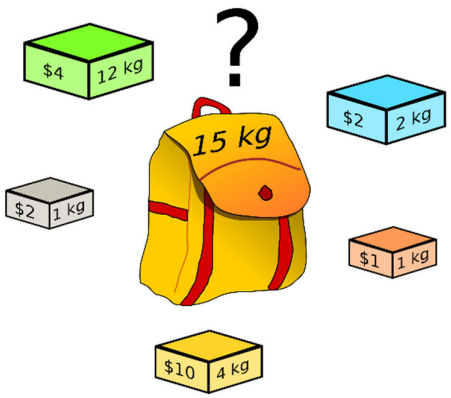
8 DÜNAAMILINE PLANEERIMINE EDASIJÕUDNUTELE

Raamatu esimeses osas oli juttu lihtsamatest dünaamilise planeerimise ülesannetest. Selles peatükis pöördutakse teema juurde tagasi ja tutvustatakse mitmeid klassikalisi dünaamilise planeerimise algoritme ja nende kasutust. Lisaks on siin mõned keerulisemad ülesanded, mille lahendamiseks tuleb tavapärasest rohkem pead murda.

> Those who cannot remember the past are condemned to repeat it.
>
> - Dynamic programming

8.1 SELJAKOTI PAKKIMINE



Seljakoti pakkimise probleem (*knapsack problem*) on ilmselt arvutiteaduse tuntumate ülesannete seas. Tegemist on optimiseerimisülesandega ja seda kasutatakse paljudes optimiseerimisvaldkondades. Probleemi on uuritud juba vähemalt aastast 1897 ning oma nime on see saanud Ameerikasse kolinud Baltisaksa matemaatiku Tobias Dantzigi töödest.

Seljakotiprobleem on NP-keeruline ja seda ei saa deterministliku algoritmiga polünomiaalse ajaga lahendada. Ülesandeklassi sagedast esinemist arvestades on aga kasulik teada, et paljude tavapäraste sisendite jaoks eksisteerib lihtne dünaamilist planeerimist kasutav algoritm. Vaatame, kuidas seda rakendada järgneva ülesande näitel.

8.1.1 Ülesanne: Kohver

Kalle on minemas IOI-le ja ta saab võtta kaasa ühe kohvri, mille kogukaal ei ületa `m` kilogrammi. Kalle tahab kaasa võtta muidugi kõige olulisemad esemed. Aita tal teha parim valik.
Sisendi esimesel real on üks täisarv `m` – kohvri maksimaalne kaal kilogrammides. Teisel real on üks täisarv `n` – Kalle kõigi esemete arv. Järgmisel `n` real on iga eseme kohta tema kaal kilogrammides ja väärtus.
Väljastada parim võimalik seljakoti väärtus ja vähemalt üks sellele vastav sisu (kotti valitavate esemete järjekorranumbrid sisendis).

NÄIDE:

```
10
8
5 7
3 9
1 5
2 2
6 4
4 5
7 8
8 9
```

Vastus:

```
21
```

Selle ülesande juures tuleb kohe tähele panna, et iga eset on täpselt üks ja on täpselt kaks võimalust – kas ese läheb kotti või mitte. Seda tüüpi seljakotiülesannet nimetatakse **0/1** seljakotiülesandeks. Jõumeetodil lahendades tuleks moodustada kõikvõimalikud kombinatsioonid esemetest ning leida iga võimaliku kombinatsiooni väärtus. Vastuseks on kõige suurem väärtus ja sellele vastav esemete kombinatsioon. Kuna iga kord tuleb teha binaarne valik (ese kuulub hulka või mitte), on erinevate kombinatsioonide arv 2^n . Muidugi saab puud kärpida selle võrra, et kui mingis harus on asjade kogukaal suurem kui lubatud maksimumkaal, siis seda haru mööda pole mõtet edasi liikuda, aga nagu teada, siis keerukusklass sellest siiski ei muutu.

8.1.2 Dünaamiline planeerimine

Oletame, et oleme juba viimase eseme juures ja kaalume, kas seda kohvrisse pakkida või mitte. Kui me viimast eset kohvrisse ei paki, siis jääb kohvri kaal ja seal olevate esemete koosseis muutumatuks ja meid huvitab hoopis, kuidas nende $n-1$ eseme hulgast optimaalne valik on saadud – ülesanne taandub veidi väiksemaks. Kui aga soovime viimase eseme lisada, siis peame arvestama selle kaaluga: see tähendab, et varem lisatud esemete kaal peab olema selle viimase eseme kaalu võrra väiksem kui lubatud kogukaal. Näiteks kui võime kohvrisse pakkida 9kg esemeid ja viimane ese kaalub 2kg, siis eelnevalt pakitud esemete kogukaal ei tohi ületada $9-2=7$ kg. Nii sõltub viimase, n-nda eseme valik sellest, kas suurema väärtusega on selline kohver, mis sai pakitud $n-1$ esemest või selline, kus eelneva $n-1$ eseme kogukaal ei ületa seitset kilogrammi. Nii on meil vaja lahendada hoopis kaks väiksemat alamülesannet. Kuna viimase eseme lisamise seisukohalt ei ole oluline, millised esemed on varem valitud (huvitab ainult nende kogukaal), siis saame ehitada oma algoritmi alt üles.

8.1.3 Rekursioonivalem

Olgu esemete kaalud positiivsed täisarvud $m_1, m_2 \dots m_n$ ja väärtused $v_1, v_2 \dots v_n$ Olgu $mv(i,m)$ suurim kohvri väärtus, mida saab moodustada kuni i-st esimesest esemest ja mille kaal ei ületa m-i. Siis:

$$mv(i,m) = \begin{cases} 0, kui\ i = 0 \\ mv(i-1,m), kui\ m_i > m \\ \max\left(mv(i-1,m), mv(i-1,m-m_i) + v_i, kui\ m_i \le m\right. \end{cases}$$

See tähendab, et ridades on meil esemed, mille lisamist kaalume. Esimeses reas esimene, teises teine jne. Veergudes on kogukaalud $0 \dots m$.

8.1.4 Tabeli täitmine

Iga lahtri täitmisel peame kõigepealt vaatama, kas vaadeldava kaaluga ese mahub vaadeldava kogukaaluga kohvrisse (kui eseme kaal on suurem kui vaadeldava kohvri kogukaal, siis seda lisada ei saa) ja väärtus jääb samaks, mis rida varem (`mv[i-1][j]`).

Kui ese mahuks kohvrisse, siis tuleb vaadata, kas seda lisades saame parema väärtuse kui lisamata jättes. Viimasel juhul on väärtuseks muidugi `mv[i-1][j]`, lisamise korral aga eseme kaalu ja selle kaalu võrra väiksema kohvri parim väärtus ehk siis, kui eseme väärtus on `v` ja kaal `m`, siis on selleks `mv([i-1], [j-m]) + v`.

Tabeli veergudes on kohvri kogukaalud ja ridades esemete kaalud ja väärtused. Tabeli lahtris `[i, j]` on maksimaalne väärtus kohvrist, mille mass on kuni `i` kg ja mis on komplekteeritud esemete hulgast ridadel `1…j`.

	0 kg	1 kg	2 kg	3 kg	4 kg	5 kg	6 kg	7 kg	8 kg	9 kg	10 kg
5kg (7)	0	0	0	0	0	7	7	7	7	7	7
3kg (9)	0	0	0	9	9	9	9	9	16	16	16
1kg (5)	0	5	5	9	14	14	14	14	16	21	21
2kg (2)	0	5	5	9	14	14	16	16	16	21	21
6kg (4)	0	5	5	9	14	14	16	16	16	21	21
4kg (5)	0	5	5	9	14	14	16	16	19	21	21
7kg (8)	0	5	5	9	14	14	16	16	19	21	21
8kg (9)	0	5	5	9	14	14	16	16	19	21	21

8.1.5 Kood

Siin on funktsioon, mis analoogse tabeli täidab ja ülesandele vastuse leiab:

```
int pakiKott(int maksKaal, int kogus, int* kaalud, int* vaartused)
{
    int** parimad = new int*[maksKaal + 1];
    for (int i = 0; i <= maksKaal; i++){
        parimad[i] = new int[kogus];
    }

    for (int i = 0; i < kogus; i++){
        parimad[0][i] = 0;
    }

    for (int i = 0; i <= maksKaal; i++){
        parimad[i][0] = 0;
        if (kaalud[0] <= i){
            parimad[i][0] = std::max(parimad[i][0], vaartused[0]);
        }
    }

    int vastus = 0;
    for (int i = 1; i <= maksKaal; i++){
        for (int j = 1; j < kogus; j++){
            parimad[i][j] = parimad[i][j - 1];
            if (kaalud[j] <= i){
                parimad[i][j] =
                    max(parimad[i][j], parimad[i - kaalud[j]][j - 1] + vaartused[j]);
            }
            vastus = max(vastus, parimad[i][j]);
        }
    }
    return vastus;
}
```

8.1.6 Tee taastamine

Mis esemed siis ikkagi kohvrisse pakiti? Seda on tervest DP tabelist päris kerge välja lugeda: kui viimase ruudu väärtus on sama, mis väärtus samal kohal eelmises reas (`mv[i-1, j] == mv[i,j]`), siis seda eset kotti ei lisatud (või vähemalt eksisteerib optimaalne lahendus ilma selle eseme lisamiseta). Kui väärtus eelmises reas on erinev, siis ese lisati ja vaatame samamoodi kotti, mille kaal on kogukaalu ja lisatud eseme kaalu `m` vahe (`mv[i-1, j-m]`).

Näidet vaadates saab vastava kombinatsiooni kõige lihtsamini leida liikudes viimast veergu mööda üles, kuni viimase väärtuseni 21 ehk 3. reale. Kuna teisel real on viimases veerus väiksem väärtus (16), siis 3. ese kuulub kohvrisse. Nüüd peame selle kaalu maha lahutama: $10 - 1 = 9$, seega

vaatame eelviimast tulpa. Siin on 2. reas väärtus 16 ja eelmises reas 7, järelikult tuleb lisada ka teine ese. Lahutame selle kaalu: $9 - 3 = 6$. Kuna esimese rea 6. veerus on väärtus 7, lisati ka esimene ese. Seega on üheks lahendiks esemed 1, 2 ja 3, kohvri kogukaal on $5 + 3 + 1 = 9$ kg ja väärtus $7 + 9 + 5 = 21$.

8.1.7 Keerukus

Dünaamilise planeerimise juures on mahulist keerukust enamasti küllaltki kerge hinnata – sellele vastab DP tabeli suurus. Seljakotiülesandeks on tabeli üheks mõõtmeks esemete arv, teine mõõde aga sõltub sellest, kui suur on lubatud kaal.

Ka ajaline keerukus sõltub selle lahenduse puhul lubatud kaalu väärtusest ja on on $O(nM)$, kus M on koti lubatud kaal.

Seega antud algoritm, lahendab seljakotiülesande **pseudopolünomiaalse** keerukusega. See tähendab, et keerukus on polünomiaalne sisendi arvulise väärtuse suhtes (seljakoti maksimaalse kaalu M suhtes), aga on siiski eksponentsiaalne sisendi mahu (bittide arvu) suhtes. Kui seljakoti maksimaalne kaal kasvab ühe biti võrra, kasvab algoritmi tööaeg 2 korda. Näiteks kui seljakoti lubatud maksimaalne kaal on 8 asemel 256, siis kasvab kuluv aeg 32 korda. Seega on keerukus täpsemalt lahti kirjutades $O(n * 2^{\log_2 M})$.

