PROGRAMAZIO DINAMIKOA VS BACKTRACK

- 2 INPLEMENTAZIO ARIKETAK -

ALGORITMOEN DISEINUA

2017-2018

Egilea:

Iker Otxoa De Latorre

20/04/2018

AURKIBIDEA

0. SARRERA

Txosten honetan *Algoritmoen Diseinua* ikasgaiko bigarren inplementazioko ariketei dagozkien azalpenak emango ditugu.

Gure kasuan, egokitu zaigun ariketa bidai agentziarena da. Enuntziatua honakoa da:

> *(1) Bidai agentzia batek Antartidara pakete turistiko berezi bat eskaini du. Aurten hara joateko, agentziak hegaldi bakarra antolatzeko gai da izan da, 80 (edo P) plaza izango dituena. Eskaintza argitaratu eta denbora gutxi barru, jaso diren eskaera kopuruak hegaldiko P plazen kopurua gainditu duela eta, bidai agentziak plaza eskatzaileen artean hautaketa bat egingo du. Negozioa negozio dela, etekin maximoa emango dioten bidaiariak aukeratuko ditu "enkante" tankerako prozesu bat erabiliz. Horretarako, plaza eskatu duen pertsona bakoitzari bi datu eskatu dizkiote: (1) plaza bat lortzeko zenbat diru ordaintzeko prest legoke, eta (2) gehienez zenbat kg. ekipaia eramango lituzkeen. Agentziak badaki hegazkinak gehienez T kilo garraia ditzakela eta bere helburua bidai-antolaketari etekin maximoa ateratzea da. Hori horrela, kalkulatu: (1) zein izango da agentziak lortu ahal izango duen etekin maximoa, eta (2) horretarako zeintzuk bidaiari aukeratu beharko lituzkeen.*

Egituraren aldetik, txostena bi zati desberdinetan banatuko dugu:

- Azterketa analitikoa:
  Atal honetan garatutako inplementazioak "teorikoki" aztertuko ditugu. Lehenik eta behin programazio dinamikoko teknika bidez garatutako soluzioa aurkeztuko dugu: erabilitako ekuazio sistema, metatze-egitura, kodea eta bere analisia.
  Ondoren backtrack bidez eraikitako soluzioak azalduko ditugu: esplorazio zuhaitza, soluzioen irudikapena, erabilitako aldagaiak/parametrizazioa, identifikatutako kimak eta kodea. Teknika honekin bi soluzio proposatuko ditugu, bata 0/1 deituriko zuhaitzak erabiliz, eta bestea 1/N zuhaitzak erabiliz.

- Azterketa esperimentala:
  Beste atal honetan aurrekoan azaldutako proposamenak praktikan jarriko ditugu, garatutako kodea sarrera datu desberdinekin exekutatuz. Exekuzio bakoitzeko erabilitako denbora neurtu eta lehen egindako azterketa analitikoarekin konparatuko dugu, bat datorren ala ez ikusteko. Amaitzeko, frogak eginez atera ditugun ondorioak aipatuko ditugu.

Inplementazioak egiteko *Java* programazio lengoaia erabili da, eta hortaz, txosten honetan agertzen diren kode zatiak lengoaia horretan idatziak egongo dira.

1. AZTERKETA ANALITIKOA

Jarraian programazio dinamiko eta backtrack bidezko soluzioak analitikoki aztertuko ditugu.

1.1. PROGRAMAZIO DINAMIKOKO (PD) SOLUZIOA

1.1.1. Ekuazioak

Proposatutako ariketa PD teknika bidez ebazteko honako ekuazioak erabili dira:

$\mathrm{EtekinMaxB(t,p,e)}$: Bidai agentziak bidai-antolaketarekin lor dezakeen etekin maximoa, jakinez:

- Aukeratzeko (1..e) bidaiari daudela,
- Guztira p plaza daudela eta
- Hegazkinak, gehienez, t kilo garraia ditzakela

Kasu nabariak:

[1KN] $\mathrm{EtekinMaxB}(0,\mathrm{p},\mathrm{e}) = 0$ (Hegazkinak 0 kg garraia ditzake)
[2KN] $\mathrm{EtekinMaxB}(\mathrm{t},0,\mathrm{e}) = 0$ (Plazarik ez dago)
[3KN] $\mathrm{EtekinMaxB}(\mathrm{t},\mathrm{p},0) = 0$ (Bidaiaririk ez dago)

Suposaketa:

Eskaitzaile bakoitzak ekipaia eramango du, hau da:

$$\forall\, i\ ekipaiKG[i] > 0$$

Kasu orokorra:

$$EtekinMaxB(t,p,e) = \begin{cases} Max\{ordaindu(e) + EtekinMaxB(t - ekipaiKG(e), p-1, e-1), EtekinMaxB(t,p,e-1)\} \\ \qquad baldin \quad ekipaiKG(e) \le T \ \wedge p \ge 1 \ \wedge e \ge 1 \\ EtekinMaxB(t,p,e-1) \quad baldin \quad ekipaiKG(e) > t \ \wedge p \ge 1 \ \wedge e \ge 1 \end{cases}$$

Non:

- $\mathrm{ordaindu(e)}$: e bezeroa/eskatzailea bidaian plaza bat lortzeko zenbat diru ordaintzeko prest dagoen.
- $\mathrm{ekipaiKG(e)}$: Gehienez zenbat kg ekipaia eramango dituen bezeroak/eskatzaileak bidaiara.

1.1.2. Metatze-egitura

Metatze-egitura bezala $(\mathrm{T}+1)\mathrm{x}(\mathrm{P}+1)\mathrm{x}(\mathrm{N}+1)$ dimentsioko kuboa erabili beharko da, proposatutako ekuazioak hiru parametro baititu. Bere izena kodean *MeE* da, eta beraz, erabilitako espazio estraren aldetik:

$$\mathrm{MeE} \in \mathrm{O(TxPxN)}$$

Errekuperazioa egiteko, beste datu egitura bat erabili behar izan dugu, aurreko *MeE* bezalakoa dena. Kasu honetan *MeErrek* deitu diogu.

