Að leysa keppnisforritunardæmi

Almenn aðferðafræði og grunnþekking

Atli Fannar Franklín

24. febrúar 2019

Efnisyfirlit

Þekking forritunarmála

- Í keppnum eru C++ og Java ávallt leyfð, oft Python líka. Stundum má nota hvað sem er. Við munum halda okkur við C++ og einstöku sinnum grípum við kannski í Python (aðallega svo við þurfum ekki að skrifa okkar eigin bigInt útfærslu).

Þekking forritunarmála

- Í keppnum eru C++ og Java ávallt leyfð, oft Python líka. Stundum má nota hvað sem er. Við munum halda okkur við C++ og einstöku sinnum grípum við kannski í Python (aðallega svo við þurfum ekki að skrifa okkar eigin bigInt útfærslu).
- Við munum nú fara snöggt í það sem þið ættuð að þekkja í því forritunarmáli sem þið munuð nota í þessum kúrs.

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:
  - `int` - 32 bita formerkjaheiltala

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:
  - `int` - 32 bita formerkjaheiltala
  - `unsigned int` - 32 bita formerkjalaus heiltala

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:
  - `int` - 32 bita formerkjaheiltala
  - `unsigned int` - 32 bita formerkjalaus heiltala
  - `long long` - 64 bita formerkjaheiltala

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:
  - `int` - 32 bita formerkjaheiltala
  - `unsigned int` - 32 bita formerkjalaus heiltala
  - `long long` - 64 bita formerkjaheiltala
  - `unsigned long long` - 64 bita formerkjalaus heiltala

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:
  - `int` - 32 bita formerkjaheiltala
  - `unsigned int` - 32 bita formerkjalaus heiltala
  - `long long` - 64 bita formerkjaheiltala
  - `unsigned long long` - 64 bita formerkjalaus heiltala
  - `double` - 32 bita fleytitala

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:
  - `int` - 32 bita formerkjaheiltala
  - `unsigned int` - 32 bita formerkjalaus heiltala
  - `long long` - 64 bita formerkjaheiltala
  - `unsigned long long` - 64 bita formerkjalaus heiltala
  - `double` - 32 bita fleytitala
  - `char` - 8 bita formerkjalaus tala

Grunntög

- Þið ættuð að þekkja eftirfarandi grunntög (eða hliðstæður þeirra) í ykkar máli:
  - `int` - 32 bita formerkjaheiltala
  - `unsigned int` - 32 bita formerkjalaus heiltala
  - `long long` - 64 bita formerkjaheiltala
  - `unsigned long long` - 64 bita formerkjalaus heiltala
  - `double` - 32 bita fleytitala
  - `char` - 8 bita formerkjalaus tala
  - `string` - Runa af `char` til að geyma texta

Gagnagrindur

- Þið ættuð að þekkja eftirfarandi gagnagrindur (eða hliðstæður þeirra) í ykkar máli:

Gagnagrindur

- Þið ættuð að þekkja eftirfarandi gagnagrindur (eða hliðstæður þeirra) í ykkar máli:
  - Fylki - `T a[n]`

Gagnagrindur

- Þið ættuð að þekkja eftirfarandi gagnagrindur (eða hliðstæður þeirra) í ykkar máli:
  - Fylki - `T a[n]`
  - Vigra - `vector<T> v`

Gagnagrindur

- Þið ættuð að þekkja eftirfarandi gagnagrindur (eða hliðstæður þeirra) í ykkar máli:
  - Fylki - `T a[n]`
  - Vigra - `vector<T> v`
  - Lista - `list<T> l`

Gagnagrindur

- Þið ættuð að þekkja eftirfarandi gagnagrindur (eða hliðstæður þeirra) í ykkar máli:
  - Fylki - `T a[n]`
  - Vigra - `vector<T> v`
  - Lista - `list<T> l`
  - Hlaða - `stack<T> s`

