1. Jõululaul (`laul`)

20 punkti

Idee: Tähvend Uustalu, teostus ja lahenduse selgitus: Jaagup Kippar

Kõigepealt tasub ülesanne käsitsi läbi mõelda. Kui jõulud piirduvad ühe päevaga, siis saadaksegi ainult üks kingitus, vastuseks on 1. Kahe päeva puhul saadakse laulusalmi järgi esimesel päeval üks esimene kingitus ning teisel päeval kaks teise päeva kingitust ja üks esimese päeva kingitus. Kokku siis esimese päeva kingitusi kaks ning teise päeva kingitusi samuti kaks. Kolmandal päeval saadakse eelnevale lisaks kolm kolmanda päeva kingitust, kaks teise päeva kingitust ja üks esimese päeva kingitus, mis vaja siis eelnevatele juurte liita. Sinnamaani saab kasvõi valikutega programmi kokku panna, nagu on tehtud näidetes `sol0.py` ja `sol0.cpp`. Niimoodi jätkates oleks täiesti võimalik (kuigi üsna tüütu) esimene alamülesanne ära lahendada.

`sol0.py`

```
n = int(input())
if n == 1:
  print(1)
if n == 2:
  print(2)
  print(2)
if n == 3:
  print(3)
  print(4)
  print(3)
```

Pikemaks arvutamiseks on kasulik programm nõnda koostada, et iga järgneva päeva kohta ei pea uut programmiosa kirjutama. Võimalik on hoida päevade/kingituste pikkuses massiivis iga päevaga alanud kingituste koguarvusid. Jõulude algusest arvutama hakates saab igal päeval lisada juurdetulnud kingituste arvud vastavatele kingituste koguarvudele ning lõpuks massiivi sisu välja trükkida, nagu on tehtud näidetes `sol1.py` ja `sol1.cpp`.

`sol1.py`

```
n = int(input())
kk = [0] * (n + 1)
# kk on nüüd nullidega täidetud massiiv
# liiget nr 0 pärast ei kasutata
for p in range(1, n + 1):
  for k in range(1, p + 1):
    kk[k] += k
for k in range(1, n + 1):
  print(kk[k])
```

Kui jõulud kestavad kuni aasta või ka tuhatkond päeva, on tulemuse nii arvutamine arvutile jõukohane. Ülesande tingimuses lubatud 100 000 päeva puhul läheb aga operatsioonide arv programmis oleva kahekordse korduse tõttu liiga suureks — viie miljardi kanti. Iga päeva kohta arvutatakse eraldi läbi kõik seni antavad kingitused. Nii tasub vaadata, kas õnnestub arvutamine miskil moel lihtsamaks muuta. Laulusalmi jälgides tuleb välja, et esimest kingitust antakse igal päeval üks — ehk siis nende koguarv on N. Teist kingitust antakse alates teisest päevast iga päev kaks — nende koguarv tuleb siis $(N-1)\cdot 2$. Kolmanda kingituse puhul saame koguarvu avaldiseks $(N-2)\cdot 3$. Ehk siis üldise avaldisena tuleb kingituse number k koguarv $(N-(k-1))\cdot k$ või ümber tõstetuna $k\cdot(N+1-k)$, nagu on tehtud näidetes `sol2.py` ja `sol2.cpp`

`sol2.py`

```
n = int(input())
for k in range(1, n + 1):
  print(k * (n + 1 - k))
```

2. Kingikott (kott)

30 punkti

Idee ja teostus: Ago Luberg, lahenduse selgitus: Ahto Truu

Üks võimalus on lahendada see ülesanne vahetult teksti järgi: vaatame läbi kõik poes olevate kaupade paarid ja arvutame iga paari jaoks kingikoti hinna nende kahe kauba hindade vahetuse järel. Nii ongi tehtud lahendustes `sol1.py` ja `sol1.cpp`, mis käivad läbi kõik $N^2/2$ paari ja iga paari korral liidavad vahetult kokku kõigi M kotis oleva kingi hinnad, tehes nii kokku suurusjärgus $N^2 \cdot M$ operatsiooni. See on piisavalt efektiivne, et lahendada esimene alamülesanne, aga jääb suuremates testides ajahätta.

Parema lahenduse saamiseks paneme tähele, et kui meil on kaupa i hinnaga H_i kotis N_i tükki ja kaupa j hinnaga H_j vastavalt N_j tükki, siis nende hindade vahetamise efekt on, nagu me võtaks kotist ära kaupa väärtuses $N_i \cdot H_i + N_j \cdot H_j$ ja paneks asemele kaupa väärtuses $N_i \cdot H_j + N_j \cdot H_i$, ehk kokku väheneb koti hind $(N_i \cdot H_i + N_j \cdot H_j) - (N_i \cdot H_j + N_j \cdot H_i) = (N_i - N_j) \cdot (H_i - H_j)$ võrra. Seda ongi kasutatud lahendustes `sol2.py` ja `sol2.cpp`, mis arvutavad koti hinna ainult ühe korra ja edasi leiavad iga paari jaoks vahetuse efekti konstantse ajaga. Nii kulutavad nad optimaalse hinna leidmiseks suurusjärgus $M + N^2$ operatsiooni, mis on piisavalt efektiivne kõigi testide läbimiseks.