1.1.3. Algoritmoa

- *Agentziak lor dezaken irabazi maximoaren kalkulua*
  Irabazi maximoa kalkulatzeko memoriadun funtzioak erabili dira, eta hortaz, kodeketa errekurtsiboa da. Iteratiboki egin genezaken ere, baina metatze-egitura kubo bat izanda errazagoa iruditu zaigu gelaxkak errekurtsio bidez betetzea.

  Lehenik eta behin metatze-egitura kasu nabariekin hasieratu dugu. Hasieraketa hauek iteratiboki egin ditugu, hau da:

$$\forall p \forall e \quad \text{EtekinMaxB}(0, \text{p}, \text{e}) = 0$$
$$\forall t \forall e \quad \text{EtekinMaxB}(\text{t}, 0, \text{e}) = 0$$
$$\forall t \forall p \quad \text{EtekinMaxB}(\text{t}, \text{p}, 0) = 0$$

  Ondoren, iteratiboki ere, gainontzeko gelaxkak -1 balioarekin bete ditugu, kasu orokorrean zein gelaxka kalkulatu ez diren jakiteko.

  Kasu orokorra kodetzeko klasean ikusitako egitura jarraitu dugu, $bete(t, p, e)$ funtzioa definituz. Honen inplementazioa ateratako ekuazioek diotena jarraituz egin behar da. Sartu behar den gauza berri bakarra errekuperaziorako arrastoa da. Kasu honetan erabilitako arrastoa 1 zenbakia izan da, *MeErrek* egituran gorde dena "**eskaitzaile bat onartzearen**" kasua eman den bakoitzean:

$$Max\{\mathbf{ordaindu}(\mathbf{e}) + \mathbf{EtekinMaxB}(\mathbf{t} - \mathbf{ekipaiKG}(\mathbf{e}), \mathbf{p} - \mathbf{1}, \mathbf{e} - \mathbf{1}), EtekinMaxB(t, p, e - 1)\}$$

- *Errekuperazioa*
  Errekuperazioa iteratiboki kodetu dugu, eta lineala atera zaigu eskatzaile kopuruan. Ideia klasean ikusitako motxilarena bezalakoa da:
  i. Agentziak eskaintzen dituen plaza kopurua aldagai batean gorde (demagun `Plaza`)
  ii. Hegazkinak gehienez garraia ditzaken kiloak aldagai batean gorde (demagun `Kilo`)
  iii. Agentziak jaso dituen eskaera kopurua aldagai batean gorde (demagun `eskatzaile`)
  iv. Begizta: `MeErrek` egiturako `MeErrek[Kilo][Plaza][eskatzaile]` gelaxkatik hasi (soluzio gelaxkaren koordenatuak) eta bira bakoitzeko …
    1. 1 balioa badauka, momentuko eskatzailea onartu dela esan nahi du. Beraz, plaza kopurua eta ekipaietarako pisua murriztu egin behar da: `Plaza - -` eta `Kilo = Kilo - ekipaiKG[eskatzaile]`
    2. Beste balio bat izatekotan (0), ez egin ezer.

3. Hurrengo biran aurretik dagoen eskatzailearen egoera aztertuko da: `eskatzailea - -`

Kodea:

```
/* Problemaren aldagaiak [GLOBALAK] */
		/* Metatze-egitura */
	private int[][][] MeE;
		/* Errekuperaziorako erabiliko den egitura */
	private int[][][] MeErrek;

	public int agentziakLorDezakeenEtekinMaxPD(String outFitxategiIzena){
		/* Metatze-egitura hasieratu */
		MeE = new int[T+1][P+1][E+1];
		for(int t=0; t<=T; t++){
			for(int p=0; p<=P; p++){
				for(int b=0; b<=E; b++){
					MeE[t][p][b]=-1;
				}
			}
		}

		/* Errekuperaziorako egitura hasieratu */
		MeErrek = new int[T+1][P+1][E+1];
		/* Hautatutako bidaiariak jasoko dituen zerrenda
		 * Errekuperazioa egiterakoan beteko da */
		LinkedList<Integer> hautatutakoBidaiariak = new LinkedList<Integer>();

		/* Metatze-egitura hasieratu kasu nabariekin */
		/* KN1 */
		for(int p=0; p<=P; p++){
			for(int e=0; e<=E; e++){
				MeE[0][p][e]=0;
			}
		}
		/* KN2 */
		for(int t=0; t<=T; t++){
			for(int e=0; e<=E; e++){
				MeE[t][0][e]=0;
			}
		}
		/* KN3 */
		for(int t=0; t<=T; t++){
			for(int p=0; p<=P; p++){
				MeE[t][p][0]=0;
			}
		}

		/* Kasu orokorra: soluzioa */
		if(MeE[T][P][E]==-1){
			bete(T,P,E);
		}

		/* Errekuperazioa */
		/* Errekuperazioa gauzatzeko aldagaiak */
		int Kilo = T;
		int Plaza = P;
		for(int eskatzailea = E-1; eskatzailea>=0; eskatzailea--){

	//bidaiariHautaketa[eskatzaile]=MeErrek[Kilo][Plaza][eskatzaile+1];
			if(MeErrek[Kilo][Plaza][eskatzailea+1]==1){
				hautatutakoBidaiariak.add(eskatzailea);
				Plaza = Plaza-1;
				Kilo = Kilo - ekipaiKG[eskatzailea];
			}
		}

		return emaitzaFitxategianIdatziPD(outFitxategiIzena,
hautatutakoBidaiariak);
	}
```

1.1.4. Analisia

Programazio dinamikoko teknika bidez kodetutako algoritmoen ordena ateratzeko nahikoa da hurrengo eragiketa egitearekin:

$$Gelaxka\ kopurua\ x\ Gelaxka\ garestienaren\ kostua$$

- Gelaxka kopurua: Guztira $(\mathrm{T}+1)\mathrm{x}(\mathrm{P}+1)\mathrm{x}(\mathrm{N}+1)$ gelaxka bete behar dira.
- Gelaxka garestienaren kostua: Kasu honetan gelaxka guztiek kostu berdina dute, eta edozein gelaxkaren balioa denbora konstantean (O(1)) kalkula daiteke.

