1. Leiutaja number üks (yks) 20 punkti

Külas on 6 leiutajat. Igaühe kohta on teada tema olemasolevate leiutiste arv N_i ja nädalaga juurde leiutatavate uute leiutiste arv M_i. Mitu leiutist on leiutajal number 0 vaja lisaks hankida, et 10 nädala pärast oleks tal kõige rohkem leiutisi?

On lihtne arvutada, et leiutajal i on 10 nädala pärast kokku $K_i = N_i + 10 \cdot M_i$ leiutist. Tähistame $M = \max(K_1, K_2, K_3, K_4, K_5)$. Kui $K_0 > M$, siis ongi leiutajal number 0 kõige rohkem leiutisi ja lisaks pole midagi vaja. Vastasel juhul peab ta $M + 1$ leiutise kokku saamiseks $M + 1 - K_0$ leiutist lisaks hankima.

2. Sõnumi kaks kuju (kaks) 30 punkti

On antud sõnum, mis koosneb algusmärgist `{`, reast täisarvudest väärtustega alla 10^{256} ja lõpumärgist `}`. Esitada see sõnum 16-numbrite jadana, kus algus- ja lõpumärke tähistavad 16-numbrid `E` ja `F` ja iga arv N esitatakse kujul `A`lP või `AO`llP, kus P on arvu N esitus 16-numbrite jadana ja l või ll on esituse P vastavalt 1-või 2-kohaline pikkus 16-arvuna. Esitus P saadakse arvust N nii, et kolm või enam järjestikust numbrit `0` esitatakse kujul `B`l või `BO`ll, kus l või ll on nullide jada vastavalt 1-või 2-kohaline pikkus 16-arvuna, aga kõik muud numbrid jäetakse muutmata. Kui kodeeritud sõnumi pikkus oleks paaritu arv, lisada selle lõppu `D`.

See on suhteliselt tehniline ülesanne, kus lahenduse ideed kui niisugust eriti polegi. Vaja on andmed sisse lugeda, nõutud kujule teisendada ja tulemus väljastada. Paar tähelepanekut:

- kuigi ülesande tekst räägib sisendi elementidena täisarvudest, toimub kogu töötlus numbrite kaupa ja kasulikum on neid elemente hoida sõnedena (tekstidena); vihje sellele lähenemisele on ka arvude pikkus — need võivad olla kuni 255-kohalised;
- pikkusi näitavate arvude esitamiseks 16-süsteemis on pea kõigis keeltes standardvahendid olemas (allolev tabel), aga seda saab teha ka "käsitsi"; üks lihtsamaid viise on kirjeldada 16-numbrid sõnena '0123456789ABCDEF', siis saab väärtusele 0. . . 15 vastava numbri sealt indeksi põhjal välja lugeda; ühekohaliste arvude teisendamiseks polegi rohkem vaja; kahekohalised arvud saab teisendada nii, et esimese numbri indeks on $arv/16$ täisosa ja teise numbri oma $arv/16$ jagamisel tekkiv jääk.

Python	`sone = format(arv, 'X')`
C	`char sone[10];` `sprintf(sone, "%X", arv);`

C++	`stringstream ss;`
	`ss << hex << uppercase << arv;`
	`string sone = ss.str();`

Java	`String sone = String.format("%X", arv);`

3. Teravnurksed kolmnurgad (kolm) 40 punkti

Tasandil on N punkti $(x_1, y_1), \ldots, (x_N, y_N)$. Kui palju saab nendest moodustada teravnurkseid kolmnurki? Kui palju saab moodustada nendest mittekongruentseid teravnurkseid kolmnurki? ($3 \leqslant N \leqslant 200$.)

