

5 DÜNAAMILINE PLANEERIMINE ALGAJATELE

Dünaamiline planeerimine on üks algoritmitehnika leivanumbreid, seda on võimalik kasutada paljudes situatsioonides otse ja see moodustab ka tähtsa koostisosa mitmes spetsiifilises algoritmis.

Inglise keeles kasutatakse enamasti terminit *dynamic programming*, aga ajalooline taust on ka seal seotud planeerimisega. Dünaamilise planeerimise leiutas ameerika rakendusmatemaatik Richard Bellman 1950. aastate alguses, kui ta töötas RAND Corporationis, mis omakorda täitis USA Õhujõudude tellimusi. Bellman uuris, kuidas paremini planeerida mitme-etapilisi protsesse, kuid leidis, et sõna „planeerimine“ oli liiga üldine ja mitmetähenduslik. Seetõttu nimetas ta oma uurimise tulemust dünaamiliseks programmeerimiseks, pidades silmas protsessi kavandamist spetsiifilisel viisil. Hiljem on sõna „programmeerimine“ saanud hoopis teiselaadse sisu, aga termin on jäänud. Eesti keeles öeldakse nii dünaamiline planeerimine kui ka dünaamiline programmeerimine. Edaspidi ütlen lihtsalt DP, igaüks võib ise otsustada, kumb sõna talle suupärasem on. Kui sa oled internetist lugenud, et DP on hoopis midagi muud, siis oled valejälgedel.

DP on rohkem üldine lähenemisviis kui konkreetne retsept või algoritm. Seepärast on siin peatükis ka ohtralt näidisülesandeid, mis illustreerivad erinevaid DP kasutamise võimalusi.

Käesolevas peatükis räägitakse tihti algoritmi keerukusest. Nagu kolmandas peatükis öeldud, on keerukuse mõõtmisel alati vaja defineerida, millise elementaaroperatsiooni suhtes me seda mõõdame. Siin peatükis on lihtsuse ja ülevaatlikkuse huvides elementaaroperatsioonina kasutatud vastava algoritmi üksikuid samme, olgu siis tegu objektide võrdlemise, ümber tõstmise või muu sarnasega. Valdava enamiku reaalse ülesannete puhul on elementaaroperatsioonid piiratud pikkusega (nt 64-bitiste) arvudega sooritatavad tehted. Nende puhul loeme lihtsuse mõttes, et nad võtavad alati konstantse aja, nii et kirjeldatud keerukus on ühtlasi ka ajaline keerukus.

5.1 SISSEJUHATAV ÜLESANNE – FIBONACCI JADA

Järgnevat ülesannet võib sageli kohata juba programmeerimise algõppe kursustel. Fibonacci jada on defineeritud lihtsalt: alustatakse nullist ja ühest ning iga järgmine element on eelmise kahe summa:

$$F_n = \begin{cases} 0, & \text{kui } n = 0 \\ 1, & \text{kui } n = 1 \\ F_{n-1} + F_{n-2} & \text{muul juhul} \end{cases}$$

Definitsioon on väga lihtne, aga matemaatikud on Fibonacci jadal leidnud tohutult palju huvitavaid omadusi ja leiavad neid edasi. Samamoodi saab Fibonacci jada abil illustreerida huvitavaid kontseptsioone programmeerimisest.

Ülesande sõnastus on samuti lihtne:

Leida Fibonacci jada liige kohal n .

NÄIDE:

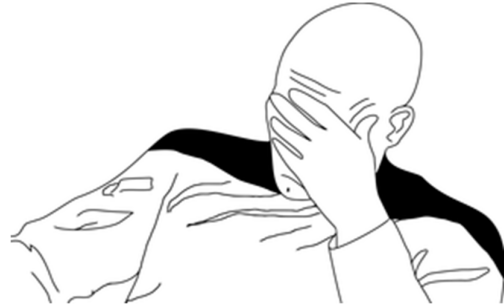
$n = 10$

Vastus: 55

5.1.1 Tavaline rekursioon ehk jõumeetod

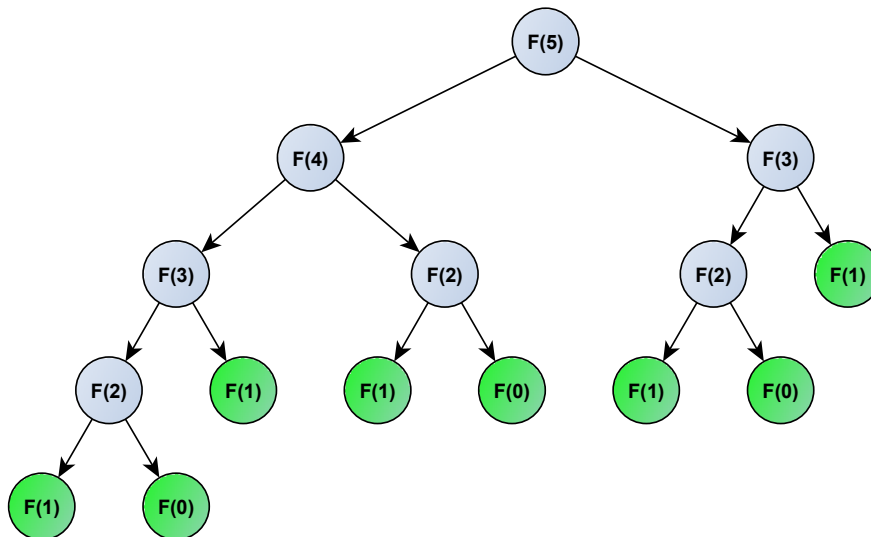
Eelneva ülesande lahendamiseks kirjutab algaja programmeerija tavaliselt definitsiooni põhjal järgmise funktsiooni:

```
int Fib1(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return Fib1(n - 1) + Fib1(n - 2);
}
```



Algoritmid programmeerimises kogenum programmeerija võtab selle koodi peale aga peast kinni. Miks? Sest näiteks $\text{Fib1}(5)$ arvutamiseks leiab programm $\text{Fib1}(4)$ ja $\text{Fib1}(3)$. $\text{Fib1}(4)$ jaoks arvutab programm (uuesti!) $\text{Fib1}(3)$ ja $\text{Fib1}(2)$ väärtused.

Kokku moodustavad väljakutsed sellise puu:



Funktsiooni väljakutsete arv (ning sellega seoses programmi tööaeg) lähevad kiiresti käest ära:

N	Vastus	Väljakutsete arv
1	1	1
2	1	3
3	2	5
4	3	9
5	5	15
6	8	25
7	13	41
8	21	67
9	34	109
10	55	177
20	6765	21891
30	832040	2692537
40	102334155	331160281

Väga kaugele ilmselt nii ei jõua.

5.1.2 Mäluga rekursioon

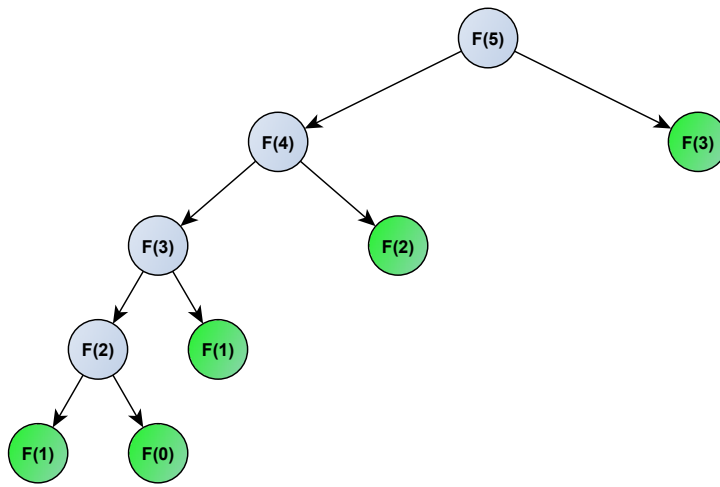
Väljakutsete arv kasvab väga kiiresti, kuid nagu eelpool juba mainitud, siis funktsiooni kutsutakse välja palju kordi täpselt sama parameetriga. Sellisel juhul on targem hoida juba arvutatud vahetulemused meeles:

```
const int Max = 1000;
long A[Max];
long Fib2(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;

    if (A[n] == 0) // Kui A[n] väärtus pole veel teada, leiame selle
        A[n] = Fib2(n - 1) + Fib2(n - 2);

    return A[n]; // Aga kui on juba teada, siis lihtsalt tagastame
}
```

Seekord on puu palju väiksem:



Ja töö palju kiirem:

N	Vastus	Väljakutsete arv
1	1	1
2	1	3
3	2	5
4	3	7
5	5	9
6	8	11
7	13	13
8	21	15
9	34	17
10	55	19
20	6765	39
30	832040	59
40	102334155	79
50	12586269025	99
60	1548008755920	119
70	190392490709135	139
80	23416728348467685	159
90	2880067194370816120	179

Meetodi väljakutsete arv kahanes eksponentsiaalsest lineaarseks! Sellist vahepealsete tulemuste meelde jätmist nimetatakse **mäluga rekursiooniks** ja see on intuitiivselt loogiline samm DP mõtlemisviisi mõistmisel. Pane ka tähele, et mäluga rekursiooni puhul vahetatakse ajaline keerukus täiendava mäluvajaduse vastu – toodud lahendus hoiab kõiki vahetulemusi meeles.

5.1.3 Alt üles DP

Eelmine lähenemine töötas „ülalt alla“ – alguses leiti suurem väärtus, millest liiguti väiksematele. Tulemuse seisukohalt võib aga sama hästi liikuda „alt üles“ ehk väiksematelt arvudelt suurematele.

Sellele vastab järgmine kood:

```
long Fib3(int n)
{
    const int Max = 100000;
    long A[Max];
    A[0] = 0;
    A[1] = 1;

    for (int i = 2; i <= n; i++) { // Arvutame algusest peale terve jada
        A[i] = A[i - 1] + A[i - 2];
    }

    return A[n];
}
```

Alt üles lähenemine on nn „stiilipuhas“ DP. Kui DP-st räägitakse, peetakse tavaliselt silmas just seda varianti. Selline lahendus töötab enamasti ka kiiremini, sest meil pole enam rekursiooni, vaid ainult lihtne tsükel. Kaasaegsed kompilaatorid suudavad küll ka eelmisest lahendusest ise rekursiooni kõrvaldada, nii et Fib2 ja Fib3 kompileeritud koodi efektiivsus on sama, kuid keerulisema näite puhul ei tarvitse see enam nii olla.

5.1.4 Kokkuhoidlik DP

Mõttearenduse viimase sammuna saab teha veel ühe olulise optimeerimise: pane tähele, et väärtuste massiivis kasutatakse korraga ainult kolme väärtust, seega pole ülejäänud massiivi vaja.

Tulemuseks on kood, mille mäluvajadus on konstantne ning tööaeg lineaarne:

```
long Fib4(int n)
{
    long A[3];
    A[0] = 0;
    A[1] = 1;

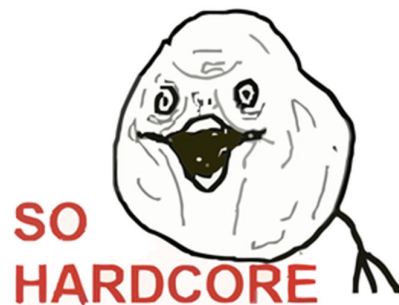
    for (int i = 2; i <= n; i++) {
        A[i % 3] =
            A[(i - 1) % 3] +
            A[(i - 2) % 3];
    }

    return A[n % 3];
}
```

Märkus: kui matemaatika appi võtta, siis saab Fibonacci arve leida veel efektiivsemalt, kasutades Binet' valem:

$$F_n = \frac{\varphi^n - \psi^n}{\varphi - \psi}, \text{ kus } \varphi = \frac{1 + \sqrt{5}}{2} \text{ ja } \psi = \frac{1 - \sqrt{5}}{2}$$

Kui vastust soovitakse piiratud täpsusega, saab Binet' valem abil leida vastuse konstantse ajaga, kuid kui soovime piiramatut tüvekohtade arvu, on ka see valem lineaarse keerukusega.



Kas kuitahes suuri Fibonacci arve saab üldse leida kiiremini kui lineaarse ajaga? Ei saa, sest F_n sõltub arvust n eksponentsiaalselt ning F_n tüvekohtade arv sõltub F_n -ist omakorda logaritmiliselt. Seega

sõltub F_n tüvekohtade arv n -ist lineaarselt, mis tähendab, et ainuüksi tulemuse välja kirjutamine (või kusagile salvestamine) võtab lineaarse aja.

Põhjalikumad Fibonacci arvude leidmise käsitlust saab lugeda ka aadressilt <https://www.nayuki.io/page/fast-fibonacci-algorithms>.

5.1.5 DP retsept

Uuritud näide oli üpris lihtne, aga paljude DP ülesannete lahenduste leidmine on taandatav samadele mõttelistele sammudele, mida ma nimetan **DP retseptiks**:

1. Mõtle, milline võiks olla jõumeetodiga rekursiivne lahendus. Kui vaja, joonista programmi töö puuna välja.
2. Proovi rekursioonipuu harusid taaskasutada, jättes vahepealsed vastused meelde.
3. Alustagi kohe nende vahepealsete vastuste väljaarvutamisest.
4. Kui võimalik, taaskasuta uute vastuste jaoks eelmiste vastuste alt vabanenud mälu.



DP tavapäraseks võtteks on, et neid vahepealseid vastuseid hoitakse **väärtuste tabelis**. Kui Fibonacci arvude puhul jäi tabel lõpuks ainult kolme lahtri suuruseks, on see tavaliselt keerulisema struktuuriga. Järgmised jaotused uurivadki mitmesuguseid kvalitatiivselt erinevaid võimalusi läbivaatuspuude ahendamiseks lihtsamatesse tabelitesse.

5.2 LINEAARNE VÄÄRTUSTE TABEL - OPTIMAALNE MAKSMINE

Järgmine optimeerimisega seotud situatsioon juhtub sageli toidupoes, aga olemuselt sarnast probleemi tuleb lahendada ka tösisemates olukordades. Tegemist on klassikalise probleemiga, mis on inglise keeles tuntud kui *change-making problem*.

On antud N erineva väärtusega münte: $V_1, V_2, \dots, V_N$. Leida minimaalne müntide arv, millega saab tasuda summa S . Iga väärtusega münte on kasutada piiramatult.