`sol1.py`

```
hh = {} #  kaupade hinnad
n = int(input())
for i in range(n):
  s = input()
  p = int(input())
  hh[s] = p

kk = [] # kinginimekiri
m = int(input())
for i in range(m):
  s = input()
  kk.append(s)

def summa():
  s = 0
  for k in kk:
    s += hh[k]
  return s

parim = summa()
for k1 in hh:
  for k2 in hh:
    if k1 == k2:
      break
    # vahetame prooviks k1 ja k2 hinnad
    hh[k1], hh[k2] = hh[k2], hh[k1]
    parim = min(parim, summa())
    hh[k1], hh[k2] = hh[k2], hh[k1]
print(parim)
```

`sol2.py`

```
hh = {} # kaupade hinnad
nn = {} # kaupade kogused nimekirjas
n = int(input())
for i in range(n):
  s = input()
  p = int(input())
  hh[s] = p
  nn[s] = 0

m = int(input())
for i in range(m):
  s = input()
  nn[s] += 1

summa = 0
for k in hh:
  summa += hh[k] * nn[k]

parim = 0
for k1 in hh:
  for k2 in hh:
    if k1 == k2:
      break
    # leiame k1 ja k2 vahetamise efekti
    dh = hh[k1] - hh[k2]
    dn = nn[k1] - nn[k2]
    parim = max(parim, dh * dn)
print(summa - parim)
```

4. Pael (`pael`) 50 punkti

Idee: Kaarel Hänni, teostus: Targo Tennisberg, lahenduse selgitus: Tähvend Uustalu

Tähistame i-nda päkapiku paela pikkust (mis on meil ette antud) P_i, i-nda päkapiku asukohta (mida meil on vaja leida) X_i ning i-nda lõikekoha asukohta C_i. Lõikekohtade asukohad on meil sisuliselt teada; need saab välja arvutada valemiga

$$C_i = P_1 + P_2 + \cdots + P_i.$$

Sageli on sellistes ülesannetes kasulik mõelda nii: kui meil on teada mingi osa lahendist (näiteks ainult X_1), kuidas sõltub sellest lahendi ülejäänud osa? Missuguseid piiranguid saab siis järeldada X_1 kohta?

Antud juhul tuleb välja, et X_1 põhjal saab kõik muud X_i väärtused teada. Tõepoolest:

- Kui on teada X_1, siis peab kehtima $C_1 - X_1 = X_2 - C_1$, sest X_1 ja X_2 peavad punktist C_1 võrdsel kaugusel olema. Seega
$$X_2 = -X_1 + 2C_1.$$
- Kui on teada X_2, siis peab kehtima $C_2 - X_2 = X_3 - C_2$. Seega
$$X_3 = -X_2 + 2C_2 = X_1 - 2C_1 + 2C_2.$$
- Kui on teada X_3, siis peab kehtima $C_3 - X_3 = X_4 - C_3$. Seega
$$X_4 = -X_3 + 2C_3 = -X_2 + 2C_1 - 2C_2 + 2C_3.$$
- jne.

Induktsiooniga saame näidata, et

$$X_i = (-1)^{i+1} X_1 + A_i,$$

kus

$$A_i = 2C_{i-1} - 2C_{i-2} + 2C_{i-3} - \cdots + (-1)^i 2C_1.$$

Sugugi mitte iga X_1 korral ei saa me nii lahendit. On mõned lisatingimused:

- Päkapikk peab seisma oma kahe lõikepunkti vahel: iga i korral peab kehtima $C_{i-1} \leqslant X_i \leqslant C_i$.
  - Piirjuhul $i = 1$ peab kehtima $0 \leqslant X_1 \leqslant C_1$.
  - Piirjuhul $i = N$ peab kehtima $C_{N-1} \leqslant X_N \leqslant C_N$, kus C_N on terve paela pikkus.
- Päkapikud peavad olema erinevates kohtades: iga i korral peab kehtima $X_i < X_{i+1}$.

Saab näidata, et kui kõik need võrratused kehtivad, siis on tegu ülesande tingimustele vastava lahendiga.

Teisendame pisut seda teist võrratust. Vastavalt ülaltoodud arutelule $X_{i+1} = 2C_i - X_i$ ehk $X_i < 2C_i - X_i$, kust $X_i < C_i$.

