- ALGORITMO JALEAK: HEDAPEN ZUHAITZ MINIMOA – KRUSKAL

```
public class GrafoEzZuzendua {
	private Erpina[] auzokideZerrenda;

	public GrafoEzZuzendua(int erpinKop, LinkedList<Ertza> ertzak){
		auzokideZerrenda= new Erpina[erpinKop];
		for(int i=0; i<erpinKop; i++){
			auzokideZerrenda[i]= new Erpina(i);
		}
		Iterator<Ertza> it = ertzak.iterator();
		while (it.hasNext()){
			Ertza e = it.next();
			int e1znbk = e.getJatorriErpinZenbakia();
			int e2znbk = e.getHelmugaErpinZenbakia();
			Erpina e1 = auzokideZerrenda[e1znbk];
			Erpina e2 = auzokideZerrenda[e2znbk];
			e1.ertzaGehitu(e);
			e2.ertzaGehitu(e);
		}
	}

	public double HZMKruskal(LinkedList<Ertza> ertzZerrenda,
LinkedList<Ertza> HZMErtzak){
		Collections.sort(ertzZerrenda, new Comparator<Ertza>(){ /*
Mergesort erabiltzen da */
			public int compare(Ertza a1, Ertza a2) {
				if(a1.getPisua()<a2.getPisua()){
					return -1;
				}else{
					return 1;
				}
			}
		});
		int ertzKop=0;
		double pisuenBatura=0;
		Partiketa partiketa = new Partiketa(auzokideZerrenda.length);
		while (ertzKop!=auzokideZerrenda.length-1){
			Ertza ertza = ertzZerrenda.removeFirst();
			int erpin1 = ertza.getHelmugaErpinZenbakia();
			int erpin2 = ertza.getJatorriErpinZenbakia();
			int etiketa1 = partiketa.bilatu3(erpin1);
			int etiketa2 = partiketa.bilatu3(erpin2);
			if (etiketa1!=etiketa2){ /* Zikloa ez da osatzen */
				partiketa.bateratu3(etiketa1, etiketa2);
				HZMErtzak.add(ertza);
				pisuenBatura = pisuenBatura + ertza.getPisua();
				ertzKop++;
			}
		}
		return pisuenBatura;
	}
}
```

- PARTIKETA

```
public class Partiketa {
	private int[] partiketa = null;

	/* Metodo Eraikitzailea */
	public Partiketa(int tamaina){
		partiketa =  new int[tamaina];
		for(int i=0; i<tamaina; i++){
			partiketa[i]=-1;
		}
	}


	public int bilatu3(int elementua){
		int aux = elementua;
		while(partiketa[aux]>=0){  /* >=0 izan behar du, gure
implementazioan 0 zenbakia duten erpinak egon daitezkeelako */
			aux=partiketa[aux];
		}
		return aux;
	}


	public void bateratu3(int e1, int e2){
		if(partiketa[e1]==partiketa[e2]){ /* Sakonera berdineko
zuhaitzak (s1==s2) */
			partiketa[e1]=partiketa[e1]-1;
			partiketa[e2]=e1;
		}else if(partiketa[e1]<partiketa[e2]){ /*
sakonera(s2)>sakonera(s1) --> Sakonera berria (batu ondoren): s2 */
			partiketa[e2]=e1;
		}else{ /* sakonera(s2)<sakonera(s1) --> Sakonera berria
(batu ondoren): s1 */
			partiketa[e1]=e2;
		}
	}
   }
```

- METAK

```
// MINIMOAREN PROPIETATEA BETETZEN DUEN META
public class Meta {
	private int v[];

	public Meta(int [] bektorea){
		v=bektorea;
		metaEraikiHondoratuz();
		//metaEraikiAzaleratuz();
	}

	public void hondoratu(int indizea){
		int i=indizea;
		int umeMin;
		int aux;
		boolean jarraitu=true;
		while ((2*i+1)<=(v.length-1) && jarraitu){
			umeMin = 2*i+1;
			if ((2*i+2)<=v.length-1 && v[2*i+2]<v[2*i+1]){
				umeMin=2*i+2;
			}
			if(v[i]>v[umeMin]){
				aux=v[i];
				v[i]=v[umeMin];
				v[umeMin]=aux;
				i=umeMin;
			}else{
				jarraitu=false;
			}
		}
	}

	public void azaleratu(int indizea){
		int i=indizea;
		int aux;
		boolean jarraitu=true;
		while (((i+1)/2)-1>=0 && jarraitu){
			if(v[i]<v[((i+1)/2)-1]){
				aux=v[i];
				v[i]=v[((i+1)/2)-1];
				v[((i+1)/2)-1]=aux;
				i=((i+1)/2)-1;
			}else{
				jarraitu= false;
			}
		}
	}

	private void metaEraikiHondoratuz() {
		for(int i=v.length-1;i>=0;i--){
			hondoratu(i);
		}
	}

	private void metaEraikiAzaleratuz() {
		for(int i=0; i<v.length; i++){
			azaleratu(i);
		}
	}

}
```