NÄIDE:

$N = 3$

$V = [1, 3, 4]$

$S = 6$

Vastus: 2 ([3, 3])

5.2.1 Ahne algoritm

Teatud väärtusega müntide (nt tavalised euromündid) puhul saab kasutada lihtsat ahnet lähenemist: võta kõige suurema väärtusega münte nii palju, kuni nende koguväärtus ei ole suurem kui vajalik summa S , seejärel lisa järgmise suurima väärtusega münte nii palju, kui mahub jne kuni soovitud summa on käes. Näiteks 84 sendi maksmiseks on vaja 5 münti: 50-, 20-, 10-, 2- ja 2-sendine.

Ahne algoritm on väga efektiivne. Kui antud müntide väärtused on suuruse järgi sorteeritud, siis on see lineaarse keerukusega $O(n)$. Kui väärtused on antud juhuslikus järjekorras, tuleb need enne sorteerida (või otsida iga kord kõige suuremat kasutamata münti, mis veel maksmata summat ei ületa) ja siis on keerukusklass $O(n \log n)$.

Kui mündid on aga suvaliste väärtustega, siis võib ahne algoritm anda vale tulemuse. Kui on antud näiteks mündid väärtustega 1, 3 ja 4 ning tarvis on saada summa $S = 6$, annab ahne lähenemine

tulemuseks 3 münti (4, 1 ja 1), mis ei ole õige, sest optimaalne lahendus on kaks kolmelist münti. Tundub, et garanteeritult õige vastuse saamiseks tuleb siiski kõik võimalused läbi vaadata.

5.2.2 Kõigi variantide läbivaatamine (rekursioon)

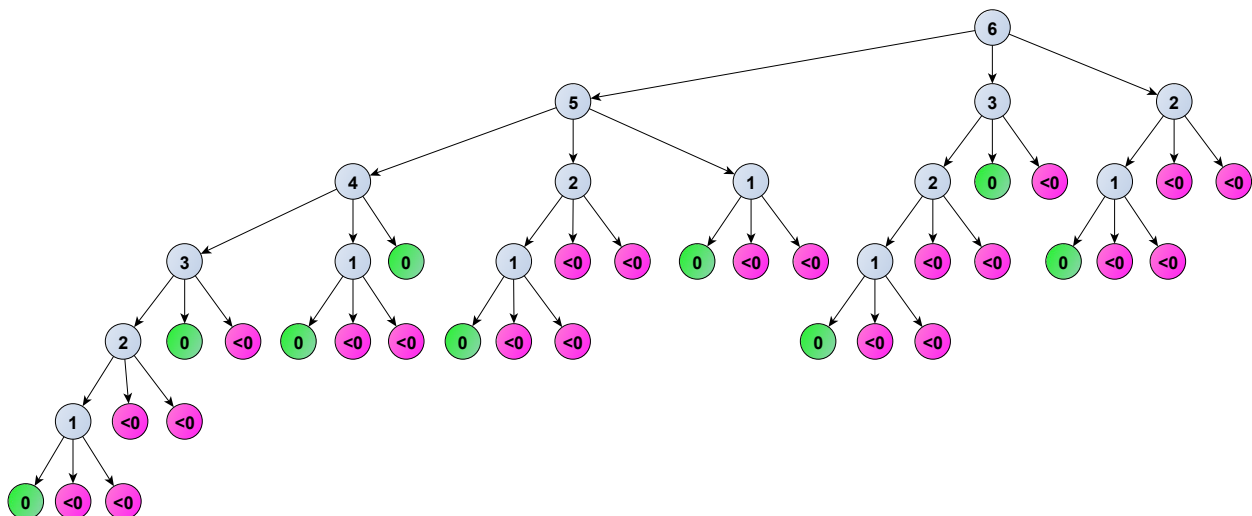
Kui teises peatükis rekursioon selgeks sai, siis antud ülesande lahendamine kõikide variantide läbivaatamise teel on lihtne. Alustame olukorrast, kus meil oleks kogu summa makstud, aga müntide arv teadmata, ning uurime, kuidas me võisime selle olukorra saavutada. Selleks oletame iga väärtuse jaoks, et viimati lisati just selle väärtusega münt, ning kordame rekursiivselt sama protseduuri. Igal järgmisel tasemel on seega üks münt vähem ja väärtuste summa selle münti väärtuse võrra väiksem. $S = 0$ jaoks on vaja võtta 0 münti. Kui aga summa läheb negatiivseks, siis sellist müntide kombinatsiooni esineda ei saa.

$$leiaKogus(S) = \begin{cases} \infty, & \text{ kui } S < 0 \\ 0, & \text{ kui } S = 0 \\ \min_{V_i \leq S} (1 + leiaKogus(S - V_i)) & \end{cases}$$

```
int leiaKogus(int S)
{
    if (S < 0) {
        return MaxInt; //tagastame "lõpmatuse"
    }
    if (S == 0) {
        return 0;
    }
    else {
        int vastus = MaxInt;

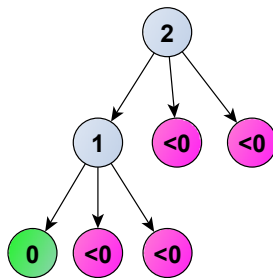
        for (int i = 0; i < N; i++) {
            vastus = min(vastus, 1 + leiaKogus(S - V[i]));
        }
        return vastus;
    }
}
```

Selline lähenemine annab küll õige vastuse, kuid on eksponentsiaalse keerukusega $O(N^S)$. Näites toodud andmetega käib programm läbi kogu järgmise puu (igas sõlmes on vastav S väärtus):



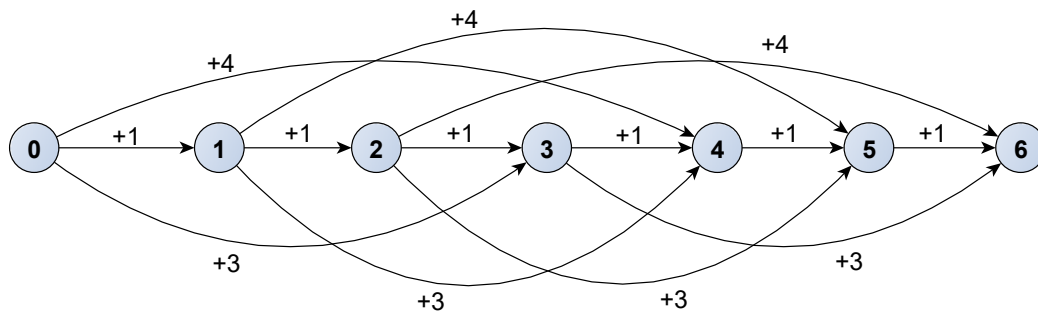
5.2.3 DP lahendus

Tähelepanelik lugeja märkab kindlasti, et selles puus mitmed harud korduvad, nt tipust 2 algavat alampuud



esineb ülaltoodud puus tervelt 4 korda! See tähendab, et tegelikult arvutab programm sama tulemust mitmeid kordi. See võiks viia mõtted kohe mälu rekursiooni ehk ülevall DP juurde. Kuna antud ülesandes järgmine samm ei sõltu sellest, kuidas eelmine tulemus saadi (iga nimiväärtusega münti on lõputul hulgal), siis piisab iga järgmise sammu leidmiseks eelmise sammu tulemuse teadmisest. Kokku on võimalikke tulemusi (müntide väärtuste summasid) 0 kuni S , kus S on otsitav summa. Niisiis piisab ülesande lahendamiseks teadmisest, kui mitu münti kulub mingi summa moodustamiseks.

Teame, et $S = 0$ korral kulub 0 münti, uurime, mis juhtub, kui siia münte lisada: saab lisada ühe münti ja saame summad 1, 3 ja 4. Nüüd uurime järgmist vähimat leitud summat ($S = 1$) ja lisame siia ühe münti (tulemuseks summad 2, 4 ja 5) jne. Seda kujutab järgmine graaf, kus tippudes on summad ning servades lisatava münti nimiväärtused:



Kuigi esmapilgul tunduvad ülaltoodud puu ja graaf üsna erinevad, on neil siiski palju ühist. Sellel puul on kõik ühe ja sama numbriga alamtipust algavad alampuud täpselt samasugused. Kui kõik sama numbriga tipud omavahel kokku viia, säilitades samas kõik servad, saame väga sarnase tulemuse üleval konstrueeritud graafiga, kus peamiseks erinevuseks on noolte suund. Samuti pole nullist alustades vaja allapoole minna (seetõttu siseneb sõlmedesse 1 ja 2 ainult üks serv, sõlme 3 kaks serva). Nii on ka visuaalselt hea aru saada, mida mõeldakse „ülevall alla“ ja „alt üles“ lähenemiste puhul. See on ka põhjus, miks me väärtuste jada üles ehitades võtame aluseks alati seni vähima uurimata väärtuse.

Funktsioon, mis leiab „alt-üles“ DP-d kasutades lahenduse:

```
int leiaKogus(int S)
{
    int kogused[S + 1];
    kogused[0] = 0;
    for (int i = 1; i <= S; i++) {
        //väärtusteks alguses „lõpmatused“, kui selle koguseni ei jõuta, siis jääb see
        //väärtus - sisuliselt tähendab see, et sellel kohal asuvat summat ei saa antud
        //müntidest moodustada
        kogused[i] = MaxInt;
    }

    for (int i = 1; i <= S; i++) {
        for (int j = 0; j < N; j++) {
            if (V[j] <= i) { //jätab vahele mündid, mille väärtus on suurem kui summa
                kogused[i] = min(kogused[i - V[j]] + 1, kogused[i]);
            }
        }
    }
    return kogused[S];
}
```

Kui sa pole alt-üles DP lähenemisega koodiga varem kokku puutunud, võib see kood paista arusaamatu. Praegune näide on üsna lihtne, kuid edaspidised lähevad keerulisemaks. Kui miski jääb segaseks, soovitan võtta siit koodi, siluris läbi käia ja kõigi vajalike muutujate väärtusi samm-sammu haaval jälgida. See on minu kogemuses parim viis DP õppimiseks.



Massiivis kogused hoitakse kõige väiksemat müntide arvu, millega senini indeksile vastav summa on saadud. Kohale 0 omistatakse väärtuseks 0, sest summa $S = 0$ saamiseks on vaja võtta 0 münti. Teistele kohtadele pannakse mingi suur väärtus, mida normaalselt maksmisel pole võimalik saada (näiteks suurem kui S – siis on kindel, et isegi ühese väärtusega müntidest ei saa vastavat väärtust kokku). Edasi vaadatakse iga summa korral läbi kõik müntide väärtused ja leitakse, kas valitud münti eelmisele seisule lisades saab vaadeldava seisu jaoks parema tulemuse, kui juba leitud on. Järgmises tabelis on toodud leitud kogused ja see, milliste müntide abil need on saadud:

Summa S	0	1	2	3	4	5	6
Kogus	0	1	2	1	1	2	2
Lisatud münti väärtus (eelmine summa)	-	1 (0)	1 (1)	3 (0)	4 (0)	4 (1)	3 (3)

Summa $S = 6$ on saadud summale $S = 3$ münti väärtusega 3 lisamise teel. Summa $S = 3$ omakorda on saadud summale $S = 0$ münti väärtusega 3 lisamise teel. Säilitades info iga summa saamiseks viimati lisatud münti väärtuse kohta, saab lahendada ka ülesandeid, kus on vaja tuua välja, mis väärtusega münte kasutati. Järgnevalt näide koodist, mis just seda teeb:

```

int leiaKogus(int S)
{
    int kogused[S + 1];
    int viimased[S + 1]; //viimasena lisatud mündi väärtus iga summa jaoks
    kogused[0] = 0;
    viimased[0] = 0;
    for (int i = 1; i <= S; i++) {
        kogused[i] = MaxInt;
        viimased[i] = 0;
    }
    for (int i = 1; i <= S; i++) {
        for (int j = 0; j < N; j++) {
            if (V[j] <= i && kogused[i - V[j]] + 1 < kogused[i]){
                kogused[i] = kogused[i - V[j]] + 1;
                viimased[i] = V[j];
            }
        }
    }
    int k = S;
    while (k > 0) {
        cout << viimased[k] << endl;
        k -= viimased[k];
    }
    return S;
}

```

Sellisel viisil läbitud tee taastamist nimetatakse **tagurdusmeetodiks** (i.k *backtracking*) ning see on sage võtte DP-d kasutavate lahenduste juures. Eelmises näites on meele peetud viimati lisatud mündi väärtus ja selle järgi saab arvutada selle mündi lisamisele eelnenud seis. Summa (=massiivi indeks) väheneb sel juhul viimasena lisatud mündi väärtuse võrra. Alternatiivina võib pidada meele ka eelmise seis indeksiks ehk eelnenud summa. Sellisel juhul tuleb arvutada kasutatud mündi väärtus, milleks on vaadeldava summa ja eelmise summa vahe.

5.3 PIKIMA KASVAVA OSAJADA LEIDMINE

Jällegi on tegemist klassikalise probleemiga, inglise keeles *the longest increasing subsequence (LIS) problem*, mis esineb sageli keerulisemate probleemide alamosana:

On antud N arvust koosnev jada. Leida selles jadas pikima kasvava osajada pikkus. Jada on kasvav siis, kui iga i korral $J_i < J_{i+1}$. Osajada elemendid ei pea olema omavahel järjest, s.t seal võib olla ka auke.

NÄIDE:

$N = 6$

Jada = [1, 3, 2, 4, 3, 7]

Vastus: 4 (näiteks [1, 3, 4, 7])

5.3.1 Kõigi võimaluste läbivaatus

Üks lahendusvõimalus on täielik läbivaatus. Seda lahendust saab näha kaasasolevas failis VP_lisad.zip nimega LIS.cpp, LIS.java või LIS.py.

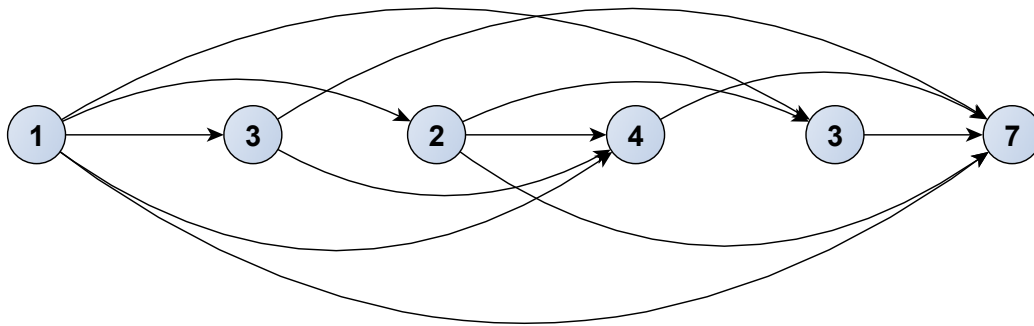
Selle lähenemise keerukus on $O(2^N)$, mis tähendab, et veidigi pikema jada korral muutub ülesanne praktiliselt lahendamatuks.

