Estructuras Discretas
Algoritmos en Pseudocodigos.

Teoría de Grafos (Implementación)

Prof. Miguel Fagundez

Algoritmo de Dijkstra

Con este algoritmo encontramos el camino mas económico o de menor costo entre un par de vértices (v_i,v_j), así que el grafo G = (V,A) debe ser un grafo pesado o etiquetado.

```
procedimiento alg_Dijkstra(G: grafo; n: entero; var Costo, Cam: vector[1..n])
inicio
   variable
   T: conj_vertices;              // En T se van agregando los vértices ya tratados
   i: entero;                     // i es contador de vértices
   inicializar Cam[i] ← 0;   ∀i∈{1, 2, …, n}   //Aquí se inicializan todos los caminos
   T ← {1};
   Costo[1] ← 0;
   para i ← 2 [incrementando] hasta n
      si ( (1, i)∈A ) entonces
         Costo[i] ← C(1, i);
         Cam[i] ← 1;
      sino
         Costo[i] ← ∞;              {un valor muy grande}
      fsi
   fpara
   para i ← 1 [incrementando] hasta n-1
      elegir un vértice  k∈(V – T)  tal que  Costo[k]  sea el menor posible;
      T ← T È {k}
      si (T ≠ V) entonces
         para  cada vértice v en (V – T)  hacer
           si ( (k, v) ∈A ) entonces
             si ( Costo[k] + C(k, v) < Costo[v] ) entonces
               Costo[v] ← Costo[k] + C(k, v);
               Cam[v] ← k;
             fsi
           fsi
         fpara
      fsi
   fpara
finProcedimiento  // fin procedimiento alg_Dijkstra //
```

Algoritmo de Kruskal

Con este algoritmo encontramos el árbol expandido mínimo de un grafo ponderado o etiquetado G = (V,A).

```
procedimiento alg_Kruskal(A: conj_arcos; G: grafo; n: entero; var Amin: conj_arcos)
inicio
   variable
      T: grafo;      // En T se va generando la secuencia de subgrafos Ti
      i: entero;   // i es contador del número de arcos añadidos al subgrafo actual Ti
      CT: real;              // CT es el costo del subgrafo expandido Ti
      AuxA: conj_arcos;      // AuxA es variable auxiliar, contiene inicialmente los arcos de A
   ei: tipo_arco;
   inicializar(T, G);             // Aquí en T se colocan los vértices de G solamente
   Amin ← f;
   CT ← 0;
   AuxA ← A;
   i ← 0;
   mientras (i < n – 1) hacer
      ei ← extraer_arco_min_costo(AuxA - Amin);
      si ¬(forma_circuito_al_añadir(ei, T)) entonces
         Amin ← Amin∪{ei};
         CT ← CT + C(ei);
         i ← i + 1;
      sino                                   // si forma circuito
         AuxA ← AuxA – {ei};
      fsi
   fmientras
   escribir("El conjunto de arcos de Tmin es: ", Amin , " con un costo de ", CT);
finProcedimiento  // fin procedimiento alg_Kruskal //
```

Algoritmo de las Componentes Conexas.

Con este algoritmo determinamos el numero de Componentes Conexas de un grafo G = (V,A)

{Sea *G* = (V,A) un grafo}

var

p: entero; {p contendrá el numero de componentes conexas de *G*}

V, B, X: conjunto; {V es el conjunto de vértices del grafo.
B tendrá inicialmente todos los vértices del grafo y luego se ira reduciendo a medida que los vértices sean tratados.
X contendrá una por una, las componentes conexas}

x,w: elemento de conjunto;

inicio

```
p←0;
B←V;
mientras B≠∅ hacer
    x←extraer_elemento(B);
    X←{x};
    B←B-X;
    para cada elemento w de B hacer
        Si existe_camino(w, algun elemento de X) entonces
            X←X∪{w};
        Fsi
    fpara
    escribir (X,"es una componente conexa")
    p←p+1;
    B←B-X;
fmientras
escribir ("El numero de componentes conexas es:",p)
fin.
```

Algoritmo para conocer el Rango Circuito de G.

Con este algoritmo determinamos el numero de circuitos independientes (Rango Circuito) de un grafo G = (V,A)

```
var
Gn:grafo;        {Gn almacenara la sucesión de subgrafos que el algoritmo va
                  creando: Parte del subgrafo formado solo por vértices (no tiene
                  arcos); en cada iteración se agrega un nuevo arco y finaliza cuando
                  todos los arcos son colocados, es decir, cuando alcanza al grafo
                  original}
R: entero;       {R contendrá el rango circuito del grafo G, es decir, e numero
                  máximo de circuios en G}
A,AuxA: conjunto; {A es el conjunto de arcos del grafo, representados por medio
                   de sus etiquetas; AuxA comenzara vació y se ira agrandando a
                   medida que este almacenando los arcos del grafo que ya han
                   sido tratados.}
e: elemento del conjunto;

inicio

    Inicializar(Gn)   {Aquí en Gn solo se colocan los vértices de G solamente}
    AuxA←∅;
    R←0;
    mientras AuxA≠A hacer
        e←extraer_arco(A-AuxA);    {siendo e = (vi,vj)}
        si existe_camino(vi,vj,Gn) entonces
            R←R+1;
            Escribir ("Circuito independiente:",camino(vi,vj) + e)
        fsi
        AuxA←AuxA∪{e};
        Agregar_arco(e,Gn);
    fmientras
    escribir("Rango Circuito de G es:",R);

fin.
```

Algoritmo de Warshall.

Este algoritmo obtiene la matriz de Alcanzabilidad de orden n. Usando el Teorema Alcanzabilidad podemos elaborar este algoritmo, pero es muy ineficiente por lo que el algoritmo de warshall es el mas utilizado. ***Modificar*** este algoritmo para que use solo la triangular superior o inferior ya que la matriz es simétrica.

```
var
    i, j, r: entero;{i,j,r variables de control para los ciclo for}
    Ady, E*: arreglo [1 .. n, 1 .. n] de bool; {Matrices de Ady y de Alcanzabilidad = E*}
comienzo
    para i := 1 hasta n hacer
        para j:= 1 hasta n hacer
            E* [i, j] := Ady [i, j];
            si i = j entonces
                E* [i, j] := 1;
            fsi
        fpara
    fpara
    para r:= 1 hasta n hacer
        para i:= 1 hasta n hacer
            para j:= 1 hasta n hacer
                si E*[i, j] = 0 entonces
                    E*[i, j] := E*[i, r] ∧ E*[r, j];
                fsi
            fpara
        fpara
    fpara
fin
```