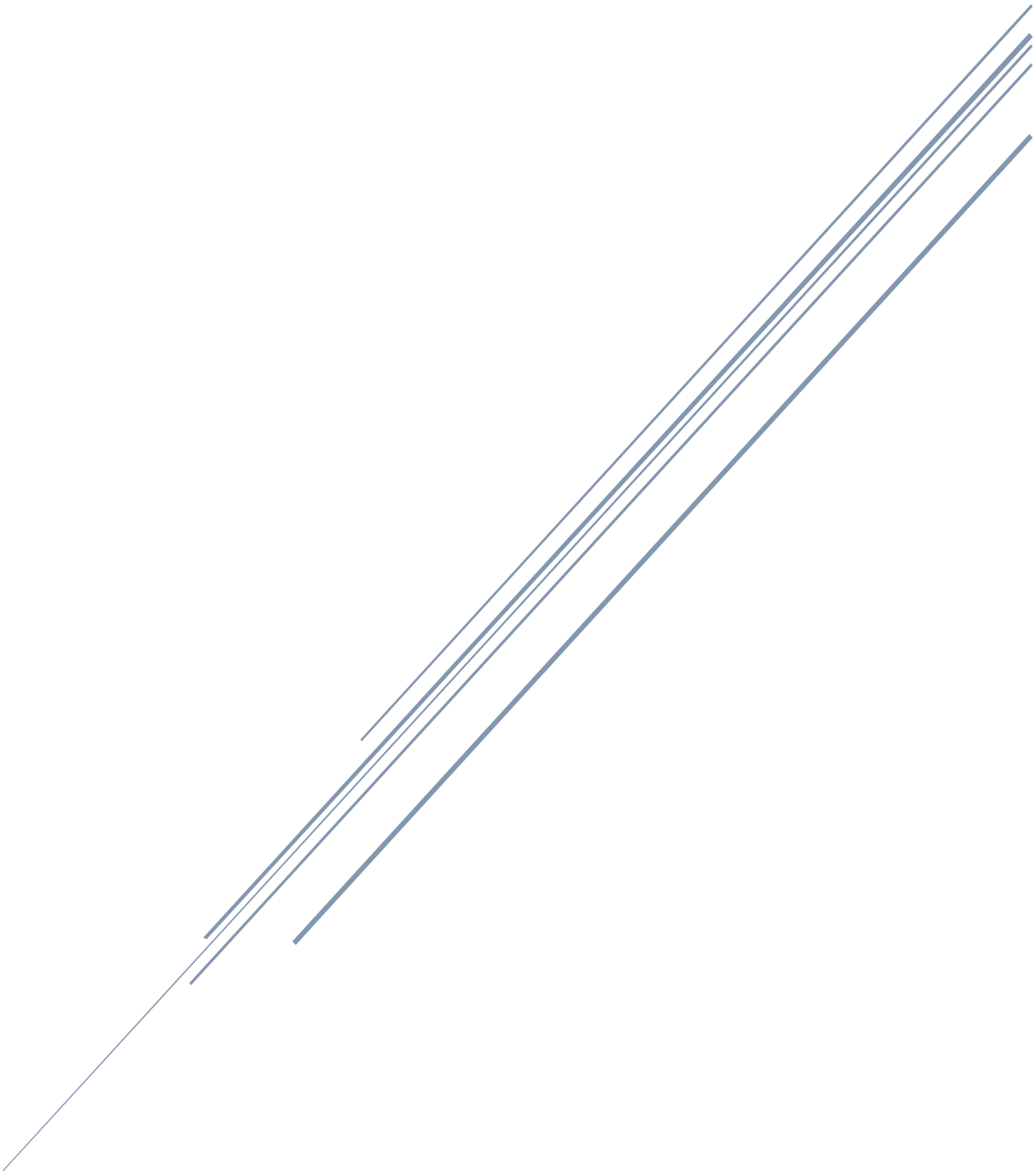
# ALGORITMOEN DISEINUA

## 2. Inplementazioaren Txostena



EHU
Urtzi Otamendi Etxabe

# 1 AURKIBIDEA

# 2 ENUNTZIATUA

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.

***SARRERA DATUAK***

- *Lehenengo lerroan*, hiru balio: **P** zenbaki naturala, hegaldian dauden plazen kopurua adierazten duena; eta **T** zenbaki positiboa, bidaiarien ekipaiek guztira pisa dezakeena (hamarrekoan jasoa) (hots, gehienez bi tona), eta **N** zenbaki positiboa, bidaian izena eman dutenen pertsonen kopurua.

- *Hurrengo N lerrotan* bina zenbaki osoko positibo: eskatzaile bakoitzak joanagatik gehienez ordaintzeko prest legokeen bidai-saria (onartuko luketen “Biletearen” gehienezko salneurria) eta eramango duten ekipaiaren gehienezko pisua hamarrekotan. Hots, 3 zenbakiak, gehienez 30 kilo eramango lituzkeela adierazten du.

***IRTEERA DATUAK***

- 1 lerroan: BK joango diren bidaiari kopurua

- Hurrengo BK lerroetan hiruna balio: eskatzaile sarituaren identifikatzailea, honek ordainduko duen bidai-saria eta bere ekipaiaren pisua.

- Hurrengo lerroan, bidai-agentziak lortuko duen irabazia, eta ekipaien pisua.

- Azkeneko lerroan: BT soluzioetan esplorazio-zuhaitzak izan dituen adabegi kopurua eta PD soluzioan P x T x N.

Analisi enpirikorako/esperimentalerako aztertu P=80, T=200, N=200 kasua, eta bat/batzuk Finko utziz bestearen/en tamaina aldatuz probak egin. Zeintzuk izan dira tamaina handieneko sarrerak ebatzi ahal izan dituzunak? Zein soluzio barianteekin?