Seega on meil hunnik võrratusi, millest igaüks on kujul

$$C_{i-1} \leqslant X_1 + A_i < C_i \qquad \text{või} \qquad C_{i-1} \leqslant -X_1 + A_i < C_i.$$

Juhul $i = N$ muutuvad ranged võrratused mitterangeteks. Nende käsitlemine on analoogiline. Teisendame iga võrratuse vastavalt kujule

$$C_{i-1} - A_i \leqslant X_1 < C_i - A_i \qquad \text{või} \qquad -C_i + A_i < X_1 \leqslant -C_{i-1} + A_i$$

ja kasutades ära seda, et lahend peab olema täisarvuline, saame vastavalt

$$C_{i-1} - A_i \leqslant X_1 \leqslant C_i - A_i - 1 \qquad \text{või} \qquad -C_i + A_i + 1 \leqslant X_1 \leqslant -C_{i-1} + A_i$$

Võtame kõikide võrratuste vasakutest pooltest suurima ja paremate pooltest vähima. X_1 peab jääma nende kahe arvu vahele. Kui see on võimatu, väljastame `EI`, vastasel juhul valime sobiva X_1 ja arvutame juba teadaolevate valemitega ülejäänud X_i. Keerukus on $\mathcal{O}(N)$.

5. *Linesweeper* (`lswp`)

100 punkti

Idee, teostus ja lahenduse selgitus: Tähvend Uustalu

Alamülesanne 1

Esimeses alamülesandes on sisuliselt ainult üks mängulaud. Strateegia on järgmine:

1. Käime läbi kõik esimest tüüpi päringud ja leiame mängulaua ülemise rea hetkel, kui tuleb esimene teist tüüpi päring.
2. Leiame ülemise rea alusel miinide asukohad.
3. Vastame leitud alumise rea põhjal teist tüüpi päringutele.

Mängulaud on mõlemas suunas lõpmatu; seejuures on alguses ja lõpus lõpmatu arv nulle. Ülemisel real kohal 0 olevat arvu päringud ei muuda, seega teame, et kohtadel 0 ja 1 miine ei ole. Oletame, et meil on mingi i korral teada järgmised asjad:

- kas kohal $i-1$ on miin;
- kas kohal i on miin;
- ülemise rea arvu kohal i.

Siis see, kas kohal $i+1$ on miin, on üheselt määratud. Võimegi siis niimoodi vasakult alustades alumise rea üles ehitada.

Alamülesanne 2

Esimeses alamülesandes panime tähele, et kui on teada, kas kohtadel i ja $i+1$ on miinid ja kui on teada, mis on kohal i olev arv, siis määrab see üheselt ära, kas kohal $i+1$ on miin. Võime mõelda nii, et arvudele $0, 1, 2, 3$ vastavad funktsioonid, mis võtavad sisendina eelmisel ja praegusel kohal oleva alumise ruudu ja tagastavad praegusel ja järgmisel kohal oleva ruudu. Kui järgmisele kohale pole võimalik midagi panna, siis on tegemist vastuoluga.

Formaalselt defineerime hulga $S = \{\texttt{oo}, \texttt{ox}, \texttt{xo}, \texttt{xx}, \lightning\}$ ja funktsioonid $f_0, f_1, f_2, f_3 \colon S \to S$ järgmise tabeliga:

	oo	ox	xo	xx	$\lightning$
f_0	oo	$\lightning$	$\lightning$	$\lightning$	$\lightning$
f_1	ox	xo	oo	$\lightning$	$\lightning$
f_2	$\lightning$	xx	ox	xo	$\lightning$
f_3	$\lightning$	$\lightning$	$\lightning$	xx	$\lightning$

Siin sümbol $\lightning$ tähistab vastuolu.

Unustame selle alamülesande piires vastuolud ära ja eeldame, et neid päringu küsimise ajal kunagi ei teki (mis ongi ju alamülesande piirang). Kui soovime leida, kas kohal i on miin, siis peaksime arvutama

$$g_i(g_{i-1}(\cdots g_1(\texttt{oo})\cdots))$$

ja vaatama saadud sümbolipaari esimest sümbolit. Siin g_i on kohal i olevale arvule vastav funktsioon, üks funktsioonidest $f_0, \ldots, f_3$.

Anname olekutele potentsiaalid: $p(\texttt{oo}) = p(\texttt{xx}) = 0$, $p(\texttt{ox}) = 1$ ja $p(\texttt{xo}) = 2$. Tabelit vaadates on selge, et f_0 ja f_3 jätavad oleku potentsiaali paigale, f_1 tõstab oleku potentsiaali ühe võrra (mod

3) ja f_2 tõstab oleku potentsiaali kahe võrra (mod 3). Ühesõnage — f_k tõstab oleku potentsiaali k võrra (mod 3). Teiselt poolt, kui meil on teada oleku

$$p(g_i(g_{i-1}(\cdots g_1(\texttt{oo})\cdots)))$$

potentsiaal ning kohal i olev arv, siis me saame selle põhjal oleku enda teada.