Gagnagrindur

- Þið ættuð að þekkja eftirfarandi gagnagrindur (eða hliðstæður þeirra) í ykkar máli:
  - Fylki - `T a[n]`
  - Vigra - `vector<T> v`
  - Lista - `list<T> l`
  - Hlaða - `stack<T> s`
  - Biðraðir - `queue<T> q`

Gagnagrindur

- Þið ættuð að þekkja eftirfarandi gagnagrindur (eða hliðstæður þeirra) í ykkar máli:
  - Fylki - `T a[n]`
  - Vigra - `vector<T> v`
  - Lista - `list<T> l`
  - Hlaða - `stack<T> s`
  - Biðraðir - `queue<T> q`
  - Tvíleitartré - `set<T> s` og `map<T, S> m`

Algengar aðgerðir

- Þið ættuð að þekkja hvernig megi framkvæma raðanir í ykkar máli (t.d. `std::sort`)

Algengar aðgerðir

- Þið ættuð að þekkja hvernig megi framkvæma raðanir í ykkar máli (t.d. `std::sort`)
- Þið ættuð að þekkja hvernig megi lesa af staðalinntaki og prenta á staðalúttak (t.d. `cin` og `cout` eða `scanf` og `printf`)

Efnisyfirlit

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?
- Það fer eftir keppninni. Við munum skoða þetta eins og það er í bæði ICPC og á Codeforces. Aðrar keppnir eru til sem fara fram öðruvísi eins og t.d. Google Code Jam, en það sem þið sjáið hér er ekki erfitt að heimfæra upp á slíkar keppnir.

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?
- Það fer eftir keppninni. Við munum skoða þetta eins og það er í bæði ICPC og á Codeforces. Aðrar keppnir eru til sem fara fram öðruvísi eins og t.d. Google Code Jam, en það sem þið sjáið hér er ekki erfitt að heimfæra upp á slíkar keppnir.
- Venjulegt keppnisforritunardæmi felur í sér eftirfarandi atriði:

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?
- Það fer eftir keppninni. Við munum skoða þetta eins og það er í bæði ICPC og á Codeforces. Aðrar keppnir eru til sem fara fram öðruvísi eins og t.d. Google Code Jam, en það sem þið sjáið hér er ekki erfitt að heimfæra upp á slíkar keppnir.
- Venjulegt keppnisforritunardæmi felur í sér eftirfarandi atriði:
  - Verkefnalýsingu

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?
- Það fer eftir keppninni. Við munum skoða þetta eins og það er í bæði ICPC og á Codeforces. Aðrar keppnir eru til sem fara fram öðruvísi eins og t.d. Google Code Jam, en það sem þið sjáið hér er ekki erfitt að heimfæra upp á slíkar keppnir.
- Venjulegt keppnisforritunardæmi felur í sér eftirfarandi atriði:
  - Verkefnalýsingu
  - Tímamörkum (og minnismörkum)

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?
- Það fer eftir keppninni. Við munum skoða þetta eins og það er í bæði ICPC og á Codeforces. Aðrar keppnir eru til sem fara fram öðruvísi eins og t.d. Google Code Jam, en það sem þið sjáið hér er ekki erfitt að heimfæra upp á slíkar keppnir.
- Venjulegt keppnisforritunardæmi felur í sér eftirfarandi atriði:
  - Verkefnalýsingu
  - Tímamörkum (og minnismörkum)
  - Inntaks- og úttakslýsing

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?
- Það fer eftir keppninni. Við munum skoða þetta eins og það er í bæði ICPC og á Codeforces. Aðrar keppnir eru til sem fara fram öðruvísi eins og t.d. Google Code Jam, en það sem þið sjáið hér er ekki erfitt að heimfæra upp á slíkar keppnir.
- Venjulegt keppnisforritunardæmi felur í sér eftirfarandi atriði:
  - Verkefnalýsingu
  - Tímamörkum (og minnismörkum)
  - Inntaks- og úttakslýsing
  - Einföld sýnidæmi