5.3.2 DP lahendus

Kui on teada pikim võimalik osajada kuni elemendini J_i , siis element J_{i+1} kas pikendab seda osajada ühe võrra või jääb pikima võimaliku osajada pikkus muutumatuks. J_{i+1} saab pikendada ainult sellist osajada, mille viimane element $J_k < J_{i+1}$.

Et seda võrratust efektiivselt kontrollida, on hea, kui teame J_k jaoks, kus $k < i + 1$, milline on pikim leitud osajada, mis lõpeb elemendiga J_k . Teiste sõnadega, hoiame mees kõiki võimalikud senised pikimad osajadad. Erinevaid võimalikke viimaseid elemente on sama palju, kui jadas elemente kokku ehk N . Üheelemendilise jada pikima kasvava osajada pikkus on muidugi 1.

Elementide lisamist illustreerib järgnev graaf:



Tippudeks on sisendjada elemendid ning kaared tähistavad, milliseid elemente saab antud elemendiga lõppevale jadale lisada. Iga serva väärtus on 1 ja vaja on leida pikim võimalik tee ühest servast teise.

Üheks võimaluseks on iga elemendi korral vaadata, millised järgmised elemendid antud elemendiga lõppevat pikimat osajada veelgi pikendavad. Kui mõni järgnev element on suurem kui vaadeldav element ja vaadeldava osajada pikendamine järgneva elemendiga annab pikema järgneva elemendiga lõppeva osajada, kui seni leitud, tähendab see, et leitud on parem vahetulemus. Siin on funktsioon, mis seda teeb:

```
int LeiaPikimPikkus(int N, int* jada) {
    int* pikkused = new int[N];
    for (int i = 0; i < N; i++) {
        pikkused[i] = 1;
    }

    int vastus = 0;
    for (int i = 0; i < N; i++) {
        vastus = max(vastus, pikkused[i]);
        for (int j = i + 1; j < N; j++) {
            if (jada[j] > jada[i]) {
                pikkused[j] = max(pikkused[j], pikkused[i] + 1);
            }
        }
    }
    return vastus;
}
```

Näitejada puhul pannakse kõigepealt igale elemendile vastavad pikima osajada pikkused võrdseks ühega, sest osajada pikkusega üks on alati võimalik. Seejärel asutakse võrdlema esimese elemendiga lõppevat osajada, milleks on [1] ja mille pikkus on 1. Nüüd käiakse tsükliliselt läbi kõik järgnevad

elemendid: antud juhul saab neid kõiki lisada vaadeldavale osajadale ja nii muudetakse pikkused 2-ks. Seejärel liigutakse järgmisele elemendile „3“ ja vaadatakse, kas sellele saab lisada järgnevaid elemente. Saame lisada elemendid 4 ja 7, nendega lõppevate osajadade pikimat pikkust muudetakse vastavalt. Järgnevas tabelis on toodud rea haaval, kuidas pikkuste jada muutub (nurksulgudes on toodud seni leitud pikim osajada; kui antud sammul on sama pikkusega alternatiivne jada, on see toodud sulgudes):

J	1	3	2	4	3	7
i = 0	1 [1]	1 [3]	1 [2]	1 [4]	1 [3]	1 [7]
i = 1	1 [1]	2 [1, 3]	2 [1, 2]	2 [1, 4]	2 [1, 3]	2 [1, 7]
i = 2	1 [1]	2 [1, 3]	2 [1, 2]	3 [1,3,4]	2[1, 3]	3 [1,3, 7]
i = 3	1 [1]	2 [1, 3]	2 [1, 2]	3 [1,3,4] ([1,2,4])	3 [1, 2, 3]	3 [1, 3, 7] ([1, 2, 7])
i = 4	1 [1]	2 [1, 3]	2 [1, 2]	3 [1,3,4]	3 [1, 2, 3]	4 [1,3,4,7] ([1,2,3,7])
i = 5	1 [1]	2 [1, 3]	2 [1, 2]	3 [1,3,4]	3 [1, 2, 3]	4 [1,3,4,7]

DP lahenduse keerukus on $O(N^2)$.

5.3.3 Tagurdusmeetodiga lahendus

Mõnikord soovitakse sarnast tüüpi ülesande vastuseks ka mõnd konkreetset osajada. Selleks on vaja pikkuse muutmisel meeles pidada viimase elemendi indeks. Osajada leidmiseks saab siis kasutada taas tagurdusmeetodit:

```
int* LeiaPikimKasvav(int N, int* S)
{
    int* pikkused = new int[N];
    int* indeksid = new int[N];

    for (int i = 0; i < N; i++) {
        pikkused[i] = 1;
        indeksid[i] = i;
    }
    int pikim = 0;
    int pikimaIndeks = 0;
    for (int i = 0; i < N; i++) {
        if (pikkused[i] > pikim) {
            pikim = pikkused[i];
            pikimaIndeks = i;
        }
        for (int j = i + 1; j < N; j++) {
            if (S[j] > S[i] && pikkused[i] + 1 > pikkused[j]) {
                pikkused[j] = pikkused[i] + 1;
                indeksid[j] = i;
            }
        }
    }

    int* osajada = new int[pikim];
    for (int i = pikim; i > 0; --i) {
        osajada[i-1] = S[pikimaIndeks];
        pikimaIndeks = indeksid[pikimaIndeks];
    }

    return osajada;
}
```

Märkus: pikima kasvava osajada ülesannet saab lahendada ka $O(n \log n)$ keerukusega, vt näiteks https://en.wikipedia.org/wiki/Longest_increasing_subsequence

5.4 KAHEMÕÖTMELINE VÄÄRTUSTE TABEL – PIKIM ÜHINE OSAJADA

On antud kaks arvuajada J1 pikkusega m ja J2 pikkusega n. Leida J1 ja J2 pikima ühise osajada pikkus.

NÄIDE:

m = 7

J1 = [1, 0, 3, 2, 5, 4, 6]

n = 6

J2 = [3, 1, 4, 0, 5, 4]

Vastus: 4 ([1, 0, 5, 4])

See on taas üks klassikaline programmeerimisülesanne, inglise keeles tuntud kui *the longest common subsequence (LCS) problem*.

5.4.1 DP lahendus

Selleks, et kasutada DP-d, on vaja kõigepealt leida, kas ülesandel on olemas väiksemad alamprobleemid, mille põhjal saaks suuremaid probleeme lahendada. Esimese ja viimase elemendi puhul on lihtne otsustada, kas see saab kuuluda pikimasse osajadasse või mitte. Seame nende jadade viimased elemendid kohakuti:

1	0	3	2	5	4	6
	3	1	4	0	5	4

On kaks võimalust – viimased elemendid on kas samad või erinevad. Kui need elemendid on erinevad, nagu toodud näites, siis vähemalt ühe jada viimane element ei sobi pikimasse ühisesse osajadasse.

Pikim ühine osajada on siis sama kui jadade

1	0	3	2	5	4
3	1	4	0	5	4

või jadade

1	0	3	2	5	4	6
		3	1	4	0	5

pikim ühine osajada.

Kindlasti tuleb proovida mõlemat võimalust, et teada saada, kumb annab parema tulemuse. Need probleemid on aga juba väiksemad kui esialgne ülesanne.

Vaatame alamülesandeid täpsemalt. Kui mõlema (alam)jada viimane element on sama, nagu näiteks:

1	0	3	2	5	4
3	1	4	0	5	4

siis see element sobib nende jadade ühisesse osajadasse, tuleb vaid leida eelnevate elementide pikim ühine osajada, eemaldades viimase elemendi:

```
1   0   3   2   5
3   1   4   0   5
```

Kuna elemente eemaldatakse ainult jada lõpust, siis jäävad jadad alati algusest kuni eemaldatava elemendini muutumatuks. Jada sellist algosa nimetatakse jada **prefiksiks**. Erinevaid prefikseid saab jadal olla sama palju, kui on jadas liikmeid (kui lugeda iga jada prefiksiks ka tühi jada, siis ühe võrra rohkem).

Nüüd tuleb veel mõelda, kust alata. Kui on antud kaks tühja jada, siis nende pikim ühine osajada on samuti tühi. Samuti, kui üks jadadest on tühi, on ka ühine osajada tühi ja pikkusega 0. Formaalset:

$$Pikkus(X_i, Y_j) = \begin{cases} 0, & \text{kui } i = 0 \vee j = 0 \\ Pikkus(X_{i-1}, Y_{j-1}) + 1, & \text{kui } X_i = Y_j \\ \max(Pikkus(X_{i-1}, Y_j), Pikkus(X_i, Y_{j-1})) \end{cases}$$

Siin on tabel, kus ridades on esimese osajada ning veergudes teise osajada kõik võimalikud prefiksid ning lahtrites on leitud nende prefiksiste pikimad ühised osajadad:

	[]	3	3, 1	3, 1, 4	3, 1, 4, 0	3, 1, 4, 0, 5	3, 1, 4, 0, 5, 4
[]	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1
1, 0	0	0	1	1	2	2	2
1, 0, 3	0	1	1	1	2	2	2
1, 0, 3, 2	0	1	1	1	2	2	2
1, 0, 3, 2, 5	0	1	1	1	2	3	3
1, 0, 3, 2, 5, 4	0	1	1	2	2	3	4
1, 0, 3, 2, 5, 4, 6	0	1	1	2	2	3	4

Täites alguses esimese rea ja veeru nullidega, saab kas mööda ridu või veerge (või ka diagonaale) liikudes täita lihtsa arvutusega kõik lahtrid: kui vaadeldavate prefiksiste viimased elemendid langevad kokku, liidetakse eelmises reas ja veerus (diagonaalis üleval vasakul) leitud pikkusele üks. Tabelis on need lahtrid tähistatud sinise taustaga. Kui aga viimased elemendid on erinevad, täidetakse lahter kas samas veerus eelmise leitud pikkusega (üleval) või samas reas eelmise leitud pikkusega (vasakul), olenevalt, kumb on suurem. Viimases lahtris on ülesande vastus.

Siin on funktsioon, mis just sel moel ülesande lahendab:

```

int leiaYhised(int* jada1, int* jada2, int p1, int p2)
{
    int** yhised = new int*[p1 + 1];
    for (int i = 0; i <= p1; i++) {
        yhised[i] = new int[p2 + 1];
    }
    for (int i = 0; i <= p1; i++) {
        yhised[i][0] = 0;
    }
    for (int j = 0; j <= p2; j++) {
        yhised[0][j] = 0;
    }
    for (int i = 1; i <= p1; i++) {
        for (int j = 1; j <= p2; j++) {
            if (jada1[i - 1] == jada2[j - 1]) {
                yhised[i][j] = yhised[i - 1][j - 1] + 1;
            }
            else {
                yhised[i][j] = max((yhised[i - 1][j]), (yhised[i][j - 1]));
            }
        }
    }
    return yhised[p1][p2];
}

```

DP lahenduse keerukus on $O(nm)$, kus n ja m on jadade pikkused.

5.5 RISTSUMMADE LOENDAMINE

Nüüd veidi keerulisem ülesanne, kus läheb tarvis kahemõõtmelist väärtuste tabelit:

Naturaalarvu ristsummaks nimetatakse selle numbrite summat. Näiteks arvu 123 ristsumma on $1 + 2 + 3 = 6$. Ülesandeks on leida, kui palju on arvust N väiksemaid arve, mille ristsumma võrdub arvu N ristsummaga (Eesti lahtiselt programmeerimisvõistluselt 2013).

NÄIDE:

$N = 123$

Vastus: 9 (Loendatavad arvud on 6, 15, 24, 33, 42, 51, 60, 105, 114).

5.5.1 Ristsummade arvutamine

Kui N ei ole kuigi suur (näiteks kuni miljon), on lihtne kõigi arvude ristsummad välja arvutada ning neid omavahel võrrelda. Edasi saab appi võtta matemaatilist trikitamist, arvestades näiteks, et arvud A ja B saavad anda sama ristsumma ainult siis, kui $|A-B|$ jagub üheksaga. Seega piisab, kui vaadata ainult iga üheksandat arvu. Lisaks leidub veel teisi mustreid, mis võimaldavad teatud arvuvahemikke vahele jätta. Nii on võimalik ülempiiri edasi venitada, näiteks miljardini. Võistlusel oli aga ülempiiriks seatud $N = 10^{18}$, mida teha?



5.5.2 DP lähenemine

Appi tuleb DP retsept. Mõtleme ülesandest arvujärkude kaupa: kõik ühelised, kõik kümnelised, kõik sajalised jne. Arvude loendamine on siis nagu rekursiivne puu läbimine: alustatakse kõrgemast järgust ja kutsutakse kümme korda välja madalamat järku, sellest kutsutakse jälle kümme korda välja järgmist järku jne. Tekib palju kordusi. Näiteks arvu 123 ristsumma arvutamisel saab teada, et arvu 23

ristsumma on 5. Seega arvu 223 ristsumma arvutamisel pole alumise kahe koha ristsummat enam vaja leida, sest see on juba teada!

Siit ka oluline tähelepanek – iga järgmise järgu juures on oluline teada ainult seda, kui palju mingi ristsumмага arve eelmises järgus oli – kuidas need täpselt leiti, ei ole oluline.

Kuidas koostada väärtuste tabelit? Kuna antud kontekstis küsitakse võimaluste koguarvu, peaks need olema ka väärtused, mida meeles hoitakse. Kuna loendamisel toimuva “rekursiooni” madalamateks tasemeteks on teadmine, mitu mingi konkreetse ristsumмага arvu väikemate arvujärkude juures oli, sobib väärtuste tabeli üheks dimensiooniks senine maksimaalne arvujärk. Teiseks dimensiooniks sobib hästi ristsumma. Erinevaid ristsummasid saab olla ainult $9 \cdot K + 1$, kus K on uuritavate arvude maksimaalne pikkus. Tabelisse paneme info, mitu mingi konkreetse ristsumмага arvu on leitud, st tabelis veerus i ja reas j on tulemus, mitu kuni i -kohalist arvu on olemas, mille ristsumma on j .