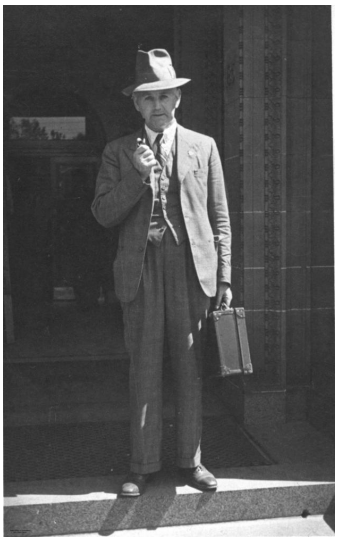
8.1.8 Seljakoti erinevad variatsioonid

Sellist tüüpi seljakoti ülesannet, kus igat eset saab lisada maksimaalselt ühe korra, nimetatakse ka **0/1** seljakotiülesandeks. Sageli kohtab aga ka **piiramata** (*unbounded*) seljakotiülesannet – sel juhul saab lisada iga eset nii palju, kui soovi on. Sellisel juhul:

$$mv(m) = \begin{cases} 0, kui\ m = 0 \\ \max\limits_{m_i \leq m}(v_i + mv(m - m_i)) \end{cases}$$

Oma olemuselt on seljakoti ülesanne väga lähedane 5. peatükis tutvustatud mündiülesandele. Müntide väärtused vastavad kaaludele ja kogusumma kogukaalule. Optimeerisime sellele, et müntide arv oleks kõige väiksem, seega võiks anda igale mündile väärtuseks -1.

8.2 RÄNDKAUPMEHE ÜLESANNE

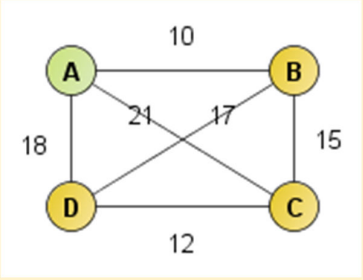


Teine klassikaline NP-keeruline optimiseerimisülesanne on rändkaupmehe ülesanne. Ülesande sõnastus on lihtne: antud on hulk linnasid koos nendevaheliste kaugustega. Leida lühim tee, mis võimaldab kõik need linnad läbi käia ja alguspunkti tagasi jõuda. Ülesanne sõnastati esmakordselt 1930. aastatel ja see on üks kõige rohkem uuritud optimiseerimisülesandeid. Hoolimata üldkujul lahendamise keerukusest on ülesandele leitud mitmeid heuristikuid, mis võimaldavad paljudel erijuhtudel leida mõistliku ajaga lahendeid, mis on ideaalsest vaid protsendi või paari võrra halvemad.

Praeguses kontekstis uurime rändkaupmehe ülesandele täpse lahendi leidmist dünaamilise planeerimise abil.

8.2.1 Ülesanne: kaupmees

Antud on linnade nimekiri, alguspunkt ja iga kahe linna vahelised kaugused. Leida lühim tee, mis läbib iga linna täpselt üks kord ning lõppeb alguspunktis.
Sisendi esimesel real on üks täisarv `n` - linnade arv. Järgneval `n-1` real on igaühel `n-i` arvu, kus `i` on reanumber: `i`-nda linna kaugus linnadest `i+1`, `i+2`…`n`
Väljundisse kirjutada üks täisarv: lühima tee pikkus, kui tee algab ja lõppeb esimeses linnas.



NÄIDE:

```
4
10 21 18
15 17
12
```

Vastus:

55

Jõumeetodit kasutades tuleks proovida kõikvõimalikud linnade järjestused läbi, neid on $(n-1)!$. Samas pole raske märgata, et kui vaatame teid `A-B-C-D-ülejäänud linnad` ja `A-C-B-D-ülejäänud linnad`, siis arvutame nende ülejäänud linnade erinevaid võimalikke järjekordi iga kord uuesti. Samas sõltub kokkulangevus kahest asjast: viimati külastatud linnast ja kõikide seni külastatud linnade hulgast.

Teame, et peame lõpetama linnas A. Enne linna A jõudmist tuleb meil läbida kõik ülejäänud linnad mingis järjekorras. Tähistame seda olukorda (A, {B,C,D}). Meid huvitab selline linnade B, C ja D järjestus, mille korral läbitakse kõige väiksem maa. On kolm erinevat võimalust vastavalt sellele, kas linna A tullakse linnast B, linnast C või linnast D.

Kui linna A tullakse linnast B, siis tee pikkuseks on tee linnast B linna A ja parima tee (B, {C,D}) pikkus. See, millisel moel C ja D kaudu linna B jõuti, ei ole sellel sammul enam oluline, meid huvitab ainult, et see oleks lühim. Lühima tee iga alamtee on ise ka lühim tee.

Analoogselt, kui linna A tullakse linnast C, siis tee pikkus on (A,C)+(C,{B,D}) ja kui linnast D, siis (A,D) + (D,{B,C}). Vastuseks on muidugi neist teedest lühim!

8.2.2 Rekursioonivalem

Olgu meil linnad $L_0..L_{n-1}$ ja mingi linnade hulk $V \in \{L_0,..,L_{n-1}\}$. (L_i, L_j) tähistab kaugust linnast L_i linna L_j ja $mk(L_i, V)$ minimaalset teed linna L_i läbi kõigi linnade hulgas V. Siis saame järgmise valemi:

$$mk(L_i, V) = \begin{cases} (L_0, L_i), kui\ V = \emptyset \\ \min\limits_{\forall L_j \in V}((L_j, L_i) + mk(L_j, V \backslash \{L_j\})) \end{cases}$$

8.2.3 Valemi kasutus

Kõigepealt leiame baasjuhud ehk teed lähtelinnast A otse igasse teise linna. Need saame kauguste tabelist:

	A	B	C	D
A	0	10	21	18
B	10	0	15	17
C	21	15	0	12
D	18	17	12	0

$$mk(B,\{\}) = (A,B) = 10$$

$$mk(C,\{\}) = (A,C) = 21$$

$$mk(D,\{\}) = (A,D) = 18$$

Järgmiseks saab arvutada minimaalsed teepikkused läbi ühe linna:

$$mk(B,\{C\}) = (C,B) + mk(C,\{\}) = 15 + 21 = 36$$

$$mk(B,\{D\}) = (D,B) + mk(D,\{\}) = 17 + 18 = 35$$

$$mk(C,\{B\}) = (B,C) + mk(B,\{\}) = 15 + 10 = 25$$

$$mk(C,\{D\}) = (D,C) + mk(D,\{\}) = 12 + 18 = 30$$

$$mk(D,\{B\}) = (B,D) + mk(B,\{\}) = 17 + 10 = 27$$

$$mk(D,\{C\}) = (C,D) + mk(C,\{\}) = 12 + 21 = 33$$

Teepikkused läbi kahe linna on juba huvitavamad:

$$mk(B,\{C,D\}) = min((C,B) + mk(C,\{D\}), (D,B) + mk(D,\{C\})) = min(15 + 30, 17 + 33) = 45$$

$$mk(C,\{B,D\}) = min((B,C) + mk(B,\{D\}), (D,C) + mk(D,\{B\})) = min(15 + 35, 12 + 27) = 39$$

$$mk(D,\{B,C\}) = min((B,D) + mk(B,\{C\}), (C,D) + mk(C,\{B\})) = min(17 + 36, 12 + 25) = 37$$

Lühim tee, mis linnast A tagasi linna A viib läbi kõigi linnade:

$$mk(A,\{B,C,D\}) =$$

$$= min\big((B,A) + mk(B,\{C,D\}), (C,A) + mk(C,\{B,D\}), (D,A) + mk(D,\{B,C\})\big) =$$
$$= min(10 + 45, 21 + 39, 18 + 37) = 55$$

8.2.4 DP tabel

Aga kuidas neid väärtusi tabelisse panna? Funktsioonil mk on 2 parameetrit – vaadeldav linn ja linnade hulk, mis on juba vaadeldud. Linnaga on lihtne, aga mida teha hulgaga? Siin tuleb kasuks meelde tuletada ülesanne „Moekunstnik ja koiliblikas“ (I osa, ptk 5.8) – hulki on hea mugav edasi anda bitimaskidena.

	0	1	2	3	4	5	6	7
	{}	{B}	{C}	{BC}	{D}	{BD}	{CD}	{BCD}
B	10	-	36	-	35	-	45	-
C	21	25	-	-	30	39	-	-
D	18	27	30	37	-	-	-	-
A	-	-	-	-	-	-	-	55

8.2.5 Kood

Siin on funktsioon, mis selle ülesande lahendab:

```
vector<int> kaupmees(int n, int ap, int** tabel)
{
    int** dp;
    int n2 = 1 << n;
    dp = new int*[n2];
    for (int i = 0; i < n2; i++) {
        dp[i] = new int[n];
        for (int j = 0; j < n; j++) {
            dp[i][j] = -1;
        }
    }
    for (int i = 0; i < n2; i++) {
        if (1 << ap & i > 0) {
            continue;
        }
        if (i == 0) {
            continue;
        }
        vector<int> cv;
        for (int j = 0; j < n; j++) {
            int linn = 1 << j;
            if (i & linn > 0) {
                cv.push_back(j);
            }
        }
        if (cv.size() == 1) {
            dp[i][cv[0]] = tabel[ap][cv[0]];
            continue;
        }
        for (int j = 0; j < cv.size(); j++) {
            int J = cv[j];
            dp[i][J] = 1000000000;
            int pi = i - (1 << J);
            for (int k = 0; k < cv.size(); k++) {
                int K = cv[k];
                if (K == J || dp[pi][K] == -1 || tabel[K][J] == -1) {
                    continue;
                }
                dp[i][J] = min(dp[i][J], dp[pi][K] + tabel[K][J]);
            }
        }
    }
    n2--;
    n2 -= (1 << ap);
    int vas = 1000000000;
    for (int i = 0; i < n; i++) {
        if (tabel[i][ap] == -1) {
            continue;
        }
        vas = min(vas, tabel[i][ap] + dp[n2][i]);
    }
```

```
    vector<int> vasvec;
    vasvec.push_back(ap);
    int rd = vas;
    int ni = ap;
    while (n2 != 0) {
        for (int i = 0; i < n; i++) {
            if (dp[n2][i] == -1 || tabel[i][ni] == -1) {
                continue;
            }
            if (dp[n2][i] + tabel[i][ni] == rd) {
                rd -= tabel[i][ni];
                vasvec.push_back(i);
                ni = i;
                n2 -= (1 << i);
                break;
            }
        }
    }
    return vasvec;
}
```

8.2.6 Tee taastamine

Tee taastamiseks on vaja teada, milline avaldis miinimumi andis. Kuna aga seda miinimumi on vaja leida päris mitme lahtri seast (nii mitme, kui mitu linna meil parasjagu vaadeldavas hulgas on), siis on taastamise vajaduse korral targem meeles hoida see linn, mille kaudu antud miinimum leiti. Nii teame taastades, et antud sammul läbiti see linn ja eelnevalt läbiti linnad hulgast $V \backslash \{L_i\}$

8.2.7 Keerukus

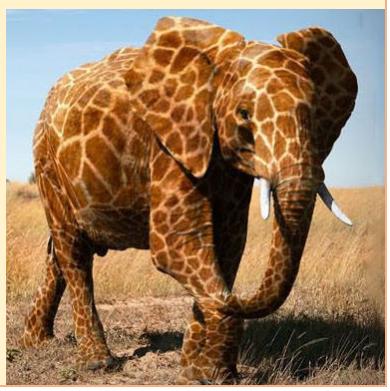
Kavalast lahendusest olenemata ei tohi linnade arv olla kuigi suur. DP tabeli üheks mõõtmeks on kombinatsioonide arv, mis on juba 2^{n-1}. Teiseks mõõtmeks on linnade arv n ja lisaks kasvab koos linnade arvuga ka see, mitmest väärtusest miinimumi võtta tuleb. Kokku on Held-Karpi algoritmi ajaliseks keerukuseks $O(n^2 2^n)$ ja mahuliseks keerukuseks $O(n2^n)$

8.3 LEVENSHTEINI KAUGUS

Paljudes ülesannetes on vaja hinnata kahe stringi omavahelist „sarnasust“. Näiteks „jalalaba“ ja „ali baba“ võiks olla omavahel sarnasemad kui „kartulipõld“ ja „veoauto“. Sellise sarnasuse hindamiseks kasutatakse sellist **Levenshteini kauguse** nimelist mõõdikut (nimetatud vene arvutiteadlase Vladimir Levenshteini järgi, kes tegeles vastava probleemiga juba aastal 1965). Sageli nimetatakse seda kaugust ka **toimetamiskauguseks** (i.k. *edit distance*).