Esimene pool

Vaatame kõigepealt, kuidas teravnurkseid kolmnurki loendada. Põhimõtteliselt võiks ülesande lahendada lihtsalt nii:

```
for (int i = 1; i <= N; i++)
  for (int j = i + 1; j <= N; j++)
    for (int k = j + 1; k <= N; k++)
      if ((x[i], y[i]), (x[j], y[j]), (x[k], y[k]) on teravnurkne)
        ans++; // liidame vastusele yhe
```

Tekib küsimus: kuidas kontrollida, kas kolm punkti moodustavad teravnurkse kolmnurga? Olgu kolmnurga küljepikkused a, b, c. Meenutame koosinusteoreemi: [1]

$$c^2 = a^2 + b^2 - 2ab\cos\gamma,$$

kus γ on kolmnurgas külje c vastasnurk. Kui γ on teravnurk, siis $\cos\gamma > 0$, vastasel juhul $\cos\gamma \leqslant 0$. Seega γ on teravnurk parajasti siis, kui $a^2 + b^2 > c^2$. Kontrollimaks, et kõik nurgad on teravnurgad piisab kontrollida, et $a^2 + b^2 > c^2$, $a^2 + c^2 > b^2$ ja $b^2 + c^2 > a^2$. Kui sorteerida küljepikkused nii, et $a \leqslant b \leqslant c$, siis piisab ka ainult kontrollida, et $a^2 + b^2 > c^2$. Küljepikkuste ruudud a^2, b^2, c^2 võime hõlpsasti arvutada Pythagorase teoreemist. Nii saame lahenduse keerukusega $O(N^3)$.

Teine pool

Nüüd uurime, kuidas loendada omavahel mittekongruentseid teravnurkseid kolmnurki. Nagu enne, võiks lahendus olla midagi sellist:

```
for (int i = 1; i <= N; i++)
  for (int j = i + 1; j <= N; j++)
    for (int k = j + 1; k <= N; k++)
      if ((x[i], y[i]), (x[j], y[j]), (x[k], y[k]) on teravnurkne)
        if (sellega kongruentset pole varem nahtud)
          ans++; // liidame vastusele yhe
```

Kuidas kontrollida, kas oleme antud kolmnurgaga kongruentset kolmnurka juba varem näinud? Teatavasti kaks kolmnurka küljepikkustega vastavalt $a \leqslant b \leqslant c$ ja $u \leqslant v \leqslant w$ on kongruentsed täpselt siis, kui $a = u$, $b = v$, $c = w$. Teisisõnu siis, kui nende vastavate külgede pikkused on samad. Aga sama hästi ka siis, kui nende küljepikkuste ruudud on samad.

Niisiis, hoiame pidevalt mälus üht hulka. Kui näeme uut teravnurkset kolmnurka küljepikkustega $a \leqslant b \leqslant c$, siis kontrollime, kas seal hulgas juba on kolmik (a^2, b^2, c^2). Kui ei ole, siis lisame ja suurendame vastust ühe võrra. Selle realiseerimiseks on iga tänapäevases keele standardteegis olemas andmestruktuur, mis võimaldab $O(1)$ või $O(\log N)$ keerukusega kontrollida, kas selles hulgas juba on antud element, ning sama keerukusega sinna uut elementi lisada. Kokku on lahenduse keerukus $O(N^3)$ või $O(N^3 \log N)$.

Teine võimalus on koostada kõigi kolmikute nimekiri, see sorteerida ja siis loendada sellised kolmikud, mis pole eelnevaga võrdsed. Kuna sorteerimisel satuvad võrdsed elemendid kõrvuti, loeme nii igast omavahel kongruentsete kolmnurkade grupist ainult ühe esindaja. See lahendus kulutab küll natuke rohkem mälu (sest unikaalsete kolmnurkade hulga asemel hoiame kõigi kolmnurkade loendit), aga efektiivse sortimisalgoritmi kasutamisel on ka selle tööaeg $O(N^3 \log N)$.

[1] Kes koosinusteoreemi ei tea, saab selle ülesande jaoks piisava järelduse ka Pythagorase teoreemist, mille kohaselt kehtib täisnurkses kolmnurgas võrrand $a^2 + b^2 = c^2$; peaks olema ilmne, et kui külgi a ja b muutmata külge c lühenda, siis muutub külje c vastas olev nurk teravamaks, ja kui külge c pikendada, muutub tema vastasnurk nürimaks, millest saamegi tervnurksuse tingimuse $a^2 + b^2 > c^2$.