Keppnisforritunardæmi

- Hvað felur keppnisforritunardæmi í sér?
- Það fer eftir keppninni. Við munum skoða þetta eins og það er í bæði ICPC og á Codeforces. Aðrar keppnir eru til sem fara fram öðruvísi eins og t.d. Google Code Jam, en það sem þið sjáið hér er ekki erfitt að heimfæra upp á slíkar keppnir.
- Venjulegt keppnisforritunardæmi felur í sér eftirfarandi atriði:
  - Verkefnalýsingu
  - Tímamörkum (og minnismörkum)
  - Inntaks- og úttakslýsing
  - Einföld sýnidæmi
- Verkefnið felur þá í sér að skrifa forrit sem leysir gefið verkefni með því að lesa inn inntak á staðalinntaki eins og í lýsingu og prenta svarið út á forminu sem lýst er í úttakslýsingu á staðalúttak. Það að það leysi verkefnið felur þá í sér að það fari ekki yfir tíma- né minnistakmarkanir.

A Different Problem (af Kattis)

Write a program that computes the difference between non-negative integers.

Input

Each line of the input consists of a pair of integers. Each integer is between 0 and 10^{15} (inclusive). The input is terminated by end of file.

Output

For each pair of integers in the input, output one line, containing the absolute value of their difference.

Sample Input 1	Sample Output 1
10 12	2
71293781758123 72784	71293781685339
1 12345677654321	12345677654320