8.3.1 Ülesanne: Levenshteini kaugus

Olgu defineeritud järgmised operatsioonid:
asendamine: üks märk asendatakse teisega, näiteks uxv → uyv
kustutamine: üks märk eemaldatakse: uxv → uv
lisamine: üks märk lisatakse: uv → uxv
Leia, mitu operatsiooni on vaja minimaalselt teha, et saada ühest stringist teine.
Sisendi esimesel real on kaks täisarvu – esimese stringi ja teise stringi pikkus. Teisel real on esimene string ja kolmandal real teine string.
Väljastada üks täisarv – vajalike operatsioonide arv, et saada ühest stringist teine.



NÄIDE:

```
7 10
elevant
kaelkirjak
```

Vastus:

```
8
```

Esimeses osas 5. peatükis oli pikima ühise osajada leidmise ülesanne. Seal meid huvitas just kokkulangevate tähtede arv, antud ülesandes on vaja loendada kohti, kus kokkulangevus puudub (märk tuleb kas lisada, eemaldada või asendada). Oma olemuselt on antud ülesanne väga sarnane pikim osajada leidmisele ning tegelikult on viimane Levenshteini kauguse leidmise erijuht.

Pikima ühise osajada leidmise ülesandes loendasime tähtede kokkulangevusi ja leidsime maksimumi, selles ülesandes on kokkulangevuse hinnaks aga 0 ja otsitakse miinimumi. Üldine idee aga jääb samaks.

Võrdleme jadades viimaseid märke a_m ja b_n. Seekord on meil neli võimalust:

- märgi a_m kustutamine esimesest stringist;
- märgi b_n lisamine esimesse stringi;
- märgi a_m asendamine märgiga b_n;
- juht, kus $a_m = b_n$.

Viimane neist on kõige lihtsam: kui on kokkulangevus, siis teisenduskauguseks on stringide $a_1 \dots a_{m-1}$ ja $b_1 \dots b_{n-1}$ teisenduskaugus. Samuti asendamise puhul on teisenduskauguseks on stringide $a_1 \dots a_{m-1}$ ja $b_1 \dots b_{n-1}$ teisenduskaugus, millele on juurde liidetud 1 toimunud asendusoperatsiooni eest. Samuti on vaja liita 1 toimunud operatsiooni eest lisamise/kustutamise korral ning võrdlemiseks jäävad stringid $a_1 \dots a_{m-1}$ ja $b_1 \dots b_n$ (kustutamisel) või $a_1 \dots a_m$ ja $b_1 \dots b_{n-1}$ (lisamisel). Kuna vaja leida on vähim võimalik toimetamiskaugus, tuleb võtta nendest neljast võimalusest see, mis annab kõige väiksema toimetamiskauguse.

Tuleb veel tähele panna, et kui üks stringidest on tühi (pikkusega 0), siis minimaalseks toimetamiskauguseks on teise stringi pikkus – tuleb teha nii mitu lisamise/kustutamise operatsiooni, kui pikemas stringis märke on.

NB! Selleks, et leida kahe stringi vahelist Levenshteini kaugust, ei ole vaja sooritada tegelikult ühtki neist operatsioonidest, sest selle ülesande käigus ei muudeta kumbagi sisendstringi!

8.3.2 Rekursioonivalem

Olgu $LK_{a,b}(i,j)$ stringide $a_1 \dots a_i$ ja $b_1 \dots b_j$ toimetamiskaugus ehk Levenshteini kaugus. Siis saame järgmise rekursioonivalemi:

$$LK_{a,b}(i,j) = \begin{cases} \max(i,j), kui \min(i,j) = 0 \\ \min \begin{cases} LK_{a,b}(i-1,j) + 1 \\ LK_{a,b}(i,j-1) + 1 \\ LK_{a,b}(i-1,j-1) + 0, kui\ a_i = b_j \\ LK_{a,b}(i-1,j-1) + 1, kui\ a_i \neq b_j \end{cases} \end{cases}$$

8.3.3 DP tabel

Järgnev tabel on täidetud seda valemit kasutades:

	[]	E	L	E	V	A	N	T
[]	0	1	2	3	4	5	6	7
K	1	1	2	3	4	5	6	7
A	2	2	2	3	4	4	5	6
E	3	2	3	2	3	4	5	6
L	4	3	2	3	3	4	5	6
K	5	4	3	3	4	4	5	6
I	6	5	4	4	4	5	5	6
R	7	6	5	5	5	5	6	6
J	8	7	6	6	6	6	6	7
A	9	8	7	7	7	6	7	7
K	10	9	8	8	8	7	7	**8**

Paneme tähele, et esimeses reas ja veerus on olukorrad, kus üks stringidest on tühi, sellisel juhul on kauguseks teise stringi pikkus. See on ka loogiline – selleks, et saada tühjast stringist näiteks string „elevant“, tuleb lisada kõik selle stringi tähed ehk toimetamiskauguseks on stringi pikkus.

Mööda diagonaali toimuvad asendamised(või kokkulangemised – kulu on 0), paremale või alla liikudes eemaldamised ja lisamised. Vastus asub tabeli alumises parempooles nurgas.

Funktsioon, mis vastava tabeli koostab, on siin:

```
int LevenshteiniKaugus(int n, int m, string s, string t)
{
    int** dp = new int*[n + 1];
    for (int i = 0; i <= n; i++) {
        dp[i] = new int[m + 1];
        for (int j = 0; j <= m; j++) {
            dp[i][j] = 0;
        }
    }
    for (int i = 0; i <= n; i++) {
        dp[i][0] = i;
    }
    for (int i = 0; i <= m; i++) {
        dp[0][i] = i;
    }

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            int sc = 1;
            if (s[i-1] == t[j-1]) {
                sc = 0;
            }
            dp[i][j] = min(min(dp[i-1][j] + 1, dp[i][j-1] + 1), dp[i-1][j-1] + sc);
        }
    }

    return dp[n][m];
}
```

8.3.4 Teisenduse taastamine tabelist

Tabelist on võimalik taastada ka tehtud teisendused, mis vähima kauguseni viisid. Lahendusi võib olla muidugi mitu.

Taastamine käib loomisega võrreldes tagurpidi, alustame kohalt `dp[m,n]` ja lõpetame kohal `dp[0,0]`. Olles lahtris `dp[i,j]`, vaatame milline väärtustest lahtrites `dp[i-1,j]`, `dp[i-1,j-1]` ja `dp[i,j-1]` on vähim.

Siin on mõned lahendused:

```
--ELEVANT- --EL--EVANT     --ELEVA-NT
KAELKIRJAK      KAELKIRJAK-     KAELKIRJAK
```

8.3.5 Ajaline ja mahuline keerukus

Selle algoritmi ajaline ja mahuline keerukus on $O(nm)$, kus n ja m on stringide pikkused. Kui vaja on leida vaid arvuline kaugus ja tee taastamine oluline ei ole, siis saab aga kasutada kokkuhoidlikku lahendust – uue lahtri arvutamiseks on vaja teada ainult eelmise rea infot ning käesoleva rea eelmist väärtust – seega piisab kauguse leidmiseks vaid eelmise ja käesoleva rea meelespidamisest.

8.3.6 Levenshteini kauguse omadused

Levenshteini kaugusel on järgmised omadused:

- See on vähemalt nii suur, kui stringide pikkuste erinevus (puuduvad tähed tuleb lisada või üleliigsed eemaldada) : $LK(a,b) \geq |\bar{a} - \bar{b}|$, kus $\bar{x}$ on stringi x pikkus.
- Ei ole kunagi suurem, kui pikema stringi pikkus (saab asendada iga märgi): $LK(a,b) \leq max(\bar{a},\bar{b})$
- See on 0 siis ja ainult siis, kui stringid on täpselt samad: $LK(a,b) = 0 \Leftrightarrow a = b$
- Esimese sõna kaugus teisest on sama, mis teise sõna kaugus esimesest: $LK(a,b) = LK(b,a)$
- Kahe stringi Levenshteini kaugus ei ole kunagi suurem kui esimese stringi kauguse mingist kolmandast stringist ja teise stringi kauguse sellest kolmandast stringist summa: $LK(a,b) \leq LK(a,c) + LK(b,c)$

8.3.7 Levenshteini kauguse variatsioone

Nagu juba mainitud, saab pikima ühise osajada leidmist vaadata kui Levenshteini kauguse erijuhtu, kus lubatud on ainult lisamised ja kustutamised (maksumusega 0), kattuvuse hinnaks on 1. Kui lubatud on ainut asendamised (stringid peavad siis muidugi olema sama pikkusega), nimetatakse sellist kaugust **Hammingi** kauguseks ning selle leidmine on triviaalne.

Mõnikord lisatakse operatsioonide hulgale veel **transpositsioon** ehk kahe kõrvuti asuva tähe vahetamine, uxyv→uyxv. Sellisel juhul omandab rekursioonivalem järgmise kuju:

$$LK_{a,b}(i,j) = \begin{cases} \max(i,j), kui \min(i,j) = 0 \\ \min \begin{cases} LK_{a,b}(i-1,j)+1 \\ LK_{a,b}(i,j-1)+1 \\ LK_{a,b}(i-1,j-1)+0, kui\ a_i = b_j \\ LK_{a,b}(i-1,j-1)+1, kui\ a_i \neq b_j \\ LK_{a,b}(i-2,j-2)+1, kui\ a_{i-1}a_i = b_j b_{j-1} \end{cases} \end{cases}$$

8.4 GEENIJÄRJESTUSTE VÕRDLEMINE

1970. aastal kasutasid Saul B. Needleman ja Christian D. Wunsch esmakordselt dünaamilist planeerimist kasutavat algoritmi bioinformaatikas nukleotiidijärjestuste vastavusse seadmiseks ehk joondamiseks.

8.4.1 Ülesanne: Sarnasus

On antud kaks nukleotiidide järjestust ning need tuleb seada üksteisega vastavusse, nii et võimalikult palju järjenditest langeks kokku. Järjendite vastavusse seades on kolm võimalust:
1. Kattuvus – mõlema järjendi nukleotiid on sama
2. Mittekattuvus – kummaski järjendis on erinev nukleotiid
3. Kustutamine – ühes järjendis puudub vastav nukleotiid

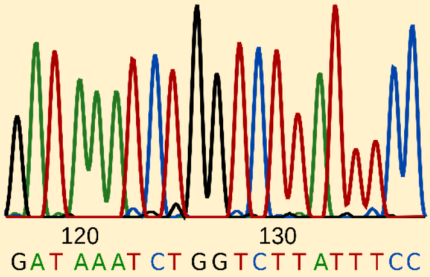
Igal juhtumil on oma hind: iga kattuvus annab 1 punkti, iga mittekattuvus -1 ning iga kustutamine samuti -1. Leia nende järjendite sarnasus ehk maksimaalne hind.