# 3 AZTERKETA ANALITIKOA

Problema hau ebazteko, bi teknika ezberdin erabili beharko dira. Horiek aplikatu aurretik, azterketa analitiko bate egin beharko da. Horretarako, enuntziatutik datu garrantzitsuenak eta deigarrienak jasoz. Erabiliko diren teknikak ezberdinak direnez, bakoitzak bere analisi berezia izango du.

## 3.1 PROGRAMAZIO DINAMIKOA

Lehenik eta behin, sarrera datuak aztertu beharko dira. Datu horietaz baliatuta, ekuazio sistema eta metatze egitura lortzeko.

- P: Plaza kopurua.
- T: Pisu maximoa.
- Bid[1..N]: Bidaiara joan nahi dutenen lista. Bertan ordaindu nahi dutena eta eramango duten pisua dago gordeta.

### 3.1.1 Ekuazio Sistema

Dei nagusia antartida izeneko funtzioa izangoda. Honek hiru parametro izango ditu, antartida(p,t,n).

- p: Geratzen diren plaza libreak.
- t: Betetzeko geratzen den pisua.
- n: Zenbatgarren bidaiaria aztertzen ari garen.

#### *3.1.1.1 Kasu Nabariak*

*Antartida(p,t,0)=0*

*Antartida(0,t,n)=0*

*Antartida(p,0,n)= mergeSort(Bid.dirua && Bid.pisua==0) -> lehen p.-ak*

#### *3.1.1.2 Kasu Orokorra*

*antartida(p,n,t)= max{ (bid[n].dirua + antartida(p-1,n-1, t-bid[n].pisua)), antartida(p,n-1, t) }*

*bid[n].pisua<=t*

*antartida(p,n,t)= antartida(p,n-1, t)* *bid[n].pisua>t*

### 3.1.2 Metatze Egitura

#### *3.1.2.1 Dimentsioak*

Gure ekuazio sistema hiru parametroren menpe dagoela ikusi dugu. Beraz, metatze egiturak 3 dimentsiokoa izan beharko du TxNxP.

*M[T][N][P]*

- *T: 0..T*
- *N: 1..N*
- *P: 1..P*

#### *3.1.2.2 Hasieratzeak*

Matrizea hiru dimentsiokoa denez, betetzea modu errekurtsiboan egin beharko da. Beraz, gelaxka guztiak (-1) balioaz hasieratuko dira.

#### *3.1.2.3 Balio Lehenetsiak*

Ekuazio sistemako kasu nabarietan oinarriturik, balio lehenetsiak honako hauek izango direla ikus dezakegu.

*Matrizea[T][0][P] = 0*

*Matrizea[T][N][0] = 0*

#### *3.1.2.4 Betetzea*

Metatze egitura tridimensional hau betetzeko, modu errekursiboan egin beharko dugu. Modu honetan, behar diren gelaxkak soilik bete beharko ditugu. Horretarako, lehenik eta behin, balio batez (-1) hasieratu beharko dugu matrize osoa. Balio hori irakurtzean, gelaxka kalkulatu behar dela esan nahiko du.

#### *3.1.2.5 Soluzioa*

Soluzioa M[T][N][P] gelaxkan egongo da.

### 3.1.3 Analisia

Bi analisi ezberdin egingo ditugu, bata denboraren analisia eta bestea erabilitako memoria estrarena.

#### *3.1.3.1 Denbora*

Denboraren analisia hiru zatitan banatuko da. Izan ere, algoritmoa hiru ataza nagusitan banatzen da.

- ***HASIERAKETAK***: Kontuan izanik bi matrize simetriko erabiliko ditugula, hauek batera hasieratu ahal izango ditugu. Gelaxka guztiak korritu behar direnez kostua O(N*T*P) izango da. Gainera, balio lehenetsiak ere ezarri beharko dira. Horren kostua, txikiagoa izanik, batuketaren erregela aplikatuz, ondoriozta dezakegu zati honen kostua honakoa dela:

  O(N*T*P)

- ***BETETZEA***: Modu errekurtsiboan inplementatu denez, kostuaren analisia zailagoa da. Izan ere, ez dira gelaxka guztiak kalkulatu beharko. Ondorioz, alferrikako deiak saihestuko dira. Motxilaren problemako arrazonamendu bera erabiliz, gelaxka bateko balioa kalkulatzeko, gehienez n gelaxken balioa kalkulatu beharko da; non n, bidaiari ezberdinen kopurua ziango den. P=0 denean, merge_sort erabiliko da, baina nola honen kostua O(lg n) den, batuketaren erregela aplikatuz ez da kontuan hartuko. Beraz ordena honakoa litzateke

  O(N*(N*T*P))

- ***ERREKUPERAZIOA***: Bidaiari lista lortzeko, lortutako soluzio optimotik errekuperazioa egin beharko da. Hori egiteko, matrize simetriko bat erabiliko da, eta hor, bidaiaria joango den ala ez gordeko da. Matrize simetriko horren korritzea, soilik N-ren menpe egongo da. Izan ere, P eta T, N. bidaiariaren informaziotik lortu bai dezakegu. Beraz ordena honakoa litzateke:

  O(N)

Batuketaren erregela aplikatuz, hiru atazen denborak gehituta, argi ikusten da zein izango den algoritmo dinamiko honen denbora kostua:

**O(N*(N*P*T))**

#### *3.1.3.2 Memoria*

Sarrera irteerako datuez gain, memoriako beste egitura batzuk erabili dira. Hain zuzen ere, metatze egiturak. Horiek, (N*T*P) tamainako bi matrize simetriko izan dira. Lehenengoa, lor daitekeen diru kopuru optimoa kalkulatzeko erabili da. Bestea berriz, soluzio optimorako aukeratu diren bidaiarien errekuperazioa egiteko.