Kokkuvõttes tekib kahemõõtmeline tabel, mille üheks mõõtmeks on ristsumma väärtus ja teiseks mõõtmeks uuritava arvu maksimaalne kohtade arv.

Tabeli suurus on seega $K \times (9K + 1)$. $K = 18$ puhul teeb see 1467 lahtrit, mille täitmine on kvintiljoni arvu loendamise kõrval mõistagi tühiasi.

5.5.3 DP Exceliga

Kui tegu on ühe- või kahemõõtmelise väärtuste tabeliga, siis on vahel väga mugav kasutada tabelarvutusprogrammi, näiteks Excelit. Selles on väärtuste tabelit hea visualiseerida, samuti on olemas lihtsalt kasutatavad vahendid arvutuste realiseerimiseks, mis ühtede väärtuste alusel teisi leiavad.

Järgmiseks on toodud näide ristsumma ülesande põhjal genereeritud Exceli tabelist (originaalfail õpiku lisamaterjalides).

Veergude indeksid on arvude pikkused: esimene veerg tähistab kuni ühekohalisi arve, teine veerg kuni kahekohalisi jne. Read tähistavad ristsummasid: rida 0 ristsummat 0, rida 1 ristsummat 1 jne.

Üksikutes lahtrites on toodud, kui mitu sellise ristsumмага arvu olemas on: näiteks see, et reas 8 ja veerus 6 on arv 1287 tähendab, et on olemas 1287 kuni 6-kohalist arvu, mille ristsumma on 8 (antud kontekstis arvestame ka lühemaid arve, näiteks 125 on nagu 000125 ja loeb samuti arvuna, mille ristsumma on 8).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	1	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136	153	171
3	1	4	10	20	35	56	84	120	165	220	286	364	455	560	680	816	969	1140
4	1	5	15	35	70	126	210	330	495	715	1001	1365	1820	2380	3060	3876	4845	5985
5	1	6	21	56	126	252	462	792	1287	2002	3003	4368	6188	8568	11628	15504	20349	26334
6	1	7	28	84	210	462	924	1716	3003	5005	8008	12376	18564	27132	38760	54264	74613	100947
7	1	8	36	120	330	792	1716	3432	6435	11440	19448	31824	50388	77520	116280	170544	245157	346104
8	1	9	45	165	495	1287	3003	6435	12870	24310	43758	75582	125970	203490	319770	490314	735471	1081575
9	1	10	55	220	715	2002	5005	11440	24310	48620	92378	167960	293930	497420	817190	1307504	2042975	3124550
10	0	9	63	282	996	2997	8001	19440	43749	92368	184745	352704	646633	1144052	1961241	3268744	5311718	8436267
11	0	8	69	348	1340	4332	12327	31760	75501	167860	352595	705288	1351909	2495948	4457175	7725904	13037606	21473856
12	0	7	73	415	1745	6062	18368	50100	125565	293380	645920	1351142	2702973	5198830	9655900	17381684	30419154	51892857
13	0	6	75	480	2205	8232	26544	76560	202005	495220	1140920	2491776	5194385	10392760	20048100	37429104	67847442	119739330
14	0	5	75	540	2710	10872	37290	113640	315315	810040	1950245	4441020	9634040	20024980	40070700	77496744	145340310	265074795
15	0	4	73	592	3246	13992	51030	164208	478731	1287484	3235727	7673744	17303416	37322208	77384340	154869456	300194262	565248708
16	0	3	69	633	3795	17577	68145	231429	708444	1992925	5223647	12889383	30180423	67484067	144841275	299671971	599811969	1164986064
17	0	2	63	660	4335	21582	88935	318648	1023660	3010150	8222357	21092292	51240891	118674570	263438325	562994016	1162635441	2327376348
18	0	1	55	670	4840	25927	113575	429220	1446445	4443725	12641772	33690306	84855615	203404215	466639050	1029313296	2191458423	4518099300
19	0	0	45	660	5280	30492	142065	566280	2001285	6420700	19013852	52611780	137299435	340409720	806551350	1835047456	4025198375	8541254700
20	0	0	36	633	5631	35127	174195	732474	2714319	9091270	28012754	80439789	217386520	557149607	1362556905	3195643120	7217572751	15753515733
21	0	0	28	592	5875	39662	209525	929672	3612231	12628000	40472894	120560088	337241320	893039018	2253099975	5444285920	12654132767	28394610894
22	0	0	21	540	6000	43917	247380	1158684	4720815	17223250	57402764	177316932	513207110	1403543155	3651444300	9086074320	21722825403	55050505050
23	0	0	15	480	6000	47712	286860	1419000	6063255	23084500	79992044	256168056	766883390	2165232160	5806283700	14872309920	36557706220	85050505050
24	0	0	10	415	5875	50877	326865	1708575	7658190	30427375	109609379	363827190	1126269560	3281867680	9068126400	23900365620	60380505050	14050505050
25	0	0	6	348	5631	53262	366135	2023680	9517662	39466306	147788201	508379664	1626975480	4891539744	13922343936	37745325216	97970505050	21050505050
26	0	0	3	282	5280	54747	403305	2358840	11645073	50402935	196198211	699354228	2313440325	7174799646	21029659515	58630143456	15050505050	35050505050
27	0	0	1	220	4840	55252	436975	2706880	14033305	63412580	256600641	947732512	3240080545	10363639300	31274624245	89641329376	22850505050	50505050500
28	0	0	0	165	4335	54747	465795	3059100	16663185	78629320	330786236	1265876976	4472267215	14751050900	45822270930	1349977E+11	37050505050	85050505050
29	0	0	0	120	3795	53262	488565	3405600	19502505	96130540	420496076	1667359200	6087014635	20700766100	66182627310	2,00373E+11	5,77885E+11	14050505050
30	0	0	0	84	3246	50877	504315	3735720	22505751	115921972	527326778	2166673224	8173248070	28656627650	94282105353	2,93293E+11	8,67982E+11	2,40505050500
31	0	0	0	56	2710	47712	512365	4038560	25614639	137924380	652623158	2778823488	10831511470	39150897800	1,3254E+11	4,23579E+11	1,28612E+12	3,70505050500
32	0	0	0	35	2205	43917	512365	4303545	28759500	161963065	797362973	3518783697	14172978235	52810668925	1,83947E+11	6,03875E+11	1,88091E+12	5,58050505050
33	0	0	0	20	1745	39662	504315	4521000	31861500	187761310	962039783	4400831436	18317641615	70361427150	2,52143E+11	8,50212E+11	2,71625E+12	8,26805050500
34	0	0	0	10	1340	35127	488565	4682700	34835625	214938745	1146551153	5437773210	23391587635	92626745225	3,41488E+11	1,18263E+12	3,87498E+12	1,2081E+13
35	0	0	0	4	996	30492	465795	4782360	37594305	243015388	1350100235	6640085244	29523293215	1,20523E+11	4,5712E+11	1,62583E+12	5,46306E+12	1,74482E+13
36	0	0	0	1	715	25927	436975	4816030	40051495	271421810	1571119110	8015006143	36838945130	1,55049E+11	6,04993E+11	2,20979E+12	7,61423E+12	2,49061E+13
37	0	0	0	0	495	21582	403305	4782360	42126975	299515480	1807222010	9565627512	45456840130	1,97265E+11	7,91895E+11	2,97041E+12	1,0495E+13	3,51557E+13
38	0	0	0	0	330	17577	366135	4682700	43750575	326602870	2055195560	11290036836	55480999990	2,48274E+11	1,02542E+12	3,95001E+12	1,431E+13	4,90864E+13
39	0	0	0	0	210	13992	326865	4521000	44865975	351966340	2311031360	13180572120	66994212910	3,09181E+11	1,3139E+12	5,19773E+12	1,93074E+13	6,78159E+13



Konkreetne valem igas lahtris on sellest ühe võrra vasakul asuva kuni kümne lahtri väärtuste summa. Näiteks real 14 ja veerus 5 on arv 2710. See on saadud summeerides veerus 4 lahtrid ridadel 5 kuni 14. Arvutuse loogika on järgmine: kuni 5-kohalise arvu saamiseks, mille ristsumma on 14, on 10 võimalust:

- Esimene number on 0 ja lisanduvad kõik neljakohalised arvud ristsummaga 14.
- Esimene number on 1 ja lisanduvad kõik neljakohalised arvud ristsummaga 13.
- ...
- Esimene number on 9 ja lisanduvad kõik neljakohalised arvud ristsummaga 5.

5.5.4 Tabeli koostamine programselt

Siin on kood samasuguse tabeli moodustamiseks:

```
int tabel[100][100];

tabel[0][0] = 1;
for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 90; j++) {
        for (int k = 0; k < 10; k++) {
            tabel[i + 1][j + k] += tabel[i][j];
        }
    }
}
```

5.5.5 Ülesande lahendus (tabeli kasutamine)

Tabelist saab vaadata, mitu antud ristsummaga arvu on n-kohaliste arvude seas. Ülesandes aga tahetakse teada, kui palju on antud arvust väiksemaid, sama ristsummaga arve. Selle leidmiseks sobib järgmine rekursiivne funktsioon:

```
int loenda(int n, int ristsumma, int jark)
{
    if (n == 0)
        return 0;
    int esimene = n / pow(10, jark); //leiame esimese numbri arvus
    if (ristsumma < esimene)
        return 0;
    int aste = rint(pow(10, jark)); //n % aste annab ülejäänud kohad arvus
    int vastus = loenda(n % aste, ristsumma - esimene, jark - 1);
    for (int i = 0; i < esimene; i++) {
        vastus += tabel[jark][ristsumma - i];
    }
    return vastus;
}
```

Tabelist õige info leidmine sarnaneb tabeli koostamise põhimõttele. Näiteks kui $N = 1234$, on vaja leida kõik 1234-st väiksemad arvud, mille ristsumma on $1 + 2 + 3 + 4 = 10$. Loomulikult on kõik sellised arvud need, kus kohtade arv väiksem kui 4 ning mille ristsumma on 10. Nende arv on tabelis 3. veerus 11. reas. Kuid lisaks sobivad ka kõik neljakohalised arvud, mis on väiksemad kui 1234. Nende leidmiseks saab kasutada tabeli koostamiseks kasutatud loogikat: 1234-st väiksemad neljakohalised arvud, mille ristsumma on 10, peavad algama 1-ga ning ülejäänud kohtade ristsumma peab olema 9 ning neist moodustatud arv peab olema väiksem kui 234. Sellisteks arvudeks on kõik kahekohalised arvud, mille ristsumma on 9 (tabelis 2. veerus 10. reas), ning need ühega algavad arvud, mille ülejäänud numbrite summa on $9 - 1 = 8$ (tabelis 2. veerus 9. reas) ning need 2-ga algavad arvud, mille ülejäänud numbrite summa on 7 ning mis ei ole suuremad kui 34. See moodustub siis kõikidest ühekohalistest arvudest, mille ristsumma on 7 (ehk siis ainult 7-st); kõigist kahekohalistest arvudest, mille esimene number on 1 ja ristsumma 6 [16]; kõigist kolmekohalistest arvudest, mille esimene number on 2 ja ristsumma 5 [25]. Siin on tabel funktsiooni „Loenda“ väljakutsetega ning tagastatava vastuse kujunemisega:

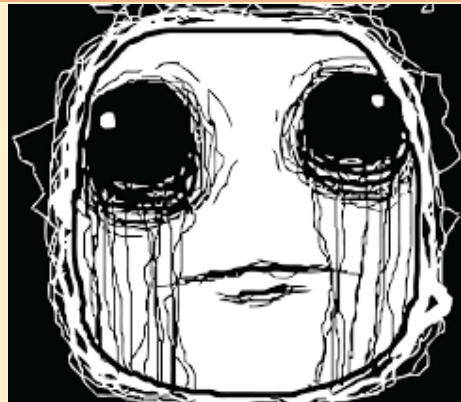
Väljakutse	Lisatavad tabeli kohad	Vastus	Näide
Loenda(1234, 10, 4)	-	85	
Loenda(1234, 10, 3)	tabel[3][10] = 63	85	Kõik kuni neljakohalised ja 1009, 1018, 1027, 1036, 1045, 1054, 1063, 1072, 1081, 1090 ja kolmekohalised: 1108, 1117, 1126, 1135, 1144, 1153, 1162, 1171, 1180 1207, 1216, 1225
Loenda(234, 9, 2)	tabel[2][9] = 10 tabel[2][8] = 9	22	Kõik vähem kui kolmekohalised: 009, 018, 027, 036, 045, 054, 063, 072, 081, 090 ja kolmekohalised: 108, 117, 126, 135, 144, 153, 162, 171, 180 Lisaks: 207, 216, 225
Loenda(34, 7, 1)	tabel[1][7] = 1 tabel[1][8] = 1 tabel[1][9] = 1	3	07, 16 ja 25
Loenda(4, 4, 0)	tabel[0][4] = 0 tabel[0][3] = 0 tabel[0][2] = 0 tabel[0][1] = 0	0	-

5.6 MITMEMÕÕTMELINE DP TABEL – MILJONÄR JA VAESLAPSED

Järgmine ülesanne on Eesti lahtiselt programmeerimisvõistluselt aastast 2015. See on kõigi aegade kõige kurvema tekstiga programmeerimisülesanne, aga selle lahendus on üle kantav paljudele teistele statistilise modelleerimise ülesannetele.

Dickensi-aegsel Inglismaal elas miljonär Mortimer. Temaga samas linnas asusid kolm lastekodu, kus elasid vaeslapsed, kellele Mortimer tavatses jõulukinke teha. Kinkide jagamise protseduur oli järgmine:

1. Iga vaeslaps saab oma lastekodust korvi, millega ta kingi järele läheb.
 2. Mortimer viskab kingitusi järjest laste sekka, mida nood oma korvidega püüavad.
 3. Iga kingitus püütakse alati kinni.
 4. Lapsed saavad kingitusi kätte juhuslikult, kuid tõenäosus, et konkreetne laps kingituse kätte saab, on võrdeline tema korvisu pindalaga.
 5. Sama lastekodu lastel on sama suurusega korvid.
 6. Kui mõni laps saab kingituse kätte, läheb ta sellega kohe lastekodusse tagasi ja rohkem püüdmises ei osale.
 7. Lapsi võib olla rohkem kui kingitusi :(
- Igal kingitusel on väärtus. Leida iga lastekodu kohta, milline on selle kodu laste poolt saadud kingituste väärtuste keskmine eeldatav summa.