Sisendi esimesel real on esimese järjestuse pikkus ja teisel real sellele vastav märgijada, mis koosneb märkidest A, C, G ja T ja U. Kolmandal real on teise järjestuse pikkus ja sellele vastav märgijada. Väljastada üks täisarv: maksimaalne hind.

NÄIDE:

7
GCATGCU
7
GATTACA
Vastus:
0

Näiteks sobib järgmine joondus:
GCAT-GCU
G-ATTACA



Kuigi nimetused on teised, pole vast raske taibata, et mittekattuvus on sama, mis eelmises ülesandes asendamine. Erinevuseks on veel operatsioonide hind: originaalses Needlemani ja Wunschi töös kasutati sellist süsteemi nagu ülesandes – kattuvus 1 punkt, mittekattuvus ja kustutamine -1. Põhimõtteliselt on võimalik kasutada ükskõik millist punktisüsteemi, kuna sellest ei muutu algoritmi ega dp-tabeli täitmise loogika. Kasutades eelnevalt kirjeldatud süsteemi, saab hinnata kattuvust 0 punktiga ja mittekattuvust ning lisamist/kustutamist ühe punktiga. Nii saame tulemuseks Levenshteini kauguse. Seetõttu algoritm ise pikemat selgitamist ei vaja:

```
int NeedlemanWunsch(int n, int m, string s, string t)
{
    int** dp = new int*[n+1];
    for (int i = 0; i <= n; i++) {
        dp[i] = new int[m+1];
        for (int j = 0; j <= m; j++) {
            dp[i][j] = 0;
        }
    }

    int vas = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            int hind = -1;
            if (s[i-1] == t[j-1]) {
                hind = 1;
            }
            dp[i][j] = max(max(dp[i-1][j] - 1, dp[i][j-1] - 1), dp[i-1][j-1] + hind);
        }
    }
    return dp[n][m];
}
```

8.4.2 Ülesanne: Mis on ühist?

On antud kaks geenijärjestust. Leida parima hinnaga joondatava alamosa hind, kui operatsioonide hinnad on järgmised:
kattuvus 3
mittekattuvus -3
kustutamine -2
Sisendi esimesel real on esimese järjestuse pikkus ja teisel real sellele vastav märgijada, mis koosneb märkidest A, C, G ja T ja U. Kolmandal real on teise järjestuse pikkus ja sellele vastav märgijada. Väljastada üks täisarv: maksimaalne lõigu hind.
NÄIDE:
9
GGTTGACTA
8
TGTTACGG
Vastus:
13

Kui Needleman-Wunschi algoritm leidis kahe järjendi parima joonduse, siis antud ülesandes on vaja leida kahest geenijärjestusest parima kattuvusega lõigud. Seetõttu nimetatakse esimest ka globaalseks joondamiseks ja teist, antud ülesannet, lokaalseks joondamiseks.

Lokaalse joondamise probleemiga tegelesid Temple F. Smith ja Michael S. Waterman, kes pakkusid 1981. aastal lahenduseks algoritmi, mis kannabki nüüd Smith-Watermani nime. Nagu aimata võid, on taas tegemist dünaamilise planeerimisega, täpsemalt jällegi Levenshteini kauguse modifikatsiooniga. Siin on oluline, et mitte soositud operatsioonid on negatiivse hinnaga, soositud (kokkulangemine) aga positiivsega. Kui mingil hetkel läheb tehingu väärtus nulli, siis seda enam ei vähendata, vaid hakatakse otsima uut parimat kokkulangevat juppi.

8.4.3 Rekursioonivalem

Tähistame kustutamise/lisamise eest lahutatavat hinda LH-ga (antud ülesandes -2) ning kattuvuse hinda KH-ga (3) ja mittekattuvuse hinda MH-ga (-3), siis:

$$SW_{a,b}(i,j) = \begin{cases} 0, kui\ i = 0\ V\ j = 0; \\ \max \begin{cases} SW_{a,b}(i-1,j) + LH \\ SW_{a,b}(i,j-1) + LH \\ SW_{a,b}(i-1,j-1) + KH, kui\ a_i = b_j \\ SW_{a,b}(i-1,j-1) + MH, kui\ a_i \neq b_j \\ 0 \end{cases} \end{cases}$$

8.4.4 Tabel

Näiteandmete põhjal tekib järgmine dp tabel:

	[]	G	G	T	T	G	A	C	T	A
[]	0	0	0	0	0	0	0	0	0	0
T	0	**0**	0	3	3	1	0	0	3	1
G	0	3	**3**	1	1	6	4	2	1	0
T	0	1	1	**6**	4	4	3	1	5	3
T	0	0	0	4	**9**	**7**	5	3	4	2
A	0	0	0	2	7	6	**10**	8	6	7
C	0	0	0	0	5	4	8	**13**	11	9
G	0	3	3	1	3	8	6	11	10	8
G	0	3	6	4	2	6	5	9	8	7

8.4.5 Kood

```
int SmithWaterman(int n, int m, string s, string t)
{
    int** dp = new int*[n+1];
    for (int i = 0; i <= n; i++) {
        dp[i] = new int[m+1];
        for (int j = 0; j <= m; j++) {
            dp[i][j] = 0;
        }
    }

    int vas = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            int sc = -3; // ei lange kokku
            if (s[i-1] == t[j-1]) {
                sc = 3; // langeb kokku
            }
            dp[i][j] =
                max(max(dp[i-1][j] - 2, dp[i][j-1] - 2), max(0, dp[i-1][j-1] + sc));
            vas = max(vas, dp[i][j]);
        }
    }
    return vas;
}
```

8.4.6 Tee taastamine

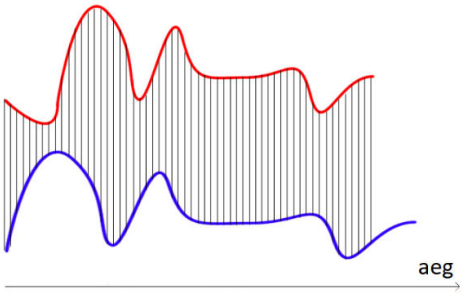
Tee taastamiseks tuleb leida suurima väärtusega lahter tabelist (seda on kaval jooksvalt meeles pidada) ning sealt saab juba tagurpidi tagasi tulles leida operatsioonid, mida kasutati. Kui jõutakse lahtrisse, kus on `0`, siis on algus leitud ja rohkem kuhugi liikuda pole vaja. Seega vastus ülesandele võib asuda suvalises tabeli lahtris ning taastamisteegi ei pea jooksma ilusti tagasi tabeli algusesse, nagu klassikalisele toimetamiskauguse algoritmile omane. Ülesande lahenduseks on näiteks järgmine joondus (parim kokkulangev lõik on märgitud rohelisega):

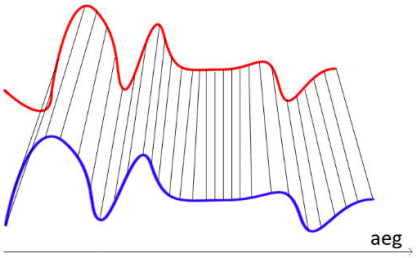
G**GTTGAC**TA
T**GTT-AC**GG

8.5 Dünaamiline Aja painutamine (Dynamic Time Warping)

Dünaamiline aja painutamine (i.k. *dynamic time warping, DTW*) on algoritm kahe aegrea sarnasuse hindamiseks. DTW algoritmi kasutati algselt just automaatses kõnetuvastuses, et hakkama saada erinevatel kiirustel rääkimisega.

Näiteks, kui sarnased on järgmistel joonistel toodud punane ja sinine funktsioon? Kui proovida võrrelda omavahel punke, mis ajas täpsel kokku langevad (esimene joonis), siis tundub, et üsna erinevad.





Parempoolsel joonisel aga võrreldakse ajas veidi üksteise suhtes nihkes olevaid punkte, mis seataksegi vastavusse nii, et erinevus oleks võimalikult väike. See saavutatakse nii, et vahepeal on aeg justkui kokkusurutud või väljavenitatud – ühele kitsale lõigule ajas vastab teisest funktsioonist hoopis laiem lõik.

8.5.1 Ülesanne: helid

On antud kaks heli aegridadena – uuritav rida ja etalon. Leia minimaalne summaarne erinevus nende ridade vahel.
Sisendi esimesel real on arv `m` – uuritava rea pikkus millisekundites ja teisel real on m arvu – igale millisekundile vastav kõrgus. Kolmandal real on arv `n` – etalonrea pikkus - ja neljandal real `n` arvu. Kõik andmed on täisarvudena.
NÄIDE:

```
6
1 1 2 3 2 0
7
0 1 1 2 3 2 1
```

Vastus:

```
2
```

Ka selle ülesande lahendamiseks saab kasutada dünaamilist planeerimist ning täpsemalt Levenshteini kauguse modifikatsiooni. Kirjeldatud kokkusurumistele ja venitamistele vastavad kustutamised ja lisamised ning asendamisi ei toimu. Operatsiooni hinnaks on punktide väärtuste kaugus.

8.5.2 Rekursioonivalem

Olgu $d(a_i, b_j) = |a_i - b_j|$, siis

$$T(i,j) = \begin{cases} 0, & kui\ i = 0\ ja\ j = 0 \\ \infty, kui\ i = 0\ või\ j = 0 \\ min \begin{cases} T(i-1,j) + d(a_i, b_j) \\ T(i-1,j-1) + d(a_i, b_j) \\ T(i,j-1) + d(a_i, b_j) \end{cases} \end{cases}$$

8.5.3 Tabel

Ülesande näide annab järgmise DP-tabeli:

	0	1	1	2	3	2	0
0	**0**	∞	∞	∞	∞	∞	∞
0	∞	**1**	2	4	7	9	9
1	∞	**1**	1	2	4	5	6
1	∞	1	**1**	2	4	5	6
2	∞	2	2	**1**	2	2	4
3	∞	3	4	2	**1**	2	5
2	∞	4	4	2	2	**1**	3
1	∞	4	4	3	4	2	**2**

8.5.4 Kood

```
int DTW(int n, int m, vector<int>, vector<int> t)
{
    int** dp = new int*[n];
    for (int i = 0; i < n; i++)
    {
        dp[i] = new int[m];
        for (int j = 0; j < m; j++) {
            dp[i][j] = 0;
        }
    }
    for (int i = 0; i < n; i++) {
        dp[i][0] = MAXINT;
    }
    for (int i = 0; i < m; i++) {
        dp[0][i] = MAXINT;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]);
            dp[i][j] = min(dp[i][j], dp[i - 1][j - 1]);
            dp[i][j] += kaugus(i, j);
        }
    }
    return dp[n][m];
}
```

8.6 LÕIKUDEGA ÜLESANDED

8.6.1 Ülesanne: palgi lõikamine

Töökoda pakub teenust, et kliendi palgist lõigatakse talle sobivast kohast sobiva pikkusega jupid. Kuna töökojas on vaid üks ühe teraga saag, saab palki lõigata korraga ühest kohast. Palgi lõikamise kulu sõltub sellest, kui pikk on parasjagu lõigatav palk. Seega, kui kliendil on 10m pikkune palk ja ta soovib saada lõiked vasakult 1 m kauguselt, 3 m kauguselt ja 5m kauguselt, siis esimene kord on vaja lõigata 10-meetrist palki, teise lõike tegemisel aga sõltub, millisest kohast lõigati esimene kord. Kui lõigati 1m kauguselt, tuleb järgmine lõige teha 9-meetrisesse palki, kui aga näiteks 5m kauguselt, siis 5-meetrisesse palki. Töökoda küsib iga meetri lõigatava palgi eest 1 ühiku raha. Leia minimaalne summa, mis etteantud palgi lõikamiseks kulub.
Sisendi esimesel real on kaks täisarvu palgi pikkus p ja lõigete arv n. Teisel real on n täisarvu – lõikekohtade kaugused palgi vasakust servast.
Väljundisse kirjutada üks arv – minimaalne hind, mis tuleb palgi jupitamise eest tasuda.
NÄIDE:

```
10 4
4 5 7 8
```

Vastus:

```
22
```

8.6.2 Tee DP-ni

Alustame taas jõumeetodile mõtlemisega. Võimalikke lõikekohti on n ja tuleks läbi proovida kõik võimalikud järjestused. Iga lõige jaotab palgi täpselt kaheks jupiks, mida tuleb siis samal moel edasi tükeldada, kuni kõik lõiked on tehtud. Kuigi see lähenemine töötab, on selle keerukuseks $O(n!)$.

