| 4 | 0 | 1 | 6 |
| **5** | **0** | **1** | **2** |

| | U | V | C |
|---|---|---|---|
| 1 | 1 | - | ∞ |
| 2 | 0 | 5 | 5 |
| 3 | 0 | 1 | 4 |
| **4** | **0** | **5** | **3** |
| 5 | 1 | - | ∞ |

| | U | V | C |
|---|---|---|---|
| 1 | 1 | - | ∞ |
| 2 | 0 | 5 | 5 |
| **3** | **0** | **1** | **4** |
| 4 | 1 | - | ∞ |
| 5 | 1 | - | ∞ |

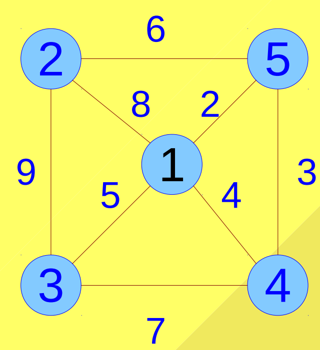
| | U | V | C |
|---|---|---|---|
| 1 | 1 | - | ∞ |
| **2** | **0** | **5** | **5** |
| 3 | 1 | - | ∞ |
| 4 | 1 | - | ∞ |
| 5 | 1 | - | ∞ |

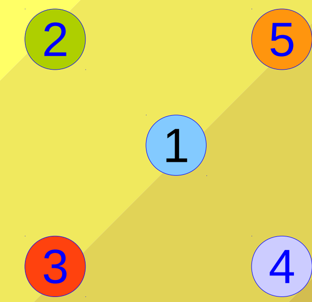
L:{ (1,5), (5,4), (1,3), (5,2) }

# Árbol abarcador mínimo

- Algoritmo de Kruskal:
  - **Inicial**: cada nodo representa una subgrafo.
  - Crear lista ordenada de todos los lados de menor a mayor coste.
  - Selecciona el lado de menor coste que une dos subgrafos distintos siguiendo el orden creciente de la lista.
  - Ahora estos dos subgrafos forman un único subgrafo.
  - Repetir hasta que quede un sólo sub-grafo.
  - Complejidad: $O(m\log_2 m)$
  - Más eficiente que Prim para grafos poco densos.

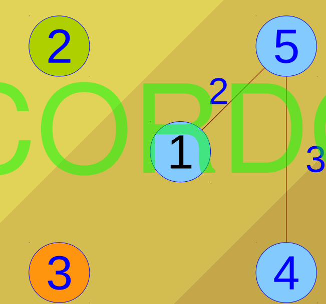
DPTO INFORMATICA - U. CORDOBA



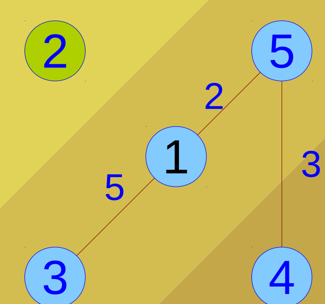





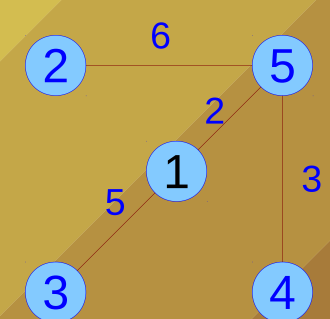
L:{**(1,5)**, (4,5), (1,4), (1,3), (2,5), (3,4), (1,2), (2,3)}




L:{(1,5), **(4,5)**, (1,4), (1,3), (2,5), (3,4), (1,2), (2,3)}




L:{(1,5), (4,5), (1,4), **(1,3)**, (2,5), (3,4), (1,2), (2,3)}




L:{(1,5), (4,5), (1,4), (1,3), **(2,5)**, (3,4), (1,2), (2,3)}

# Referencias

- Lecturas recomendadas:
  - Apuntes de la asignatura (moodle).
  - Caps. 14, 15 y 16 de “Estructuras de Datos”, A. Carmona y otros. U. de Córdoba. 1999.

DPTO INFORMATICA - U. CORDOBA