ABELIAN GROUP

Dossier de consulta

AdaByron 2019

Alejandro García Carretero

Laura Medina Henche

Yihui Xia

ALGUNAS FORMULAS MATEMÁTICAS

$$\sum_{1}^{n} i = \frac{n(n+1)}{2} \ (Num\ triangulares) \qquad \sum_{1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad a + \cdots + b = \frac{n(a+b)}{2}$$

$$Pirámide = \frac{n(n+1)(n+2)}{6} \qquad \sum_{1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 \qquad \binom{a}{b} = \frac{a!}{b!\,(a-b)!}$$

$$Cn = \frac{(2n)!}{(n+1)!\,n!} \ (Nº\ de\ paréntesis\ balanceados\ o\ árboles\ binarios\ de\ n\ nodos(Tb\ rooted))$$

//Número primo

```
boolean esPrimo(int n){
        if(n<2) return false;
        for(int i=2;i2<=n;i++){
                if(n%i==0) return false;
        }
        return true;
}
```

//Eratostenes

```
//sieve es un array de números inicializado a
//0, de longitud n(numero hasta el que se
//quiere comprobar)
for (int x = 2; x <= n; x++) {
      if (sieve[x]) continue;
      for (int u = 2*x; u <= n; u += x) {
           sieve[u] = x;
      }
}
```

//MCD

```
static int mcd(int a, int b) {
        int aux;
        while (b != 0){
                a %= b;
                aux = b;
                b = a;
                a = aux;
        }
        return a;
}
```

//MCM

```
static int mcm (int a, int b) {
          return a * b / mcd(a, b);
}
```

//Fibonacci logn

```
static int fibonacci(int n){
    int h=1,i=1,j=0,k=0,aux;
    while (n > 0){
       if ( n % 2 != 0){
          aux = h*j
          j = h*i + j*k + aux
          i = i*k + aux
       }
       aux = h*h
       h = 2*h*k + aux
       k = k*k + aux
       n = n/2
    }
    return j;
}//Fin fibonacci
```

//Fibonacci aproximado

```
static double fibonacci(int n){
   double raiz = sqrt(5);
   double fib = ((1/raiz)*(pow((1+raiz)/2,n)) -
(1/raiz)*(pow((1-raiz)/2,n)))
   return fib;
}
```

//Numeros catalanes

```
int catalan(int n) {
      int res = 0;
      if (n <= 1) {
         return 1;
      }
      for (int i = 0; i < n; i++) {
         res += catalan(i) * catalan(n - i - 1);
      }
      return res;
   }
```

GRAFOS

//DIJSKTRA (matriz)

```
PriorityQueue<State> cola = new PriorityQueue<>();
boolean visitado[] = new boolean[N];
int distancia[] = new int[N];
int INF = -1;
for (int i = 1; i < N; i++) {
        distancia[i] = INF;
}
cola.offer(new State(0,0));
while(!cola.isEmpty()){
        State actual = cola.poll();  //a[0] i del grafo a[1] valor del grafo
        for(int j = 0;j<N;j++){
                if(actual.nodo!= j && grafo[actual.nodo][j]!=INF && !visitado[actual.nodo]){
                    if(distancia[j]>grafo[actual.nodo][j] + actual.distancia || distancia[j] == INF){
                                distancia[j] = grafo[actual.nodo][j] + actual.distancia;
                    }
                    cola.offer(new State(j,distancia[j]));
                }
        }
        visitado[actual.nodo]=true;
}//Fin del while
```

//DIJSKTRA (listas)

```
PriorityQueue<State> cola = new PriorityQueue<>();
boolean visitado[] = new boolean[N];
int distancia[] = new int[N];
int INF = -1;
for (int i = 1; i < N; i++) {
        distancia[i] = INF;
}
cola.offer(new State(0,0));
```

```
while(!cola.isEmpty()){
        State actual = cola.poll();  //a[0] i del grafo a[1] valor del grafo
        for(Arista arista: grafo[actual.nodo]){
                int destino = arista.to;
                int peso = arista.peso;
                if(!visitado[actual.nodo]){
                    if(distancia[destino]>peso + actual.distancia || distancia[destino] == INF){
                                distancia[destino] = peso + actual.distancia;
                    }
                    cola.offer(new State(destino,distancia[destino]));
                }
        }
        visitado[actual.nodo]=true;
}
```

