LECCIÓN 11: ALGORITMOS VORACES

-Exploración de Grafos

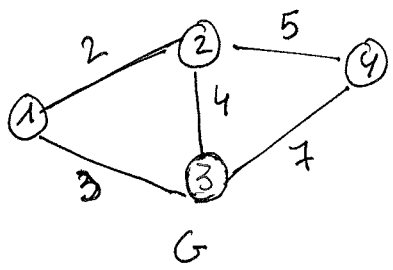
Grafo $G=(V,A)$ V: conjunto de vertices $v_1, v_2, \dots v_n$

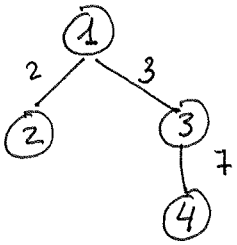
A: conjunto de aristas (v_i, v_j)

CONCEPTOS:

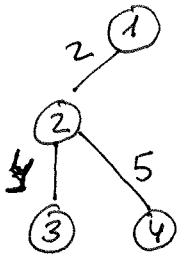
Árbol de recubrimiento de G.- Se define al subgrafo de G sin ciclos (cada vértice puede aparecer como final de una arista una sola vez) y contiene todos sus vertices.

Ej





Arbol de recubrimiento de G (1)
coste = 12



Arbol de recubrimiento de G (2)
coste 11

En caso de existir varios árboles de recubrimiento, nos interesa aquel de coste mínimo. La suma de los pesos de las aristas en el arbol es el menor posible.

Algoritmos que obtiene el Arbol de recubrimiento Minimal: Alg Prim y Kruskal. En ambos algoritmos en cada paso se añade una arista o arco. La forma de escoger esa arista es lo que distinguen a los dos Algoritmos ⇐ DISTINTAS FUNCIONES de SELECCIÓN.

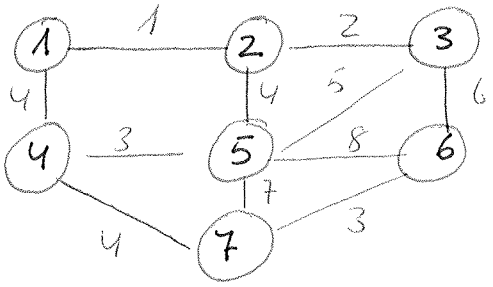
(2)

LECCION 11: ALGORITMOS VORACES

Arboles de recubrimiento minimal de G

ALG de KRUSKAL

Ejemplo



Paso 1: Ordenar las aristas de forma creciente del peso

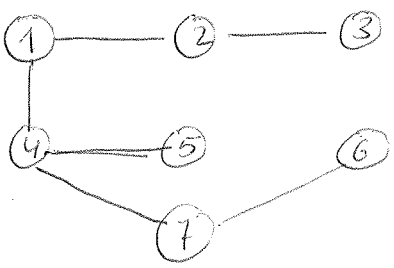
(1,2), (2,3) (4,5) (6,7) (1,4) (2,5); (4,7)
(3,5) (3,6) (5,7) (5,6)

- Los siguientes pasos escoge una arista con vértices en dos componentes conexas diferentes
- Inicialmente cada vertice forma una componente conexa.

{1}, {2}, {3}, {4}, {5}, {6} {7}

- El algoritmo para cuando nos quede una componente conexa.

PASO	ARISTA SELECCIONADA	COMPONENTES CONEXAS
1	(1,2)	{1,2}, {3}, {4} {5}, {6}, {7}
2	(2,3)	{1,2,3} {4} {5} {6} {7}
3	(4,5)	{1,2,3} {4,5} {6} {7}
4	(6,7)	{1,2,3} {4,5} {6,7}
5	(1,4)	{1,2,3,4,5} {6,7}
6	(4,7)	{1,2,3,4,5,6,7}



LECCION 11.- ALGORITMOS VORACES

Pseudo código - Kruskal

```
FUNCION KRUSKAL (G:<N,A>)
    //INICIALIZACION
1)  Ordenar A por valor del peso de forma creciente
2)  n ← |N| //nº de vertices   T ← ∅ //arbol recubrimiento
3)  Iniciar n componentes conexas cada una contiene
    un vértice de N
4)  repetir
5)      e ← {u,v} // la arista de menor peso en A
6)      A ← A - {u,v}
7)      compu ← Componente_conexa(u)
8)      compv ← Componente_conexa(v)
9)      si compu ≠ compv
10)         fusionar (compu, compv)
11)         T ← T U {e}
12) hasta que T tenga n-1 aristas.
end.
```

- El árbol de recubrimiento encontrado por Kruskal es minimal.

- **EFICIENCIA**. Sea a el nº de aristas del Grafo y n el nº de vertices.