Beraz, kodetutako algoritmoaren ordena $O(TxPxN)$ da, non:

$$\mathrm{T} = \text{Hegazkinak gehienez garraia dezaken pisua}$$

$$\mathrm{P} = \text{Pakete turistikoak dauzkan plaza kopurua}$$

$$\mathrm{N} = \text{Agentziak jaso dituen eskaera kopurua}$$

1.2. BACKTRACK BIDEZKO (BT) SOLUZIOA

Hurrengo ataletan backtrack bidez garatu diren bi soluzioak aurkeztuko dira. Lehenengoan backtrack zuhaitza 0/1 motakoa izango da, eta bigarrenean aldiz, 1/N motakoa. Analisiaren aldetik badaude hainbat teknika estatistiko erabil daitezkeenak algoritmoen kostua estimatzeko (*Monte Carlo* edo *Las Vegas* adibidez), baina ez ditugu erabiliko. Hala eta guztiz ere, jakin badakigu backtrack bidezko algoritmoak esponentzialak direla.

1.2.1. 0/1 ZUHAITZA

1.2.1.1. Soluzioen formatua (partzialak eta optimoa)

Soluzioak irudikatzeko N tamainako bektore bat (N osagai dituen sekuentzia) erabiliko da: **[x1,x2,…,xi,…,xN]**. **Xi** bakoitzak eskatzaile baten eskaera bat irudikatzen du eta har ditzakeen balioak **[0/1]** dira, eskaera **[baztertzen/onartzen]** den arabera.

Soluzio partziala noiz da soluzioa? Sekuentziak N osagai dituenean, hau da, eskaera bakoitza aztertu denean (onartu ala baztertu). Soluzio partzialean lortu dugun etekina orain arteko optimoa baino handiagoa bada, soluzio partziala orain arteko optimoa bezala erregistratu beharko da.

Soluzio optimoa lortzeko hedatuz joango garen zuhaitzean, maila bakoitzean eskaera bat aztertuko da.

1.2.1.2. Kimak

1. Zuhaitzeko i mailan (i. eskaera aztertzen ari garenean), dagoeneko P eskaera onartu baditugu (P eskaitzaile hautatu ditugu bidaiara joateko), orduan kimatu, hegaldiko P plazak bete baitira.
2. Zuhaitzeko i mailan (i. eskaera aztertzen ari garenean), eskatzaileak bidaiara eraman nahi duen ekipaiaren pisua gehi onartutako eskaeren ekipaien pisua hegazkinak gehienez eraman dezaken pisua gainditzen badu, orduan kimatu.
3. Zuhaitzeko i mailan (i. eskaera aztertzen ari garenean), [xi,…,xN] eskaeren bidez lor daitekeen irabazia (oraindik tratatu gabe daudenak) gehi soluzio partzialean daramagun irabazia, orain arte optimotzat daukagun soluzioa ezin badu hobetu, orduan kimatu.

1.2.1.3. Aldagaiak eta parametrizazioa

ALDAGAIA	DESKRIPZIOA	LOKALA/GLOBALA
`int E`	Agentziak jaso dituen eskaera kopurua	Globala
`int P`	Hegaldiak dituen plaza kopurua	
`int T`	Hegazkinak garraia dezaken gehienezko pisua	
`int[] ordaindu`	Eskatzaile bakoitza ordaintzeko prest dagoena jasotzen duen bektorea	
`int[] ekipaiKG`	Eskatzaile bakoitzak bidaiara eraman nahi duen ekipaiaren pisua	
`int[] SP`	Soluzio partziala	Lokala – Funtzioko parametroa
`int[] SOPT`	Soluzio optimoa	
`int[] SOPTE`*	Soluzio optimoaren etekina	
`int[] SOPTB`*	Soluzio optimoko bidaiari kopurua	
`int[] adabegiKop`*	Backtrack esplorazio zuhaitza korritzean sortuz doazen adabegiak zenbatzeko erabiliko den aldagaia	
`int outEtekina`	3 kimaketa gauzatzeko aldagaia. [x*esk*,…,x*N*] eskaeren mailen batura.	
`int esk`	Tratatuko den eskaeraren indizea	
`int SPBK`	Soluzio partzialean daramagun bidaiari kopurua	
`int SPEK`	Soluzio partzialean daramagun ekipai kilogramoak	

`int SPE`	Soluzio partzialean daramagun etekina	

*__OHARRA__: *Javan objektuak ez diren parametroak (adibidez int datuak) balioz pasatzen dira, eta ez erreferentziz. Horregatik, soluzio optimoaren mailaren balioa posizio bakarreko bektore batean gorde behar izan dugu, erreferentziz pasa dadin.*

Beraz, erabiliko den parametrizazioa honakoa izango da:

