2. [Az4.18'] MergeSort_K_Heap garatzea eta aztertzea eskatzen zaizu, non tamaina berdineko K zatitan banatuko duen sarrerako taula eta Meta (Heap) egitura nonbait erabiliko duena. Ideia: Izan bitez K taula ordenatuak, guztira n osagai dituztenak. K osagaien arteko txikiena minimoen metak erabiliz muga dezakegu. Taula horien merge' bidezko bateraketa metak erabiliz aztertu.

MergeSort-a garatzea falta da.

MergeSort_K_Heap garatzeko ordenatuak dauden K listen bateraketa inplementatu behar dugu. Ohizko mergesort-an bi lista bakarrik bateratu egiten ziren (merge prozedura), baina oraingo honetan k lista zuzenean ordenatzea bilatzen da.

Ariketa hau metekin egin daiteke. Ideia honakoa da:

1. Minimoaren propietatea betetzen duen meta bat eraiki K listen lehenengo elementua hartuz. Metan ez ditugu zenbakiak soilik sartuko, baizik eta pareak. Ikusiko dugunez, zenbaki bakoitza ze listatik datorren jakin behar dugu, eta beraz, lista zenbakia zenbakiarekin batera gordeko dugu.
2. Metaren erroa beti izango da zenbakirik txikiena, eta beraz, emaitza modura bueltatuko dugun lista ordenatuan sartu behar den hurrengo elementua izango da.
3. Metaren erroan dagoen zenbakia ateratzerakoan, beste elementu bat sartu behar dugu…
   3.1. Erroan zegoen zenbakiaren jatorri lista hartu behar dugu eta elementu gehiago baldin badaude, hurrengo elementua hartu, metaren erroan jarri eta metaren hondoratu prozedurarei deituko diogu.
   3.2. Erroan zegoen zenbakiaren jatorri lista hartzerakoan elementu gehiago ez badaude, gure kasuan, balio oso handi bat sartu (Integer.MAX_VALUE) eta metan hondoratu dugu.

2 eta 3 pausoak behin eta berriz errepikatu behar dira emaitza bektorea (k*n) elementu izan arte.

Kodea:

```
public class Frogak {

	public static void main(String[] args) {
		int[][]b = {
				{0,2,41,52,55,78,92},
				{1,4,14,17,24,30,31},
				{3,6,8,11,12,62,95},
				{9,10,18,27,39,49,100},
				};
		/*int[][]b = {
				{0,2,41},
				{1,4,14},
				{3,6,8},
				};*/
		KBateraketa kb = new KBateraketa();
		System.out.println(Arrays.toString(kb.kBateraketa(b)));
	}

}
```



```
public class KBateraketa {

	public int[] kBateraketa(int [][] listak){
		int[] emaitza = new int[listak.length*listak[0].length];
		int emaitzaInd = 0;
		Bikotea [] metarakoBektorea = new Bikotea[listak.length];
		for(int i=0; i<listak.length; i++){
			Bikotea aux= new Bikotea(listak[i][0],i);
			metarakoBektorea[i]=aux;
			listak[i][0]=1;
		}
		Meta meta = new Meta(metarakoBektorea);
		while (emaitzaInd!=emaitza.length){
			Bikotea minimoa=meta.minimoaItzuli();
			emaitza[emaitzaInd]=minimoa.getZenbakia();
			emaitzaInd++;
			int jatorriLista=minimoa.getJatorriLista();
			if (listak[jatorriLista][0]==listak[0].length){
				meta.erroaAldatuEtaHondoratu(new
Bikotea(Integer.MAX_VALUE,-1));
			}else{
				int elementuBerria =
listak[jatorriLista][listak[jatorriLista][0]];
				listak[jatorriLista][0]++;
				meta.erroaAldatuEtaHondoratu(new
Bikotea(elementuBerria, jatorriLista));
			}
		}
		return emaitza;
	}
}
```

```
public class Bikotea {
	private int zenbakia;
	private int jatorriLista;

	public Bikotea(int z, int jl){
		zenbakia=z;
		jatorriLista=jl;
	}

	public int getZenbakia() {
		return zenbakia;
	}
	public void setZenbakia(int zenbakia) {
		this.zenbakia = zenbakia;
	}
	public int getJatorriLista() {
		return jatorriLista;
	}
	public void setJatorriLista(int jatorriLista) {
		this.jatorriLista = jatorriLista;
	}
}
```

```
import java.util.Arrays;

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

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

	public void hondoratu(int indizea){
		int i=indizea;
		int umeMin;
		Bikotea 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].getZenbakia()<v[2*i+1].getZenbakia()){
				umeMin=2*i+2;
			}
			if(v[i].getZenbakia()>v[umeMin].getZenbakia()){
				aux=v[i];
				v[i]=v[umeMin];
				v[umeMin]=aux;
				i=umeMin;
			}else{
				jarraitu=false;
			}
		}
	}

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