A Different Problem sýnilausn

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    // ATH: long long er nauðsynlegt
    // því int mun valda overflow
    long long a, b;
    while(cin >> a >> b) {
        cout << abs(a - b) << endl;
    }
}
```

Dómar

- Ef sýnilausninni að ofan er skilað fæst grænn haki við skilin og textann 'Accepted'. En hvað annað hefði geta komið upp?

Dómar

- Ef sýnilausninni að ofan er skilað fæst grænn haki við skilin og textann 'Accepted'. En hvað annað hefði geta komið upp?
- Möguleg svör (á Kattis) eru:
  - Accepted
  - Compile Error
  - Run Time Error
  - Time Limit Exceeded
  - Wrong Answer
  - Output Limit Exceeded
  - Memory Limit Exceeded
  - Judge Error (ætti aldrei að koma upp)

Dómar

- Ef sýnilausninni að ofan er skilað fæst grænn haki við skilin og textann 'Accepted'. En hvað annað hefði geta komið upp?
- Möguleg svör (á Kattis) eru:
  - Accepted
  - Compile Error
  - Run Time Error
  - Time Limit Exceeded
  - Wrong Answer
  - Output Limit Exceeded
  - Memory Limit Exceeded
  - Judge Error (ætti aldrei að koma upp)
- Sumar keppnir nota önnur kerfi en Kattis, t.d. Domjudge, eða jafnvel sín eigin kerfi. En þau eru yfirleitt svipuð þessu.

Að prófa forritið sitt

- Eins og þið sáuð hér fyrr þurfti að nota `long long` til að fá ekki overflow, en sýnidæmin sýndu það ekki.

Að prófa forritið sitt

- Eins og þið sáuð hér fyrr þurfti að nota `long long` til að fá ekki overflow, en sýnidæmin sýndu það ekki.
- Yfirleitt eru sýnidæmin lítil hjálp og maður þarf að passa sig að prófa forritið sitt á sem flestum mögulegum gildrum áður en maður skilar.

Að prófa forritið sitt

- Eins og þið sáuð hér fyrr þurfti að nota `long long` til að fá ekki overflow, en sýnidæmin sýndu það ekki.
- Yfirleitt eru sýnidæmin lítil hjálp og maður þarf að passa sig að prófa forritið sitt á sem flestum mögulegum gildrum áður en maður skilar.
- Yfirleitt fer maður í gegnum hluti eins og 'Hvað með mjög stór inntök?', 'Hvað með mjög lítil inntök?', 'Hvað með neikvæðar tölur?' etc. etc.

Að prófa forritið sitt

- Eins og þið sáuð hér fyrr þurfti að nota `long long` til að fá ekki overflow, en sýnidæmin sýndu það ekki.
- Yfirleitt eru sýnidæmin lítil hjálp og maður þarf að passa sig að prófa forritið sitt á sem flestum mögulegum gildrum áður en maður skilar.
- Yfirleitt fer maður í gegnum hluti eins og 'Hvað með mjög stór inntök?', 'Hvað með mjög lítil inntök?', 'Hvað með neikvæðar tölur?' etc. etc.
- Það að sjá út svona hluti sem þarf að prófa er bara æfing, en það er mikilvægt að hafa það í huga.

Nokkur Time Limit Exceeded ráð

- Yfirleitt þegar maður fær þetta svar er tímaflækja forritsins ekki nógu góð (skoðum það é eftir), en stundum dugar að kreista bara smá meiri hraða úr forritinu. Þá koma hér nokkur ráð:

Nokkur Time Limit Exceeded ráð

- Yfirleitt þegar maður fær þetta svar er tímaflækja forritsins ekki nógu góð (skoðum það é eftir), en stundum dugar að kreista bara smá meiri hraða úr forritinu. Þá koma hér nokkur ráð:
  - Ef mikið er lesið prentað til skiptis, notið `cin.tie(NULL)`. Ef mikið er lesið eða prentað, notið `ios::sync_with_stdio(false)`. Ekki nota `std::endl`, notið frekar `'\n'`.

Nokkur Time Limit Exceeded ráð

- Yfirleitt þegar maður fær þetta svar er tímaflækja forritsins ekki nógu góð (skoðum það é eftir), en stundum dugar að kreista bara smá meiri hraða úr forritinu. Þá koma hér nokkur ráð:
  - Ef mikið er lesið prentað til skiptis, notið `cin.tie(NULL)`. Ef mikið er lesið eða prentað, notið `ios::sync_with_stdio(false)`. Ekki nota `std::endl`, notið frekar `'\n'`.