### 3.1.4 Algoritmoa

```
Void Optimoa(Bid[], N, T,P){
Int[T][N][P] M;
Int[T][N][P] ME;
Dirua=0;
bidaiariLista= new BidaiariLista();

if(N==0 || P==0){
        return (dirua, bidaiariLista);
}
Else if( T==0){
        pisuGabekoak=pisurikGabekoakHartu(Bid);
        merge_sort(pisuGabekoak, 1, pisuGabekoak.Size);
        luz= pisuGabekoak.Size;
        if( luz >=P)
                luz=P;
        for( i in 1…luz){
                bidaiariLista.add(pisuGabekoak[i])
                dirua += pisuGabekoak[i].dirua;
        }
        Return (dirua, bidaiariLista);
}
Else{
        //Hasieraketak
        For( t in 0..T)
                For(n in 1..N)
                        For(p in 1..P)
                                M[t][n][p]=-1;
                                ME[t][n][p]=0;

        //Balio lehenetsiak

        For( t in 0..T)
                For(n in 1..N)
                        M[t][n][0]=0;

        For( t in 0..T)
                For(p in 1..P)
                        M[t][0][P]=0;

        If(M[T][N][P]==-1) antartida(T,N,P);
        Dirua=M[T][N][P];
        Errekuperazioa();
        Return (dirua, bidaiariLista);
}
}
Void antartida(t,p,n){
Int sol_partz;
If(bid[n].pisua <= t)
```

```
        If(M[t- bid[n].pisua][n-1][p-1]==-1)  antartida(t- bid[n].pisua, n-1, p-1);
        If(M[t][n-1][p]==-1) antartida(t,n-1,p);

        Int aux1= M[t- bid[n].pisua][n-1][p-1];
Int aux2= M[t][n-1][p];
If(aux1>aux2)
        sol_partz=aux1;
        ME[t][n][p]=1;
Else
        Sol_partz=aux2;
Else
        Sol_partz= M[t][n-1][p];

M[t][n][p]=Sol_partz;
}

Void errekuperazioa(){
Int p=P;
Int t=T;
For( n in N..1){
        If(ME[t][n][p]==1)
                t -= Bid[n].pisua;
                p--;
                bidaiariLista.add(Bid[n]);
}

}
```

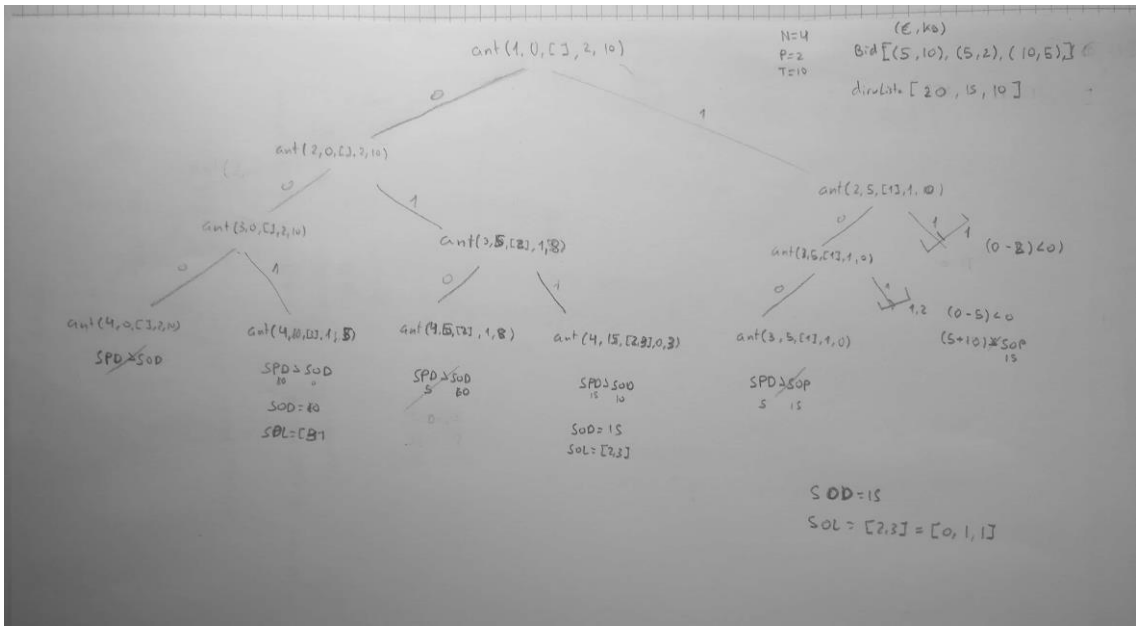
## 3.2 BACKTRACKING 01

Lehenik eta behin, sarrera datuak aztertu beharko dira. Datu horietaz baliatuta, zuhaitza irudikatu ahal izango dugu.

- P: Plaza kopurua.
- T: Pisu maximoa.
- Bid[1..N]: Bidaiara joan nahi dutenen lista. Bertan ordaindu nahi dutena eta eramango duten pisua dago gordeta.

Bi backtracking teknika daude. Lehenengo hau, 01 bat teknika sinpleena da. Kasu honetan, mailaka aztertzen joango gara, hau da, bidaiariak banan bana aztertuz. Bi aukera izango ditugu, bidaiaria soluzioan sartu (1) edo ez sartu (0).

### 3.2.1 Zuhaitza




### 3.2.2 Aldagaiak

#### *3.2.2.1 Globalak*

- SOL[]: Soluzio optimoko bidaiari lista. Listaren luzaera N izango da, eta bertan, joango diren bidaiariek 1 balioa izango dute, eta joan behar ez direnek 0.

- SOD: Soluzio optimoan lortu daitekeen irabazi maximoa.

- N: Bidaiara joan nahi duten bidaiari kopurua.

- P: Plaza kopurua.

- T: Pisu maximoa.