```
	public Bikotea minimoaItzuli(){
		return v[0];
	}

	public void erroaAldatuEtaHondoratu(Bikotea bikoteBerria){
		v[0]=bikoteBerria;
		hondoratu(0);
	}

	public void print(){
		System.out.println(Arrays.toString(v));
	}

}
```

K_Bateraketaren analisia: K lista ordenatu dauzkagu, bakoitza n tamainakoa.

1. Metarako bektorea eraiki lista bakoitzeko lehenengo elementuarekin: O(k)
2. Minimoaren propietatea betetzen duen meta eraiki, hondoratze prozesuarekin: O(k), metak k elementu dituelako.
3. Emaitza moduan bueltatu behar dugun lista k*n elementu izango ditu. Beraz, bukleak k*n iterazio emango ditu, honakoa eginez:
   a. Metaren erroa atera eta emaitza modura bueltatuko dugun listan dagokion posizioan sartu. Denbora konstantean egin daiteke hau (O(1)).
   b. Metaren erroan elementu berri bat jarri behar dugu. Horretarako, atera dugun elementua zein listatik datorren jakin behar dugu. Listara joan eta…
      i. Oraindik elementuak baldin baditu, hurrengo elementua hartu, metako erroan jarri eta hondoratu.
      ii. Elementuak ez baditu (lista hutsa), konputagailuan errepresenta daitekeen zenbaki osoko handiena hartu, metako erroan jarri eta hondoratu.

   Bi kasuetan, listan elementurik dagoen begiratzeak denbora konstantea hartzen du, eta bi kasuetan ere metan hondoratze eragiketa gauzatzen da. Hondoratzea da kostu gehien daukan eragiketa: O(lg k).

➔ Baturaren erregela aplikatuz a. eta b. artean: O(lg k)

Guztira:

k + k + kn*lgk = 2k + kn*lgk = O(kn lgk)

Algoritmoaren kostua O(kn lgk) da.

4. Izan bedi G grafo zuzendu pisuduna, baina pisua erpinei esleituak dituena. Programa bat idatz ezazu grafoko bide luzeena kalkulatzeko jakinik soilik erpin batetik pisu handiagoa duen erpin batera pasa gaitezkeela. Zure algoritmoaren analisia egizu.

Grafo mota: Grafo zuzendua.

Korritze mota: Sakonerako korritzea.

Deskripzioa: Erpin bakoitzean honako informazioa gorde behar dugu: aztertua izan den ala ez, eman ahal diren salto maximo kopurua (bide luzeena erpin horretatik hasita) eta zein den bere semea (korritzerako orduan balioak jarriko dizkiogu parametro honi, eta korritzea bukatu eta gero bidea ateratzeko balio izango du). Gainera, erpin bakoitzak bere auzokideen lista, pisua, eta identifikazio zenbakia izango ditu.

Sakonerako korritzea hasi baino lehen erpin guztiak ez direla aztertu jarri behar dugu, eta salto kopurua eta semea -1 baliora hasieratu. Problema honen gakoa sakonerako korritzeak ematen digun atze-prozesaketan edo post-prozesaketan dago.

Sakonerako korritzean, erpin bakoitzarentzako eta soilik aztertua izan ez bada, honako hau egingo dugu:

1. Aztertua izan dela markatu.
2. Bere auzokideak hartu. Baina hauek tratatzerako orduan bakarrik erpina baino pisu handiagoa dutenak hartu behar ditugu kontuan (besteetaz ahaztu behar gara).
3. Auzokide bakoitzeko, aztertua izan ez bada, erpinaren semea auzokidea bera dela jarriko dugu eta sakonerako korritzearen funtzioari errekurtsiboki deituko diogu.
4. Dei errekurtsiboa bukatzerakoan, atze-prozesaketa dator. Auzokideak (baliteke SEMEA EZ IZATEA) lortu dituen salto kopurua (bide luzeena) erpinak dauzkanak baino gehiago badira, erpinaren seme berria auzokide hori izango da eta erpinaren salto kopuru berria auzokidearen berbera izango da (*).
5. Azkenik, erpinaren auzokide guztiak tratatu ditugunean, erpinaren bide kopuruari bat gehituko diogu (lehen (*) ez dugulako gehitu).

Sakonerako korritzea bukatzerakoan, salto kopuru maximoa lortu duen erpina bilatu behar dugu, erpin hori izango baita bilatutako bidearen abiapuntua. Behin aurkituta, bueltatu behar den lista eraiki dezakegu, erpin bakoitzak korritzean jarri diogun semea gordeta baitauka.

Algoritmoaren analisia:

1. Erpinen hasieraketa (ez aztertua, semea=saltokopurua=-1): O(n)
2. Sakonerako korritzea gauzatzea salto kopuruen zenbaketa eta bidea eraikitzen dugun bitartean. Egiten diren eragiketak (konparazioak, batuketak, erpinen salto kopuruen kontsultak) denbora konstantean egiten dira. Beraz: O(n+a)
3. Salto kopuru maximoa lortu duen erpina bilatu korritzea bukatu denean. Kasu txarrenean: O(n)
4. Bueltatu behar den lista eraiki (bektore batean). Kasu txarrenean bidean erpin guztiek parte hartuko dute... ¿O(n)?

Baturaren erregela aplikatuz: n+n+a+n+n=4n+a=O(n+a)

Kodea:

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

	public Grafoa(Erpina [] erpinak, int [][] auzokideak){
		auzokideZerrenda = new Erpina[erpinak.length];
		for(int i=0; i<erpinak.length; i++){
			erpinak[i].setAuzokideak(auzokideak[i]);
			auzokideZerrenda[i]=erpinak[i];
		}
	}

	public int[] bideLuzeenaAurkitu(){
		int [] bidea;
		for (Erpina e: auzokideZerrenda){
			e.setAztertua(false);
			e.setSaltoKop(-1);
			e.setSemea(-1);
		}
		for(Erpina e: auzokideZerrenda){
			if(!e.aztertua()){
				sk(e);
			}
		}

		Erpina bideHasiera =auzokideZerrenda[0];
		//Salto kopuru maximoa duen erpina bilatu
		for(Erpina e: auzokideZerrenda){
			if (e.getSaltoKop()>bideHasiera.getSaltoKop()){
				bideHasiera=e;
			}
		}
		bidea= new int[bideHasiera.getSaltoKop()+1];
		int i=0;
		while(bideHasiera.getSemea()!=-1){
			bidea[i]=bideHasiera.getZenbakia();
			bideHasiera=auzokideZerrenda[bideHasiera.getSemea()];
			i++;
		}
		bidea[i]=bideHasiera.getZenbakia();
		return bidea;
	}

	public void sk(Erpina e){
		e.setAztertua(true);
		int [] auzokideak = e.getAuzokideak();
		for(int i=0; i<auzokideak.length; i++){
			Erpina auzokidea=auzokideZerrenda[auzokideak[i]];
			if(auzokidea.getPisua()>e.getPisua()){
				if(!auzokidea.aztertua()){
					e.setSemea(auzokidea.getZenbakia());
					sk(auzokidea);
				}
				if(auzokidea.getSaltoKop()>e.getSaltoKop()){
					e.setSaltoKop(auzokidea.getSaltoKop());
					e.setSemea(auzokidea.getZenbakia());
				}
			}
		}
		e.setSaltoKop(e.getSaltoKop()+1);
	}
}
```

```
public class Erpina{
            private int zenbakia;
            private int [] auzokideak;
            private boolean aztertua;
            private int pisua;
            private int saltoKop;
            private int semea;

            public Erpina (int num, int p){
                  zenbakia=num;
                  aztertua=false;
                  pisua=p;
                  saltoKop=0;
                  semea=-1;
            }

            public int getZenbakia(){
                  return zenbakia;
            }

            public void setAztertua (boolean b){
                  aztertua=b;
            }

            public boolean aztertua(){
                  return aztertua;
            }

            public int [] getAuzokideak() {
                  return auzokideak;
            }

            public void setAuzokideak(int [] auzokideak) {
                  this.auzokideak = auzokideak;
            }

            public int getPisua() {
                  return pisua;
            }

            public void setPisua(int pisua) {
                  this.pisua = pisua;
            }

            public int getSaltoKop() {
                  return saltoKop;
            }

            public void setSaltoKop(int saltoKop) {
                  this.saltoKop = saltoKop;
            }

            public int getSemea() {
                  return semea;
            }

            public void setSemea(int semea) {
                  this.semea = semea;
            }
      }
```



```
import java.util.Arrays;


public class Frogak {

      public static void main(String[] args) {
            int [][] auzokideak = {
                                                {1},
                                                {2,4},
                                                {3},
                                                {5},
                                                {2,3},
                                                {}
                                                };
            Erpina [] erpinak = new Erpina[6];
            erpinak[0]= new Erpina(0,3);
            erpinak[1]= new Erpina(1,5);
            erpinak[2]= new Erpina(2,7);
            erpinak[3]= new Erpina(3,8);
            erpinak[4]= new Erpina(4,6);
            erpinak[5]= new Erpina(5,10);
            Grafoa g = new Grafoa(erpinak, auzokideak);

      System.out.println(Arrays.toString(g.bideLuzeenaAurkitu()));
      }

}
```