  - Forðist þess að dýrar aðgerðir eins og deiling og modulus séu framkvæmdar ef nota má einfaldari aðgerðir eins og samanburð og samlagningu í staðinn.

Nokkur Time Limit Exceeded ráð

- Yfirleitt þegar maður fær þetta svar er tímaflækja forritsins ekki nógu góð (skoðum það é eftir), en stundum dugar að kreista bara smá meiri hraða úr forritinu. Þá koma hér nokkur ráð:
  - Ef mikið er lesið prentað til skiptis, notið `cin.tie(NULL)`. Ef mikið er lesið eða prentað, notið `ios::sync_with_stdio(false)`. Ekki nota `std::endl`, notið frekar `'\n'`.
  - Forðist þess að dýrar aðgerðir eins og deiling og modulus séu framkvæmdar ef nota má einfaldari aðgerðir eins og samanburð og samlagningu í staðinn.
  - Forreiknið eitthvað, ef þið reiknið sömu gildi á $\cos$ oft t.d., forreiknið þau og flettið upp.

Efnisyfirlit

Tímaflækjur

- Tímaflækjur ættu að vera kunnuglegar þar sem þær eru teknar fyrir í stærðfræðimynstrum.

Tímaflækjur

- Tímaflækjur ættu að vera kunnuglegar þar sem þær eru teknar fyrir í stærðfræðimynstrum.
- Þær lýsa hraða forritsins sem fall af inntaksstærð. Ef forrit keyrir í $\mathcal{O}(f(n))$ þýðir það að tíminn vaxi í versta falli eins og $f(n)$ þegar inntaksstærð n vex. Svipað segir $\Theta(f(n))$ að hraðinn vaxi eins og $f(n)$ og $o(f(n))$ segir að hraðinn vaxi í besta falli eins og $f(n)$.

Tímaflækjur

- Tímaflækjur ættu að vera kunnuglegar þar sem þær eru teknar fyrir í stærðfræðimynstrum.
- Þær lýsa hraða forritsins sem fall af inntaksstærð. Ef forrit keyrir í $\mathcal{O}(f(n))$ þýðir það að tíminn vaxi í versta falli eins og $f(n)$ þegar inntaksstærð n vex. Svipað segir $\Theta(f(n))$ að hraðinn vaxi eins og $f(n)$ og $o(f(n))$ segir að hraðinn vaxi í besta falli eins og $f(n)$.
- Þetta hunsar alla fasta og við lítum svo á að allar grunnaðgerðir eins og samlagning og margföldun (á 32 og 64 bita heiltölum eða fleytitölum, ekki á bigInt) taki fastan tíma.

Tímaflækjur

- Tímaflækjur ættu að vera kunnuglegar þar sem þær eru teknar fyrir í stærðfræðimynstrum.
- Þær lýsa hraða forritsins sem fall af inntaksstærð. Ef forrit keyrir í $\mathcal{O}(f(n))$ þýðir það að tíminn vaxi í versta falli eins og $f(n)$ þegar inntaksstærð n vex. Svipað segir $\Theta(f(n))$ að hraðinn vaxi eins og $f(n)$ og $o(f(n))$ segir að hraðinn vaxi í besta falli eins og $f(n)$.
- Þetta hunsar alla fasta og við lítum svo á að allar grunnaðgerðir eins og samlagning og margföldun (á 32 og 64 bita heiltölum eða fleytitölum, ekki á bigInt) taki fastan tíma.
- Því er það að leggja saman n tölur $\mathcal{O}(n)$ en það að leggja þær saman 100 sinnum er líka $\mathcal{O}(n)$. Hins vegar væri það að leggja þær saman m sinnum $\mathcal{O}(mn)$.

Tímaflækja algengra hluta

- Að gera eitthvað sem tekur $\mathcal{O}(f(n))$ tíma n sinnum tekur $\mathcal{O}(nf(n))$ tíma.

Tímaflækja algengra hluta

- Að gera eitthvað sem tekur $\mathcal{O}(f(n))$ tíma n sinnum tekur $\mathcal{O}(nf(n))$ tíma.
- Að raða lista af tölum með góðu reikniriti tekur $\mathcal{O}(n \log(n))$ tíma. Hér skiptir grunntala lograns ekki máli því það munar bara fasta.

Tímaflækja algengra hluta

- Að gera eitthvað sem tekur $\mathcal{O}(f(n))$ tíma n sinnum tekur $\mathcal{O}(nf(n))$ tíma.
- Að raða lista af tölum með góðu reikniriti tekur $\mathcal{O}(n \log(n))$ tíma. Hér skiptir grunntala lograns ekki máli því það munar bara fasta.
- Að helmingunarleita (meir um það síðar) í röðuðum lista að staki er $\mathcal{O}(\log(n))$.

Tímaflækja algengra hluta

- Að gera eitthvað sem tekur $\mathcal{O}(f(n))$ tíma n sinnum tekur $\mathcal{O}(nf(n))$ tíma.
- Að raða lista af tölum með góðu reikniriti tekur $\mathcal{O}(n\log(n))$ tíma. Hér skiptir grunntala lograns ekki máli því það munar bara fasta.