- diruLista[]: Listaren luzaera N izango da. Bertan, edozein i. elementuak, i. bidaiaritik N. bidaiarira lor daitekeen etekin maximoa gordeko du, hau da, kasurik onenean denak sartzea lortuz gero.

- bidaiariLista: Bidaiari lista duen bektore bat.

#### *3.2.2.2 Lokalak*

- i: Zenbatgarren bidaiaria aztertzen ari garen.

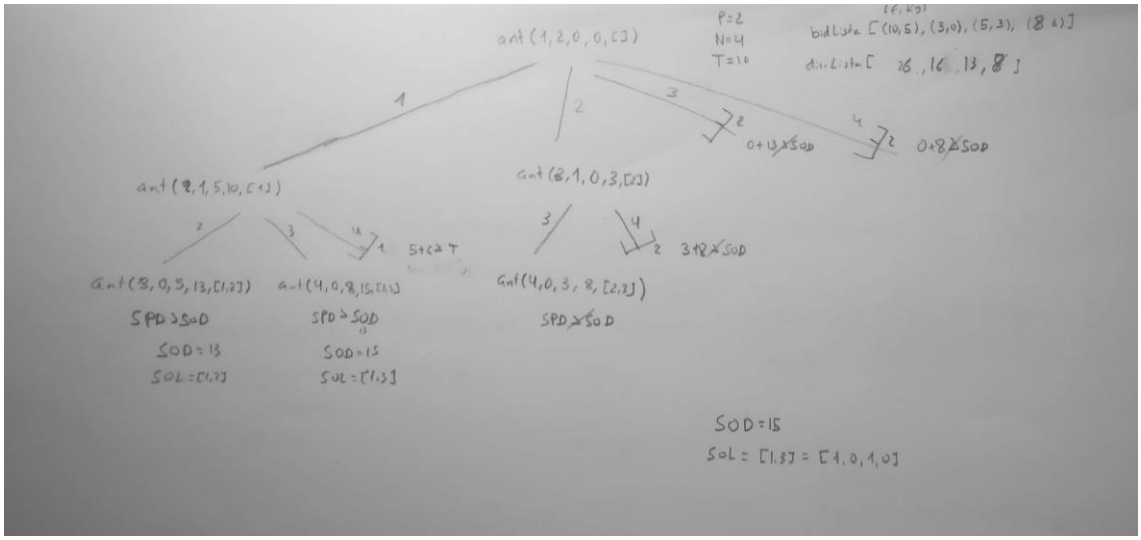
- SPB: Soluzio partzialean dagoen diru kantitatea.
- SPL: Soluzio partzialaren bidaiari lista.
- SPxP: Soluzio partzialean betetzeko falta diren plaza kopurua.
- SPxT: Soluzio partzialean betetzeko falta diren kilo kopurua.

#### 3.2.2.2.1 Deia

***ANTARTIDA(I,SPD,SPL,SPXP,SPXT)***

Hasierako deia:

*Antartida(1,0,[N],P,T);*




### 3.2.3 Kasuak

Kasu nabariak eta orokorrak aztertu aurretik, kontuan izan behar da, Programazio Dinamikoko planteamenduan bezala, N edo P zero balioa dutenean, soluzio optimoa ere zero izango dela. Bestalde, T-ren balioa zerokoa denean, aurreko planteamenduan bezala, merge sort teknika erabiliz, pisurik eramango ez duten eta gehien ordaindu nahi duten bidaiariak hartuko direla.

#### *3.2.3.1 Nabariak*

> *SPxP==0 edo i>N denean, aztertu ea soluzio partziala (SPD) optimoa baino hobea den (SOD). Ala bada, SPD eta SPL, soluzio optimoaren balioak izango dira.*

##### *3.2.3.1.1 Orokorra*

*Kimaketak kontuan izanik, kasu orokorrean bi aukera izango ditugu, i. elementua soluzioan sartu edo ez. Biak probatu beharko dira. Baina, kontuan izanik, elementua listan kimaketak gainditzean soilik sartuko dela.*

*If(kimakGainditu==true)*

*SPL[i]=1;*

*Anartida(i+1,SPD+bidairiLista[i].dirua, SPL, SPxP-1, SPxT- bidaiariLista[i].pisua);*

*SPL[i]=0;*

*Antartida(i+1,SPD,SPL,SPxP,SPxT);*

### 3.2.4 Kimak

1. Aukeragai den bidaiariaren pisuak, soluzio partzialean geratzen den kapazitatea gainditzen duenean.

*SPxT-bidaiariLista[i].pisua<0*

2. Aukeragai den bidaiaria eta ondorengo guztiak sartuz gero; kasurik onenean, momentuko Soluzio Optimoa gainditzen ez duenean.

*SPD+bidaiariLista[i].dirua<SOD*

### 3.2.5 Algortimoa

```
Void Optimoa(bidaiariLista [], N, T,P){
SOD=0;
SOL[N];
if(N==0 || P==0){
        return (SOD, SOL);
}
Else if( T==0){
        pisuGabekoak=pisurikGabekoakHartu(bidaiariLista);
        merge_sort(pisuGabekoak, 1, pisuGabekoak.Size);
        luz= pisuGabekoak.Size;
        if( luz >=P)
                luz=P;
        for( i in 1…luz){
                SOL[i]=1;
                SOD += pisuGabekoak[i].dirua;
        }
        Return (SOD, SOL);
}
Else{
        Gehienez();
        Antartida(1,0,[N].fill(0), P, T);
```

```
        Return (SOD,SOL);
}
}

Void antartida(i,SPD,SPL,SPxP,SPxT){

If(SPxP==0 || i>N)
        If(SPD> SOD)
                SOD=SPD;
                SOL=SPL;
Else
        If(SPxT-bidaiariLista[i].pisua>=0)
                If(diruLista[i]+SPD>SOD)
                        SPL[i]=1;
Anartida(i+1,SPD+bidairiLista[i].dirua, SPL, SPxP-1, SPxT- bidaiariLista[i].pisua);
                        SPL[i]=0;
Antartida(i+1,SPD,SPL,SPxP,SPxT);


}



Void gehienez(){
        diruLista[N]=bidaiariLista[N].dirua;
        for ( i in N-1…1)
                diruLista[i]=diruLista[i+1]+bidaiariLista[i].dirua;
}
```