- ALGORITMO JALEAK: HEDAPEN ZUHAITZ MINIMOA – PRIM

```
public double HZMPrim(int[][] grafoa, LinkedList<Ertza> HZMErtzak){
            int[] pisuMin = new int[grafoa.length];
            int[] auzokide = new int [grafoa.length];

            for(int i=1; i<grafoa.length; i++){
                  pisuMin[i]=grafoa[0][i];
                  auzokide[i]=0;
            }

            double pisuenBatura=0;
            for(int i=1; i<grafoa.length; i++){
                  int min = Integer.MAX_VALUE;
                  int erpina=-1;
                  for(int j=1; j<grafoa.length; j++){
                        if(pisuMin[j]>0 && pisuMin[j]<min){
                              min=pisuMin[i];
                              erpina=j;
                        }
                  }
                  pisuenBatura=pisuenBatura+min;
                  HZMErtzak.add(new Ertza(erpina, auzokide[erpina],
min));
                  pisuMin[erpina]=-1;

                  for(int j=1; j<grafoa.length; j++){
                        if(grafoa[erpina][j]<pisuMin[j]){
                              pisuMin[j]=grafoa[erpina][j];
                              auzokide[j]=erpina;
                        }
                  }
            }
            return pisuenBatura;
```

- DISTANTZIA MINIMOAK – DIJKSTRA

```
private final int INF = 100;

	public int[] dijkstraDistantziaMinimoak(int[][] grafoa){
		int[] disTaula = new int[grafoa.length];
		boolean[] hautagaiak = new boolean [grafoa.length];
		disTaula[0]=0;
		hautagaiak[0]=false;
		/* 0 erpinetik hasten da (abiapuntua 0 da) */
		for(int i=1; i<grafoa.length; i++){
			disTaula[i]=grafoa[0][i];
			hautagaiak[i]=true;
		}
		for(int k=1; k<grafoa.length-2; k++){
			int min = INF;
			int gertuenDagoenErpina = 0;
			/* Gertuen dagoen erpina atera */
			for(int i=1; i<grafoa.length; i++){
				if(hautagaiak[i] && disTaula[i]<min){
					min = disTaula[i];
					gertuenDagoenErpina = i;
				}
			}
			/* Hautagaietatik kendu */
			hautagaiak[gertuenDagoenErpina]=false;
			/* Distantziak hobe daitezke? */
			for(int i=1; i<grafoa.length; i++){
				if(hautagaiak[i] &&
disTaula[gertuenDagoenErpina]+grafoa[gertuenDagoenErpina][i]<disTaula[
i]){

	disTaula[i]=disTaula[gertuenDagoenErpina]+grafoa[gertuenDagoenEr
pina][i];
				}
			}
		}
		return disTaula;
	}
```

- BIDE MOTZENAK – FLOYD

Ekuazioak

$$d_{i,j}^{k} = \begin{cases} pisua(i,j) & if\ k = 0 \\ \min\left(d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\right) & if\ k \geq 1 \end{cases}$$

Metatze-egitura

Ez da metatze-egitura estrarik behar.

Kodea

```
public int[][] floyd (int[][] grafoa){
            int[][] distantziak = new
int[grafoa.length][grafoa[0].length];
            for(int i=0; i<grafoa.length; i++){
                  for(int j=0; j<grafoa.length; j++){
                        distantziak[i][j]=grafoa[i][j];
                  }
            }

            for(int k=0; k<grafoa.length; k++){
                  for(int i=0; i<grafoa.length; i++){
                        for(int j=0; j<grafoa.length; j++){

      distantziak[i][j]=Math.min(distantziak[i][j],
distantziak[i][k]+distantziak[k][j]);
                        }
                  }
            }

            return distantziak;
      }
```

- KOSTUAK

- **Partiketa** *[s=zuhaitzaren sakonera; osagaiKop=partiketak dauzkan elementu kopurua].*

ALGORITMOA	ORDENA [Kasu txarrena]
Bilatu3	O(s) = O(lg osagaiKop)
Bateratu3	O(1)

[Bilatu3/Bateratu3]^n sekuentzia izatekotan, ordena ∈ O(n lgn)

- **Metak** *[n=metako elementu kopurua]*

PROZEDURA	ORDENA [Kasu txarrena]
Hondoratu	O(lgn)
Azaleratu	O(lgn)
MetaEraikiHondoratuz	O(n)
MetaEraikiAzaleratuz	O(n lgn)

- **Algoritmo Jaleak Grafoetan** *[a=arku/ertz kopurua; p=erpin kopurua].*

ALGORITMOA	ORDENA [Kasu txarrena]
HZM Kruskal	O(a lgp)
HZM Prim	O(p^2)
Grafoak ertz asko (dentsoa)→ PRIM erabili Grafoak ertz gutxi → KRUSKAL erabili	
Dijkstra	O(p^2)

- **Programazio Dinamikoa Grafoetan** *[p=erpin kopurua].*

ALGORITMOA	ORDENA [Kasu txarrena]
Floyd	O(p^3)

- ALGORITMO JALEEN OSAGAI KOMUNAK

1. **Hautagaien multzoa**
   Aukeran ditugun “elementuak”.

2. **SP: Soluzio partziala**
   Jadanik aukeratu eta onartu diren hautagaien multzoa. Hautagaien azpimultzoa da.

3. **SoluzioaDa?**
   Hautagaien azpimultzo bat gure arazoaren soluzioa den ala ez erabakitzen duen funtzioa. [Normalean hautagaiak bukatu zaizkigunean → SoluzioaDa? True]

4. **Hautesle-Prozedura**
   Oraindik aukeratu ez diren hautagaietatik soluzio onenaren/optimoaren bideratzailea den hautagaia aukeratuko duen prozedura.

5. **Helburu-Funtzioa**
   Soluzio bati dagokion balioa/kostua itzultzen duena

6. **Osogarria funtzioa**
   Aukeratutako azken hautagaia tarteko emaitzari (soluzio partzialari) eransgarria den ala ez erabakitzen duena.