6. Ariketa 3.16 [Dasgupta et al.] Suposatu GII kurrikulumak N irakasgaiz osaturik dagoela, eta denak derrigorrezkoak direla, gainera eta A eta B irakasgaien arkua badago, A irakasgaia Bren aurrebaldintza dela adieraziko du. Egizu algoritmo bat erabakitzeko gutxienez zenbat lauhilabete beharko dituen GIIko ikasle batek kurrikulumeko irakasgai guztiak egiteko (suposatu ikasleek lauhile batean nahi adina ikasgai egin ditzakeela). Soluzioaren denbora-analisia egizu.

Grafo mota: Grafo zuzendua.

Erabilitako adierazpidea: Auzokideen zerrendak.

Korritze mota: Sakonerako korritzea.

Soluzioaren deskribapena: Problema honen soluzioa ateratzeko nahikoa da klasean emandako ordenazio topologikoa aplikatzearekin. Sakonerako korritzearen eskema berrerabiliz, atze-prozesaketa egiten denean (erpin bakoitzaren prozesaketa amaitzear dagoenean), pila batean sartu behar dugu erpina (edo pilaren antzeko datu egitura batean, burutik txertatzea uzten diguna). Korritzea amaitu eta gero, ikasgaiak pilan gordeta egongo dira (gauzatu behar ditugun ordenean). Zenbat lauhilabete behar ditugun zenbatzeko, pilan dauden ikasgaiak korritu behar ditugu. Ikasgaiak korritzen ditugun bitartean, zenbat “lauhilabete aldaketa” (1etik 2ra edo alderantziz) dauden zenbatu beharko dugu, zenbaki hori izango baita behar ditugun lauhilabete kopurua.

NOTA: Ordenazio topologikoa frogatzeagatik sartu/irten denbora markak jarri dizkiegu erpinei (denbora konstantean egiten dena). Baina problema hau ebazteko ez da behar, eta pila erabiltzearekin nahikoa da.

Algoritmoaren analisia:

1. Erpin guztiak hasieratu (ez dira aztertuak izan): O(n).
2. Sakonerako korritzea gauzatu. Erpin eta arku guztiak behin bisitatuko ditugu. Pilan elementuak sartzea denbora konstantean egiten da. Beraz: O(n+a).
3. Korritzea bukatzerakoan pila guztiz korritu behar dugu: ikasgaiak (erpinak) gauzatu behar diren ordenean daude, eta lauhilabete aldaketak zenbatu nahi ditugu lauhilabete kopurua zenbatzeko. Beste modu batean ikusita, erpin guztiak bisitatu behar ditugu berriz ere. Beraz: O(n).

Baturaren erregela aplikatuz: n+n+a+n=3n+a= O(n+a).