Mõlemal juhul on hea mõte hoida just kolmikuid (a^2, b^2, c^2), mitte (a, b, c). Seda sellepärast, et kuna koordinaadid on täisarvud, siis ka a^2, b^2 ja c^2 on alati täisarvud. Veelgi enam, neid saab arvutada nii, et kõik vahetulemused on ka täisarvulised. Kui arvud ei ole täisarvud, siis hoitakse neid arvuti mälus nn ujukomaarvudena (*floating-point number*), mis on ligikaudsed ja millega arvutamisel võivad tekkida väikesed ebatäpsused. Kahe tegelikult sama pika sirglõigu pikkusi arvutades võib arvuti saada natuke erinevad vastused (kuigi vahe on tavaliselt üliväike) ja lugeda siis lõigud eripikkusteks. Antud ülesandes ei ole see nii oluline, sest kui lõigu pikkuse ruut on täpne täisarvuna, siis sellest ruutjuurt võttes on tulemus siiski alati sama. Aga üldiselt tasub reaalarve vältida, sest niipea kui mõni vahetulemus on realaarv, hakkavad edasi arvutamisel ümardamisvead liituma.

4. Igav tund nelja ruuduga (`neli`)

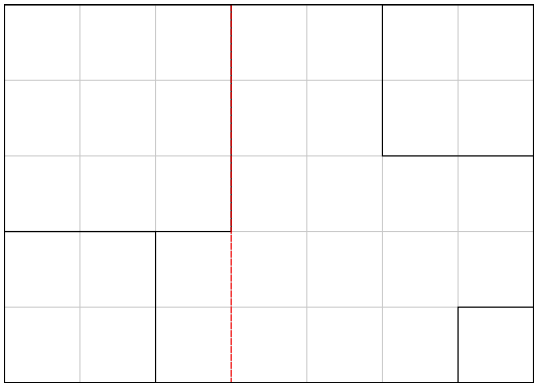
60 punkti

On antud $N \times M$ ruudustik. Mitu võimalust on joonistada ruudustiku igasse nurka ruut nii, et need ruudud omavahel ei kattu? ($2 \leqslant N, M \leqslant 10^7$.)

Lahendus

Eeldame üldisust kitsendamata, et $N \leqslant M$ — võime ju ruudustikku 90 kraadi pöörata, vastus sellest ei muutu.

Kui meil on mingi korrektne joonistus, võime alati tõmmata vertikaalse joone, mis eraldab vasakusse ja paremasse serva joonistatud ruudud. Seda vertikaalset joont võib liigutada vasakule nii kaua, kuni puutub ühe vasakpoolse ruuduga kokku.



Loendame, kui palju on selliseid joonistusi, kus see joon asub mingil fikseeritud positsioonil i. Selleks on $V[i] \cdot P[M - i]$ võimalust, kus:

- $V[i]$ loendab, kui mitu võimalust on panna vasakusse serva kaks ruutu nii, et nendest suurema küljepikkus on **täpselt** i;
- $P[i]$ loendab, kui mitu võimalust on panna paremasse serva kaks ruutu nii, et nendest suurema küljepikkus on **ülimalt** i.

Kui me oskame arvutada $V[i]$, siis me oskame arvutada ka $P[i]$. On kerge aru saada, et:

$$P[i] = V[1] + V[2] + \cdots + V[i].$$

Niisiis, kuidas arvutada massiivi $V[\cdot]$ elemente? Peame panema $N \times i$ ribale kaks ruutu nii, et üks nendest oleks laiusega täpselt i. Paneme tähele, et:

- kui $i \geqslant N$, siis see on võimatu;
- kui ülemise ruudu laius on i, siis alumise ruudu valimiseks on $\min(i, N-i)$ võimalust;
- kui alumise ruudu laius on i, siis ülemise ruudu valimiseks on $\min(i, N-i)$ võimalust;
- praegult loendasime topelt neid olukordi, kus nii ülemise kui alumise ruudu laius on i, lahutame ühe sellise maha (kui neid on).

Seega $V[i]$ on võimalik iga i korral valemiga välja arvutada; selle põhjal saame arvutada iga i jaoks ka $P[i]$. Lõppvastus on siis [2]

$$\sum_{i=1}^{M-1} V[i] \cdot P[M-i].$$

Keerukus $O(\max(N, M))$.

Väljakutse

Lahenda ülesanne keerukusega $O(1)$, s.t. et toimiks ka näiteks juhul $N, M \leqslant 10^{18}$.