NÄIDE:

Esimesest lastekodust on kohal $L_1 = 1$ laps korvi pindalaga $K_1 = 1$.

Teisest lastekodust on kohal $L_2 = 1$ laps korvi pindalaga $K_2 = 2$.

Kolmandast lastekodust on kohal $L_3 = 1$ laps korvi pindalaga $K_3 = 3$.

Mortimeril on $N = 2$ kingitust väärtustega $V = [10, 20]$.

Vastus:

L_1 : 6,666667

L_2 : 11,333333

L_3 : 12

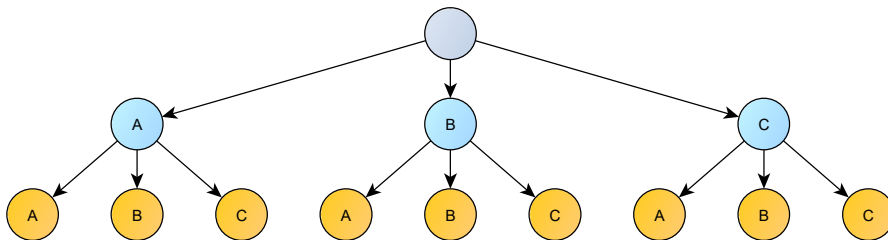
Näiteandmeid kasutades kujuneb vastus esimese lastekodu kohta järgnevalt:

- Esimese lastekodu ainus laps saab esimese kingituse kätte tõenäosusega $\frac{1}{6}$, see annab oodatavaks väärtuseks $10 * \frac{1}{6} = \frac{5}{3} \approx 1,666667$ ning teiste kingituste püüdmises ta ei osaleks.
- Tõenäosusega $\frac{1}{3}$ saab esimese kingi teise lastekodu laps ja läheb ära. Sel juhul on esimesel lapsel teise kingi saamise tõenäosus $\frac{1}{4}$. Oodatav väärtus on siis $20 * \frac{1}{4} * \frac{1}{3} = \frac{5}{3} \approx 1,666667$.
- Tõenäosusega $\frac{1}{2}$ saab esimese kingi kolmanda lastekodu laps ja lahkub. Sel juhul on esimesel lapsel teise kingi saamise tõenäosus $\frac{1}{3}$, oodatav väärtus $20 * \frac{1}{2} * \frac{1}{3} = \frac{10}{3} \approx 3,333333$.

Kokku tulebki vastuseks $\frac{5}{3} + \frac{5}{3} + \frac{10}{3} = \frac{20}{3} \approx 6,666667$. Samasugust arutlust saab kasutada ka teiste lastekodude jaoks.

5.6.1 Kõikide võimaluste läbivaatus

Tõenäosus, et kingi saab kätte mingi konkreetse lastekodu laps, sõltub kogu selle lastekodu püüdmisel osalevate korvide pindala suhtest kõigi veel püüdmisel osalevate korvide pindalasse. Iga kord, kui kink kätte saadakse, väheneb ühe lastekodu korvide kogupindala, kuna kingi saanud laps lahkub koos oma korviga. Seetõttu väheneb ka kõigi korvide pindala. Kinkide püüdmisele vastab järgmine puu:

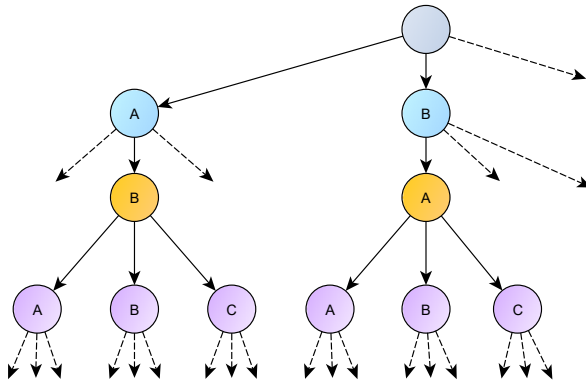


A, B ja C tähistavad erinevaid lastekodusid, esimene tase (sinised tipud) esimest kingitust, teine tase (oranžid tipud) teist kingitust. Igas tipus on sellele lastekodule vastav täht, kes vastava taseme kingituse sai. Kõikide variantide läbivaatamise keerukus sõltub kinkide arvust N ja on $O(3^N)$.

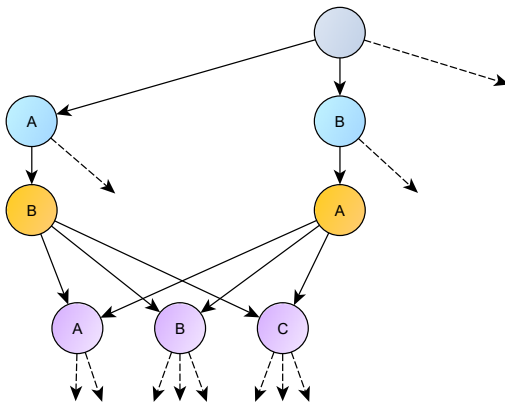
5.6.2 Korduvad harud

Kas läbivaatuste puus hakkavad mingid osad korduma? Lapsed saavad ükshaaval kinke ning lahkuvad seejärel. i -nda kingi jagamise ajaks on lahkunud i last ja lastekodude ning ka kogu korvide pindala on

vähenenud vastavalt sellele, milliste lastekodude lapsed olid juba lahkunud. Samas ei sõltu allesjäänud laste ja korvide arv sellest, millises järjekorras eelmised lapsed oma kinke said. Siin on alamosa sügavamast rekursioonipuust:



Ülesande seisukohast on pärast oranži taset toimuv kinkide jagamine täpselt sama:



5.6.3 DP tabeli koostamine

Kui kõik kingid on jagatud, siis kingi saamise tõenäosus on 0. Kui on alles viimane kink, siis tõenäosus, et selle saab mõni esimese lastekodu laps, on esimese lastekodu veel kohale jäänud laste arv LJ_1 korrutatud esimese lastekodu korvi pindalaga K_1 ning jagatud kõikide allesjäänud korvide pindalaga. Et seda leida, on vaja teada, mitu last igast lastekodust alles on. Hea on koostada kolmemõõtmeline massiiv, kus iga mõõde vastab ühe lastekodu kohal olevate laste arvule. Näiteks, kui on vaja jagada 5 kinki ja lastekodudest on kohal vastavalt 2, 5 ja 1 last, siis viimase kingi jagamise ajaks võib olla esimesest lastekodust alles 0 kuni 2 last, teisest 1 kuni 4 last ja kolmandast 0 või 1 last. Kokku on järel $2 + 5 + 1 - 4 = 4$ last. Järgnevas tabelis on toodud viimase kingi jagamisest saadavad keskmised eeldatavad summad esimese lastekodu jaoks:

A	B	1	2	3	4	5
	C					
0	0	-	-	-	0	-
	1	-	-	0	-	-
1	0	-	-	$\frac{K_1}{K_1 + 3 * K_2} * V$	-	-
	1	-	$\frac{K_1}{K_1 + 2 * K_2 + K_3} * V$	-	-	-
2	0	-	$\frac{2 * K_1}{2 * K_1 + 2 * K_2} * V$	-	-	-
	1	$\frac{2 * K_1}{2 * K_1 + K_2 + K_3} * V$	-	-	-	-

A, B ja C tähistavad vastavalt esimesest, teisest ja kolmandast lastekodust alles jäänud laste arvu. K_1 , K_2 ja K_3 tähistavad vastavalt lastekodude korvi pindalasid ning V viimase kingituse väärtust.

Kuidas leida, milline on oodatav kinkide väärtus üks käik enne seda, näiteks juhul, kui alles on kaks last esimesest, kaks teisest ja üks kolmandast lastekodust? Siis on 3 võimalust, millise lastekodu laps kingi saab:

1. Kingi saab esimese lastekodu laps ning pärast seda on situatsioon, kus lapsi on järel $A = 1$, $B = 2$ ja $C = 1$ (tabelis oranži taustaga lahter).
2. Kingi saab teise lastekodu laps ning pärast seda on lapsi järel $A = 2$, $B = 1$ ja $C = 1$ (tabelis rohelise taustaga lahter).
3. Kingi saab kolmanda lastekodu laps ning pärast seda on lapsi $A = 2$, $B = 2$ ja $C = 0$ (tabelis sinise taustaga lahter).

Tähistagu kingi saamise tõenäosusi antud sammul vastavalt T_A , T_B ja T_C ja parasjagu jagatava kingituse väärtust VP , siis

$$E(A_i, B_j, C_k) = \begin{cases} 0, & \text{kui } i = 0, j = 0, k = 0 \\ T_A * VP + T_A * E(A_{i-1}, B_j, C_k) + T_B * E(A_i, B_{j-1}, C_k) + T_C * E(A_i, B_j, C_{k-1}) \end{cases}$$

Ülesandes tahetakse vastust kõigi kolme lastekodu jaoks, seega tuleb teha 3 tabelit: üks iga lastekodu jaoks.

5.6.4 Lahendus

Siin on funktsioon, mis antud ülesande lahendab:

```

int kingid(int La, int Lb, int Lc, int Ka, int Kb, int Kc, int N, int* kingid)
{
    double**** kv = new double****[La + 1]; //1. lastekodust kohalolevate laste arv
    for (int i = 0; i <= La; i++) {
        kv[i] = new double**[Lb + 1]; //2. lastekodust kohalolevate laste arv
        for (int j = 0; j <= Lb; j++) {
            kv[i][j] = new double*[Lc + 1]; //3. lastekodust kohalolevate laste arv
            for (int k = 0; k <= Lc; k++) {
                kv[i][j][k] = new double[3]; // lastekodu
            }
        }
    }

    int lapsiKokku = La + Lb + Lc;
    //Alustame sellest, kui kõik lapsed on lahkunud kuni selleni, kui kõik veel alles
    for (int al = lapsiKokku - N; al <= lapsiKokku; al++) {
        int m1 = min(La, al); //nii palju saab maksimaalselt olla alles 1. lastekodust
        for (int i = 0; i <= m1; i++) {
            //nii palju saab olla 2. lastekodust, kui esimesest on alles i last:
            int m2 = min(Lb, al - i);
            for (int j = 0; j <= m2; j++) {
                if (al - i - j <= Lc) {
                    int k = al - i - j; //nii palju on 3. lastekodust kohal
                    kv[i][j][k][0] = 0;
                    kv[i][j][k][1] = 0;
                    kv[i][j][k][2] = 0;
                    if (al == lapsiKokku - N) {
                        continue; //kedagi ei ole, jätkame
                    }
                    double p1 = i * Ka;
                    double p2 = j * Kb;
                    double p3 = k * Kc;
                    double kp = p1 + p2 + p3; //alles korvide kogupindala
                    double t1 = p1 / kp; //Tõenäosus, et kingi saab 1. lastekodu
                    double t2 = p2 / kp; //2. lastekodu
                    double t3 = p3 / kp; //3. lastekodu
                    int jk = kingid[lapsiKokku - al]; //jagatava kingi väärtus

                    if (i != 0) {
                        kv[i][j][k][0] += t1 * (jk + kv[i - 1][j][k][0]);
                        kv[i][j][k][1] += t1 * kv[i - 1][j][k][1];
                        kv[i][j][k][2] += t1 * kv[i - 1][j][k][2];
                    }
                    if (j != 0) {
                        kv[i][j][k][0] += t2 * kv[i][j - 1][k][0];
                        kv[i][j][k][1] += t2 * (jk + kv[i][j - 1][k][1]);
                        kv[i][j][k][2] += t2 * kv[i][j - 1][k][2];
                    }
                    if (k != 0) {
                        kv[i][j][k][0] += t3 * kv[i][j][k - 1][0];
                        kv[i][j][k][1] += t3 * kv[i][j][k - 1][1];
                        kv[i][j][k][2] += t3 * (jk + kv[i][j][k - 1][2]);
                    }
                }
            }
        }
    }

    cout << fixed << setprecision(10) << kv[La][Lb][Lc][0] << endl <<
        kv[La][Lb][Lc][1] << endl << kv[La][Lb][Lc][2] << endl;
    return 0;
}

```

Selles lahenduses on DP tabeli dimensioonideks kasutatud küll lastekodulaste arvu, kuid tegelikud arvutused toimuvad ainult laste arvu ja kinkide arvu vahe osas (täpsemalt võiks mõõtmeteks olla nt lahkunud laste arv, mis saab maksimaalselt võrduda kinkide arvuga). Seega on kogu selle lahenduse keerukus $O(N^3)$.

5.7 TÕEVÄÄRTUSTE TABELIGA DP - ÕIGLANE JAGAMINE

See ülesanne on teisend klassikalisest seljakoti pakkimise ülesandest. Seljakoti pakkimisest tuleb hiljem rohkem juttu kaheksandas peatükis – DP edasijõudnutele.

Kaks venda, Albert ja Benno, tahavad jagada omavahel hulka kingitusi. Iga kingituse peab andma kas Albertile või Bennole; kingitusi poolitada ei saa. Igal kingitusel on väärtus. A ja B tähistavad vastavalt Albertile ja Bennole antud kingituste kogusummat. Minimeerida vahe A - B absoluutväärtus. Kingituste arv N ei ületa 100. Kingituste väärtused on positiivsed täisarvud, mis ei ületa 200.

NÄIDE:

N = 4

V = [2, 3, 4, 7]

Vastus: 2 (näiteks jaotused [2, 7] ja [3, 4] või [7] ja [2, 3, 4])

5.7.1 Kõik kombinatsioonid

Esimese hooga tuleb mõtte vaadata läbi kõik kombinatsioonid. Kuna kinkide jagamisel tekkinud vahe on sümmeetriline (st ei ole vahet, kas konkreetse komplekti kingitusi saab Albert ja ülejäänud jääb Bennole või vastupidi), siis piisab pooltest võimalustest. Näiteandmete jaoks on need toodud järgmises tabelis:

Alberti kinkide väärtused	2, 3, 4, 7	3, 4, 7	2, 4, 7	2, 3, 7	2, 3, 4	4, 7	3, 7	3, 4
Benno kinkide väärtused	0	2	3	4	7	2, 3	2, 4	2, 7
Vahe	16	12	10	8	2	6	4	2

Tundub hea plaan. Erinevaid kombinatsioone kinkide jagamiseks on 2^{N-1} . Kui N on väike, nagu antud näites, siis töötab kõik suurepäraselt, kuid kui N = 100, teeb see juba 2^{99} erinevat kombinatsiooni ja see on isegi masinale liiga palju tööd.