Como maximo puede considerar todas las aristas del grafo. Al fusionar dos componentes conexas. nos quedamos con una menos.
Si tenemos los vertices ordenados buscar(u) buscar(v) es log(n). Fusionar puede realizarse en O(1). Luego en total $O(a \log n)$ donde a

está comprendido en $(n-1) \leq a \leq n\left(\frac{n-1}{2}\right)$

(n-1): Grafo que es un camino.

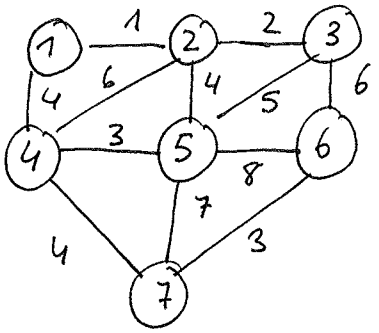
$n\left(\frac{n-1}{2}\right)$: Grafo completo

4

LECCION - 11 ALGORITMOS VORACES

Arbol de Recubrimiento Minimal. Algoritmo de Prim

- La diferencia es que la arista que se coge uno de los nodos pertenece al árbol de recubrimiento en el paso i-th y el otro nodo/vertice no.



$B \leftarrow \{1\}$

Paso	$\{u,v\}$	B
1	$\{1,2\}$	$\{1,2\}$
2	$(2,3)$	$\{1,2,3\}$
3	$(1,4)$	$\{1,2,3,4\}$
4	$(4,5)$	$\{1,2,3,4,5\}$
5	$(4,7)$	$\{1,2,3,4,5,7\}$
6	$(7,6)$	$\{1,2,3,4,5,7,6\}$

Orden de las aristas

(1,2), (2,3) (4,5) (6,7)
(1,4) (2,5) (4,7) (3,5)
(3,6). (2,4) (5,7) (5,6)

Funcion Prim ($G=\langle N,A\rangle$)

$T \leftarrow \emptyset$

$B \leftarrow$ { un miembro arbitrario de N }

mientras $B \neq N$ hacer

buscar $e=\{u,v\}$ de longitud mínima tq $u \in B$ y $v \in N \setminus B$

$T \leftarrow T \cup \{e\}$

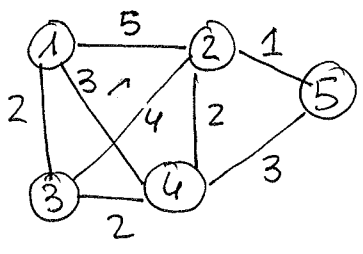
$B \leftarrow B \cup \{v\}$

end

devolver T

end.

Ej 2



aristas ordenadas: (2,5) (1,3) (3,4) (4,2) (1,4) (4,5) (3,2) (2,1)

$B \leftarrow \{1\}$

Paso	(u,v)	B
1	(1,3)	{1,3}
2	(3,4)	{1,3,4}
3	(4,2)	{1,3,4,2}
4	(2,5)	{1,3,4,2,5}

LECCION 11 : ALGORITMOS VORACES

Alg. Prim (continuación)

Una implementación sencilla del algoritmo se podría llevar a cabo suponiendo que los nodos de G están enumerados de 1 a n $N=\{1,2,\dots,n\}$. Además vamos a tener una matriz simétrica L que da el peso de las aristas, y $L[i,j]=\infty$ si no existe la arista (i,j). Supongamos que tenemos un vector más-proximo[i] que para todo $i \in N \setminus B$ proporciona el nodo de B más próximo a i. Además disponemos del vector distmin[i] da la distancia desde $i \in N \setminus B$ hasta el nodo más próximo en B. Para todo nodo en B hacemos distmin[i] = -1.

```
typedef pair<int,int> arista;
void Prim (Matriz& L, list<arista>& T, int N){
    vector<int> mas_proximo(N+1); //vamos a indexar desde 1
    vector<float> distmin(N+1); //vamos a indexar desde 1

    for (int i=1; i<=N; i++){
        mas_proximo[i]=1;
        distmin[i]=L(i,1);
    }
    int k;
    for (int l=1; l<=N-1; l++){                          O(n^2)
        float min=numeric_limits<float>::max();
        for (int j=1; j<=N; j++){                        O(n)
            if (distmin[j]>=0 && distmin[j]<min){
                min=distmin[j];
                K=j;
            }
        }
        arista a(mas_proximo[K],K);
        T.insert(T.end(),a);
        distmin[K]=-1;
        for (int j=2; j<=N; j++)                         O(n)
            if (L[j,K]<distmin[j]){
                distmin[j]=L[j,K];
                mas_proximo[j]=K;
            }
    }
}
```

Se corrige la distancia mínima a los nodos aún no incluidos en B.