Siingi hakkavad alamprobleemid kattuma, sest kui mingil hetkel on järel jupp lõikekohast i kuni lõikekohani j, siis pole tegelikult vahet, mil moel selle seisuni jõuti – optimaalse lahenduse saamiseks on vaja lõigata see tekkinud jupp võimalikult väikese kuluga.

8.6.3 Rekursioonivalem

Tähistame $h(i,j)$ parimat hinda lõikekohast i kuni lõikekohani j, kus $0 \le i < j \le n+1$. 0 on palgi vasak ja $n+1$ parem otspunkt. Otspunktide vahel on lõikekohad $1 \dots n$, järjestatud rangelt vasakult paremale. Sellisel juhul

$$h(i,j) = \begin{cases} 0, kui\ i+1 = j \\ p(i,j) + \min\limits_{\forall k \in \{i+1,\dots,j-1\}} \big(h(i,k) + h(k,j)\big) \end{cases}$$

kus $p(i,j)$ on palgi pikkus i. lõikepunktist j. lõikepunktini.

8.6.4 Tabeli täitmine

Kasutame rekursioonivalemit DP tabeli täitmiseks. Siin tuleb tähele panna, et alustada tuleb lühematest lõikudest, kuna neid kasutatakse pikemate lõikude arvutamiseks. See tähendab, et tabelit täidetakse sisuliselt mööda diagonaali – lühemad lõigud asuvad diagonaali lähedal, pikemad sellest eemal. Praktikas on lihtsam täita veerg-haaval, täites iga veeru diagonaalist ülespoole liikudes. Ka vastus ei ole tavapärases kohas ehk tabeli lõpus, vaid hoopis esimese rea viimases veerus:

	1 (4)	2 (5)	3 (7)	4 (8)	5 (10)
0 (0)	0	5 + min([0,1],[1,2]) = 5 + min(0, 0) = **5** L=1	7 + min([0,1]+[1,3], [0,2]+[2,3]) = 7+ min(0+3, 5+0) = **10** L=1	8 + min([0,1]+[1,4], [0,2]+[2,4], [0,3]+[3,4]) = 8 + min(0+7, 5+3, 10+0) = **15** L=1	10 + min([0,1]+[1,5], [0,2]+[2,5], [0,3]+[3,5], [0,4]+[4,5]) = 10 + min(0+12, 5+8, 10+3, 15+0) = **22** **L=1**
1 (4)		0	3 + min([1,2]+[2,3]) = 3 + min(0+0) = **3** **L=2**	4 + min([1,2]+[2,4], [1,3]+[3,4]) = 4 + min(0+3, 3+0) = **7** **L={2,3}**	6 + min([1,2]+[2,5], [1,3]+[3,5], [1,4]+[4,5]) = 6 + min(0+8, 3+3, 7+0) = **12** **L=3**
2 (5)			0	3 + min([2,3]+[3,4]) = 3 + min(0+0) = **3** **L=3**	5 + min([2,3]+[3,5], [2,4]+[4,5]) = 5 + min(0+3, 3+0) = **8** **L={3,4}**
3 (7)				0	3+ min([3,4]+[4,5]) = 3 + min(0+0) = **3** **L=4**
4 (8)					0

Iga arvutatud lahtri viimasel real on toodud ka lõikekoht (või lõikekohad), mis parima tulemuse andis. Nii on lihtne leida, mis järjekorras tuleb palki tegelikult tükeldada.

8.6.5 Kood

```
int loika(int pikkus, int loikeid, int* loikekohad) {

    int** dp = new int*[loikeid + 4];
    for (int i = 0; i < loikeid + 4; i++) {
        dp[i] = new int[loikeid + 4];
    }
    for (int i = 0; i < loikeid + 4; i++) {
        for (int j = 0; j < loikeid + 4; j++) {
            dp[i][j] = MAXINT;
        }
    }

    for (int i = 0; i <= loikeid + 1; i++) {
        dp[i][i] = 0;
        dp[i][i + 1] = 0;
        dp[i][i + 2] = loikekohad[i + 2] - loikekohad[i];
    }

    for (int k = 3; k <= loikeid + 1; k++) {
        for (int i = 0; i <= loikeid + 1 - k; i++) {
            for (int j = i + 1; j <= i + k - 1; j++) {
                if ((dp[i][j]+dp[j][i+k]+loikekohad[i+k]-loikekohad[i])<dp[i][i+k]) {
                    dp[i][i+k] = dp[i][j]+dp[j][i+k]+loikekohad[i+k]-loikekohad[i];
                }
            }
        }
    }

    return dp[0][loikeid+1];
}
```

8.6.6 Tee taastamine

Tee taastamiseks on vaja teada, milline lõige andis vaadeldava miinimumi. Kui vaatame tabeli ülemist parempoolset lahtrit ehk seda, kus on terve palgi jupitamine ja seega kogu ülesande vastus, siis see jaotus, mis andis tol sammul miinimumi, ongi esimeseks lõikeks optimaalse lahenduse puhul. Järgmiste lõigete jaoks tuleb siis asuda järelejäänud juppe uurima.
Kuigi on võimalik see miinimum tagurpidi taastada (proovime taas kõik lõiked läbi ja vaatame, milline neist andis miinimumi), siis juhul, kus taastamine on vajalik, on sageli kasulikum meeles pidada see üks arv – optimaalse lõike asukoht.

Antud näites siis on optimaalseks hinnaks 22 ja see saadi siis, kui viimane lõige tehti 1. lõikekohta. Seega järgmiseks lõikeks jäävad järele jupid 0-1 ja 1-5. Mis järjekorras neid edasi tükeldada, ei oma tähtsust. Antud juhul esimest juppi rohkem tükeldada pole vajagi. Teise tüki jaoks vaatame tabelis rea 1 veergu 5. Seal on näha, et minimaalse hinna (12) andis lõige 3. lõikekohast. Seega jääb vaadata juppe 1-3 ja 3-5. Neil on nagunii vaid täpselt üks lõikekoht ja nagu öeldud – järjekord ei ole oluline. Seega sobivad lahenduseks lõiked järjestuses 1, 3, 2, 4 ja 1, 3, 4, 2.

Kontrollime, kas esimene järjestus annab oodatud tulemuse: Alguses on palgi pikkus 10 ja peame arvestama kuluga 10. Lõigates 1. lõikekohast, saame palgid pikkusega 4 ja 6. Esimesel neist lõikekohti rohkem pole. Teise palgi lõikame lõikekohast 3 ja saame kaks palgijuppi pikkusega 3. Kuna lõigatava palgi pikkus oli 6, teeb see kuluks 10+6=16. Nüüd on vaja jupitada veel mõlemad 3-se pikkusega palgitükid. Täpne lõikekoht polegi enam oluline, kuna lõikamise kulu on igal juhul 3, seega on kogukulu 16+3+3 =22.

8.6.7 Keerukus

Mahuline keerukus on $O(n^2)$. Ajalise keerukuse leidmiseks tuleb tähele panna, et ühe tabeli lahtri arvutamiseks võib vaja minna kuni n operatsiooni, seega on ajaliseks keerukuseks $O(n^3)$.

8.7 MAATRIKSITE KORRUTAMINE

Paljud ülesanded füüsikast majandusteooriani taanduvad lineaaralgebrale, kus omakorda on sagedaseks ülesandeks erinevate maatriksite korrutamine. Seetõttu on ka vastavaid algoritme palju uuritud ja optimiseeritud.

8.7.1 Ülesanne: maatriksid

On antud n maatriksit A_1, A_2…A_n. i. maatriksi suurus on P_{i-1}*P_i. Leida vähim operatsioonide arv, mida on vaja teha, et maatriksid omavahel korrutada. Kui maatriks A on mõõtmetega n*m ja maatriks B on mõõtmetega m*k, siis AxB on mõõtmetega n*k ning korrutamiseks vajalike operatsioonide arv on n*m*k.

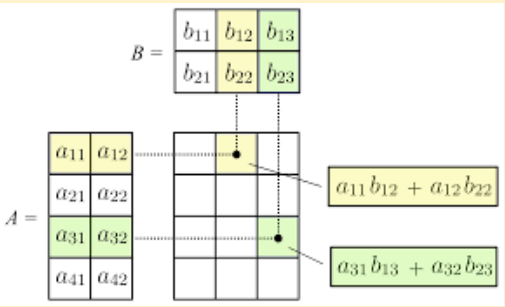
Sisendi esimesel real on antud maatriksite arv n.
Järgmisel real on antud n+1 tühikuga eraldatud arvu P_0…P_n, kus P_{i-1} ja P_i tähistavad i. maatriksi mõõte.
Väljastada vähim operatsioonide arv.

NÄIDE:

```
5
5 2 7 4 3 5
```

Vastus:

```
160
```



8.7.2 Võimaluste arv

Esimese hooga tuleb ehk tahtmine proovida kõikvõimalikke variante. Näiteks nelja maatriksi korrutamiseks on 5 varianti:
`((ab)c)d, (a(bc)d), (ab)(cd), a((bc)d)` ja `a(b(cd))`.

Võimalike erinevate variantide arv on C_{n-1}, kus C_i tähistab i. Catalani arvu ja n maatriksite arvu. Catalani arv avaldub

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n}\frac{n+k}{k}$$

Ja esimesed 20 Catalani arvu on:
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190.

See tähendab, et kõigi võimaluste läbivaatamine tuleb kõne alla vaid siis, kui korrutatavate maatriksite arv on väga väike.

8.7.3 Tee DPni

Vaatame mingit teatud pikkusega alamosa maatriksite jadast. Olgu esimene maatriks selles A_i ja viimane A_j. Tähistame korrutise $A_i \times A_{i+1} \times \ldots \times A_j$ leidmiseks vajalike tehete arvu $op(i,j)$. Siis

$$op(i,j) = \begin{cases} 0, kui\ i = j \\ \min\limits_{\forall k \in \{i,\ldots,j\}} (op(i,k) + op(k+1,j) + P_{i-1} \cdot P_k \cdot P_j) \end{cases}$$

8.7.4 Tabeli täitmine

Selle ülesande juures tuleb tähele panna, et tabeli peadiagonaalil, kus i=j, asuvad 0-väärtused. Tabeli täitmisel liigutakse peadiagonaalist eemale ning vastus on hoopis kohal [1, n], mis loogiliselt vastabki maatriksite jadale $A_i \ldots A_n$.