5.7.2 DP lahendus

Eelmises tabelis on näha, et üks kinkide jaotamisel tekkinud väärtuste vahe kordub. Kuna kinkide lubatud väärtus on kuni 200, siis nii väikese kinkide arvu juures nagu näites (N = 4) see ei pruugi ilmselgelt olla, kuid suurema arvu kinkide jagamisel ilmselt peavad hakkama vahed korduma, sest kõigi võimalike vahede arv on fikseeritud. Näiteks maksimumsisendi juures, kus kinke on 100 ja igaüks väärtusega 200, siis juhul kui Albert ahnitseb kõik kingitused endale, on Alberti kingituste koguväärtus kõigi kingituste väärtuste summa, mis on maksimaalselt $100 * 200 = 200\,000$. Benno kingituste koguväärtus on loomulikult 0, seega maksimaalne vahe saab olla 200 000. See tähendab aga, et nende kingituste ükskõik millisel jagamisel ei saa tekkida rohkem kui 200 001 erinevat vahet (0 sobib ka). Pane tähele: see kehtib seetõttu, et kingituste väärtused ja seega ka vahed saavad olla ainult täisarvud, reaalarvuliste väärtuste korral saab sellist lähenemist kasutada ainult teatud tingimustel.

Näites on kingituste kogusumma $2 + 3 + 4 + 7 = 16$, järelikult võimalikud vahed on $0 \dots 16$. Kuidas muutuvad vahed kingituste jagamise käigus?

Kui jagatakse ainult üks kingitus, saab vaheks olla ainult selle kingituse väärtus (tabelis '+' tähistab võimalikku vahet, '-' võimatut. Reas on kingituse nr(= väärtus)):

Vahe	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Kui jagatakse järgmine kingitus, siis kõik võimalikud vahed saavad kas kasvada või kahaneda täpselt selle kingituse väärtuse võrra, nt kui esimese kingituse väärtusega 2 sai Albert ja teise, väärtusega 3, saab samuti Albert, on vahe $2 + 3 = 5$. Kui teine kink läheb aga Bennole, on vahe $|2 - 3| = 1$.

Vahe	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-

Kolmanda kingituse jagamisel saab vahe muutuda 4 võrra, sest see on kolmanda kingituse väärtuseks. Kui enne sai olla väärtuste vahe kas 1 või 5, siis nüüd on võimalikud vahed:

- $1 + 4 = 5$,
- $|1 - 4| = 3$,
- $5 + 4 = 9$ ja
- $5 - 4 = 1$.

Samamoodi saab arvutada kõik võimalikud vahed ka viimase kingituse jaoks. Siin on kogu täidetud tabel näites toodud 4 kingi jaoks:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-
3	-	+	-	+	-	+	-	-	-	+	-	-	-	-	-	-	-
4	-	-	+	-	+	-	+	-	+	-	+	-	+	-	-	-	+

Siin on ka võimalus kokkuhoidliku DP kasutamiseks: iga kord kasutatakse järgmise rea arvutamiseks ainult eelmisel real olevat infot – see tähendab, et kui ülesande vastuseks soovitakse ainult minimaalset vahet, piisab tegelikult kahe reaga tabelist: vaja on meeles pidada eelmine rida ning käesolev rida.

Järgmine funktsioon illustreerib kokkuhoidlikku DP lahendust:

```

int LeiaVahe(int mituPakki, int* pakid)
{
    int suurimVahe = 0;
    for (int i = 0; i < mituPakki; i++) {
        suurimVahe += pakid[i]; //Leiame suurima võimaliku vahe
    }
    bool** tabel = new bool*[suurimVahe + 1];
    for (int i = 0; i <= suurimVahe; i++) {
        tabel[i] = new bool[2]; //kokkuhoidliku DP jaoks on vaja vaid 2 rida
    }
    tabel[0][1] = 1; //kui midagi jagada pole, on vahe 0
    for (int i = 1; i <= suurimVahe; i++) {
        tabel[i][1] = 0; //ülejäänud vahed on võimatud
    }
    for (int i = 0; i < mituPakki; i++) {
        int praeguneRida = i % 2;
        int eelmineRida = (i + 1) % 2;
        for (int j = 0; j < suurimVahe; j++) {
            int eelmineVahe = j - pakid[i];
            if (eelmineVahe < 0){
                eelmineVahe = -eelmineVahe;
            }
            if (tabel[eelmineVahe][eelmineRida]) {
                tabel[j][praeguneRida] = 1;
                continue;
            }
            eelmineVahe = j + pakid[i];
            if (eelmineVahe > suurimVahe || !tabel[eelmineVahe][eelmineRida]) {
                tabel[j][praeguneRida] = 0;
                continue;
            }
            tabel[j][praeguneRida] = 1;
        }
    }
    int vastus;
    for (int i = 0; i < suurimVahe; i++) {
        int praeguneRida = (mituPakki - 1) % 2;
        if (tabel[i][praeguneRida]) {
            vastus = i; //leiame esimese võimaliku vahe, see ongi vastuseks
            break;
        }
    }
    return vastus;
}

```

5.8 BITIMASKIDE PÕHINE VÄÄRTUSTE TABEL - MOEKUNSTNIK JA KOILIBLIKAS

Järgmine ülesanne on samuti Eesti lahtiselt programmeerimisvõistluselt aastast 2013. Ülesande originaaltekst kuulub selgelt fantastika valdkonda, kuid samal põhimõttel saab lahendada mitmesuguseid mänguteooria või riskianalüüsi situatsioone.

Moekunstnik Märdil on kapis N ilusat salli, millega ta käib laupäeviti moekunstnike koosolekul. Igal sallil on oma moeväärtus ja koosolekul saab Märt vastava hulga feimi. Kaks korda sama salliga kohale tulla oleks suur *faux pas* ja Märt ei tee seda kunagi. Kantud sallid paneb ta kappi tagasi, aga rohkem neid ei kanna. Märdi kapis elab ka koiliblikas Kärt, kes sööb igal pühapäeval ühele sallile augu sisse. Auguga salli enam kanda ei saa. Kärt moeväärtusest ei hooli, vaid sööb selle juhuslikult. Mingile sallile augu söömise tõenäosus on võrdeline salli pikkusega. Kärt võib sama salli süüa ka mitu korda. On selge, et Märt saaks koosolekul käia ülimalt N korda, kui Kärt sööks ainult juba kantud salle. Kui Kärt sööb mõnikord ka kandmata salle, jääb koosolekute arv sellevõrra väiksemaks. Kirjutada programm, mis leiab, kui palju Märt keskmiselt feimi saab, kui ta kasutab parimat võimalikku strateegiat, aga Kärt sööb salle juhuslikult. Tegevus algab nädala alguses, seega esimese koosoleku ajaks on kõik sallid veel terved. Sisendina on antud kõigi sallide pikkused ja moeväärtused.

Sisendi esimesel real on sallide moeväärtused ja teisel real nende pikkused.

NÄIDE:

Pikkused: 3 2 1

Moeväärtused: 10 20 30

Vastus:

48,333333

(Pildil Märt, Kärt ja ülesande autor Targo).



Näite selgitus: Antud näites on Märdi optimaalne strateegia võtta kõigepealt teine sall. Pärast seda on:

- tõenäosusega $\frac{1}{2}$ auk esimeses sallis, teiseks koosolekuks alles ainult kolmas sall ja saame kokku 50 feimi;
- tõenäosusega $\frac{1}{3}$ auk teises sallis ja kaks salli alles; võtame neist kõigepealt kolmanda ning $\frac{1}{2}$ tõenäosusega saame ka esimese ära kanda; keskmiselt kokku 55 feimi; tõenäosusega $\frac{1}{6}$ auk kolmandas sallis, alles ainult esimene ja saame kokku 30 feimi.

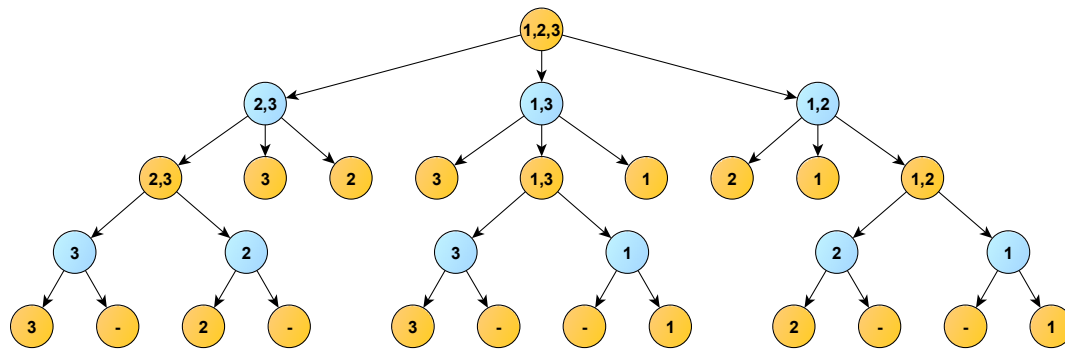
Nende variantide kaalutud keskmine on $48\frac{1}{3}$ feimi.

5.8.1 Kõigi läbivaatuste puu

Nagu ikka, alustame jõumeetodi analüüsiga.

Alguses on Märdil valida kõikide sallide vahel. Ta valib neist ühe ning järgmiseks korra jääb N - 1 valikut. Nüüd sööb Kärt ühele sallile augu sisse. Kui Kärt valis einestamiseks juba kantud salli, siis

Märdi valikud ei muutu. Kui aga Kärt sõi mõne kandmata salli, jääb Märdile $N - 2$ valikut jne. Siin on puu kolme salli jaoks:



Numbrid tippudes näitavad, millised sallid on veel Märdile kasutatavad. Oranžides ringides on Märdi käikudele, sinistes Kärdi käikudele eelnev seis.

Märt saab igal sammul valida järelejäänud sallide hulgast – Kärt valib küll kõigi sallide hulgast, kuid tegelikult pole ülesande seisukohalt oluline, millise katkestest või kantud sallidest Kärt valib, neid võib vaadelda ühe juhuna. Sellisel juhul kõikide variantide läbivaatamisega lahenduse keerukus on $O((N!)^2)$. Seega juba $N = 20$ juures tuleb teha $(20!)^2 = 5,919 \cdot 10^{36}$ läbivaatust. Arvestades 10^8 läbivaatust sekundis, kulub $2 \cdot 10^{21}$ aastat!

5.8.2 Tee DP poole

Kui vaadata hoolega eespool joonistatud puud, pole raske märgata, et puus hakkavad harud korduma. Sisuliselt tähendab see seda, et kui mingil hetkel on Märdil valida täpselt samade sallide vahel, siis on tal tark täpselt samamoodi käituda ning ta saab keskmiselt samapalju feimi juurde eelnevalt kogutule: nii näiteks kolmanda päeva valik ei sõltu enam sellest, mida Märt ja Kärt esimesel kahel päeval ette võtsid, vaid ainult sellest, millised sallid veel selleks ajaks järel on. Piisab, kui samasuguseid alampuid arvutame vaid ühel korral, edaspidi võib kasutada meelde jäetud väärtust. Seega võiks mõtte minna DP lahenduse peale. Erinevaid seise on sama palju kui parajasti alles olevate sallide võimalikke kombinatsioone ehk 2^N . Iga kombinatsiooni kohta on vaja meeles pidada parim tulemus. Kuidas aga järjestada kombinatsioone?

5.8.3 Bitimaskide kasutamine tabeli indeksitena

Siin on olukord, kus süsteemi võimalikud olekud on kirjeldatavad bitimaskidena: mingi hulk objekte on parajasti kas aktiivsed või mitteaktiivsed. N objekti puhul on selliseid võimalusi 2^N . Sellisel juhul on kaval luua 2^N elemendiga massiiv, kus iga element tähistab üht võimalikku kombinatsiooni. Kui massiivi indeksid on 0 kuni $2^N - 1$, annab see meile ka loomuliku viisi kombinatsioonide järjestamiseks: iga massiivi element tähistab parajasti nende objektide olemasolu, millele vastavad bitid on indeksi kahendesituses ühed.

Näiteks nelja objekti puhul tähendaks see järgmist tabelit:

Massiivi indeks	Kahendesitus	Allesolevad objektid
0	0000	Mitte ühtegi
1	0001	1.
2	0010	2.
3	0011	1. ja 2.
4	0100	3.
5	0101	1. ja 3.
6	0110	2. ja 3.
7	0111	1., 2. ja 3.
8	1000	4.
9	1001	1. ja 4.
10	1010	2. ja 4.
11	1011	1., 2. ja 4.
12	1100	3. ja 4.
13	1101	1., 3. ja 4.
14	1110	2., 3. ja 4.
15	1111	Kõik objektid

Madalamad bitid tähistavad esimesi objekte, sest siis me ei piira objektide koguarvu – kui neid on rohkem, saab võtta lihtsalt pikema arvu.

Tavapäraste operatsioonide sooritamiseks on kasulikud bitikaupa tehked:

Operatsioon	Operaator	Näide kahendesituses	Näide kümnendesituses
Bittide nihutamine vasakule. $M \ll N$ on ekvivalentne tehkega $M * 2^N$.	$\ll$	$0101 \ll 2 == 010100$	$5 \ll 2 == 20$
Bitikaupa korrutamine	$\&$	$0101 \& 0111 == 0001$	$5 \& 7 == 1$
Bitikaupa liitmine	$ $	$0101 0011 == 0111$	$5 3 == 7$
Bitikaupa eitus	$\sim$	$\sim 0101 == 1010$	Vastus oleneb arvu pikkusest
Bitikaupa välistav või	$\wedge$	$0101 \wedge 0011 == 0110$	$5 \wedge 3 == 6$

Nende abil saab teha järgmisi operatsioone väga efektiivselt:

- Kõigi kombinatsioonide arvu (2^N) leidmine.

$$1 \ll N$$

- Konkreetselt objektile vastava arvu leidmine. Tehe tagastab kahendarvu, kus altpoolt m. bitt on 1 ja teised bitid nullid.

$$1 \ll (m - 1)$$

- Kombinatsiooni arvu leidmine, kus on olemas kõik objektid peale ühe. Tegemist on eelmise tehke tulemuse bittide ümberpööramisega.

$$\sim(1 \ll (m - 1))$$

- Tsükkel, mis genereerib üksikutele objektidele vastavaid kahendarve (sisuliselt kahe astmeid):

```
for (int mbit = 1; mbit <= max; mbit *= 2)
```

- Kontroll, kas kombinatsioon i sisaldab objekti number m. Kui kombinatsioonile vastavat arvu bitikaupa korrutada ühele objektile vastava arvuga, on vastus positiivne parajasti siis, kui kombinatsiooni arvus on ka m. bitt seatud.

```
i & 1 << (m - 1) > 0
```

- Objekti eemaldamine kombinatsioonist (täpsemalt, kombinatsiooni i alusel sellise kombinatsiooni leidmine, kus puudub objekt m).

```
i & ~(1 << (m - 1))
```

- Objekti lisamine kombinatsioonile (täpsemalt, indeksi i alusel sellise indeksi leidmine, kuhu on lisatud objekt m):

```
i | 1 << (m - 1)
```

- Objekti sisse-väljalülitamine (lihtsustab koodi juhul, kui me juba ette teadsime, kas see objekt oli enne aktiivne või ei):

```
i ^ 1 << (m - 1)
```

Selline lähenemine on arvutuslikult üliefektiivne. Konkreetsetele objektikombinatsioonidele vastavate massiivi indeksite leidmine ja nendega opereerimine taanduvad täisarvutehetele, mille jaoks kaasaegsetes protsessorites on hästi optimeeritud meetodid ja mille täitmist mõõdetakse üksikutes nanosekundites.