- Að helmingunarleita (meir um það síðar) í röðuðum lista að staki er $\mathcal{O}(\log(n))$.
- Þessir reikningar geta aðstoðað mikið við að segja hvort forrit muni keyra innan gefinna tímamarka. Við miðum við að tölva geti framkvæmt 10^8 aðgerðir á sek. sem þumalputtareglu.

Hvað er hægt að gera í 10^8 aðgerðum?

Stærð n	Versta leyfilega tímaflækja	Dæmi
≤ 10	$\mathcal{O}(n!)$	Brute force TSP
≤ 15	$\mathcal{O}(n^2 2^n)$	DP TSP
≤ 20	$\mathcal{O}(n2^n)$	DP á hlutmengjum
≤ 100	$\mathcal{O}(n^4)$	Blossom reikniritið
≤ 400	$\mathcal{O}(n^3)$	Floyd-Warshall reikniritið
≤ 10000	$\mathcal{O}(n^2)$	LCS
≤ 100000	$\mathcal{O}(n\sqrt{n})$	Rótarþáttun
≤ 1000000	$\mathcal{O}(n \log(n))$	Byggja Fenwick-tré
≤ 100000000	$\mathcal{O}(n)$	NGE
> 100000000	$\mathcal{O}(\log(n))$, $\mathcal{O}(1)$	Stærðfræðileg lausn

Efnisyfirlit

Tegundir dæma

- Í framhaldinu munum við skoða ýmsar tegundir dæma og tegundir lausna á þeim.

Tegundir dæma

- Í framhaldinu munum við skoða ýmsar tegundir dæma og tegundir lausna á þeim.
- Nú fyrst munum við bara skoða svokölluð 'Ad hoc' dæmi. Skilgreining frá Merriam-Webster: 'formed or used for specific or immediate problems or needs', t.d. 'ad hoc solutions'.

Tegundir dæma

- Í framhaldinu munum við skoða ýmsar tegundir dæma og tegundir lausna á þeim.
- Nú fyrst munum við bara skoða svokölluð 'Ad hoc' dæmi. Skilgreining frá Merriam-Webster: 'formed or used for specific or immediate problems or needs', t.d. 'ad hoc solutions'.
- Þetta eru s.s. dæmi sem krefjast ekki sérþekkingar og snúast bara um að útfæra það sem þú ert beðinn um að útfæra. Þetta eru yfirlétt léttustu dæmin á keppnum, en þó ekki alltaf.

Dæmi um ad hoc

- Færð 3×3 mylluborð, er einhver búinn að vinna?

Dæmi um ad hoc

- Færð 3×3 mylluborð, er einhver búinn að vinna?
- Raðaðu gefin spil eftir tegund og svo innbyrðis í vaxandi röð.

Dæmi um ad hoc

- Færð 3×3 mylluborð, er einhver búinn að vinna?
- Raðaðu gefin spil eftir tegund og svo innbyrðis í vaxandi röð.
- Hvaða reit þarf biskup á skákborði að millilenda á ef hann á að fara frá a til b (ef hann getur farið frá a til b).

Dæmi um ad hoc

- Færð 3×3 mylluborð, er einhver búinn að vinna?
- Raðaðu gefin spil eftir tegund og svo innbyrðis í vaxandi röð.
- Hvaða reit þarf biskup á skákborði að millilenda á ef hann á að fara frá a til b (ef hann getur farið frá a til b).
- Leysum þrjú svona ad-hoc dæmi saman.

Mixed Fractions - Lýsing

You are part of a team developing software to help students learn basic mathematics. You are to write one part of that software, which is to display possibly improper fractions as mixed fractions. A proper fraction is one where the numerator is less than the denominator; a mixed fraction is a whole number followed by a proper fraction. For example the improper fraction 27/12 is equivalent to the mixed fraction 2 3/12. You should not reduce the fraction (i.e. don't change 3/12 to 1/4).

Mixed Fractions - I/O

Input

Input has one test case per line. Each test case contains two integers in the range $[1, 2^{31} - 1]$. The first number is the numerator and the second is the denominator. A line containing 0 0 will follow the last test case.

Output

For each test case, display the resulting mixed fraction as a whole number followed by a proper fraction, using whitespace to separate the output tokens.

Sample Input 1	Sample Output 1
27 12	2 3 / 12
2460000 98400	25 0 / 98400
3 4000	0 3 / 4000
0 0	

Mixed Fractions - Sýnilausn

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b, c;
    cin >> a >> b;
    while(a != 0 || b != 0) {
        cout << (a / b) << " " << (a % b) << " / " << b << endl;
        cin >> a >> b;
    }
}
```