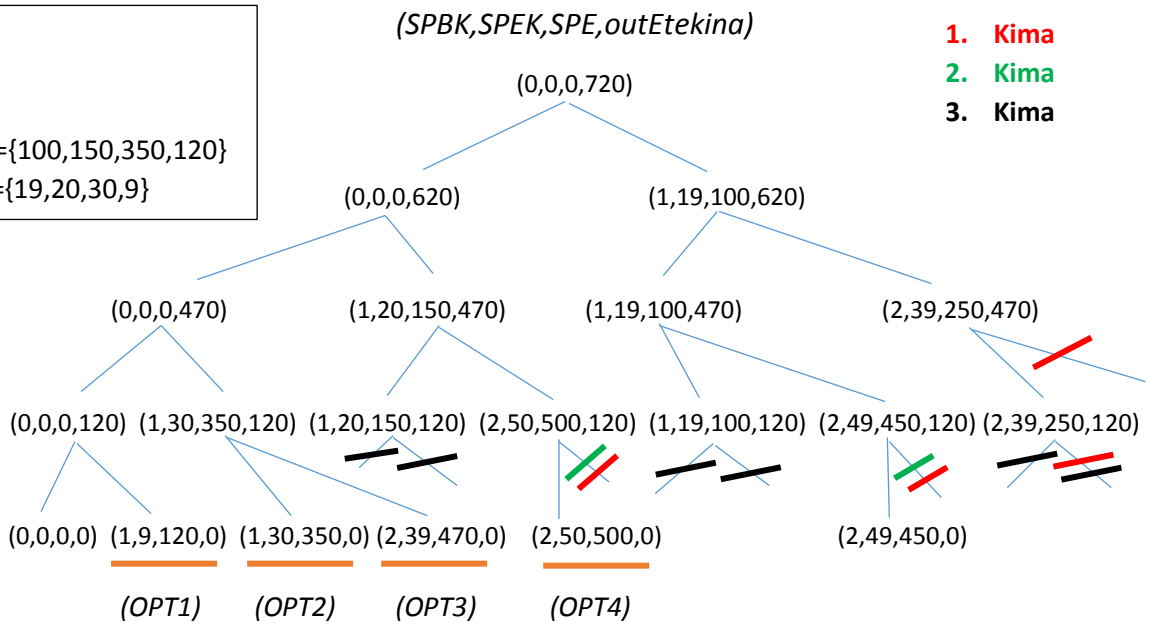
```
bidaiariakHautatu01(int SPBK, int SPEK, int SPE, int[] SP, int esk,
int[] SOPTE, int[] SOPT, int[] SOPTB, int outEtekina, int[]
adabegiKop)
```

Lehenengo deia:

```
bidaiariakHautatu01(0, 0, 0, SP=new int[E], 0, SOPTE={0}, SOPT= new
int[E], SOPTB={0}, outEtekina=∑_{esk=1}^{n} ordaindu[esk], adabegiKop={1}
/*Erroa zenbatzeko*/);
```

1.2.1.4. Zuhaitza

E=4
P=2
T=50
ordaindu={100,150,350,120}
ekipaiKG={19,20,30,9}



1.2.1.5. Kodea

```
public int agentziakLorDezakeenEtekinMaxBT01(String outFitxategiIzena){
            int[] SP = new int[E];
            int[] SOPT = new int[E];
            int[] SOPTE = {0};
            int[] SOPTB = {0};
            int outEtekina=0;
            for(int eskaera=0; eskaera<E; eskaera++) outEtekina=outEtekina +
ordaindu[eskaera];
            int[] adabegiKop = {1}; /* 1etik hasi, zuhaitzaren erroa
zenbatzeko */
            bidaiariakHautatu01(0, 0, 0, SP, 0, SOPTE, SOPT, SOPTB,
outEtekina, adabegiKop);
            return emaitzaFitxategianIdatziBT01(outFitxategiIzena,SOPT,
SOPTE[0], SOPTB[0], adabegiKop[0]);
}

private void bidaiariakHautatu01(int SPBK, int SPEK, int SPE, int[] SP, int
esk, int[] SOPTE, int[] SOPT, int[] SOPTB, int outEtekina, int[]
adabegiKop){
      if (esk==E){
            if(SPE>SOPTE[0]){
                  SOPTE[0]=SPE;
                  SOPTB[0]=SPBK;
                  kopiatu(SP, SOPT);
            }
      }else{
            for(int j=0; j<=1; j++){
                  if(SPBK+(j*1)<=P /*Kima1*/ &&
                     SPEK+(j*ekipaiKG[esk])<=T /*Kima2*/ &&
                     SOPTE[0]<SPE+outEtekina /*Kima3*/){
                        adabegiKop[0]++;
                        SP[esk]=j;
                        bidaiariakHautatu01(SPBK+(j*1),
SPEK+(j*ekipaiKG[esk]), SPE+(j*ordaindu[esk]), SP, esk+1, SOPTE, SOPT,
SOPTB, outEtekina-ordaindu[esk], adabegiKop);
                  }
            }
      }
}
```

1.2.2. 1/N ZUHAITZA

1.2.2.1. Soluzioen egitura

Kasu honetan soluzioak irudikatzeko tamaina aldakorreko zerrendak erabiliko ditugu, handituz/txikituz joango direnak eskaerak onartzen/baztertzen ditugun arabera (1/0 bertsioan erabilitako sekuentziak tamaina finkoa zuten (N)). Beraz, zerrendako elementu bakoitzak onartutako eskaera bat irudikatzen du, eta gehienez N elementu izango ditu (eskaera guztiak onartzen badira).

Soluzio partziala noiz da soluzioa? 0/1 bertsioarekin konparatuta, kasu honetan ez dugu itxaron behar eskaera guztiak tratatu arte soluzio optimoa hobe daitekeen ala ez erabakitzeko. Aldiz, saiakuntza bakoitzarekin egiten den soluzio partzialaren hedapena soluzio optimoa izan liteke (soluzio partzialeko zerrendan eskaerak sartuz/kenduz goaz), eta kasu bakoitzean soluzio optimotzat daukagun etekina hobetzen bada, soluzio partziala optimo bezala erregistratu beharko da.

Zuhaitza eraikitzerakoan egin beharreko adarketa [i..N] izango da, *i* momentuan tratatzen ari garen eskaera izanez. Eskaerak onartzen diren ordena ez da garrantzizkoa.