5.8.4 DP lahendus

Siin on põhjalike kommentaaridega funktsioon, mis antud ülesandele vastuse leiab:

```
double leiaFeim(int N, int* moevaartused, int* pikkused)
{
    // võimalike kombinatsioonide arv, millised sallid võivad söödud olla = 2^n
    int kombinatsioonid = 1 << N;
    int kogupikkus = 0;
    for (int i = 0; i < N; i++) {
        kogupikkus += pikkused[i];
    }
    // Siin massiivis hoiame parimaid võimalikke tulemusi, mida on võimalik saavutada,
    // kui meil on alles mingi konkreetne kombinatsioon sulle.
    // Massiivi indeksid vastavad sallikombinatsioonide kahendarvulistele väärtustele.
    // Näiteks feimid[37] sisaldab maksimaalset võimalikku väärtust, mida Märdil on
    // võimalik saavutada juhul, kui alles on 1., 3. ja 6. sall (sest 10-süsteemis 37
    // on kahendsüsteemis 00100101, seatud on altpoolt lugedes 1., 3. ja 6. bitt)
    double* feimid = new double[kombinatsioonid];
    feimid[0] = 0;
```

```

// konstrueerime kõik kombinatsioonid alates lihtsamatest
for (int i = 1; i < kombinatsioonid; i++) {
    feimid[i] = 0;

    // Proovime uue kombinatsiooni üles ehitada väiksematest.
    // mvalik on Märdi tehtav valik, st mitmenda salli ta võtab.
    // Proovime kõiki Märdi valikuid
    for (int mvalik = 0, mbit = 1; mbit <= i; mbit *= 2, mvalik++) {
        // bitt pole seatud, st vaadeldav kombinatsioon ei sisalda seda salli
        if ((mbit & i) == 0){
            continue;
        }
        double alles = 1; // tõenäosus, et ühtki head salli ei sööda ära
        double feim = moevaartused[mvalik]; // Märdi käigu väärtus
        // kvalik on Kärdi tehtav valik, st mitmenda salli sisse ta augu sööb
        for (int kvalik = 0, kbit = 1; kbit <= i; kbit *= 2, kvalik++) {
            if ((i & kbit) == 0 || mvalik == kvalik) continue;
            double prob = (double)pikkused[kvalik] / kogupikkus;
            // väärtusele lisandub selle allesjääva kombinatsiooni väärtus, kust
            // on eemaldatud Märdi ja Kärdi sall
            feim += prob * feimid[i ^ mbit ^ kbit];
            alles -= prob;
        }
        // Kui Kärt ei söönud ühtki head salli ära, lisandub selle kombinatsiooni
        // väärtus, kus on kõik sallid peale Märdi praeguse valiku
        feim += alles * (feimid[i ^ mbit]);
        if (feim > feimid[i]) {
            feimid[i] = feim;
        }
    }
}
return feimid[kombinatsioonid - 1];
}

```

5.9 KUIDAS JA MILLAL DP-D KASUTADA

Eelnevalt toodud näidetest selgus, et DP aitab algoritmi keerukust sageli dramaatiliselt parandada:

Ülesanne	Jõumeetodiga keerukus	DP keerukus
Fibonacci	$O(\varphi^N)$	$O(N)$
Optimaalne maksmine	$O(2^N)$	$O(N)$
Pikim kasvav osajada	$O(2^N)$	$O(N^2)$
Pikim ühine osajada	$O(2^{N+M})$	$O(NM)$
Ristsumma	$O(N)$	$O(\log N)$
Õiglane jagamine	$O(2^N)$	$O(N \cdot \log(W))$, kus W on kingi maksimaalne väärtus
Miljonär ja vaeslapsed	$O(3^N)$	$O(N^3)$
Moekunstnik ja koiliblikas	$O(N!^2)$	$O(2^N)$

Kuidas aga ära tunda ülesandeid, kus on võimalik DP-d kasutada?

- Probleemi saab jagada väiksemateks alamprobleemideks.
- Suurema probleemi lahenduse saab leida ühe või enama alamprobleemi lahenduse põhjal.
- Alamprobleemid on järjestatavad.

- Suurema probleemi lahendus sõltub ühest või enamast alamprobleemi lahendusest, kuid mitte sellest, kuidas need lahendused täpselt saadi.

Tavalisi DP ülesannete struktuure:

Sisend	DP tabelis kasutatav(ad) väärtus(ed)	Üleminek	Näiteid
Jada $x_1, x_2, \dots, x_n$	Jada element i	Vali jada esimesed $i-1$ elementi, töötle i , lisa i eelmisele seisule või mitte.	Pikim kasvav osajada
Kaks jada $x_1, x_2, \dots, x_n$ ja $y_1, \dots, y_m$	Jadade elementide paar i (esimesest jadast) ja j (teisest jadast)	Nagu eelmine, aga muuda üht indeksit korraga.	Pikim ühine osajada
Jada $x_1, \dots, x_i, \dots, x_j, \dots, x_n$	Alamjada $x_i, \dots, x_j$	Jaga lõik $i, \dots, j$ lõikudeks $i, \dots, k+1, \dots, j$	Maatriksite jada korrutamine (vt peatükk 8)
Suunatud tsükliketeta graaf	Tipp graafis	Töötle vaadeldava tipu naabrid	Pikimate või lühimate teede leidmine graafis, teede loendamine
Piirav arvuline väärtus	Piiriväärtusest väiksem (või suurem) arv	Suurenda (või vähenda) vaadeldavat väärtust kuni jõuad piiriväärtuseni.	Seljakotiülesanne, mitmesugused maksmisülesanded
Objektide hulk	Objektide alamhulk (tavaliselt bitimaskina)	Lülita alamhulgas elemente sisse-välja	

5.9.1 Ült alla vs alt üles DP

	Ült alla	Alt üles
Plussid	- Loomulik järgmine samm täisläbivaatusega lahendusest. - Leiab alamvastused ainult siis, kui see on vajalik (mõnedes olukordades kiirem).	- Kiirem juhul kui paljusid alamprobleeme „külastatakse“ korduvalt - Võimaldab mälu kokkuhoidu, kui kasutame „kokkuhoidliku DP“ trikki.
Miinused	- Rekursiivsel funktsioonide väljakutsumisel on lisakulu. - Kõigi olekute meelespidamine võib põhjustada mäluprobleeme.	- Lugejale, kes pole DP-ga tuttav, on koodi intuitiivne mõistmine raskem. - Leiab kõik alamvastused ka juhul, kui see pole otseselt vajalik.

5.9.2 Kidunud DP

Vahel on võimalik leida DP ka olukordades, kus seda esmapilgul ei ole. Olgu ülesandeks näiteks:

Leida arvude jadas maksimaalne element.

Loogiline ja intuitiivne lahendus on meeles pidada, mis on seni leitud maksimaalne element, võrrelda seda kõigi järgmiste elementidega ning kui leiame suurema, siis meeles peetud element selle suurema väärtuse vastu vahetada. Samas võib ka sellest algoritmist mõelda kui DP algoritmist, kus „tabel“ koosnebki sellest samast seni leitud maksimaalsest elemendist. Alternatiivne „jõumeetodiga“ lahendus samal ülesandele oleks iga elemendi jaoks proovida, kas ta on kõigist teistest suurem või ei: $O(N^2)$ ajalise keerukusega algoritm.

5.10 DP PRAKTILISED KASUTUSALAD

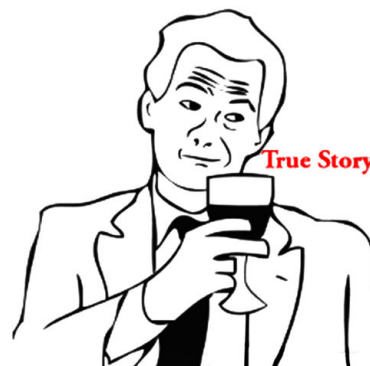
Me kasutame sageli tööriistu, mis sisaldavad endas ühte või teist DP-l põhinevat algoritmi.

Mõned huvitavad DP kasutusala:

- diff (https://en.wikipedia.org/wiki/Diff_utility) ja tema sugulased, mis kasutavad pikima ühise osajada leidmist.
- Polünoomide interpoleerimine (https://en.wikipedia.org/wiki/Polynomial_interpolation) on kergesti realiseeritav DP abil. Polünoomide interpoleerimine ise leiab kasutust mitmel pool arvutigraafikas, kus meil on vaja keerulisi jooni lihtsa vaevaga kuvada. Näiteks fontide skaleerimist saab realiseerida, kasutades polünoomide interpoleerimist.
- Spordiennustus ja modelleerimine, eriti mängudes, mis koosnevad diskreetsetest etappidest, näiteks kriket ja pesapall. Vt nt [https://en.wikipedia.org/wiki/WASP_\(cricket_calculation_tool\)](https://en.wikipedia.org/wiki/WASP_(cricket_calculation_tool))
- Teksti joondamise ja poolitamise algoritmid (kuidas lõpetada ridu nii, et raisataks võimalikult vähe ruumi), mida kasutavad TeX ja teised küljendusprogrammid.
- Plagiaadi leidmise tööriistad kasutavad teisenduskauguse (i.k *edit distance*) leidmise algoritmi, mis põhineb DP-l. Teisenduskaugusest tuleb rohkem juttu peatükis „DP edasijõudnutele“.
- Lühima tee leidmise algoritmid kaardirakendustes.
- Logistika planeerimine.
- Markovi ahelad DNA-s mustrite leidmiseks.
- Soovituste andmine otsingumootorites.
- Võimsuse optimaalne planeerimine elektrijaamades.

Üks päriselu olukord, kus mul isiklikult DP-laadsest lähenemisest kõige rohkem kasu on olnud, oli seotud andmebaasidega. Tegu oli suure, paljude klientidega olmeettevõttega. Kõigil klientidel oli palju mitmesuguseid tarbimisi. Iga kuu lõpus oli ärianalüütikutel vaja teha musttuhat raportit ja päringut erinevate tarbimisliikide, kliendisegmentide jm kohta.

Täiendavalt tuli teha simulatsioonid, et mis juhtuks, kui hinnad muutuksid. Algne lahendus sisaldas hulka andmebaasipäringuid, mis töötasid aga terve nädalavahetuse, enne kui raportid valmis said. Kui asja uurisin, siis selgus, et üsna paljusid andmeid päriti kogu selles protsessis palju kordi. Et seda vältida, ehitasin eelpool kirjeldatud DP-tabelite analoogina hulga ajutisi tabeleid, milles jupphaaval lõplikku statistikat koostati. Kokkuvõttes valmisid raportid nüüd päevade asemel minutitega.



Selline lähenemine, kus andmebaasis andmeid teatud päringute huvides denormaliseeritakse, on küllaltki levinud (vt nt <https://en.wikipedia.org/wiki/Denormalization>), aga selle kombineerimine DP mõtteviisiga lisab su tööriistakasti veel mõned huvitavad võimalused.

5.11 KONTROLLÜLESANDED

5.11.1 Aktsiaturg

Kui siin õpikus oleks kirjas kindel retsept aktsiaturul raha teenimiseks, võiks selle raamatu iga eksemplari eest küsida tuhandeid eurosid. Kuna aga raamatutest pole võimalik kindlaid meetodeid leida, uuravad paljud kauplejad üksteise järel, püüdes jälile saada võitvatele strateegiatele.

Sul on antud võistleva kaupleja tehingute nimekiri, mille igal real on üks number tehingu kohta – täisarvuline kasum või kahjum.

Eesmärgiks on leida sellest jadast maksimaalse summaga alamjada. Kui kasumlikke tehinguid ei ole, väljastada 0.

Sisendi esimesel real on tehingute arv N ($1 \leq N \leq 10000$). Teisel real on igast tehingust saadud kasum või kahjum.

NÄIDE 1:

5

12 -4 -10 4 9

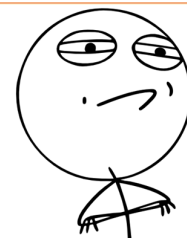
Vastus: 13

NÄIDE 2:

20

-896 995 -588 -23 648 -980 51 -705 -620 -640 915 663 444 307 -207 81 -763 -993
24 665

Vastus: 2329



CHALLENGE ACCEPTED

5.11.2 Protsessori planeerimine

Operatsioonisüsteem peab järjekorda seadma hulga tegevusi, mille sooritamiseks on vaja protsessoriaega. Tegevusi on kaht sorti, esimeste täitmine võtab n millisekundit ja teiste täitmine m millisekundit.

Eesmärgid (prioriteedijärjekorras) on esiteks minimeerida aega, mil protsessor on jõude (ideaalis null, kui null pole võimalik, siis nii väike kui saab) ning teiseks maksimeerida täidetud ülesannete arvu. Töö tegemiseks on aega t millisekundit. Tööd ei tohi pooleli jääda, see oleks sama hea kui jõudeolek.

Kui antud on arvud m , n ja t (täisarvud lõigust 0 - 10000), leida mitu ülesannet on nende reeglite alusel võimalik maksimaalselt lõpetada.