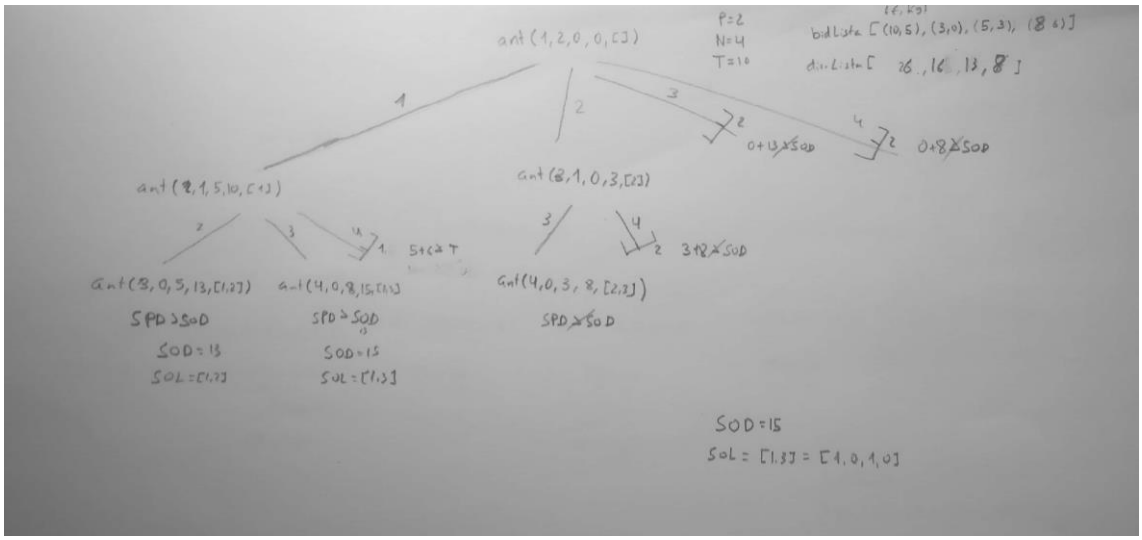
## 3.3 BACKTRACKING 1N

Lehenik eta behin, sarrera datuak aztertu beharko dira. Datu horietaz baliatuta, zuhaitza irudikatu ahal izango dugu.

- P: Plaza kopurua.
- T: Pisu maximoa.
- Bid[1..N]: Bidaiara joan nahi dutenen lista. Bertan ordaindu nahi dutena eta eramango duten pisua dago gordeta.

Teknika hau, aurrekoa baino konplexuagoa da. Bertan, aztertzen gauden bidaiaritik azkenekora sartuko dira; posible izanez gero, dei errekurtsibo ezberdinetan bakoitza.

### 3.3.1 Zuhaitza




### 3.3.2 Aldagaiak

#### *3.3.2.1 Globalak*

- SOL[]: Soluzio optimoko bidaiari lista. Listaren luzaera N izango da, eta bertan, joango diren bidaiariek 1 balioa izango dute, eta joan behar ez direnek 0.
- SOD: Soluzio optimoan lortu daitekeen irabazi maximoa.
- N: Bidaiara joan nahi duten bidaiari kopurua.
- P: Plaza kopurua.
- T: Pisu maximoa.
- diruLista[]: Listaren luzaera N izango da. Bertan, edozein i. elementuak, i. bidaiaritik N. bidaiarira lor daitekeen etekin maximoa gordeko du, hau da, kasurik onenean denak sartzea lortuz gero.
- bidaiariLista: Bidaiari lista duen bektore bat.

#### *3.3.2.2 Lokalak*

- i: Zenbatgarren bidaiaria aztertzen ari garen.
- SPB: Soluzio partzialean dagoen diru kantitatea.

- SPL: Soluzio partzialaren bidaiari lista.
- SPxP: Soluzio partzialean betetzeko falta diren plaza kopurua.
- SPxT: Soluzio partzialean akumulatutako kiloak.

#### 3.3.2.2.1 Deia

**Antartida(i,SPxT,SPxP,SPB,SPL)**

Hasierako deia:

*Antartida(1,P,0,0,[N]);*

### 3.3.3 Kasuak

Kasu nabariak eta orokorrak aztertu aurretik, kontuan izan behar da, Programazio Dinamikoko planteamenduan bezala, N edo P zero balioa dutenean, soluzio optimoa ere zero izango dela. Bestalde, T-ren balioa zerokoa denean, aurreko planteamenduan bezala, merge sort teknika erabiliz, pisurik eramango ez duten eta gehien ordaindu nahi duten bidaiariak hartuko direla.

#### *3.3.3.1 Nabariak*

*SPxP==0 edo i>N denean, aztertu ea soluzio partziala (SPD) optimoa baino hobea den (SOD). Ala bada, SPD eta SPL, soluzio optimoaren balioak izango dira.*

##### *3.3.3.1.1 Orokorra*

*Kimaketak kontuan izanik, aztergai den bidaiaria sartzeko aukera badugu, lehenik ta behin SPL-an adierazi beharko da, eta dei errekurtsiboa egin ostean, berriz aurreko egoerara itzuli.*

```
If(kimakGainditu==true)
        SPL[i]=1;
        Anartida(i+1, SPxP-1, SPxT+ bidaiariLista[i].pisua, SPD+bidairiLista[i].dirua,
SPL);
        SPL[i]=0;
```

### 3.3.4 Kimak

1. Aukeragai den bidaiariaren pisuak, soluzio partzialean akumulatutako pisuari gehituz, kapazitate maximoa gainditzen duenean.