1.2.2.2. Kimak

Erabilitako kimak 0/1 bertsioko berdinak dira:

1. Dagoeneko P eskaera onartu baditugu (P eskaitzaile hautatu ditugu bidaiara joateko, eta hortaz, soluzio partzialeko zerrendaren tamaina P izango da), orduan kimatu, hegaldiko P plazak bete direlako.
2. Onartutako eskaeren ekipaien pisua gehi eskatzaile batek eraman nahi duen ekipaiaren pisua hegazkinak gehienez eraman dezaken pisua gainditzen badu, orduan kimatu.
3. Oraindik tratatu ez diren eskaerek eman dezaketen irabazia gehi soluzio partzialean daramagun irabazia, orain arte optimotzat daukagun soluzioa ezin badu hobetu, orduan kimatu.

1.2.2.3. Aldagaiak eta parametrizazioa

ALDAGAIA	DESKRIPZIOA	LOKALA/GLOBALA
`int E`	Agentziak jaso dituen eskaera kopurua	Globala
`int P`	Hegaldiak dituen plaza kopurua	
`int T`	Hegazkinak garraia dezaken gehienezko pisua	
`int[] ordaindu`	Eskatzaile bakoitza ordaintzeko prest dagoena jasotzen duen bektorea	
`int[] ekipaiKG`	Eskatzaile bakoitzak bidaiara eraman nahi duen ekipaiaren pisua	

`LinkedList<Integer> SP`	Soluzio partziala	Lokala – Funtzioko parametroa
`LinkedList<Integer> SOPT`	Soluzio optimoa	
`int[] SOPTE`*	Soluzio optimoaren etekina	
`int[] adabegiKop`*	Backtrack esplorazio zuhaitza korritzean sortuz doazen adabegiak zenbatzeko erabiliko den aldagaia	
`int[] outEtekina`*	3 kimaketa gauzatzeko aldagaia. Bere kalkulua 0/1 bertsioarekin konparatuta desberdina da, zuhaitzaren egitura desberdina delako	
`int esk`	Tratatuko den eskaeraren indizea	
`int SPBK`	Soluzio partzialean daramagun bidaiari kopurua	
`int SPEK`	Soluzio partzialean daramagun ekipai kilogramoak	
`int SPE`	Soluzio partzialean daramagun etekina	

OHARRA: *Javan objektuak ez diren parametroak (adibidez int datuak) balioz pasatzen dira, eta ez erreferentziz. Horregatik, soluzio optimoaren mailaren balioa posizio bakarreko bektore batean gorde behar izan dugu, erreferentziz pasa dadin.*

Beraz, erabiliko den parametrizazioa honakoa izango da:

```
bidaiariakHautatu1N(int SPBK, int SPEK, int SPE, LinkedList<Integer>
SP, int esk, int[] SOPTE, LinkedList<Integer> SOPT, int[] outEtekina,
int[] adabegiKop)
```

Lehenengo deia:

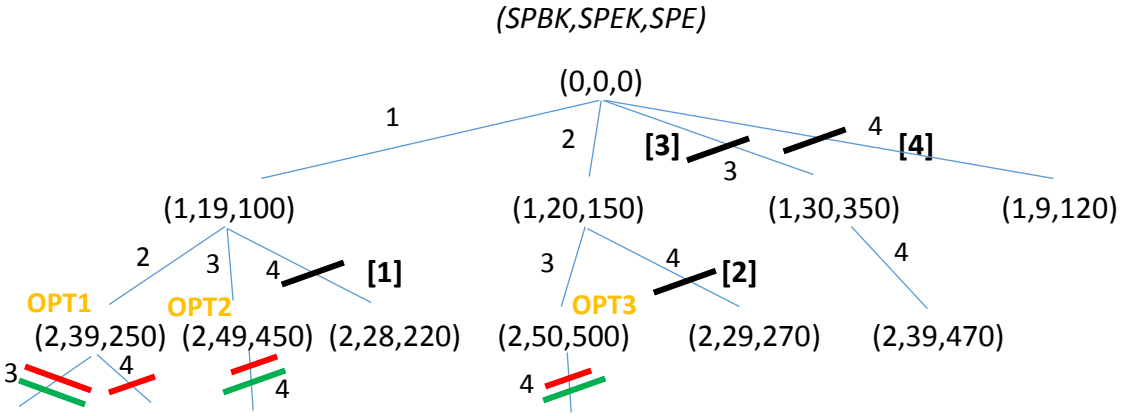
```
bidaiariakHautatu1N(0, 0, 0, SP=new LinkedList<Integer>(), 0,
SOPTE={0}, SOPT==new LinkedList<Integer>(), outEtekina*,
adabegiKop={0});
```

* Taulan adierazi dugunez, hirugarren kima gauzatzeko oraindik tratatu ez diren eskaerek eman dezaketen irabazia jakin behar dugu. Honen kalkulua 0/1 zuhaitzean sinplea zen maila bakoitzean eskaera bat tratatzen zelako, 1/N zuhaitzetan gertatzen ez dena (maila bakoitzean eskaera desberdinak aztertzen dira). Horregatik lehen deia egin baino lehen eta kima eraginkorki gauza dadin, i eskaera bakoitzeko $\sum_{j=i}^{n} ordaindu[j]$ kalkulua egingo dugu, emaitza *outEtekina* izeneko bektore batean gordez.