NÄIDE 1:

3 5 54

Vastus: 18 (kõik kolmesed ülesanded)

NÄIDE 2:

3 5 55

Vastus: 17 (kaks viiest ja 15 kolmest ülesannet).

NÄIDE 3:

886 340 6177

Vastus: 10

5.11.3 Maksmise võimalused

Soomes ei ole ühe- ja kahesendised euromündid üldiselt kasutusel. Ignoreerides ühe- ja kahesendiseid (kõiki teisi europangatähti ja münte võib kasutada, hulk ei ole piiratud), leida, mitu võimalust on konkreetse summa maksmiseks. Maksimaalne uuritav summa on 500 eurot. Sisendiks on summa sentides.

NÄIDE 1:

20

Vastus: 4 (1x20, 2x10, 1x10 + 2x5 ning 4x5 senti)

NÄIDE 2:

79

Vastus: 24

5.11.4 Statistika manipuleerimine

Hoolimata sellest, et statistika on täppisteadus ega saa kuidagi „valetada“, nimetatakse statistikat siiski sageli valelikuks. Tegelikult on valelikud muidugi hoopis need, kes statistikat väärkasutavad, valides välja ainult osad andmed, võrreldes omavahel asjaolusid, mis on tegelikult erinevad, või visualiseerides andmeid eksitavalt.

Vaatame käesolevas ülesandes üht sellist statistika manipuleerimise viisi. Oletame, et keegi tahab näidata, et kallite autodega juhtub vähem õnnetusi. Selleks oleks efektne meetod luua nimekiri automarkidest, mis on üheaegselt hinna poolest kasvavas, aga õnnetuste arvu poolest kahanevas järjekorras.

Antud on hulk automarke koos nende hindade (tuhandetes eurodes) ja õnnetuste arvuga (miljoni auto kohta).

Ülesandeks on leida nende markide seast maksimaalse pikkusega jada, kus automargid on hinna poolest kasvavas, aga õnnetuste poolest kahanevas järjekorras. Jada väljastada esialgse jada järjekorranumbrina. Kõik võrratused on ranged, s.t hinnad peavad olema rangelt kasvavas ja õnnetused rangelt kahanevas järjekorras.

Sisendi esimesel real on automarkide arv N ($1 \leq N \leq 1000$). Järgmisel N real on täisarvupaarid, kus esimene arv tähistab hinda ja teine õnnetuste arvu A ($1 \leq A \leq 10000$).

NÄIDE:

9

13 6008

21 6000

20 500

40 1000

30 1100

20 6000

14 8000

12 6000

19 2000

Vastus:

7 2 5 4



PS! Kui sul tegelikult kästakse kunagi statistikat võltsida, siis intellektuaalselt aus oleks töökohta vahetada 😊

5.11.5 Lennukikütus

Reisilennukid püüavad teatavasti oma kütusekulu võimalikult hästi optimeerida. Oluliseks faktoriks on siin tuule kiirus. Ilmateatest on teada, milline tuul eri paikades ja kõrgustel puhub, ülesandeks on koostada optimaalne lennutrajektor.

Iga 10 lennukilomeetri jaoks on kolm võimalust: hoida olemasolevat kõrgust (kulub 30 liitrit kütust), tõusta kilomeetri võrra (kulub 60 liitrit kütust) ja laskuda kilomeetri võrra (kulub 20 liitrit kütust).

Tuule kiirus erinevatel kõrgustel on antud 10 km/h täpsusega ja tuule kiirus võib olla -100 km/h (vastutuul) kuni 100 km/h (pärituul). Iga 10 km/h tuult suurendab või vähendab kütusekulu ühe liitri võrra. Tuult arvestatakse algekõrgusest lähtuvalt.

Lubatud on lennukõrgused 1 km kuni 9 km.

Sisendi esimesel real on antud soovitud lennukaugus L (kümnega jaguv positiivne täisarv kuni 10000).

Järgmisel 10 real on igaühel $\frac{L}{10}$ täisarvu, mis tähistavad tuulekiirusi vastaval kõrgusel ja teelõigul (kõrgemad enne). Esimene rida tähistab tuult kõrgusel 9 ja alumine rida tuult kõrgusel 0. Reisi algus ja lõpp on mõlemad kõrgusel 0, mis tähendab, et alguses ja lõpus tuleb kindlasti tõusta ja laskuda.

NÄIDE 1:

```
40
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 90 90 10
10 -90 -90 10
```

Vastus: 120 (Alustame +10 pärituulega, tõuseme +90 tuulega kõrgusele, hoiame kõrgust ja siis laskume).

NÄIDE 2:

```
100
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
70 70 70 70 70 70 70 70 70 70
-50 -50 -50 -50 -50 -50 -50 -50 -50 -50
-70 -30 -70 -70 -70 -70 -70 -70 -70 -70
-90 -90 -90 -90 -90 -90 -90 -90 -90 -90
```

Vastus: 354

5.11.6 Kahjuritõrje

Farmer Fredil on ristkülikukujuline põld, mis on jagatud 10×10 m ruutudeks. Ruute on kokku $M \times N$. Nüüd on tal vaja põldu taimekaitsevahenditega pritsida. Pritsimine toob nii kasu (hävitades kahjureid) kui kahju (suurendades saagi kemikaalisaldust). Iga ruudu kohta on teada, kui palju kasu või kahju selle ruudu pritsimine kokku annab.

Fred saab tellida lennuki, mis pritsib üle parajasti mingi ühe konkreetse ristkülikukujulise osa põllust (võib ka terve). Leida, kui palju kasu on võimalik lennuki tellimisest saada. Formaalsemalt väljendudes: leida maksimaalse summaga alamristkülik.

Sisendi esimesel real on täisarvud M ja N ($1 \leq M, N \leq 100$). Järgmistel M real on igaühel N arvu, mis tähistavad pritsimisväärtusi (täisarvud -127 kuni 127).

NÄIDE 1:

```
4 4
0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2
```

Vastus:

15

(maksimaalse summaarse väärtusega alamristkülik asub vasakus alumises nurgas:

```
9 2
-4 1
-1 8
```

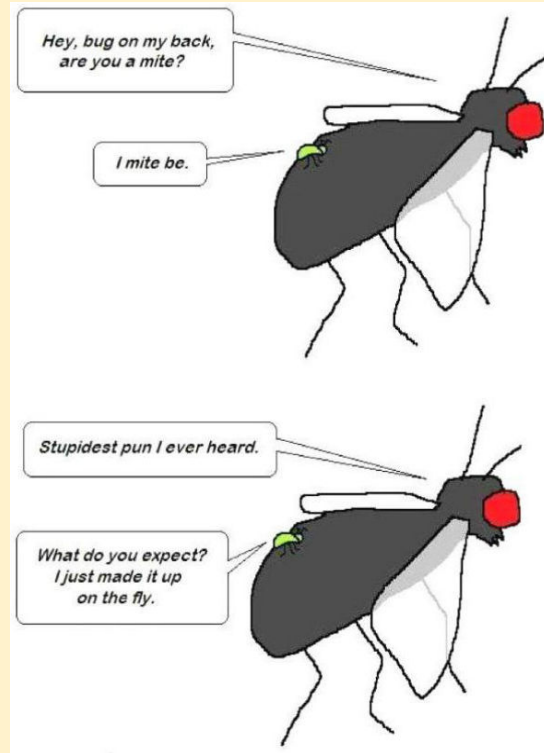
ja selle summa on 15)

NÄIDE 2:

```
5 5
125 85 -120 -114 63
-110 -77 49 27 47
13 -10 101 -116 -34
60 111 -1 61 -24
-120 110 80 69 107
```

Vastus:

513



5.11.7 Torni ehitamine

Väike Jaak ehitab klotsidest torni. Kõik klotsid on risttahukakujulised. Kuna Jaak on alles noor, suudab ta laduda klotse ainult nii, et asetab väiksema klotsi suurema peale. Täpsemalt tähendab see, et pealmise klotsi horisontaalsed mõõtmed (pikkus ja laius) peavad olema rangelt väiksemad kui alumise klotsi pikkus ja laius.

Samas võib klotse igat pidi pöörata, nii et nende pikkus, laius ja kõrgus on omavahel vahetatavad. Jaagu vanem vend Ants soovib nooremat venda aidata ja annab talle kätte klotse sellises järjekorras ja niipidi pööratult, et Jaak saaks kokku võimalikult kõrge torni.

Kui vendadel on kokku N sorti klotse ja igat klotsi on võimalik kasutada piiramatul hulgal, leida, kui kõrge torni nad kokku saaksid.

Sisendis on antud arv N ning seejärel iga N klotsi kohta tema dimensioonid.

NÄIDE 1:

1

30 20 10

Vastus: 40 (kõigepealt klots lapiti, kõrgus 10, siis teine klots püsti esimese otsa, kõrgus 30, kokku 40)

NÄIDE 2:

2

6 8 10

5 5 5

Vastus: 21 (kõrgused 6, 10 ja 5)

NÄIDE 3:

7

1 1 1

2 2 2

3 3 3

4 4 4

5 5 5

6 6 6

7 7 7

Vastus: 28 (suuremast väiksemani üksteise otsas)

NÄIDE 4:

5

31 41 59

26 53 58

97 93 23

84 62 64

33 83 27

Vastus: 342



5.11.8 Putin sukeldumas

Vene Föderatsiooni president Vladimir Vladimirovitš Putin on teatavasti tuntud oma füüsiliste vägitükkide poolest, olgu siis tegu ratsutamise, lendamise või võitlemisega. Muuhulgas on ta paistnud silma oma sukeldumisoskusega, tuues Musta mere põhjast ära antiikseid vaase.

Vähem on aga teada, et sukeldumisele eelnes põhjalik planeerimine, kui palju ja millist varandust Putin ära saaks tuua.

Sul on unikaalne võimalus olla V.V. Putini sukeldumisülem ja planeerida võimalikult tulus sukeldumine. On teada, et merre on eelnevalt paigutatud N vaasi, sügavustele vastavalt $s_1, \dots, s_N$ meetrit ning väärtustega $v_1, \dots, v_N$. Putinil on hapnikuballoon, millest piisab, et olla vee all t sekundit. Vaasid tuleb ära tuua ükshaaval. Meres laskumine võtab aega w sekundit meetri kohta ning tõusmine $2 \cdot w$ sekundit meetri kohta.

Sul tuleb leida parim strateegia, millises järjekorras peaks Putin vaase ära tooma, et nende koguväärtus oleks maksimaalne.

Sisendi esimesel real on arvud t , w ja N . Järgmisel N real on vaaside sügavused ning väärtused.

Väljastada iga vaasi sügavus ja väärtus, mille Putin saab ära tuua, kasutades parimat võimalikku strateegiat. $1 \leq t \leq 1000$, $N \leq 30$.

NÄIDE 1:

210 4 3
10 5
10 1
7 2

Vastus:

10 5
7 2

NÄIDE 2:

656 2 5
79 74
97 75
37 81
21 99
38 25

Vastus:

37 81
21 99
38 25



5.11.9 Arvude liitmine

Antud on positiivsed täisarvud K ja N ($1 \leq K$, $N \leq 100$).

Mitu võimalust on arvu N esitamiseks K mittenegatiivse täisarvu summana? Sisendis on antud arvud K ja N , väljastada üks arv vastusena.

NÄIDE 1:

2 5

Vastus: 6 (0 + 5, 1 + 4, 2 + 3, 3 + 2, 4 + 1, 5 + 0)

NÄIDE 2:

3 4

Vastus: 15

NÄIDE 3:

100 3

Vastus: 5151

5.11.10 Templisambad

Antiikse templi restaureerimiseks on antud N marmorist silindrikujulist kivi, mis võivad olla erineva kõrgusega. Silindritest tuleb laduda K millimeetri kõrgune samm. Kui täpselt K kõrgust sammast pole võimalik ehitada, tuleb mõni kivi madalamaks lihvida, mis tähendab, et kallist materjali läheb raisku. Eesmärgiks on minimeerida esiteks kaotsimineva materjali hulka ning teiseks vajaminevate kivide arvu. Sisendi esimesel real on arv K , teisel real arv N ($N \leq 100$). Järgmisel N real on kivide kõrgused (täisarvud kuni 2000). Väljastada vajaminevate kivide arv ja nende kõrgus enne madalamaks lihvimist.

NÄIDE 1:

1400
3
500 1000 2000

Vastus:

2 1500

NÄIDE 2:

2508
5
331 1585 1395 278 537

Vastus:

4 2541

(Pildil on Apolloni templi varemed Delfis).



5.12 VIITED LISAMATERJALIDELE

Kaasasolevas failis VP_lisad.zip, peatükk5 kaustas on abistavad failid käesoleva peatüki materjalidega põhjalikumaks tutvumiseks:

Fail	Kirjeldus
Fib.cpp, Fib.java, Fib.py	Fibonacci arvu leidmine rekursiooniga
Fib2.cpp, Fib2.java, Fib2.py	Fibonacci arvu leidmine mäluga rekursiooniga
Fib3.cpp, Fib3.java, Fib3.py	Fibonacci arvu leidmine alt-üles DP-ga
Fib4.cpp, Fib4.java, Fib4.py	Fibonacci arvu leidmine kokkuhoidliku DP-ga
Myndid1.cpp, Myndid1.java, Myndid1.py	Mündiülesande lahendus jõumeetodil
Myndid2.cpp, Myndid2.java, Myndid2.py	Mündiülesande lahendus DP-ga
Myndid3.cpp, Myndid3.java, Myndid3.py	Mündiülesande lahendus koos tagurdusmeetodiga (tagastab müntide väärtused)
LIS.cpp, LIS.java, LIS.py	Pikima kasvava osajada pikkuse leidmine jõumeetodil
LIS2.cpp, LIS2.java, LIS2.py	Pikima kasvava osajada pikkuse leidmine DP-ga
LIS3.cpp, LIS3.java, LIS3.py	Pikima kasvava osajada leidmine DP-ga
LCS.cpp, LCS.java, LCS.py	Pikima ühise osajada pikkuse leidmine DP-ga
Ristsumma.cpp, Ristsumma.java, Ristsumma.py	Ristsumma ülesande DP lahendus
ristsumma.xlsx	Ristsumma Excelis
Mortimer.cpp, Mortimer.java, Mortimer.py	Miljonäri ja vaeslaste ülesande lahendus
Vennad.cpp, Vennad.java, Vennad.py	Õiglase jagamise ülesande lahendus
Sallid.cpp, Sallid.java, Sallid.py	Moekunstniku ja koiliblika ülesande lahendus