Kodea:

```
import java.util.LinkedList;
public class Grafoa {
	private Erpina [] auzokideZerrenda;

	public Grafoa(Erpina [] erpinak, int [][] auzokideak){
		auzokideZerrenda = new Erpina[erpinak.length];
		for(int i=0; i<erpinak.length; i++){
			erpinak[i].setAuzokideak(auzokideak[i]);
			auzokideZerrenda[i]=erpinak[i];
		}
	}

	public int ikasgaiPlangintza(){
		int lauhilabeteKop=0;
		for (Erpina e: auzokideZerrenda){
			e.setAztertua(false);
			e.setMarka1(0);
			e.setMarka2(0);
		}
		int denboraMarka=0;
		LinkedList<Erpina> zerrenda = new LinkedList<Erpina>();
		for(Erpina e: auzokideZerrenda){
			if(!e.aztertua()){
				denboraMarka=sk(e, zerrenda, denboraMarka);
			}
		}
		int lauhilabetea=0;
		while (zerrenda.size()!=0){
			Erpina e=zerrenda.removeFirst();
			if(e.getLauhilabetea()!=lauhilabetea){
				lauhilabetea=e.getLauhilabetea();
				lauhilabeteKop++;
			}
		}
//		int [] v=new int[zerrenda.size()];
//		for(int i=0; i<zerrenda.size(); i++){
//			v[i]=zerrenda.get(i).getZenbakia();
//		}
//		return v;
		return lauhilabeteKop;
	}

	public int sk(Erpina e, LinkedList<Erpina> zerrenda, int marka){
		e.setAztertua(true);
		e.setMarka1(marka);
		marka=marka+1;
		int [] auzokideak = e.getAuzokideak();
		for(int i=0; i<auzokideak.length; i++){
			Erpina auzokidea=auzokideZerrenda[auzokideak[i]];
			if(!auzokidea.aztertua()){
				marka=sk(auzokidea, zerrenda, marka);
			}
		}
		e.setMarka2(marka);
		zerrenda.addFirst(e);
		marka=marka+1;
		return marka;
	}
}
```

```
public class Erpina{
		private int zenbakia;
		private int [] auzokideak;
		private boolean aztertua;
		private int kurtsoa;
		private int marka1;
		private int marka2;
		private int lauhilabetea;


		public Erpina (int num, int k, int l){
			zenbakia=num;
			aztertua=false;
			kurtsoa=k;
			lauhilabetea=l;
			marka1=0;
			marka2=0;
		}

		public int getZenbakia(){
			return zenbakia;
		}

		public void setAztertua (boolean b){
			aztertua=b;
		}

		public boolean aztertua(){
			return aztertua;
		}

		public int [] getAuzokideak() {
			return auzokideak;
		}

		public void setAuzokideak(int [] auzokideak) {
			this.auzokideak = auzokideak;
		}

		public int getKurtsoa() {
			return kurtsoa;
		}

		public void setKurtsoa(int kurtsoa) {
			this.kurtsoa = kurtsoa;
		}

		public int getLauhilabetea() {
			return lauhilabetea;
		}

		public void setLauhilabetea(int lauhilabetea) {
			this.lauhilabetea = lauhilabetea;
		}

		public int getMarka1() {
			return marka1;
		}

		public void setMarka1(int marka1) {
			this.marka1 = marka1;
		}

		public int getMarka2() {
			return marka2;
		}

		public void setMarka2(int marka2) {
			this.marka2 = marka2;
		}
	}
```



```
public class Frogak {

	public static void main(String[] args) {
		int [][] auzokideak = {
							{5},
							{5},
							{4},
							{9},
							{12},
							{10},
							{8},
							{8},
							{12},
							{},
							{13},
							{14},
							{},
							{},
							{},
							{}
							};
		Erpina [] erpinak = new Erpina[16];
		erpinak[0]= new Erpina(0,1,1);
		erpinak[1]= new Erpina(1,1,1);
		erpinak[2]= new Erpina(2,1,2);
		erpinak[3]= new Erpina(3,1,2);
		erpinak[4]= new Erpina(4,2,1);
		erpinak[5]= new Erpina(5,2,1);
		erpinak[6]= new Erpina(6,2,2);
		erpinak[7]= new Erpina(7,2,2);
		erpinak[8]= new Erpina(8,3,1);
		erpinak[9]= new Erpina(9,3,1);
		erpinak[10]= new Erpina(10,3,2);
		erpinak[11]= new Erpina(11,3,2);
		erpinak[12]= new Erpina(12,4,1);
		erpinak[13]= new Erpina(13,4,1);
		erpinak[14]= new Erpina(14,4,2);
		erpinak[15]= new Erpina(15,4,2);
		Grafoa g = new Grafoa(erpinak, auzokideak);
		System.out.println(g.ikasgaiPlangintza());
	}

}
```

8. [az3.21] Zuhaitz baten diametroa edozein erpin bikote arteko distantzia guztietatik distantzia handiena da. Zuhaitz baten diametroa kalkulatzen duen algoritmo lineal bat eraikitzea eskatzen da.

Grafo mota: Grafo ez-zuzendua.

Erabilitako adierazpidea: Auzokideen zerrendak.

Korritze mota: Sakonerako korritzea.

Soluzioaren deskribapena: Sakonerako korritzea zuhaitzaren errotik hasiko dugu. Sakonerako korritzeak daukan atze- edo post-prozesaketa egitean, erpin bakoitzeko, zein den eman dezakegun salto kopuru maximoa kalkulatuko dugu (hau da, erpin horretatik hasia zein den biderik luzeena). Grafoa ez-zuzendua denez, prozesaketa egiterakoan honako hiru egoerak desberdindu ditugu:

- Erpinak auzokide bakarra badauka ez dugu ezer berezirik egingo (sakonerako korritzearekin jarraitu).
- Erpinak bi auzokide baditu, erpinetik hasia eman daitezkeen salto kopuru maximoa honakoa izango da: erpinak dagoeneko dauzkan saltoak + auzokideak dauzkan salto kopurua +1.
- Erpinak hiru edo auzokide gehiago baditu, auzokideen arteko salto kopuru maximoarekin gelditu beharko gara (eta salto kopuru horri bat gehitu).