//DFS

```
public static void DFS(ArrayList<Arista>[]
grafo,int N){
      ArrayDeque<State> pila = new
ArrayDeque<>();
      boolean visitado[] = new boolean[N];
      pila.push(new State(0,0));
      visitado[0] = true;
      while(!pila.isEmpty()){
          State aux = pila.pop();
          for(Arista ady: grafo[aux.nodo]){
                int destino = ady.to;
                if(!visitado[destino]){
                      pila.push(new
State(destino,0));
                      visitado[destino] =
true;
                }
          }//for
      }//while
   }//DFS
```

//BFS

```
public static void BFS(ArrayList<Arista>[]
grafo,int N){
      ArrayDeque<State> cola = new
ArrayDeque<>();
      boolean visitado[] = new boolean[N];
      cola.offer(new State(0,'0'));
      visitado[0] = true;
      while(!cola.isEmpty()){
         State aux = cola.poll();
         System.out.println(aux.nodo);
         for(Arista ady: grafo[aux.nodo]){
            int destino = ady.to;
            if(!visitado[destino]){
               cola.offer(new
State(destino,'0'));
               visitado[destino] = true;
            }
         }
      }
   }
```

//PRIM

```
//Previo
class State{  int nodo, distancia; }
class Arista{ int from,to,peso; }
ArrayList<Arista>[] grafo = new ArrayList[N];
grafo[ini].add(new Arista(ini,fin,coste));
grafo[fin].add(new Arista(fin,ini,coste));
```

```
//Algoritmo
PriorityQueue<State> cola = new PriorityQueue<>();
        boolean visitado[] = new boolean[N];
        int suma = 0;
        cola.offer(new State(0,0));
        while(!cola.isEmpty()){
                State aux = cola.poll();
                if(!visitado[aux.nodo]){
                        suma+=aux.distancia;
                        visitado[aux.nodo] = true;
                        for (int i = 0; i < grafo[aux.nodo].size(); i++) {
                                int destino = grafo[aux.nodo].get(i).to;
                                int peso = grafo[aux.nodo].get(i).peso;
                                if(!visitado[destino]){
                                        cola.offer(new State(destino,peso));
        }       }       }       }
        int f = 0;
        for (int i = 0; i < N; i++) {
                if(!visitado[i]){
                        f = 1;
                        break;
                }
        }
```

//FIN ALGORITMO

//Ciclo (y Camino) Euleriano: solo si C(v) es par (C(v) par con impar en 2 vertices)

//Ciclo (u Camino) Hamiltoniano:

//Componentes Conexas: modificar dfs, pasando por referencia visitado[] y el nodo inicio

```
public static int cc(ArrayList<Arista>[] grafo,int N){
      boolean visitado[] = new boolean[N];
      int cuenta = 0;
      for (int i = 0; i < N; i++) {
          if(!visitado[i]){
              DFS(grafo,N,i,visitado);
```

```
                cuenta++;
            }
        }
        return cuenta;
}
```

//Sobre matrices: Grafos Dirigidos

//TopoSort

```
public static void topoUtil(int[][] grafo,int actual,boolean[] visitado,ArrayDeque<Integer> pila){
        visitado[actual] = true;
        for (int i = 0; i < grafo.length; i++) {
            if(grafo[actual][i]>0 && !visitado[i]){
                topoUtil(grafo,i,visitado,pila);
            }
        }
        pila.push(actual);
}
public static ArrayDeque<Integer> topoSort(int[][] grafo){
        ArrayDeque<Integer> pila = new ArrayDeque<>();
        ArrayDeque<Integer> output = new ArrayDeque<>();
        boolean visitado[] = new boolean[grafo.length];
        for (int i = 0; i < grafo.length; i++) {
            if(!visitado[i]){
                topoUtil(grafo,i,visitado,pila);
            }
            while(!pila.isEmpty()){
                output.offer(pila.pop());
            }
        }
        return output;
}
```

//SCC

```
public static int scc(int[][] grafo,int N){
        boolean visitado[] = new boolean[N];
        ArrayDeque<Integer> cola = topoSort(grafo);
        int cuenta = 0;
        for (int i: cola) {
                if(!visitado[i]){
                        DFSt(grafo,N,i,visitado);
                        cuenta++;
        }       }
        return cuenta;  }//Fin scc
```

//DFST

```
public static void DFSt(int[][] grafo,int N,int inicio,boolean visitado[]){
        ArrayDeque<Integer> pila = new ArrayDeque<>();
        pila.push(inicio);
        visitado[inicio] = true;
        while(!pila.isEmpty()){
                int aux = pila.pop();
                for(int ady=0;ady<N;ady++){
                        if(grafo[ady][aux]>0){
                                int destino = ady;
                                if(!visitado[destino]){
                                        pila.push(destino);
                                        visitado[destino] = true;
}       }       }       }       }
```

//Máscaras de bits

Si tuviésemos 3 nodos y estuviésemos interesados en chequear un ciclo hamiltoniano tendríamos que usar el entero 7, (2^3-1), cuya representación de bits es 111.