Baby Bites - Lýsing

Arild just turned 1 year old, and is currently learning how to count. His favorite thing to count is how many mouthfuls he has in a meal: every time he gets a bite, he will count it by saying the number out loud.

Unfortunately, talking while having a mouthful sometimes causes Arild to mumble incomprehensibly, making it hard to know how far he has counted. Sometimes you even suspect he loses his count! You decide to write a program to determine whether Arild's counting makes sense or not.

Baby Bites - I/O

Input

The first line of input contains an integer n $(1 \leq n \leq 1000)$, the number of bites Arild receives. Then second line contains n space-separated words spoken by Arild, the i'th of which is either a non-negative integer a_i $(0 \leq ai \leq 10000)$ or the string "mumble".

Output

If Arild's counting might make sense, print the string "makes sense". Otherwise, print the string "something is fishy".

Sample Input 1
5
1 2 3 mumble 5

Sample Output 1
makes sense

Sample Input 2
8
1 2 3 mumble mumble 7 mumble 8

Sample Output 2
something is fishy

Sample Input 3
3
mumble mumble mumble

Sample Output 3
makes sense

Baby Bites - Sýnilausn

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    string s;
    cin >> n;
    for(int i = 1; i <= n; ++i) {
        cin >> s;
        if(s == "mumble" || s == to_string(i)) continue;
        cout << "something is fishy" << endl;
        return 0;
    }
    cout << "makes sense" << endl;
}
```

Run-Length Encoding, Run! - Lýsing

Forrest lives in a prehistoric era of "dial-up Internet." Unlike the fast streaming of today's broadband era, dial-up connections are only capable of transmitting small amounts of text data at reasonable speeds. Forrest has noticed that his communications typically include repeated characters, and has designed a simple compression scheme based on repeated information. Text data is encoded for transmission, possibly resulting in a much shorter data string, and decoded after transmission to reveal the original data.

The compression scheme is rather simple. When encoding a text string, repeated consecutive characters are replaced by a single instance of that character and the number of occurrences of that character (the character's run length). Decoding the encoded string results in the original string by repeating each character the number of times encoded by the run length. Forrest calls this encoding scheme run-length encoding. (We don't think he was actually the first person to invent it, but we haven't mentioned that to him.)

For example, the string HHHeellllo is encoded as H3e2l3o1. Decoding H3e2l3o1 results in the original string. Forrest has hired you to write an implementation for his run-length encoding algorithm.

Run-Length Encoding, Run! - I/O

Input

Input consists of a single line of text. The line starts with a single letter: E for encode or D for decode. This letter is followed by a single space and then a message. The message consists of 1 to 100 characters. Each string to encode contains only upper- and lowercase English letters, underscores, periods, and exclamation points. No consecutive sequence of characters exceeds 9 repetitions. Each string to decode has even length. Its characters alternate between the same characters as strings to encode and a single digit between 1 and 9, indicating the run length for the preceding character.

Output

On an input of E output the run-length encoding of the provided message. On an input of D output the original string corresponding to the given run-length encoding.

Sample Input 1	Sample Output 1
E HHHeellloWooorrrrlld!!	H3e2l3o1W1o3r4l2d1!2
Sample Input 2	Sample Output 2
D H3e2l3o1W1o3r4l2d1!2	HHHeellloWooorrrrlld!!

Run-Length Encoding, Run! - Sýnilausn

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c;
    string s;
    int cnt = 1;
    cin >> c >> s;
    if(c == 'E') {
        s.push_back('#');
        for(int i = 0; i < s.size() - 1; ++i) {
            if(s[i] == s[i + 1]) cnt++;
            else {
                cout << s[i] << cnt;
                cnt = 1;
            }
        }
        cout << '\n';
    } else {
        for(int i = 0; i < s.size(); i += 2) {
            for(int j = 0; j < s[i + 1] - '0'; ++j) cout << s[i];
        }
        cout << '\n';
    }
}
```