Niisiis selleks, et leida, kas kohal i on miin, arvutame ülemise rea kohtadel $1, 2, \ldots, i$ olevate arvude summa (mod 3) ja leiame vastuse saadud summa ning ülemise arvu põhjal. Vastus on nende andmete põhjal üheselt määratud.

Nüüd on meil vaja lahendada järgmine ülesanne. Antud on massiiv pikkusega $N = 10^5$ ja $Q \leqslant 2 \cdot 10^5$ päringut, millest igaüks on ühel kujudest:

1. Antud i ja c. Sea massiivi i-ndaks elemendiks c.
2. Antud i. Leia massiivi elementide $1, 2, \ldots, i$ summa (mod 3).

See on täpselt Fenwicki puu tüüpülesanne. Fenwicki puu täidab iga päringu keerukusega $\mathcal{O}(\log N)$, seega kokku keerukus on $\mathcal{O}(Q \log N)$.

Täislahendus

Kasutame teise alamülesande ideed edasi. Iga teist tüüpi päringu täitmiseks:

- Leiame funktsiooni
$$h_N = g_N \circ g_{N-1} \circ \cdots \circ g_2 \circ g_1$$
ja kontrollime, kas $h_N(\texttt{oo}) = \lightning$. Kui nii, siis väljastame `!`.
- Leiame funktsiooni
$$h_i = g_i \circ \cdots \circ g_2 \circ g_1$$
ja väljastame $h_i(\texttt{oo})$ esimese sümboli.

"Funktsiooni leidmise" all peame siinkohal silmas tabeli koostamist, kus on kirjas igale sisendile vastav väljund. Kuna määramispiirkond on väike — viieelemendiline — siis on see tehtav.

Funktsioonide leidmiseks saame kasutada lõikude puud, mille igas tipus hoiame viieelemendilise massiivina meeles mingit funktsiooni. Täpsemalt hoiame lõigule $[l \cdots r]$ vastavas tipus meeles funktsiooni $g_r \circ g_{r-1} \circ \cdots \circ g_l$.

Alternatiivne lahendus

Alternatiivina on võimalik järgmine lähenemine: klassifitseerime ära kõik ülemised read, millele alumist rida ei leidu. Seejärel leiutame andmestruktuuri, mis vastavalt klassifikatsioonile suudab kontrollida, kas ülemisele reale vastavaid alumisi ridu leidub. Kui ei leidu, väljastame `!`; kui leidub, leiame vastava sümboli (näiteks alamülesande 2 lahenduse abil).

Saab näidata, et ülemine rida on vastuoluline parajasti siis, kui ülemisel real kehtib vähemalt üks järgmistest:

1. 0 ja 2 on kõrvuti;
2. 0 ja 3 on kõrvuti;
3. 1 ja 3 on kõrvuti;
4. prefiksisumma on nullist erinev (mod 3) seal, kus ülemisel real on 0;

5. prefiksisumma on nullist erinev (mod 3) seal, kus ülemisel real on 3;
6. prefiksisumma on null (mod 3) seal, kus ülemisel real on kõrvuti 1 ja 2;
7. prefiksisumma on null (mod 3) seal, kus ülemisel real on kõrvuti 2 ja 1.

Jooksvalt tingimuste 1-3 kontrollimine on triviaalne. Tingimuste 4-7 kontrollid on kõik täiesti analoogilised, seega keskendume vaid tingimusele 4.

Meile piisaks andmestruktuurist, mis hoiab endas massiivi pikkusega $N = 10^5$ ja oskab efektiivselt teostada järgmisi operatsioone:

- Antud i. Märgi i-s element *eriliseks*.
- Antud i. Eemalda i-ndalt elemendilt erilisuse märge.
- Antud i ja c. Liida elementidele $i, i+1, i+2, \ldots, N$ arv c (mod 3).
- Kontrolli, kas kõikides erilistes punktides on arv 0.

Tõepoolest, siis võime märkida erilisteks parajasti need ruudud, kus ülemisel real on 0.

Kirjeldame sellist andmestruktuuri. Tükeldame massiivi tükkideks pikkusega $\sqrt{N}$. Igas blokis hoiame meeles, kui palju erilisi punkte on kohal, kus väärtus on 0, kui palju kohal, kus 1 ning kui palju kohal, kus 2. Kui tuleb uuendus, mis mõjutab mingeid selle bloki elemente (aga mitte kõiki), siis käime elemendid läbi ja uuendame neid, mida on vaja. Kui tuleb uuendus, mis mõjutab kõiki selle bloki elemente, siis jätame liidetava arvu meelde ja kasutame seda väärtust ainult päringutele vastates.