- Por cada nodo que visitemos lo marcamos a 0
  - mask = 7 // 111
  - mask ^ (1<<i) donde i es el nodo
- Si queremos saber si el i-ésimo bit está encendido:
  - mask & (1<<i)

i. Desplazamos 1 i veces a la izquierda

ii. Para i=2; 100 (4)

iii. 7 & 4 = 111 & 100 => (4)

iv. Mientras que mask & (1<<i) no tome un valor de 0, el i-ésimo bit está encendido.

●Si queremos convertir el i-ésimo bit a 0 (sabiendo que está encendido de antes):

○ mask ^ (1<<i)

i. Desplazamos 1 i veces a la izquierda

ii. Para i=2; 100 (4)

iii. 7 ^ 4 = 111 ^ 100 => (011)

●En caso del camino hamiltoniano si mask=0 hemos terminado

● En caso de ciclo si mask=0 tenemos que comprobar que el nodo actual es igual al nodo de inicio

//TopoSortW

```
// A Java program to print topological sorting of a DAG
// This class represents a directed graph using adjacency list representation
class Graph {
	private int V; // No. of vertices
	private LinkedList<Integer> adj[]; // Adjacency List
	Graph(int v){
		V = v;
		adj = new LinkedList[v];
		for (int i=0; i<v; ++i)
			adj[i] = new LinkedList();
	}
	// Function to add an edge into the graph
	void addEdge(int v,int w) { adj[v].add(w); }
	void topologicalSortUtil(int v, boolean visited[],Stack stack){
		// Mark the current node as visited.
		visited[v] = true;
		Integer i;
		// Recur for all the vertices adjacent to this vertex
		Iterator<Integer> it = adj[v].iterator();
		while (it.hasNext()){
			i = it.next();
```

```
                if (!visited[i])
                        topologicalSortUtil(i, visited, stack);        } //Fin del while
        // Push current vertex to stack which stores result
        stack.push(new Integer(v));
}

// The function to do Topological Sort. It uses recursive topologicalSortUtil()
void topologicalSort(){
        Stack stack = new Stack();
        // Mark all the vertices as not visited
        boolean visited[] = new boolean[V];
        for (int i = 0; i < V; i++)
                visited[i] = false;
        // Call the recursive helper function to store Topological Sort starting from all
        vertices one by one
        for (int i = 0; i < V; i++)
                if (visited[i] == false)
                        topologicalSortUtil(i, visited, stack);
        // Print contents of stack
        while (stack.empty()==false)
                System.out.print(stack.pop() + " ");
}
// Driver method
public static void main(String args[]){
        // Create a graph given in the above diagram
        Graph g = new Graph(6);
        g.addEdge(5, 2); //Y etcetera
        g.topologicalSort();
}
```

PROGRAMACIÓN DINÁMICA

//Problema de la mochila (PD)

```
int N; //Numero de objetos
int precios[]; //precios de cada producto
int pesos[]; //pesos de cada producto
int memo[][]; //inicializado todo a -1
static int mochila(int id, int w) {
        if(id == N || w == 0){ return 0; }
        if(memo[id][w] != -1){ return memo[id][w]; }
        if(pesos[id] > w){  memo[id][w] = mochila(id + 1, w);
        }else{
                memo[id][w] =  MAX(mochila(id + 1, w), precios[id] + mochila(id + 1, w -
                pesos[id]));
                return memo[id][w];
        }
}
```

//Problema del cambio(PD)

//Dada una cantidad V de centavos, y una lista de n monedas existentes, determinar cual es la mínima cantidad de monedas que debe usarse para completar V.

```
int monedas[n];
int mem[MAX];
static int cambio(int k) {
        if (k == 0){ return 0; }
        if(k < 0){ return INF; }
        if(mem[k] != -1){ return memo[k]; }
        int change = INF;
        for(int i=0;i<n;i++){
                change=min(change, cambio(k-monedas[i]);
        }
        mem[k]=change+1;
        return change+1;
}
```