Esan den guztia auzokidea aztertua izan ez bada egin behar da, eta dei errekurtsiboa bukatu eta ondoren:

[….] if(!auzokidea.aztertua()){

Sk(auzokidea);

/*Hemen egin*/

} […]

Algoritmoaren analisia:

1. Erpin guztiak hasieratu: Salto kopurua zerora jarri eta aztertuak ez direla izan jarri. Ordena: O(n).
2. Sakonerako korritzea gauzatu. Erpin bakoitzetik eman daitezkeen salto kopuru maximoa kalkulatzeko egin behar diren eragiketak denbora konstantean egiten dira (konparazioak, erpinak aztertuak izan diren ala ez kontsultatu/aldatu, zenbaki osokoen batuketa, …). Beraz: O(n+a).

Baturaren erregela aplikatuz: n+n+a=2n+a= O(n+a).

Kodea:

```
public class Frogak {
	public static void main(String[] args) {
		int [][] auzokideak = {{1,2},{0,3,4,5},{0,6},{1},{1,7,8},{1,10},
		{2},{4},{4,9},{8},{5}};
		Grafoa g = new Grafoa(11,auzokideak);
		System.out.println("Zuhaitzaren diametroa:
"+g.zuhaitzarenDiametroa());
	}
}
```



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

	public Grafoa(int erpinKop, int [][] auzokideak){
		auzokideZerrenda = new Erpina[erpinKop];
		for(int i=0; i<erpinKop; i++){
			Erpina e= new Erpina(i);
			e.setAuzokideak(auzokideak[i]);
			auzokideZerrenda[i]=e;
		}
	}

	public int zuhaitzarenDiametroa(){
		int diametroa=0;
		for(Erpina e:auzokideZerrenda){
			e.setAztertua(false);
			e.setSaltoKop(0);
		}
		Erpina erroa = auzokideZerrenda[0];
		sk(erroa);
//		for(Erpina e:auzokideZerrenda){
//			System.out.println("Erpina: "+e.getZenbakia()+"  saltoak:
"+e.getSaltoKop());
//		}
		diametroa=erroa.getSaltoKop();
		return diametroa;
	}

	public void sk(Erpina e){
		e.setAztertua(true);
		int[] auzokideak = e.getAuzokideak();
		for(int i=0; i<auzokideak.length; i++){
			Erpina auzokidea= auzokideZerrenda[auzokideak[i]];
			if(!auzokidea.aztertua()){
				sk(auzokidea);
				if(auzokideak.length==2){

	e.setSaltoKop(e.getSaltoKop()+auzokidea.getSaltoKop()+1);
				}else if(auzokideak.length>=3){

	if(e.getSaltoKop()<auzokidea.getSaltoKop()+1){

	e.setSaltoKop(auzokidea.getSaltoKop()+1);
					}
				}
			}
		}
	}
}
```

ALGORITMOEN DISEINUA – 2. MINTEGIA [C atala]

```
public class Erpina{
            private int zenbakia;
            private int [] auzokideak;
            private boolean aztertua;
            private int saltoKop;

            public Erpina (int num){
                  zenbakia=num;
                  aztertua=false;
                  saltoKop=0;
            }

            public int getZenbakia(){
                  return zenbakia;
            }

            public void setAztertua (boolean b){
                  aztertua=b;
            }

            public boolean aztertua(){
                  return aztertua;
            }

            public int [] getAuzokideak() {
                  return auzokideak;
            }

            public void setAuzokideak(int [] auzokideak) {
                  this.auzokideak = auzokideak;
            }

            public int getSaltoKop() {
                  return saltoKop;
            }

            public void setSaltoKop(int saltoKop) {
                  this.saltoKop = saltoKop;
            }

      }
```

9. Demagun abizen jakin bateko zuhaitza genealogikoa dugula, erroa ezaguna izanik (hau da, abizen hori eraman zuen lehenengo arbasoa). Helburua sendiko edozein bi ahaideren arteko hurbilen duten lehengo arbaso komuna finkatuko duen prozedura egitea da. Proposatutako soluzioaren denbora analisia eta ordena kalkula itzazu.

Grafo mota: Grafo zuzendua.

Erabilitako adierazpidea: Auzokideen zerrendak.

Korritze mota: Sakonerako korritzea.

Soluzioaren deskribapena: Grafo ariketetan, normalean, erpin bakoitzak aztertua izan den ala ez adierazten duen marka dauka bakarrik. Problema hau ebazteko, ordea, ohikoa den markaz gain beste marka bat erabili dugu. Marka horri *marka* deitu diogu eta hasiera batean pasa dizkiguten bi ahaideek izango dute soilik.

Gakoa atze- edo post-prozesaketan dago. Hura gauzatzean, *marka* marka bi ahaideetatik bere arbasoetarantz hedatuz joango da. Kasu hauek bereizten dira:

- Erpinaren auzokide batek (seme batek) *marka* jarrita badauka eta erpinak *marka* ez badauka → Erpinari *marka* jarri.
- Erpinaren auzokide batek (seme batek) *marka* jarrita badauka eta erpinak dagoeneko *marka* badauka → Erpina bilatutako arbaso komuna da.
- Beste kasuetan ez da ezer berezirik egin behar (sakonerako korritzearekin jarraitu).

Dakigunez, sakonerako korritzea errekurtsiboki egiten da. Hori "arazoa" bilakatzen da arbaso komuna identifikatzen dugunean, erantzun moduan bueltatu eta bere bilaketa nolabait geldiarazi nahi dugulako… Baina erditik errekurtsio pila daukagu. Horretarako, kodean, *aurkitua* aldagaia erabili dugu.

Algoritmoaren analisia:

1. Erpin guztiak hasieratu: ez dira aztertuak izan eta ez daukate *marka* marka. Beraz: O(n).
2. Bi ahaideei *marka* marka jarri. Denbora konstantea behar da: O(1).
3. Sakonerako korritzea gauzatu. Gehienez erpin eta arku guztiak bisitatuko dira. Atze-prozesaketan egiten diren eragiketak (erpin batek *marka* marka duen ala ez begiratu, erpin bati marka *marka* jarri, …) denbora konstantean egiten dira. Beraz: O(n+a).

Baturaren erregela aplikatuz: n+1+n+a=2n+a+1=O(n+a)

Kodea:

```
public class Frogak {
	public static void main(String[] args) {
		int [][] auzokideak = {
{1,2},
{3,4},
{},
{},
{5,6},
{},
{}
};
		Grafoa g = new Grafoa(7, auzokideak);
		System.out.println(g.biAhaideenArtekoHubirlenArbasoa(2, 5));
	}
}
```



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

	public Grafoa(int erpinKop, int [][] auzokideak){
		auzokideZerrenda = new Erpina[erpinKop];
		for(int i=0; i<erpinKop; i++){
			Erpina e= new Erpina(i);
			e.setAuzokideak(auzokideak[i]);
			auzokideZerrenda[i]=e;
		}
	}

	public int biAhaideenArtekoHubirlenArbasoa(int a1, int a2){
		for (Erpina e: auzokideZerrenda){
			e.setAztertua(false);
			e.setMarka(false);
		}
		Erpina ahaide1=auzokideZerrenda[a1];
		ahaide1.setMarka(true);
		Erpina ahaide2=auzokideZerrenda[a2];
		ahaide2.setMarka(true);
		//Sakonerako korritzea zuhaitzaren errotik hasi
		return sk(auzokideZerrenda[0]);
	}

	public int sk(Erpina e){
		int aurkitua=-1;
		e.setAztertua(true);
		int [] auzokideak = e.getAuzokideak();
		for(int i=0; i<auzokideak.length && aurkitua==-1; i++){
			Erpina auzokidea=auzokideZerrenda[auzokideak[i]];
			if(!auzokidea.aztertua()){
				aurkitua= sk(auzokidea);
			}
			if(auzokidea.markatuta() && aurkitua==-1){
				if(!e.markatuta()){
					e.setMarka(true);
				}else{
					aurkitua= e.getZenbakia();
				}
			}
		}
		return aurkitua;
	}
}
```

ALGORITMOEN DISEINUA – 2. MINTEGIA [C atala]

```
public class Erpina{
            private int zenbakia;
            private int [] auzokideak;
            private boolean aztertua;
            private boolean marka;

            public Erpina (int num){
                  zenbakia=num;
                  aztertua=false;
                  marka=false;
            }

            public int getZenbakia(){
                  return zenbakia;
            }

            public void setAztertua (boolean b){
                  aztertua=b;
            }

            public boolean aztertua(){
                  return aztertua;
            }

            public int [] getAuzokideak() {
                  return auzokideak;
            }

            public void setAuzokideak(int [] auzokideak) {
                  this.auzokideak = auzokideak;
            }

            public boolean markatuta() {
                  return marka;
            }

            public void setMarka(boolean marka) {
                  this.marka = marka;
            }
      }
```

10. Gr=(N, E) grafo zuzendu pisudunak komunikazio sare bat adierazten du. (u,v)∈E arkuaren pisua f(u,v) da, bere balioa 0≤f(u,v)≤1 tartean egonik, u-tik v-raino komunikazio kanalaren fidagarritasuna adierazten du, hots, u-v kanalaren ez hutsegitearen probabilitatea. Honela, f(u,v)=0 denean, kanala ez dela batere fidagarria adieraziko du, eta aldiz, f(u,v)=1 denean, komunikazio kanala erabat fidagarria dela adieraziko du. Demagun Xetik Yra doan kanalaren fidagarritasunaren huts egitearen probabilitatea kanala osatzen duten zatien probabilitateen biderkadura dela. Adibidez, [x,y] bideak (x,z) eta (z,y) bi kanalak zeharkatzen dituen bidea bada, [x,y]=(x,z)(z,y), orduan f[x,y]= f(x,z)*f(z,y).
A eta B erpinen arteko bide fidagarriena balioa kalkulatzeko algoritmo eraginkor bat egizu eta hartuko duen denbora kostua aztertu.

Grafo mota: Grafo zuzendu pisuduna.

Erabilitako adierazpidea: Auzokideen zerrendak. Erpin bakoitzak bere arku zerrenda dauka.

Korritze mota: Sakonerako korritzea.

Soluzioaren deskribapena:

Erpin bakoitzak honako atributuak/aldagaiak izango ditu bere barne:

- Zenbakia: Erpinak daukan zenbakia.
- Aztertua: Aztertua izan den jakitzeko.
- Arku zerrenda: Bere auzokideengana heldu ahal izateko.
- bideanDago: Erpina Atik Bra doan kanal baten barne dagoen ala ez jakiteko.
- Probabilitatea: Hasieran 0 izango da, baina erpina Atik Bra doan kanal baten parte bada (bideanDago), probabilitate bat izango du.

Prozedura: Ikusiko dugunez, gakoa atze- edo post-prozesaketan dago.