5. Kui palju võimalusi? (`palju`)

100 punkti

On antud $N \times N$ "klaviatuur" — ruudustik, kus iga rida algab eelmisest poole ruudu võrra paremal. Tehti $K+1$ klahvivajutust; meil on antud vajutatud klahvide omavahelised kaugused $A[1], \ldots, A[k]$. Kui palju on võimalikke klahvivajutuste jadasid, mis tekitavad sellise kauguste jada? ($2 \leqslant N \leqslant 300, 1 \leqslant K \leqslant 300$.)

Lahendus

Esiteks lepime kokku klaviatuuri klahvide nummerduse. See on selline:

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)
(5, 0)	(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)

Lahendame ülesande dünaamilise planeerimise abil. Teeme kolmemõõtmelise tabeli dp ja seame eesmärgiks täita tabelit nii, et $\mathrm{dp}[t][x][y]$ vastab küsimusele "mitu võimalust on teha esimesed t klahvivajutust nii, et jõuame pärast t-ndat vajutust klahvile (x, y)".

[2] Kirjutis $\sum_{i=A}^{B}$ tähistab summat, kus liidetakse kokku $\sum$-märgi järel oleva avaldise väärtused $i = A$, $i = A+1$, $\ldots$, $i = B$ jaoks.

On selge, et $\mathrm{dp}[1][x][y] = 1$ iga (x, y) korral. Samuti võime tähele panna, et kui $t > 1$, siis

$$\mathrm{dp}[t][x][y] = \sum_{(u,v)} \mathrm{dp}[t-1][u][v],$$

kus summa on võetud üle selliste ruutude (u, v), mille kaugus ruudust (x, y) on täpselt $A[t-1]$. Siis ülesande vastuseks on

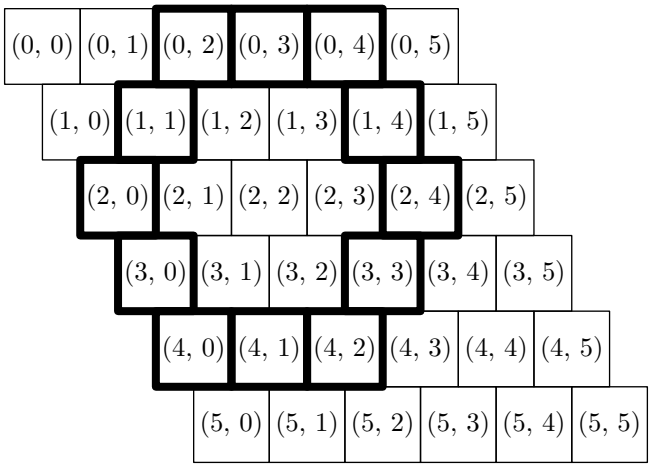
$$\sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathrm{dp}[K+1][x][y].$$

Kõige naiivsem viis DP tabeli täitmiseks on selline:

```
for (int t = 2; t <= K + 1; t++)
  for (int x = 0; x < N; x++)
    for (int y = 0; y < N; y++)
      for (int u = 0; u < N; u++)
        for (int v = 0; v < N; v++)
          if (dist(x, y, u, v) == A[t - 1])
            dp[t][x][y] += dp[t - 1][u][v];
```

Siin `dist` on mingi funktsioon, mis arvutab kahe ruudu vahelist kaugust. On selge, et sellise lahenduse keerukus on $O(N^4K)$. Sellest piisab, et esimene alamülesanne ära lahendada.

Aga nii külastame igas tsüklis palju ruute, mille kaugus ruudust (x, y) on vale, ja raiskame palju aega. Paneme tähele, et ruudud, mis on ruudust (x, y) kaugusel r, moodustavad kuusnurkse "rõnga":



Kuusnurgal asub vaid $O(N)$ ruutu. Samas on suhteliselt lihtne kirjutada tsükkel, mis itereerib üle selle kuusnurga, sest see koosneb kuuest sirgest lõigust:

```
for (int t = 2; t <= K + 1; t++)
  for (int x = 0; x < N; x++)
    for (int y = 0; y < N; y++)
      for ((u, v) in hexagon(x, y, A[t - 1]))
        // kaib labi O(N) ruutu
        dp[t][x][y] += dp[t - 1][u][v]
```