   *SPxT+bidaiariLista[i].pisua>P*

2. Aukeragai den bidaiaria eta ondorengo guztiak sartuz gero; kasurik onenean, momentuko Soluzio Optimoa gainditzen ez duenean.

   *SPD+bidaiariLista[i].dirua<SOD*

### 3.3.5 Algortimoa

```
Void Optimoa(bidaiariLista [], N, T,P){
SOD=0;
SOL[N];
if(N==0 || P==0){
        return (SOD, SOL);
}
Else if( T==0){
        pisuGabekoak=pisurikGabekoakHartu(bidaiariLista);
        merge_sort(pisuGabekoak, 1, pisuGabekoak.Size);
        luz= pisuGabekoak.Size;
        if( luz >=P)
                luz=P;
        for( i in 1…luz){
                SOL[i]=1;
                SOD += pisuGabekoak[i].dirua;
        }
        Return (SOD, SOL);
}
Else{
        Gehienez();
        Antartida(1,P,0,0,[N].fill(0));
        Return (SOD,SOL);
}
}

Void antartida(i, SPxP,SPxT ,SPD,SPL){

If(SPxP==0 || i>N)
        If(SPD> SOD)
                SOD=SPD;
                SOL=SPL;
Else
        For( n in i…N)
                If(bidaiariLista[n].pisua+SPxT<=T)
                        If(SPB-bidaiariLista[n]>SOD)
                                SPL[n]=1;
```

```
                                    Antartida(n+1,SPxP-1,SPxT+bidairiLista[n].pisua,
SPB+bidaiariLista[n].dirua, SPL);
                                    SPL[t]=0;


        If(SPD>SOD)
                SOD=SPD;
                SOL=SPL;

}
Void gehienez(){
        diruLista[N]=bidaiariLista[N].dirua;
        for ( i in N-1…1)
                diruLista[i]=diruLista[i+1]+bidaiariLista[i].dirua;
}
```

# 4 AZTERKETA ENPIRIKOA

Aurreko hiru algoritmoak Java-ko ingurune batean inplementatu ostean, proba programa bat sortu da, edonork nahi izanez gero proba batzuk egiteko. Programa horretan, erabiltzaileak, sarrera balioak aldatzeko aukera izango du, hau da, bidaiara joan nahi dutenen kantitatea, bidaian guztira eraman daitekeen pisu maximoa edota plaza kopurua. Kalkulu horiek egiteko datuak, proba fitxategi batetik lortzen dira. Datu horiek, enuntziatuko formatua mantentzen dute. Kontuan izanik, bidaiariak ordainduko duen prezioa 1 eta 100 arteko zenbaki bat dela, eta pisua berriz 0 eta 100 artekoa.

Proba programa hori exekutatu ahal izateko, kodearekin batera pasatako 3 fitxategi beharko dira. Programaren ".jar" fitxategia, proben ".txt"-a eta exekuzio agindua duen ".bat" fitxategia. Azken hori klikatuz gero, proba programa exekutatuko da. Baina, kontuan izan, horretarako hiru fitxategiak leku berean gordeak egon behar dutela.

Programa exekutatzean, nahi diren balioak sartu ahal izango dira. Ondoren, programak hiru algoritmoak exekutatuko ditu. Lehenik, SPD eta ondoren *backtracking*-eko biak. Bakoitzaren exekuzioa amaitzean, honen bidez lortutako soluzio optimoa inprimatuko digu pantailan, eta kalkulatzeko behar izan duen denbora.

### 4.1.1 Denbora Neurketak

Exekuzioen ostean jasotako denboren analisia egingo da orain. Kontuan izanik, algoritmo bakoitzak bere denbora behar izango duela, balio ezberdinetarako, hiru grafika ezberdin egitea erabaki da. Izan ere, problemak hiru parametro ditu, eta horien aldaketarekin, denbora ezberdinen analisia eta azterketa egingo da.

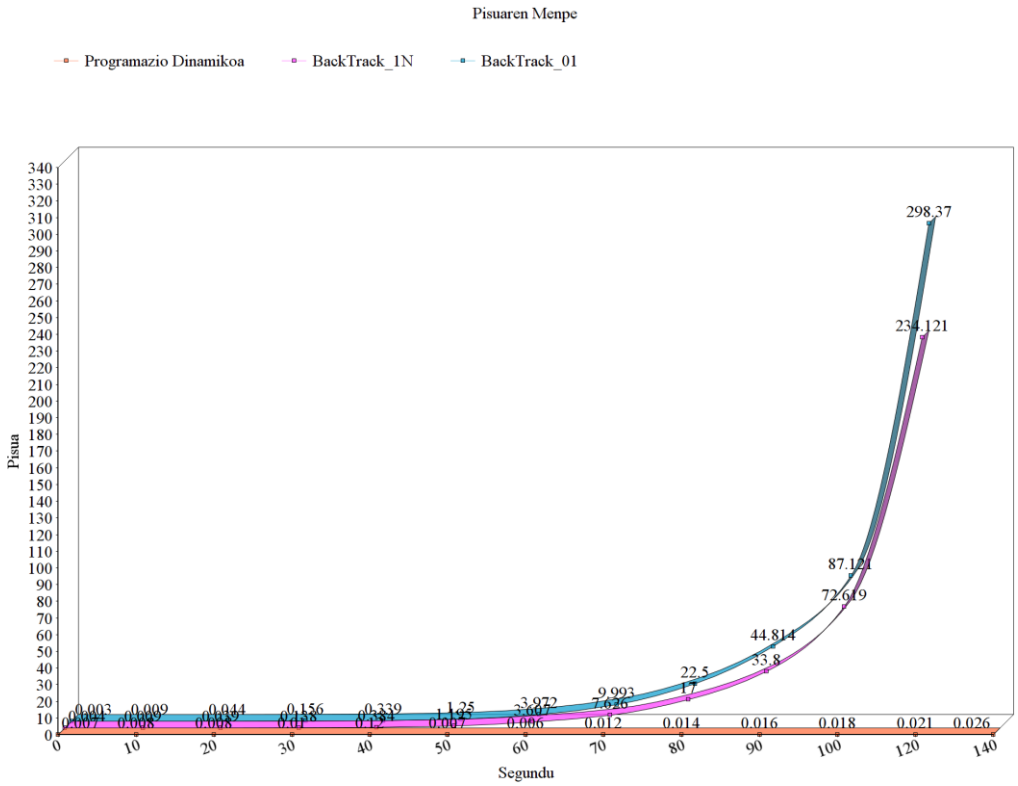
Exekuzioen alderaketak modu egokian egiteko, aldagaien defektuzko balioak honako hauek izango dira:

- N = 100
- P = 10
- T = 50

Grafika bakoitzean, parametro bat soilik aldatuko da, eta besteak balio hauekin mantenduko dira. Ez dira balio handiagoak erabili, neurketa handiegiak lortzen zirelako.

#### 4.1.1.1 Pisuaren Menpe

Grafika honetan, pisu maximoaren balio ezberdinetarako, gure algoritmoek behar izango duten denbora aztertuko dugu. Pisu maximoari, 0 eta 140 arteko balioak eman dizkiogu, grafika osatzeko.



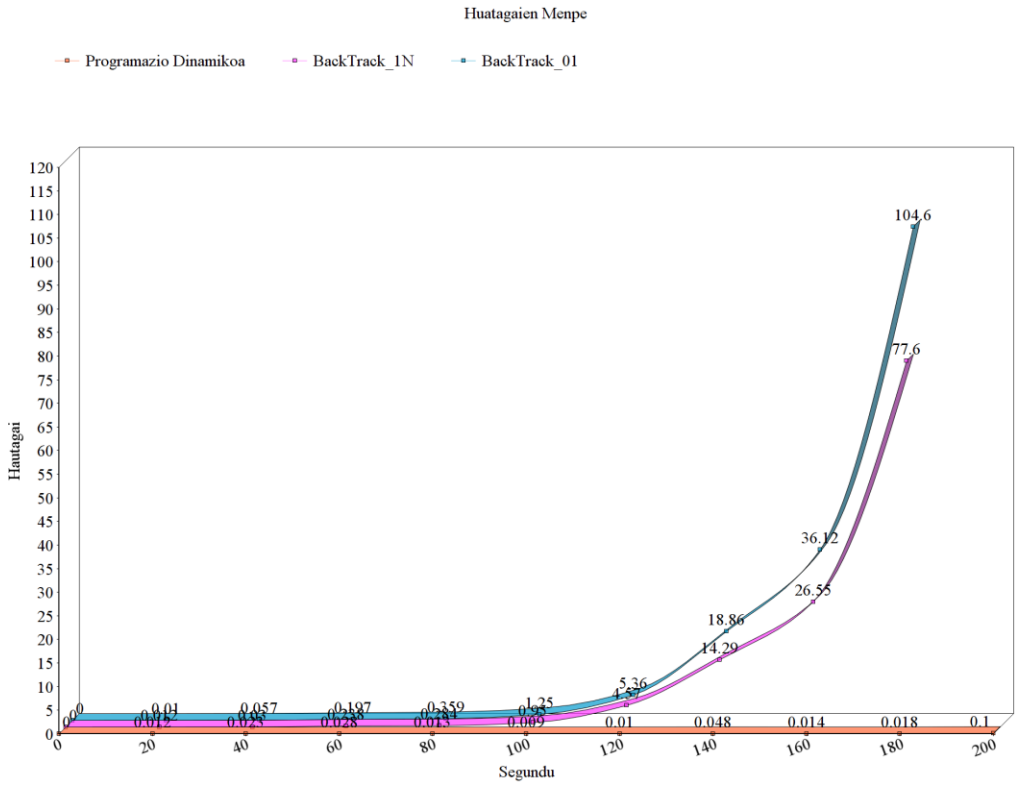

Ikus daitekeen moduan, hasiera batean, hiru algoritmoek, denbora bera behar dute kalkuluak egiteko. Baina, balioa handitzen joan ala, garbi ikusten da, bi *backtrackin* algoritmoek geroz eta denbora gehiago behar dutela. Esponentzialki handitzen dela ondoriozta dezakegu.

Programazio dinamikoa erabili den algoritmoak aldiz, denbora ia konstantean funtzionatzen duela ikus dezakegu. Gogoan izanik, algoritmoaren kostua O(N*N*M*P) dela, balio aldaketa txiki hauek ez dute eragin handirik exekuzio denboran.

Azkenik, azter dezakegu, 120-tik aurrera, *backtracking* algoritmoek behar duten denbora ez dela lortu, exekuzioak denbora asko behar baitzuen.

#### *4.1.1.2 Hautagaien Menpe*

Kasu honetan, hautagai kopuru ezberdinekin egingo dugu azterketa. Horretarako, N-ri balio ezberdinak emango dizkiogu, 0 eta 200 artean.