1. Erpin guztiak hasieratu behar ditugu (aztertua=false; probabilitatea=0.0 eta bideanDago=false).
2. Kanala Atik Bra doanez, B erpina hartu eta aztertua izan dela jarriko dugu, probabilitate moduan 1 jarriko diogu (Btik Bra joatea tribiala da) eta bidearen parte dela jarriko diogu (bideanDago=true).
3. A erpina abiapuntutzat hartuz, sakonerako korritzea gauzatuko dugu. Bukatzerakoan, kanalaren fidagarritasuna (biderik hoberena hartuta) A erpinean gordeta egongo da. Dagoeneko egin duguna kontuan hartuz, egin beharrekoa honakoa da:
   3.1. Sakonerako korritzean erpina (A hasieran) aztertua izan dela markatu.
   3.2. Erpinaren auzokideak hartu. Auzokide bakoitzeko:
      3.2.1. Auzokidea aztertu gabe badago, sakonerako korritzea gauzatu berarekin.
      3.2.2. Auzokidea bide batean sartuta badago (bideanDago, hau gertatuko den lehenengo aldia B aurkitzen dugunean izango da) eta erpina bide batean sartuta ez badago, erpina bidean dagoela markatuko dugu eta probabilitate bat esleituko diogu. Probabilitate hori honakoa izango da: Auzokideak daukan probabilitatea * Auzokide horretara eraman gaituen arkuaren fidagarritasuna.
      3.2.3. Auzokidea bidean badago eta erpina ere bidean badago (bideanDago=true), horrek esan nahi du erpinetik gutxienez B erpinera bi bide daudela. Beraz, bide fidagarrienarekin geldituko gara. Horretarako honakoa egingo dugu:

ALGORITMOEN DISEINUA – 2. MINTEGIA [C atala]

Erpinaren probabilitatea= max{erpinak daukan probabilitatea, auzokideak daukan probabilitatea * auzokide horretara eraman gaituen arkuaren fidagarritasuna}.

      3.2.4. Auzokidea eta erpina edozein bidetik kanpo badaude ez da ezer berezirik egin behar (sakonerako korritzearekin normal jarraitu).

Algoritmoaren analisia:

1. Erpin guztiak hasieratu (aurreko ataleko lehengo puntua). Erpin bakoitzak dituen aldagaiak hasieratzea denbora konstantean egiten da, eta beraz, guztientzat (n-rentzat) egiten bada: O(n).
2. B erpina hartu eta aztertua izan dela jarri. Bere probabilitatea 1 dela eta bidean dagoela jarriko dugu ere. Denbora konstantea behar du honek: O(1).
3. Sakonerako korritzea gauzatu A erpinetik. Sakonerako korritzean egiten ditugun eragiketak denbora konstantean egiten dira: markak jarri (aztertua edo bideanDago), markak kontsultatu, konparazioak (maximoa atera adibidez), biderketak, ... Guztia O(n+a) denboran egiten da.

Baturaren erregela aplikatuz: n+1+n+a=2n+a+1= O(n+a).

Kodea:

```
public class Arkua {
	private int erpinZnbk;
	private double fidagarritasuna;

	public Arkua(int erpina, double fid) {
		this.erpinZnbk = erpina; this.fidagarritasuna = fid;
	}

	public int getErpinZenbakia() {return erpinZnbk;}
	public void setErpinZenbakia(int hiria){this.erpinZnbk = hiria;}
	public double getFidagarritasuna() {return fidagarritasuna;}
	public void setFidagarritasuna(double fidagarritasuna) {
		this.fidagarritasuna = fidagarritasuna;
	}
}
```

```
public class Erpina{
		private int zenbakia;
		private Arkua[] auzokideak;
		private boolean aztertua;
		private boolean bideanDago;
		private double probabilitatea;

		public Erpina (int num){
			zenbakia=num;
			aztertua=false;
			probabilitatea=0.0;
			bideanDago=false;
		}


	// GETTERRAK eta SETTERRAK
```

```
public class Grafoa {

	private Erpina [] auzokideZerrenda;

	public Grafoa (int erpinKop, Arkua [][] auzokideak){
		auzokideZerrenda = new Erpina [erpinKop];
		for(int i=0; i<erpinKop; i++){
			Erpina e = new Erpina(i);
			e.setAuzokideak(auzokideak[i]);
			auzokideZerrenda[i]=e;
		}
	}

	public double kanalarenFidegarritasuna(int e1, int e2){
		for(Erpina e: auzokideZerrenda){
			e.setAztertua(false);
			e.setProbabilitatea(0);
			e.setBideanDago(false);
		}
		Erpina erpina2 = auzokideZerrenda[e2];
		erpina2.setBideanDago(true);
		erpina2.setAztertua(true);
		erpina2.setProbabilitatea(1);
		Erpina erpina1 = auzokideZerrenda[e1];
		sk(erpina1);
		return erpina1.getProbabilitatea();
	}

	public void sk(Erpina e){
		e.setAztertua(true);
		Arkua [] arkuak = e.getAuzokideak();
		for(Arkua a: arkuak){
			Erpina auzokidea = auzokideZerrenda[a.getErpinZenbakia()];
			if(!auzokidea.aztertua()){
				sk(auzokidea);
			}
			if(!e.bideanDago() && auzokidea.bideanDago()){

	e.setProbabilitatea(a.getFidagarritasuna()*auzokidea.getProbabilitatea()
);
				e.setBideanDago(true);
			}else if(e.bideanDago() && auzokidea.bideanDago()){

	e.setProbabilitatea(Math.max(e.getProbabilitatea(),a.getFidagarritasuna(
)*auzokidea.getProbabilitatea()));
			}
		}
	}

}
```



```
public class Frogak {

	public static void main(String[] args) {
		Arkua[][] auzokideak = new Arkua[6][];
		//0 Erpina
		Arkua [] e0=new Arkua[2];
		e0[0]=(new Arkua(2,0.8));
		e0[1]=(new Arkua(5,0.1));
		auzokideak[0]=e0;
		//1 Erpina
		Arkua [] e1=new Arkua[3];
		e1[0]=(new Arkua(3,0.5));
		e1[1]=(new Arkua(4,1));
		e1[2]=(new Arkua(5,0.7));
		auzokideak[1]=e1;
		//2 Erpina
		Arkua [] e2=new Arkua[0];
		auzokideak[2]=e2;
		//3 Erpina
		Arkua [] e3=new Arkua[1];
		e3[0]=(new Arkua(2,0.9));
		auzokideak[3]=e3;
		//4 Erpina
		Arkua [] e4=new Arkua[1];
		e4[0]=(new Arkua(0,0.6));
		auzokideak[4]=e4;
		//5 Erpina
		Arkua [] e5=new Arkua[1];
		e5[0]=(new Arkua(2,0.4));
		auzokideak[5]=e5;

		Grafoa gp = new Grafoa(6, auzokideak);
		System.out.println(gp.kanalarenFidegarritasuna(1, 2));
	}
}
```