1.2.2.4. Zuhaitza

E=4
P=2
T=50
ordaindu={100,150,350,120}
ekipaiKG={19,20,30,9}

1. **Kima**
2. **Kima**
3. **Kima**



[1] 100+120=220 < 450

[2] 150+120=270 < 500

[3] 0+470=470 < 500

[4] 0+120=120 < 500

1.2.2.5. Kodea

```
public int agentziakLorDezakeenEtekinMaxBT1N(String outFitxategiIzena){
            LinkedList<Integer> SP = new LinkedList<Integer>();
            LinkedList<Integer> SOPT = new LinkedList<Integer>();
            int[] SOPTE = {0};
            int[] outEtekina = new int[E];
            outEtekina[E-1]=ordaindu[E-1];
            for(int eskaera=E-2; eskaera>=0; eskaera--){

      outEtekina[eskaera]=ordaindu[eskaera]+outEtekina[eskaera+1];
            }
            int[] adabegiKop = {1}; /* 1etik hasi, zuhaitzaren erroa
zenbatzeko */
            bidaiariakHautatu1N(0, 0, 0, SP, 0, SOPTE, SOPT, outEtekina,
adabegiKop);
            return emaitzaFitxategianIdatziBT1N(outFitxategiIzena,SOPT,
SOPTE[0], adabegiKop[0]);
}

private void bidaiariakHautatu1N(int SPBK, int SPEK, int SPE,
LinkedList<Integer> SP, int esk, int[] SOPTE, LinkedList<Integer> SOPT,
int[] outEtekina, int[] adabegiKop){
      if(esk==E){
            if(SPE>SOPTE[0]){
                  SOPTE[0]=SPE;
                  kopiatu(SP,SOPT);
            }
      }else{
            for(int eskaera=esk; eskaera<E; eskaera++){
                  if(SPBK+1<=P && /* Kima 1 */
                        SPEK+ekipaiKG[eskaera]<=T && /* Kima 2 */
                        SPE+outEtekina[eskaera]>SOPTE[0]){ /* Kima 3*/
                        adabegiKop[0]++;
                        SP.addFirst(eskaera);
                        bidaiariakHautatu1N(SPBK+1, SPEK+ekipaiKG[eskaera],
SPE+ordaindu[eskaera], SP, eskaera+1, SOPTE, SOPT, outEtekina, adabegiKop);
                        SP.removeFirst();
                  }
            }
            if(SPE>SOPTE[0]){
                  SOPTE[0]=SPE;
                  kopiatu(SP,SOPT);
            }
      }
}
```

2. AZTERKETA ESPERIMENTALA

Atal honetan aurrekoan aurkeztutako kodeketak praktikan jarriko ditugu, eta hainbat exekuzio egingo ditugu sarrera datu desberdinekin.

1. Datu Fitxategien sorrera

Frogak egin baino lehen datu fitxategiak eraiki behar izan ditugu. Horretarako ausazko zenbakiak lortzeko Javak ematen dizkigun baliabideak erabiliko ditugu. Datuak nolabait koherentzia izan dezaten, honako mugak ezarri ditugu zenbakien sorreran:

- Eskatzaileak bidaiagatik ordaindu nahi dutena: 150 eta 600 € artean.
- Eskatzaile bakoitzak eraman nahi duen pisua: 10 eta 60 kg artean.

Orokorrean, ausazko zenbakiak lortzeko honako Java funtzioa erabili dugu:

```
int ausZenb = ThreadLocalRandom.current().nextInt(BeheLimt, GoiLimt + 1);
```

Guztira 17 fitxategi sortu ditugu, honako eskaera kopuruekin*: 20, 40, 50, 60, 80, 100, 150, 200, 350, 500, 700, 900, 1000, 2000, 3000, 4000, 5000, 7000 eta 10000. Fitxategiak sortzerakoan P eta T aldagaiek 10 eta 100 balioak izango dituzte, hurrenez hurren.

2. Frogekin lortutako denborak

Lehenik eta behin sortu berri ditugun fitxategiekin frogak egin ditugu. Beraz, parametroen balioak P=10, T=100 eta E={*Aipatutakoak**} izango dira. Gure lehen ondorioak ateratzeko balio izango digu.

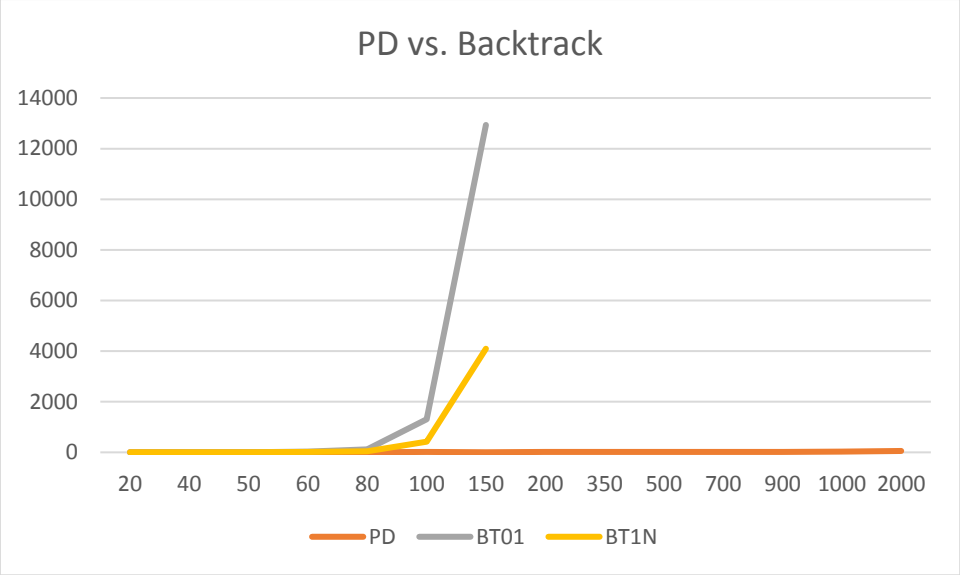
Hona hemen lortutako denborak (milisegundotan):

P=10, T=100

E	PD (ms)	BT01 (ms)	BT1N (ms)
20	6	3	6
40	4	3	6
50	5	6	4
60	8	9	6
80	8	35	14
100	5	114	44
150	11	1307	422
200	8	12934	4094
350	14	*Denbora Asko > 10-15min*	
500	16		
700	20		
900	21		
1000	20		
2000	32		
3000	55		
4000	*StackOverflow*		

- Sortutako arazoak:
  1. Backtrack bidezko soluzioak 350 eskaerekin ordu erdi pasa eta gero ez dute bukatu.
  2. Programazio dinamikoko teknika bidez, 4000 eskaera dituen fitxategiarekin *StackOverflow* errorea ematen da, hots, errekurtsio pila bete delako. Beraz, nahiz eta PD soluzioaren kodeketa errazagoa izan memoriadun funtzioak (errekurtsioa) erabilita, hobe da iteratiboki egitea honako mugaketa ez izateko. Programazio dinamikoko teknikak, berez, espazio estra dezente har dezake, eta hortaz hobe da gure aldetik errekurtsio pila gehiegi ez betearaztea, oraindik memoria gehiago ez erabiltzearren.