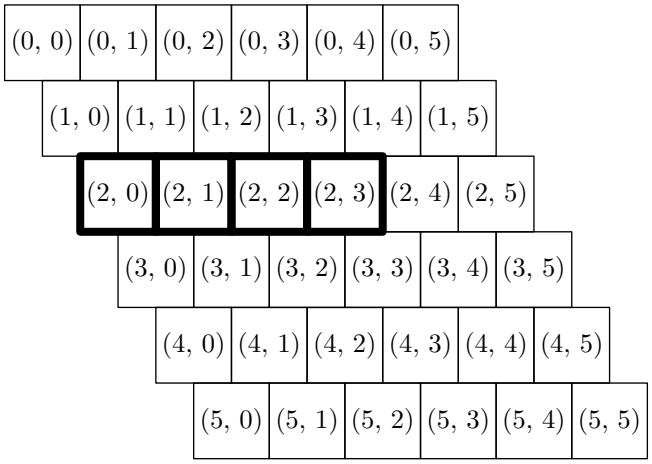
Nii on keerukus $O(N^3K)$. Sellest piisab, et teine alamülesanne ära lahendada.

Täispunktide saamiseks kasutame osasummade ehk prefiksisummade võtet. Oletame, et oleme arvutanud tabeli kihi $\mathrm{dp}[t-1][\cdot][\cdot]$ ja soovime nüüd hakata täitma kihti $\mathrm{dp}[t][\cdot][\cdot]$. Enne, kui

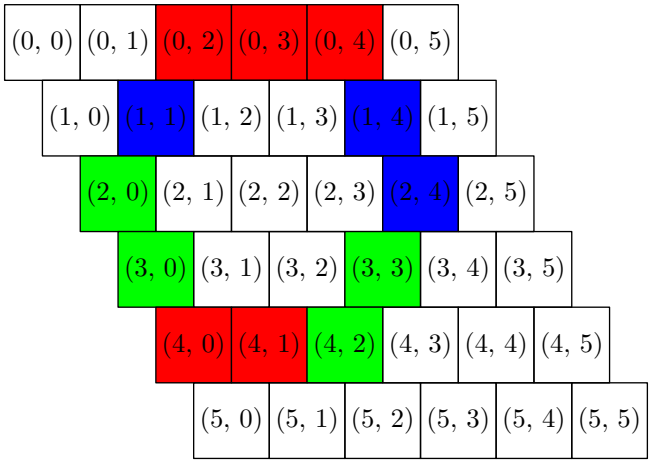
täitma asume, teeme teise tabeli pref. Täidame selle nii, et

$$\text{pref}[x][y] = \sum_{u=0}^{y} \text{dp}[t-1][x][u].$$

Näiteks pref[2][3] on dp[$t-1$][2][0]+dp[$t-1$][2][1]+dp[$t-1$][2][2]+dp[$t-1$][2][3] ehk dp[$t-1$][$\cdot$][$\cdot$] summa üle nende ruutude, mis märgitud rasvaselt:



Analoogiline osasummade tabel tuleb teha kummagi diagonaalse suuna jaoks. Oletame, et nüüd soovime arvutada dp[t][x][y] väärtust. Vaatleme klahvist (x, y) kaugusel $A[t-1]$ asuvat rõngast. Jaotame selle kuueks lõiguks:



Siis näiteks ülemisel vertikaalsel lõigul olevate DP väärtuste summa saab lihtsalt arvutada valemiga pref[0][4] − pref[0][1], sest

$$\begin{aligned}\text{pref}[0][4] - \text{pref}[0][1] = {} & \text{dp}[u][0][0] + \text{dp}[u][0][1] + \text{dp}[u][0][2] + \text{dp}[u][0][3] + \text{dp}[u][0][4] \\ & - \text{dp}[u][0][0] - \text{dp}[u][0][1] \\ = {} & \text{dp}[u][0][2] + \text{dp}[u][0][3] + \text{dp}[u][0][4]\end{aligned} =$$

ja see on otsitav summa. Analoogiliste tabelitega saab arvutada ülejäänud viiel lõigul oleva summa. Nii saame dp tabeli täita keerukusega $O(N^2K)$, millega laheneb ülesanne täispunktidele.