11. Hiri handi bateko metro planoan N estazio ditugu. Metroko estazio bakoitzak estaldura zenbaki osoko bat erlazionatua du, estazio horrek zenbat biztanleei/bezeroei eman diezaiokeen zerbitzua adierazten duena. E estazioa emanik eta X osokoa (X<N) algoritmo bat egizu kalkula dezan E estaziotik eta (gehienez) X geltoki bitarteko estazio guztien estalduren balioen batura. Algoritmoaren exekuzio denbora aztertu.

Grafo mota: Grafo ez-zuzendua.

Erabilitako adierazpidea: Auzokideen zerrendak.

Korritze mota: Sakonerako korritzea.

Soluzioaren deskribapena: Grafoaren sakonerako korritze bat egin dugu E estaziotik hasita. Sakonerako korritzeak duen eskema edo egitura errekurtsiboa ondo etorri zaigu pauso bakoitzean ematen diguten X balioa unitate batean murrizteko eta estazioaren auzokideekin dei errekurtsiboa egiteko. Hau da:

$$sk(E, x) = ErenEstaldura + sk(ErenEstazioAuzokideak, x-1)$$

Auzokideak tratatzerakoan kontuz ibili behar gara, aztertuak izan diren estazioen estaldurak berriz ez gehitzeko.

Algoritmoaren analisia:

1. Erpin (edo estazio) guztiak hasieratu behar ditugu (aztertuak ez direla izan jarri). Ordena: O(n).
2. Sakonerako korritzea gauzatu. Estalduren batura kalkulatzeko egiten diren eragiketak (hainbat konparazio, batuketa, kenketa, erpinei aztertu marka jarri/kontsultatu, ...) denbora konstantean egiten dira. Gainera, X balioaren arabera baliteke grafoko erpin guztiak ez bisitatzea. Halaber, ordena O(n+a) dela esan dezakegu.

Baturaren erregela aplikatuz: n+n+a=2n+a= O(n+a)

Kodea:

```
public class Frogak {
	public static void main(String[] args) {
		Erpina [] erpinak = new Erpina[6];
		//Erpina 0
		erpinak[0]= new Erpina(0,350);
		//Erpina 1
		erpinak[1]= new Erpina(1,520);
		//Erpina 2
		erpinak[2]= new Erpina(2,300);
		//Erpina 3
		erpinak[3]= new Erpina(3,420);
		//Erpina 4
		erpinak[4]= new Erpina(4,490);
		//Erpina 5
		erpinak[5]= new Erpina(5,1000);
		int [][] auzokideak = {
					{2,4,5},{2,5},{0,1},
					{4,5},{0,3},{0,1,3}
					};
		Grafoa g = new Grafoa(erpinak,auzokideak);
		System.out.println(g.estalduraBalioenBatura(4, 2));
	}
}
```



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

	public Grafoa(Erpina[] erpinak, int [][] auzokideak){
		auzokideZerrenda = new Erpina[erpinak.length];
		for(int i=0; i<erpinak.length; i++){
			erpinak[i].setAuzokideak(auzokideak[i]);
			auzokideZerrenda[i]=erpinak[i];
		}
	}

	public int estalduraBalioenBatura(int e, int x){
		for(Erpina er: auzokideZerrenda){
			er.setAztertua(false);
		}
		Erpina estazioa = auzokideZerrenda[e];
		return sk(estazioa,x);
	}

	public int sk(Erpina estazioa, int x){
		estazioa.setAztertua(true);
		int emaitza = estazioa.getEstaldura();
		if(x==0){
			return emaitza;
		}else{
			int [] auzokideak = estazioa.getAuzokideak();
			for(int i=0; i<auzokideak.length; i++){
				Erpina auzokidea =
auzokideZerrenda[auzokideak[i]];
				if(!auzokidea.aztertua()){
					emaitza= emaitza + sk(auzokidea,x-1);
				}
			}
		}
		return emaitza;
	}
}
```

```
public class Erpina{
            private int zenbakia;
            private int [] auzokideak;
            private boolean aztertua;
            private int estaldura;

            public Erpina (int num, int est){
                  zenbakia=num;
                  aztertua=false;
                  estaldura=est;
            }

            public int getZenbakia(){
                  return zenbakia;
            }

            public void setAztertua (boolean b){
                  aztertua=b;
            }

            public boolean aztertua(){
                  return aztertua;
            }

            public int [] getAuzokideak() {
                  return auzokideak;
            }

            public void setAuzokideak(int [] auzokideak) {
                  this.auzokideak = auzokideak;
            }

            public int getEstaldura() {
                  return estaldura;
            }

            public void setEstaldura(int estaldura) {
                  this.estaldura = estaldura;
            }
      }
```