	A1	A2	A3	A4	A5
A1	0				
A2	[A1,A1] + [A2,A2] + P0*P1*P2 = **70**	0			
A3	min([A1,A2]+[A3,A3]+P0*P2*P3; **[A1,A1]+[A2,A3] + P0*P1*P3**) = min(70+140, **56+40**) = **96**	[A2,A2] + [A3,A3] +P1*P2*P3= **56**	0		
A4	min([A1,A3]+[A4,A4]+P0*P3*P4; [A1,A2]+[A3,A4]+P0*P2*P4; **[A1,A1]+[A2,A4]+P0*P1*P4**)=min(96+60, 70+84+105, **80+30**) =**110**	min(**[A2,A3]+[A4,A4]]+ P1*P3*P4**; [A2,A2]+[A3,A4]+P1*P2*P4)=min(**56+24**,84+42)=**80**	[A3,A3] + [A4,A4] + P2*P3*P4 = **84**	0	
A5	min(**[A1,A1]+[A2,A5]+P0*P1*P5**; [A1,A2]+[A3,A5]+P0*P2*P5; [A1,A3]+[A4,A5]+P0*P3*P5; [A1,A4]+[A5,A5]+P0*P4*P5)= min(**110+50**; 70+189+175; 96+60+100; 110+75)=**160**	min(**[A2,A4]+[A5,A5]+P1*P4*P5**; [A2,A3]+[A4,A5]+P1*P3*P5; [A2,A2]+[A3,A5]+P1*P2*P5)= min(**80+30**, 56+60+40, 189+70)= **110**	min(**[A3,A4]+[A5,A5]+ P2*P4*P5**; [A3,A3]+[A4,A5]+P2*P3*P5) = min(**84+105**,60+140)= **189**	[A4,A4] + [A5,A5] + P3*P4*P5 = **60**	0

Kui huvitab, milline korrutis siis sellise tulemuse andis, siis tabelis on märgitud rasvaselt ka see summa, mis miinimumini viis. Lahtrist [A1,A5] näeb, et miinimum tuli summast [A1,A1]+[A2,A5]+P0*P1*P5 ehk siis maatriksi A1 korrutisest maatriksite A2…A5 korrutisega. A2 ja A5 parim korrutis tuli lahti [A2,A5] põhjal maatriksi A5 korrutisest maatriksite A2..A4 korrutisega. A2…A4 saab vähimate operatsioonide arvuga leida, korrutades maatriksi A4 maatriksite A2 ja A3 korrutisega. Seega parima tulemuse andis korrutis

$$A_1 \times \Big(\big((A_2 \times A_3) \times A_4 \big) \times A_5 \Big).$$

8.7.5 Kood

Siin on funktsioon, mis antud tabeli täidab:

```
int matMul(int n, vector<int> ml)
{
    int** dp = new int*[n+1];
    for (int i = 0; i <= n; i++) {
        dp[i] = new int[n+1];
    }

    for (int vh = 0; vh <= n; vh++) { // vh - vahe ehk kui pikki lõike täidetakse
        for (int i = 1; i + vh <= n; i++) {
            int j = i + vh;
            dp[i][j] = (i == j) ? 0 : 1000000000;
            for (int k = i; k < j; k++) { // iga lõikekoha k jaoks
                dp[i][j] = min(dp[i][j], dp[i][k]+dp[k+1][j]+(ml[i-1]*ml[k]*ml[j]));
            }
        }
    }
            return dp[1][n];
}
```

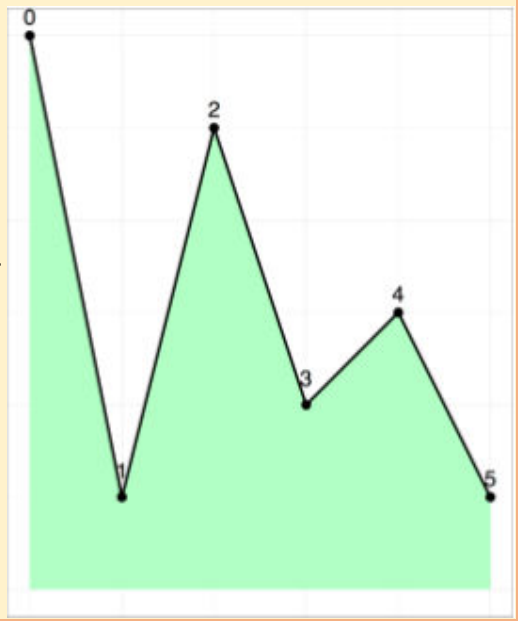
8.7.6 Keerukus

Mahuline keerukus on $O(n^2)$. Ajalise keerukuse leidmiseks tuleb tähele panna, et ühe tabeli lahtri arvutamiseks võib vaja minna kuni n operatsiooni, seega on ajaliseks keerukuseks $O(n^3)$.

8.8 Ülesanne: mäed

Järgnev ülesanne oli 2017. aastal Teheranis toimunud rahvusvahelise informaatikaolümpiaadi proovivoorus:

Kolmas Pärsia kuningas Tahmuras on võitnud lahingu suure deevade (deemonite) armeega. Ta soovib vangistada nii palju deevasid kui võimalik Alborzi mägedesse ja lasta ülejäänud vabaks. Alborz on mäeahelik, mis näeb välja nagu n-tipuga ($1 \le n \le 2000$) hulknurkne ahel. i. tipu koordinaadid on `(i, y[i])`, kus `i` on kaugus aheliku algusest ja `y[i]` ($0 \le y[i] \le 10^9$) tipu kõrgus.
Deevasid saab vangistada erinevatesse tippudesse. Ainus tingimus on, et deevad ei tohi üksteist näha, sest siis saavad nad teha põgenemisplaane. Kaks deevat ei näe teineteist, kui nende vahel on vähemalt üks tipp, mis on rangelt kõrgemal kui nende deevade asukoha tippude vahele tõmmatud joon.
Järgneval joonisel deeva tipus 0 näeb deevasid tippudes 1 ja 2, aga ei näe neid, kes asuvad tippudes 3, 4 ja 5, kuna tipp 2 on kõrgemal ja varjab vaate tipust 0 tippudesse 3, 4 ja 5.



Aita Tahmurasel leida, kui mitu deevat saab ta Alborzi mäeahelikku vangistada. Sisendi esimesel real on mäetippude arv N. Järgmisel N real on tippude kõrgused.

***NÄIDE 1** (joonisel):*

```
6
6 1 5 2 3 1
```

Vastus:

3 (Näiteks tippudele 1, 3 ja 5)

NÄIDE 2:

```
3
0 1 2
```

Vastus:

1

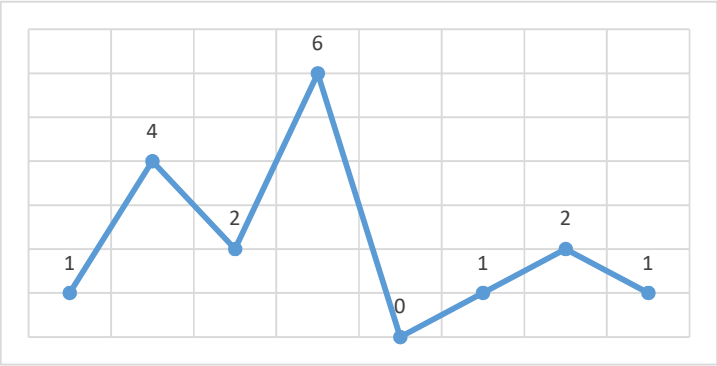
Esimese asjana on meil vaja vastust küsimusele kas tipp i paistab tipust j? Näitame hiljem ülesande lahenduses, kuidas seda teha, esialgu lihtsalt eeldame, et meil on nxn maatriks, mis vastab antud küsimusele. Näiteandmete puhul on see siis järgmine:

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

Pealtnäha ei ole võimalik seda ülesannet lahendada muud moodi, kui jõumeetodiga proovides. Kuid tuleb tähele panna, et kui proovime deeva panna mingisse tippu k, ei saa teisi deevasid olla neis tippudes, mis on näha tipust k, aga deevad saavad olla nende nähtavate tippude vahelistel lõikudel.

Eraldi lõigud tekivad siis, kui nende vahel on nii kõrge punkt, et kumbki pool ei saa teist näha, mistõttu need kaks alamosa ei mõjuta üksteist. Seega, kui me paneme ühe deeva tippu k, siis maksimaalselt saame paigutada mäestikusse nii mitu deevat, kui on igale tipust k nähtamatule

tippude lõigule maksimaalselt paigutatavate deevade summa pluss veel see deeva, mille just paigutasime tippu k.



Näiteks kui panna deeva joonisel viimasesse, 8. tippu (kõrgusega 1), siis on sealt näha 4. tipp kõrgusega 6 ja 7. tipp kõrgusega 2. Deevasid saab paigutada veel lõikudele [1,3] ja [4,5], kuna need ei ole nähtavad viimasest tipust. Nagu jooniselt näha, saab esimesele neist [1,3] paigutada 2 deevat.

8.8.1 Rekursioonivalem

Tähistagu $D(i,j)$ seda, mitu deevat saab panna tippude lõigule i-st j-ni vastavalt ülesande tingimustele. On lihtne taibata, et ühetipulisele lõigule ($i = j$) saab panna täpselt ühe deeva. Tegelikult ka kahetipulisele lõigule saab panna täpselt ühe deeva, kuna teine tipp on esimesest ja esimene tipp teisest tipust näha.

Olgu k tipp, mis asub lõigul $[i,j]$ ehk $i \leq k \leq j$. Tähistagu LK_{ijk} selliste maksimaalse pikkusega mittekattuvate lõikude hulka, kus iga lõik $[h,g] \in [i,j]$ ja ükski lõigu $[h,g]$, $h \leq g$ punktidest ei ole tipust k nähtav. Saame järgmise rekursioonivalemi:

$$D(i,j) = \begin{cases} 1, kui\ i = j \\ 1 + \max\limits_{\forall k \in \{i \ldots j\}} \sum\limits_{\forall [h,g] \in LK_{ijk}} D(h,g) \end{cases}$$

8.8.2 Tabeli täitmine

Tabeli täitmine sarnaneb kahele eelmisele ülesandele:

	0	1	2	3	4	5
0	**1**	1+max(0,0) = **1**	1+max(0,0,0) =**1**	1+max([3,3], [3,3], 1, [0,1]) =**2**	1+max([3,4], [3,4], 0, [0,1] [0,1]) =**2**	1+ max([3,5], [3,5], [5,5], [0,1]+[5,5], [0,1], [0,3]) =**3**
1		**1**	1+max(0,0) = **1**	1+max([3,3], 0, [1,1]) = **2**	1+max([3,4], 1, [1,1], [1,1]) =**2**	1+max([3,5], [5,5], [1,1]+[5,5], [1,1], [1,3]) = **3**
2			**1**	1+max(0,0) = **1**	1+max(0,0,0) = **1**	1+max([5,5], [5,5], 0, [2,3]) = **2**
3				**1**	1+max(0,0) = **1**	1+max([5,5], 0, [3,3]) =**2**
4					**1**	1+max(0,0) = **1**
5						**1**