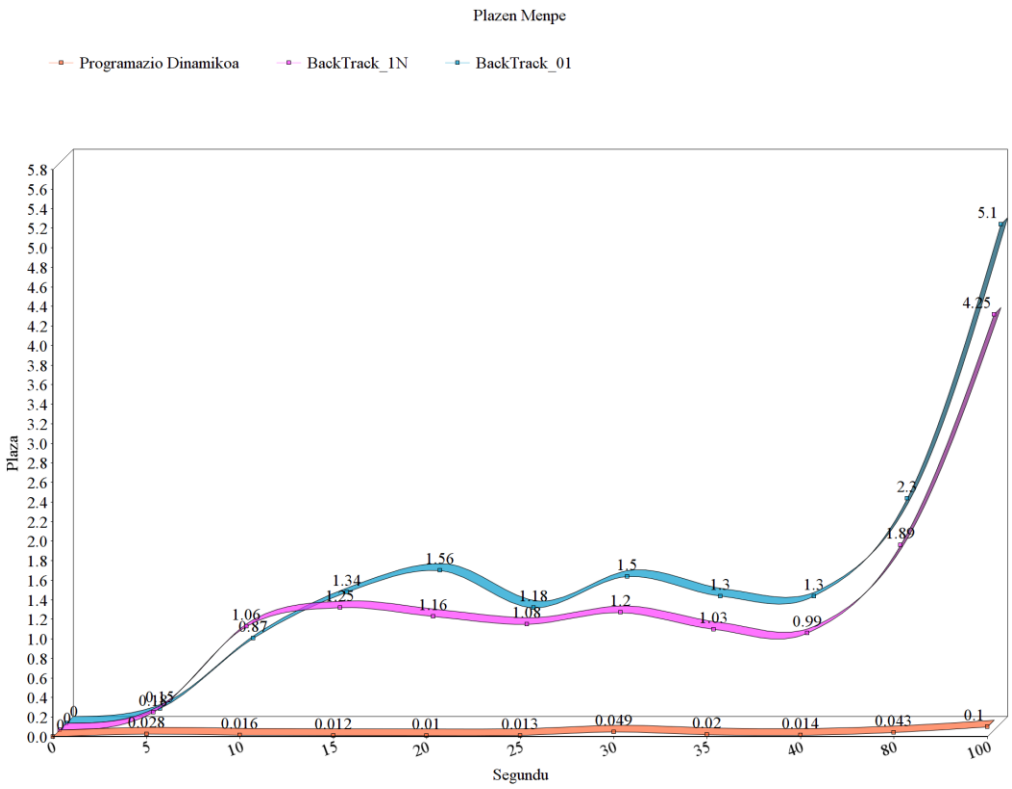



Aurreko grafikan bezala, argi eta garbi ikusten da, backtracking teknika erabili den algoritmoek behar duten exekuzio denbora esponentzialki handitzen dela. Baina, lehen gertatu den bezala, programazio dinamikoan behar izan den denbora, konstante mantendu da.

Hasiera batean, hirurek denbora berdina izan badute ere, 100 hautagaitik aurrera, bi teknikak ezberdintzen joan direla argi ikusten da.

#### *4.1.1.3 Plazen Menpe*

Azken grafika honetan, plaza kopuruarekin egingo ditugu azterketak. Horretarako, balio ezberdinak emango dizkiogu, baina kasu honetan, lehen emanikoak baino txikiagoak.




Hasiera batean, grafika honen emaitzak arraroak direla pentsa dezakegu, izan ere, besteetan ez bezala, kasu honetan backtracking algoritmoak ez baitute esponentzial itxura hori. Kasu honetan, hasieratik ezberdintze dira bi teknikak.

Aztertzerakoan, konturatzen gara, kimaketen ondorioz, backtracking teknikek denbora gutxiago behar izango dutela, izan ere, naiz eta plaza kopurua handitzen joan, kasu honetan exekuzioak limitatzen dituena pisu maximoa baitda. Hau da, naiz eta bidaiara 40 pertsona joan ahal izan, kilo kopurua 50 bada, seguruena 5-10 plazekin nahikoa izatea, pisu ezak amaituko baitdu prozesuekin.

### 4.1.2 Balio Handienak

Orain, proba batzuk egingo dira, algoritmoek jasan ahal dituzten balio handienak gutxi gora-behera zein izango liratekeen jakiteko. Kontuan izanik, guk daukagun datu listan gehienez 1011 hautagai daudela.

#### *4.1.2.1 Programazio Dinamikoa*

| DENBORA | N | T | P |
|---|---|---|---|
| **1M 17S** | 1011 | 1000 | 100 |
| **35S** | 500 | 2000 | 100 |
| **-** | 1011 | 10000 | 100 |
| **0.008S** | 100 | 10000 | 99 |

| | | | |
|---|---|---|---|
| **-** | 1011 | 1000 | 500 |
| **2.865S** | 200 | 500 | 500 |

#### *4.1.2.2 BackTrack*

| **DENBORA(1N/01)** | **N** | **T** | **P** |
|---|---|---|---|
| **-/-** | 1011 | 100 | 5 |
| **3S/4S** | 100 | 10000 | 5 |
| **-/-** | 500 | 10000 | 5 |

### 4.1.3 Ondorioak

Aurreko grafikak ikusirik, denboraren analisia egin da parametro bakoitzaren balio ezberdinetarako. Argi eta garbi ikusi da, programazio dinamikoa erabili den algoritmoak parametroen balio handietarako ere, denbora tarte zuzen batean egiten dituela kalkuluak.

Backtrackinean berriz, adarkatze bakoitzeko dei errekurtsibo bat dagoenez, zenbat eta balio handiagoak jarri, orduan eta denbora gehiago behar dute. Ikusi den moduan, N eta T-k 100 dik gorako balioak hartzen dituenean, exponentzialki handitzen dela exekuzio denbora.

Beraz, ondoriozta dezakegu, datu tamaina handietarako, teknika egokiena, programazio dinamikoa dela, izan ere, honek denbora nahiko berdintsuan egiten baititu exekuzioak. Kontuan izanik, probatu diren balioetarako, ez duela behin ere segundo erdia baino gehiago behar izan.