- Lehenengo ondorioak:
  1. Programazio dinamikoaren teknika hobeagoa da backtracking-en teknika baino. Sarreraren tamaina handitzean, backtracking teknika bidez ebatzitako algoritmoek behar duten denbora izugarri igotzen da PDkoaren teknikarekin konparatuta. Ondoren agertzen den grafikoan aipatutakoa erraz ikus daiteke:



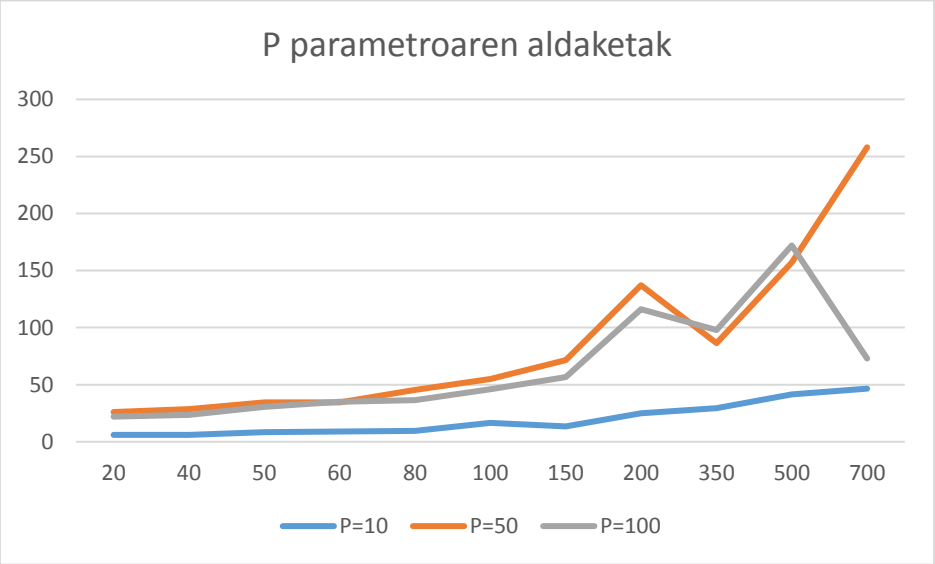
  2. Backtracking-en bi bertsioen artean, 1/N teknika hobeagoa da 0/1a baino, azkarragoa baita. Izan ere, 1/N bidez garatutako esplorazio-zuhaitzek dituzten adabegi kopurua askoz txikiagoa da 0/1 bidez hedatutako zuhaitzekin konparatuta. Ondorio hau aurreko grafikoan ere erraz ikus daiteke, eta azterketa analitikoan irudikatutako zuhaitzekin bat dator.

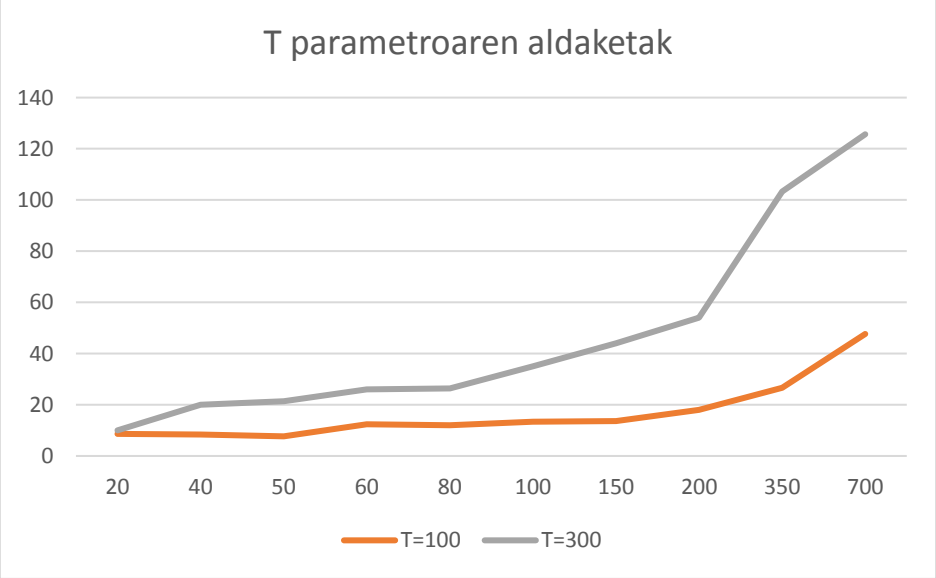
Jarraian T eta P parametroak zertxobait aldarazi eta algoritmoak berriz exekutatuko ditugu, denbora neurketak berriro eginez.

Gure kasuan aipatutako parametroak gutxi eraldatu ditugu, nahiz eta jakin froga/analisi zehatzagoak egiteko exekuzio eta aldagaien bariazioa handiagoa izan behar dela *. Orora, honako parametroekin gauzatu ditugu froga berriak: P=10 – T=300, P=50 – T=300, P=100 – T=300, P=50 – T=100 eta P=100 – T=100.