8.8.3 Kood

Koodi juures tuleb juhtida tähelepanu kahele asjale: esiteks ei ole antud maagilist nähtavuse tabelit ja see, mis tipud mingist tipust näha on, tuleb ise arvutada. Teiseks, kuna tabeli täitmisel tuleks päris palju kordi järgi vaadata, millised järjestikused tipud ei ole nähtavad, siis on lihtsam hoida iga tipu kohta meeles, millised lõigud sellest tipust nähtavad ei ole.

```
int maximum_deevs(std::vector<int> y)
{
    int n = y.size();
    vector<pair<int, int> >* nl = new vector<pair<int, int> >[n];
    for (int i = 0; i < n; i++) {
        vector<pair<int, int> > vi;
        nl[i] = vi;
    }
    int** dp = new int*[n];
```

```
    for (int i = 0; i < n; i++)
    {
        dp[i] = new int[n];
        double bs = -3; // (-pi)
        int ch = y[i]; // vaadeldava tipu kõrgus
        for (int j = i + 1; j < n; j++) {
            int nh = y[j]; // järgmine tipp
            int cy = nh - ch; // kõrguste vahe
            int cx = j - i; // kauguste vahe
            double cs = atan2(cy, cx); // tippude vaheliste sirgete nurk radiaanides
            if (cs >= bs) { // nurk on suurem, kui seni leitud
                bs = cs;
                int nljs = nl[j].size();
                if (nljs > 0 && nl[j][nljs - 1].second == i - 1) {
                    nl[j][nljs - 1].second = i;
                    continue;
                }
                nl[j].push_back(make_pair(i, i));
            }
        }
    }

    for (int i = 0; i < n; i++) { // esimesest tipust viimaseni
        for (int j = i; j >= 0; j--) { // vaatame eelmisi tippe
            if (i == j) {
                dp[j][i] = 1; // ühetipulisele lõigule saab panna 1 deemoni
                continue;
            }
            //saame panna lõigule j..i vähemalt sama palju deemone, kui lõigule j..i-1
            dp[j][i] = dp[j][i - 1];
            int ov = 1;
            int si = j;
            for (int k = 0; k < nl[i].size(); k++) {
                if (nl[i][k].first < si) { // kui algus on enne vaadeldavat indeksit
                    continue;
                }
            // kui on esimene lõik või eelmise lõigu lõpp on enne vaadeldavat tippu
                if (k == 0 || nl[i][k - 1].second < si) {
                     if (nl[i][k].first <= si && nl[i][k].second >= si) {
                        continue; // oleme lõigu sees, ei muutu midagi
                    }
                    // liidame need lõigud juurde, mis tuleb arvesse võtta
                    ov += dp[si][nl[i][k].first - 1];
                    continue;
                }
                if (nl[i][k].first - 1 == nl[i][k - 1].second) {
                    continue;
                }
                ov += dp[nl[i][k - 1].second + 1][nl[i][k].first - 1];
            }
            dp[j][i] = max(dp[j][i], ov);
        }
    }
    int vas = dp[0][n - 1];
    return vas;
}
```

8.8.4 Keerukus

Mahuline keerukus on muidugi taas tabeli suurus ehk $O(n^2)$. Lahtri täitmiseks on vaja vaadata kõiki eelnevaid tippe ehk teha kuni n arvutust, kogu ajaline keerukus on seega $O(n^3)$.

8.9 KONTROLLÜLESANDED

8.9.1 Torn

Külma sõja ajal üritasid USA ning NSVL üksteist igas asjas üle trumbata. Seega, kui Leonid Brežnev kuulis, et üks Ameerika koolipoiss on ehitanud maailma kõrgeima pappkastidest koosneva torni, teatas ta kohe, et Nõukogude Liit ei saa jääda teiseks. Oma pappkasttorni ehitamiseks said nõukogude teadlased `N(1≤N≤1000)` erinevat sorti kasti, mis on nummerdatud arvudega `1...N` ning mida on lõputul hulgal saadaval. Iga kasti kohta on teada selle kaal ning maksimaalne kaal, mida see kannatada suudab. Kastide paigutamiseks on järgmised reeglid: väiksema numbriga kasti ei tohi suurema numbriga kasti peale panna ning ühegi kasti peal ei tohi kokku olla rohkem raskust, kui see kannatada suudab. Leida maksimaalne arv kaste, mida on võimalik üksteise peale laduda.
Sisendi esimesel real on arv `N`. Järgmisel `N` real on kaks tühikutega eraldatud täisarvu: kasti kaal ning kandmisvõime.

Väljastada üks täisarv: maksimaalne arv kaste, mida on võimalik üksteise peale laduda.

NÄIDE:

```
5
19 15
7 13
5 7
6 8
1 2
```

Vastus:

```
4
```

8.9.2 Bussireisid

Bussifirma tahab saata ühe bussi mööda teedevõrku sõitma, nii et see jõuab lõpuks ka bussijaama tagasi. Teedevõrgus on $n(1 \le n \le 50)$ ristmikku, $m(1 \le m \le 1000)$ kahesuunalist teed kahe ristmiku vahel, mille kohta on teada sõidukulu teed mööda sõitmiseks ning $p(1 \le p \le 12)$ peatust, mis asuvad ristmikutel ning mille kohta on teada, kui palju tulu saab firma, kui buss sinna sõidab. Buss alustab teekonda ristmikult nr 0. Leida suurim kasum, mida firma saab teenida.
Sisendi esimesel real on kaks tühikuga eraldatud täisarvu: n ja m. Järgmisel m real on kolm tühikutega eraldatud täisarvu: tee vasak otspunkt, tee parem otspunkt ning tee sõidukulu sentides. Siis on üksikul real täisarv p. Järgmisel p real on kaks tühikuga eraldatud täisarvu: peatuse asukoht ning peatuses peatumise eest saadav tulu sentides.
Väljundisse kirjutada üks täisarv: firma maksimaalne kasum sentides.

NÄIDE 1:

```
4 5
0 1 100
1 2 300
1 3 200
2 4 400
3 4 325
3
2 150
3 700
4 900
```

Vastus:

```
350
```

NÄIDE 2:

```
1 1
0 1 150
1
1 299
```

Vastus:

```
0
```

8.9.3 Sillad

Pärast üht looduskatastroofi on Räniorg jagunenud kaheks osaks, mille vahel on kuristik. Kummalegi poole kuristikku on jäänud firmad, millest igaühel on oma tootlus ning usk mingisse IT suurtegijasse. Otsustati, et oleks hea mõte ehitada kuristikule firmade vahele sildasid selleks, et nad saaksid ühineda ning oma tootlust kahekordistada. Samas on teada, et ei ole hea mõte ühendada erineva usuga firmasid ning sillad üksteist ületada ei saa. Aidake firmajuhtidel leida, kuidas maksimeerida tootlikkuse kasvu ning mis on vähim vajalik sildade arv, et nõutud tootlikkust saada.
Sisendi esimesel real on arv $v (1 \le v \le 1000)$, vasakul äärel olevate firmade arv. Järgmised v rida sisaldavad kolm tühikutega stringi: firma nimi, firma kirikupea ning firma tootlikkus. Firmad on antud nende paiknemise järjekorras serval.
Nendele järgneb arv $p (1 \le p \le 1000)$ - paremal äärel olevate firmade arv. Järgmised p rida sisaldavad nende firmade informatsiooni samas stiilis, kui vasakul äärel olevate firmade kohta.
Väljastada kaks tühikutega eraldatud täisarvu: maksimaalne võimalik tootlikkuse kasv ning selleks minimaalne sildade arv.

NÄIDE :

```
3
Apple Jobs 1000000
Lääne-Google Lee 1000
Micro Gates 400
4
Googol Lee 5000
Pear Jobs 1200
Soft Gates 100
Ida-Google Lee 50
```

Vastus:

```
1002250 2
```

8.9.4 Mäng

4*4 laual mängitakse järgmist mängu: mängija saab kas katta ühe vabalt valitud ruudu või valida kaks või kolm järjestikust tühja ruutu, millest moodustatud rida on ühe otsaga vastu ruudustiku serva, ning kõik selles reas olevad ruudud katta. Kaotab see, kes katab kogu ruudustiku. Kirjutada programm, mis vaatab juba kaetud ruute, ning ütleb, kas antud seis on selline, et parima strateegia korral võidab praegu käija või teisena käija.
Sisendi esimesel real on arv `n(1≤n≤100 000)`, erinevate laudade arv. Järgmistel ridadel on n 4*4 lauda, kus 'X' tähistab kaetud ruutu ning '.' katmata ruutu. Iga laua ees on tühi rida.
Väljastada n rida. i-ndal real on kirjas üks sõna: `ALUSTAJA`, kui praegu käija võidab või `TEINE`, kui teisena käija võidab.
NÄIDE:

```
3

XXX.
XXX.
.XXX
.XXX

XXXX
…X
XX.X
XX.X

….
….
….
….
```

Vastus:

TEINE
ALUSTAJA
TEINE

8.9.5 Uuring

KAPO on oma korruptsiooniuuringutega saanud kätte ühe märkmiku, kuhu on kirja pandud kõik tehingud, mida kahtlusalune organisatsioon on viimase kuu jooksul teinud. Iga tehing on mingi arv, mis näitab, kas organisatsioon on kulutanud tehinguga x eurot ära või saanud x eurot juurde. Iga lehekülje lõpus on märgitud kõigi tehingute summa ehk voog. Näiteks kui on leheküljel olevate tehingutega saadud 7€, saadud veel 2€, kulutatud 3€, saadud 1€ ning kulutatud 11€, on lehekülje lõpus arv 7 + 2 – 3 + 1 – 11 = -4. Samas ei ole iga tehingu juurde märgitud, kas see oli kulu või tulu. Kuna KAPO tahab seda teada, on vaja kirjutada programm, mis leiab kas mingi tehing oli kulu, tulu või ei ole teada, kumb tehing see on.

Sisendi esimesel real on kaks tühikuga eraldatud arvu: `N (2≤N≤40)` ning `F (-16000≤F≤16000)`, leheküljel märgitud voog. Järgmisel `N` real on arv `T (1≤T≤400)`, tehingu absoluutväärtus.

Väljastada `N`-märgiline string. `i`-s märk peab olema ’+’, kui on teada, et `i`-s tehing on tulu, ’-’, kui `i`-s tehing on kulu ning ’?’, kui ei ole võimalik seda informatsiooni teada. Kui ei ole võimalik märgitud voogu nende tehingutega saada, kirjutada väljundi ainsale reale üks märk ’*’.