Interesekoak izatekotan, lortutako denbora taulak eranskin moduan atxikitu dira txosten honen amaieran. Hemen aldiz, lortutako datuak aztertu ondoren atera ditugun ondorioak adieraziko ditugu:

1. Datuak aztertu ondoren eta *Excel* kalkulu horri batez denboren batezbesteko batzuk eginez, P eta T parametroen igoerak denboran duen eragina ikusi izan dugu. Hona hemen lortu ditugun grafikoak:





* Gainera, parametroetarako erabili ditugun balioak errealitatetik zerbait urruntzen dira. Adibidez, gaur egungo distantzia ertain/txikietarako erabiltzen diren hegazkinak, gutxienez ~ 180 (P) bidaiarientzako eserlekua badute, eta guztira 2500kg (T) baino pisu gehiagoko ekipaia eramateko gai dira.

Grafikoetan argi ikus daiteke nola parametro hauen balioen igoerak zuzenean algoritmoek exekutatzeko behar duten denbora handitzen duten. Gainera, T parametroak denboran daukan eragina handiagoa da P parametroarekin konparatuta. Azken hau argi ikus daiteke ere grafikoetan, lerroen artean dagoen distantzia kontuan izanez gero (P parametroaren grafikan lerroak elkartuagoak daude).

Amaitzeko, praktika hau eginez eta denborekin erlazioa duten beste ondorio batzuk aipatuko ditugu.

1. Datuen arabera, nahiz eta tamaina bereko fitxategiak izan, lortzen diren denborak oso desberdinak izan daitezke. Ondorio hau beste ikasleen fitxategiak erabiliz atera izan dugu: nahiz eta P,T,E parametro berdinak izan, ateratzen ziren denborak oso desberdinak ziren kasu gehienetan. Hortaz, analisi enpiriko on bat egiteko (guk egin dugun baino hobea), fitxategi desberdinak erabili beharko ziren.

2. Eranskineko tauletan ikus dezakegun moduan (gorriz), exekuzioak egiterakoan bi errore desberdin agertu zaizkigu, beti programazio dinamikoaren teknika erabiltzen ari genuenean:
   - *StackOverflow*: Errekurtsio pilarentzat erreserbatuta dagoen memoria bete da.
   - *OutOfMemory*: Javak bere makina birtualean exekuzioak egiteko erabiltzen duen memoria bete da.

   Hortaz, programazio lengoaiak ere mugak ezarri dizkigu. Baliteke Java ez izatea teknika hauek frogatzeko lengoai aproposena, bi arrazoi hauengatik:

   - Javak bere makina birtual propioa dauka, eta horren gainean exekutatzen ditu eraikitako programak. Makina horrek bere muga propioak izan ditzake, eta hortaz, gure ordenagailuaren (gure benetako makinaren) baliabide guztiak ez erabiltzea gerta liteke.
   - Web orrialde batzuetan ikusi dugunez, ordenagailu pertsonaletarako (edo “PC”) erabiltzen den Java “bertsioa” bere murriztapenak ditu makina mota hauentzako. Hau da, makina motaren arabera Javak ezartzen dituen mugak aldatuz doaz. Eta normalean alda daitezkeen ere, horretan ez sartzea erabaki dugu.

3. Orokorrean, Programazio Dinamikoaren “erabilgarritasuna” mugatzen duen baliabidea memoria da, eta Backtracking teknikaren kasuan, konputazio edo kalkulu baliabideak, esplorazio-zuhaitzak berez ez direlako memorian gordetzen (inplizituak dira).

3. ERANSKINA: DENBORA TAULAK

Ondoren lortutako denbora taulak atxikitzen dira:

- ***P=10 – T=100***

P=10, T=100

E	PD (ms)	BT01 (ms)	BT1N (ms)
20	6	3	6
40	4	3	6
50	5	6	4
60	8	9	6
80	8	35	14
100	5	114	44
150	11	1307	422
200	8	12934	4094
350	14	*Denbora Asko > 10-15min*	
500	16		
700	20		
900	21		
1000	20		
2000	32		
3000	55		
4000	*StackOverflow*		

- ***P=50 – T=100***

P=50; T=100

E	PD (ms)	BT01 (ms)	BT1N (ms)
20	10	6	2
40	8	3	4
50	6	6	7
60	8	12	8
80	10	37	17
100	16	114	44
150	8	1286	421
200	18	12888	4022
350	24	*Denbora Asko > 10-15min*	
500	35		
700	39		
900	50		
1000	52		
2000	90		
3000	112		
4000	*StackOverflow*		

- ***P=100 – T=100***

P=100; T=100

E	PD (ms)	BT01 (ms)	BT1N (ms)
20	10	2	2
40	13	3	3
50	12	5	4
60	21	11	8
80	18	38	12
100	19	114	44
150	22	1297	416
200	28	12842	3951
350	42	*Denbora Asko > 10-15min*	
500	53		
700	84		
900	73		
1000	63		
2000	115		
3000	*OutOfMemory*		

- ***P=10 – T=300***

P=10; T=300

E	PD (ms)	BT01 (ms)	BT1N (ms)
20	3	3	3
40	8	2826	2092
50	7	42505	27738
60	9	371455	219008
80	10	*Denbora Asko > 10-15min*	
100	14		
150	22		
200	19		
350	36		
500	43		
700	63		
900	72		
1000	66		
2000	166		
3000	194		
4000	*StackOverflow*		

- ***P=50 – T=300***

P=50; T=300

E	PD (ms)	BT01 (ms)	BT1N (ms)
20	8	8	3
40	21	1803	1574
50	22	24217	19198
60	29	280334	225197
80	17	*Denbora Asko > 10-15min*	
100	37		
150	40		
200	58		
350	84		
500	30		
700	54		
900	258		
1000	225		
2000	*OutOfMemory*		

- ***P=100 – T=300***

P=100; T=300

E	PD (ms)	BT01 (ms)	BT1N (ms)
20	19	5	8
40	31	1796	1589
50	35	23799	19170
60	40	275268	218831
80	52	*Denbora Asko > 10-15min*	
100	54		
150	70		
200	85		
350	190		
500	143		
700	260		
900	*OutOfMemory*		