NÄIDE 1:

```
5 7
1
2
3
4
5
```

Vastus:

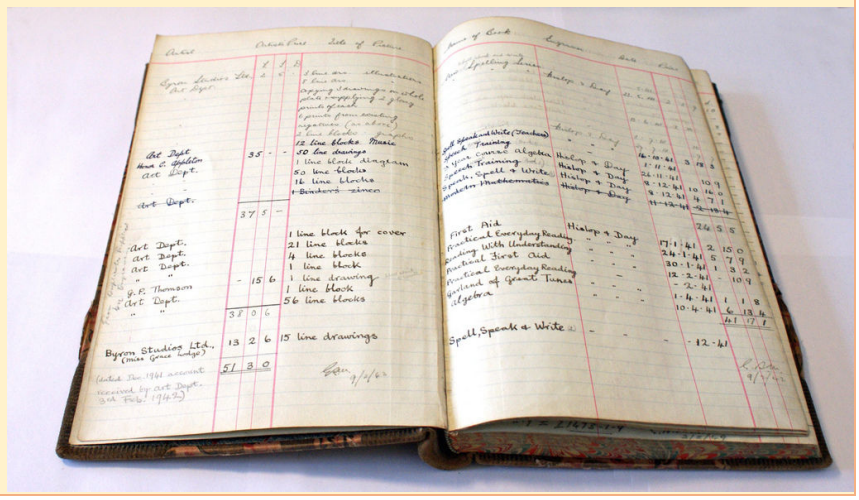
```
?+??+
```

NÄIDE 2:

```
4 15
3
5
7
11
```

Vastus:

```
*
```



NÄIDE 3:

```
5 12
6
7
7
1
7
```

Vastus:

```
+??-?
```

8.9.6 Alkeemik

Alkeemialaboris on hulk erinevaid kemikaale. Alkeemik tahab need kahekaupa omavahel kokku segada, et alles jääks vaid üks kemikaal. Iga kahe kemikaali reageerimisel tekib mingisugune kolmas kemikaal ning eraldub teatud hulk soojust, mida saab tabelist järgi vaadata:

Kemikaalid	1		2		3	
	Uus kemikaal	Eraldunud soojus	Uus kemikaal	Eraldunud soojus	Uus kemikaal	Eraldunud soojus
1	1	0	3	-10	3	3000
2	3	-10	2	0	1	-500
3	3	3000	1	-500	3	0

Ülalolevas tabelis on kolm erinevat kemikaali: 1, 2 ja 3. Kui kokku segada kemikaalid 1 ja 3, tekib 3000 ühikut soojust ning uus segu on kemikaal 3. Vahel võib soojus neelduda. Näiteks tekib kemikaalide 2 ja 3 segamisel kemikaal 1 ning temperatuur langeb 500 soojusühiku võrra. Alkeemik tahab tekitada võimalikult kõrget temperatuuri. Seega, kui laboris on kemikaalid 1, 2, 2 ja 3, on tal võimalik nendest saada erinevaid temperatuure. Kui valida järjekord (2(1(23))), on temperatuuri muut kokku -510, kui see on aga ((12)(23)), on temperatuuri muut 2490.

Sisendi esimesel real on kemikaalide arv `N (1≤N≤6)`. Järgmisel `N` real on N tühikutega eraldatud arvupaari, kus `i`-nda rea `j`-nda paari esimene arv on kemikaalide `i` ja `j` segamisel tekkiv uus kemikaal ning teine arv on sellel reaktsioonil toimuv temperatuuri muut. Järgmisel real on katseklaaside arv `K (2≤K≤10)`. Sellele järgneb rida, kus on `K` arvu: igas katseklaasis olev kemikaali number.

Väljastada üks arv: maksimaalne võimalik temperatuuri tõus, mida on võimalik tekitada.

***NÄIDE 1*:**

```
3
1 0 3 -10 3 3000
3 -10 2 0 1 -500
3 3000 1 -500 3 0
4
1 2 2 3
```

Vastus:

```
2500
```

***NÄIDE 2*:**

```
3
1 0 3 500 3 -250
3 500 2 0 1 10
3 -250 1 00 3 0
6
1 1 1 2 2 3
```

Vastus:

```
1000
```

8.9.7 Puhkus

Lõbus Albert otsustas minna mõneks päevaks puhkusele. Puhkusel on tal võimalik teha erinevaid tegevusi, mille hind ja lõbuväärtus on teada. Albert teeb iga päev täpselt üht tegevust, kuid kui teha üht tegevust teist päeva järjest, saab Albert sellest 50% esialgsest lõbuväärtusest ning kolmandal järjestikusel päeval pole see tegevus enam lõbus. Seega, kui mingi tegevuse lõbuväärtus on v ning Albert teeb seda kolm päeva järjest, saab ta kokku lõbuväärtuse 1,5v. Aita Albertil oma reisi planeerida, nii et tal oleks võimalikult lõbus ning ta ei ületaks oma eelarvet.
Sisendi esimesel real on kolm tühikutega eraldatud arvu: reisil olemise päevade arv k (1≤k≤21), erinevate tegevuste arv n(1≤n≤50) ning Alberti eelarve m (0≤m≤100).
Järgneval n real on kaks tühikuga eraldatud täisarvu: c (1≤c≤50), tegevuse hind ning v (1≤v≤10000), tegevuse lõbuväärtus.
Väljundisse kirjutada kaks tühikuga eraldatud arvu: maksimaalne lõbuväärtus, mida Albert saab endale lubada ning minimaalne hind, millega ta saab seda endale lubada. Kui ükski võimalik tegevuskava ei mahu eelarvesse sisse, väljastada arvupaar 0 0.

***NÄIDE 1*:**

```
2 1 5
3 5
```

Vastus:

```
0 0
```

***NÄIDE 2*:**

```
3 5 20
2 5
18 6
1 1
3 3
2 3
```

Vastus:

```
13 6
```

8.9.8 Tango

Sel ajal, kui muusika algab, on peol võrdselt poisse ning tüdrukuid. Nad moodustavad omavahel poiss-tüdruk paarid, nii et keegi ei jää tantsust kõrvale. On teada, et mõned paarid on sellised, et paarilised vihkavad teineteist (vihkav paar) ning teised paarid on sellised, et paarilised armastavad üksteist (armastav paar). Ülejäänud paarides on paariliste omavaheline suhe neutraalne (neutraalne paar). Leida, mitu erinevat võimalust on paaride moodustamiseks, kus kas ei ole ühtegi vihkavat paari või on vähemalt üks armastav paar.

Sisendi esimesel real on arv `n` (`1≤n≤15`), mis näitab poiste (ja seega ka tüdrukute) arvu. Järgmisel `n` real on `n` tühikutega eraldatud arvu (`0`, `1` või `2`). real `i` veerus `j` tähistab `0`, et `i`-s poiss ning `j`-s tüdruk moodustavad vihkava paari, `1`, et moodustub neutraalne paar ning `2` tähistab armastavat paari.

Väljundisse kirjutada üks arv: võimalike paarikomplektide koguarv.

***NÄIDE 1*:**

```
3
0 1 1
1 1 0
1 0 1
```

Vastus:

```
2
```

NÄIDE 2:

```
3
1 1 2
2 1 0
1 1 2
```

Vastus:

```
4
```

8.9.9 Valimised

FIPA (*Federation Internationale de Programmation Association*) korraldab hääletuse, et teada saada, mis riik korraldab tulevast programmeerimise MM-i ning EIO žürii on nõus tegema kõik, mis võimalik, et saada Eestit korraldajaks. Selleks teevad nad rahalisi annetusi erinevatele riikidele. Erinevate meetodite kaudu on teada, mitu eurot on vaja mingile riigile annetada, et nad hääle Eestile annaksid. Samas on teada, et hääle saamiseks ei pea igale riigile raha annetama, sest väiksemad ja vaesemad riigid on suuremate mõju all ning hääletavad nii, nagu nende mõjutaja. Seega, kui annetada mingile riigile, saab Eesti selle riigi hääle, kõigi sellele alluvate riikide hääled jne. Näiteks kui on teada, et Moldova on Rumeenia mõju all ning Rumeenia on Venemaa mõju all, on võimalik kõigi kolme riigi hääled saada siis, kui annetada vaid Venemaale. Ühelgi riigil pole mitut mõjutajat ning ükski mõjuahel ei moodusta tsüklit. EIO žürii palub abi, et teada saada, mis on vähim hulk raha, mida nad peavad kulutama, et Eesti võidaks. Kandidaadina ei ole Eestil hääleõigust.

Sisendi esimesel real on kaks arvu: n (1≤n≤200) ning m (0≤m≤n), mis on hääletusel osalevate riikide arv ning Eesti valimiseks vajaminev häälte arv. Järgmisel n real on info iga hääleõigusliku riigi kohta. Esimesena on riigi nimi, siis on hääle muutmiseks vajaminev raha hulk, siis on number 0 või 1, mis näitab, kas antud riigil on ülemus: 0 tähistab, et ülemust pole ja 1, et ülemus on. Riigi ülemuse nimi (kui see on olemas) on neljas string reas. Iga riigi ülemus on sisendis eelnevalt mainitud.

Väljundisse kirjutada üks arv: Mitu eurot on vaja kulutada, et Eesti saaks MM-i korraldajaks.

NÄIDE:

```
3 2
Läti 15 0
USA 20 0
Kanada 10 1 USA
```

Vastus:

```
20
```

8.9.10 Plaan

Kuulus programmeerija ning kolmekordne IOI kuldmedalist Martin Pettai töötab nüüd väikeses idufirmas Pisiraamat, kus tal tuleb tegeleda kahe projektiga: „Pehme“ ja „Nägu“. Martin peab kummagi projekti kallal töötama D (D ≤ 33) päeva. Ta ei saa ühel päeval mitme projekti kallal töötada ning ta peab varem alustatud projekti ka varem lõpetama. Samuti on Martin ainus inimene, kes projekte töös hoiab, mistõttu ta ei tohi rohkem kui G (G ≤ 33) päeva kummagi projekti kallal mitte töötada.

Kuna mõlema projekti juhid tahavad, et nende projekt valmis saaks, ei saa Martin ühtegi vaba päeva võtta. Seega, kui D = 3 ja G = 2, on tal võimalik järgneva kuue päeva jaoks kümme erinevat töögraafikut koostada:

	1. päev	2. päev	3. päev	4. päev	5. päev	6. päev
1.	„Pehme“	„Pehme“	„Nägu“	„Pehme“	„Nägu“	„Nägu“
2.	„Nägu“	„Nägu“	„Pehme“	„Nägu“	„Pehme“	„Pehme“
3.	„Pehme“	„Pehme“	„Nägu“	„Nägu“	„Pehme“	„Nägu“
4.	„Nägu“	„Nägu“	„Pehme“	„Pehme“	„Nägu“	„Pehme“
5.	„Pehme“	„Nägu“	„Pehme“	„Pehme“	„Nägu“	„Nägu“
6.	„Nägu“	„Pehme“	„Nägu“	„Nägu“	„Pehme“	„Pehme“
7.	„Pehme“	„Nägu“	„Pehme“	„Nägu“	„Pehme“	„Nägu“
8.	„Nägu“	„Pehme“	„Nägu“	„Pehme“	„Nägu“	„Pehme“
9.	„Pehme“	„Nägu“	„Nägu“	„Pehme“	„Pehme“	„Nägu“
10.	„Nägu“	„Pehme“	„Pehme“	„Nägu“	„Nägu“	„Pehme“

Sisendi ainsal real on arvud D ja G.

Väljundisse kirjutada üks arv: mitu erinevat võimalust on Martinil töögraafiku koostamiseks.

NÄIDE:

```
3 2
```

Vastus:

```
10
```

8.10 VIITED LISAMATERJALIDELE

Kaasasolevas failis VP_lisad.zip, peatükk8 kaustas on abistavad failid käesoleva peatüki materjalidega põhjalikumaks tutvumiseks:

Fail	Kirjeldus
Kohver.cpp, Kohver.java, Kohver.py	0/1 seljakoti ülesnne
Kaupmees.cpp, Kaupmees.java, Kaupmees.py	Rändkaupmees Held-Karpi algoritmiga
Levenshtein.cpp, Levenshtein.java, Levenshtein.py	Levenshteini kaugus ehk toimetamiskaugus
Geenid1.cpp, Geenid1.java, Geenid1.py	Needleman-Wunchi algoritm
Geenid2.cpp, Geenid2.java, Geenid2.py	Smith-Watermani algoritm
DTW.cpp, DTW.java, DTW.py	Dünaamiline aja painutamine
Palk.cpp, Palk.java, Palk.py	Lõikudega dp, palgi saagimine
Maatriksid.cpp, Maatriksid.java, Maatriksid.py	Maatrikisite korrutamine (lõikudega dp)
Maed.cpp, Maed.java, Maed.py	Ülesanne “Mäed” (lõikudega dp)