# PSA2: DN1

*Pa se vrnimo v zgodovino nazaj,*
*kako se začelo je in pa kdaj.*

DJ Jure

Jože Gorišek že dolgo več ne dela niti v TAM-u niti na fuš, saj se je zaposlil kot anketar. Skrbi za preprosto spletno stran, kamor lahko ljudje vpišejo svoje najljubše število.

Anketiranci niso ravno izvirni: vsi vnašajo le naravna števila ($0 \notin \mathbb{N}$), prav tako pa nikogaršnje najljubše število ne presega $10^6$.

Jožetovi nadrejeni bi radi odkrili, kakšna je zgodovina okusa ljudi, zato mu pošiljajo številne poizvedbe oblike $(t, n)$, kjer jih zanima *stopnja zgoščine* $Z(t, n)$ okoli števila $n$ do (vključno) časa $t$. Stopnjo zgoščine Jože izračuna v dveh korakih.

1. Najprej med vsemi odgovori, ki so bili podani do vključno časa $t$ (in so bili vnešeni pred poizvedbo), najde tiste štiri (označimo jih z $x_1$, ..., $x_4$), ki so številu $n$ najbližje na številski osi.

2. Izračuna stopnjo zgoščine $Z(t, n) = \sum_{i=1}^{4} |x_i - n|$.

Nižje vrednosti $Z(t, n)$ pripadajo popularnejšim območjem najljubših števil. Jožeta jutri čaka kosilo pri najboljšem kolegu, zato mu pomagajte pri hitrem procesiranju novih vnosov anketirancev in pri odgovorih na poizvedbe.

## Naloga A (5 točk)

Vse poizvedbe se zgodijo šele po vseh vnosih in so take, da je treba upoštevati vse vnose. Z drugimi besedami: na časovno komponento lahko pozabimo.

## Naloga B (5 točk)

Vnosi v bazo in poizvedbe prihajajo v mešanem vrstnem redu. Ko se zgodi poizvedba $(t, n)$, je treba nanjo odgovoriti takoj in pri tem upoštevati le tiste odgovore anketirancev, ki so že vneseni in so se zgodili do vključno časa $t$.

# Vhodni podatki

Vsaki od petih podnalog (A1, A2, A3, B1 in B2) pripada vhodna datoteka s končnico `.in` in izhodna datoteka s končnico `.out`. V vhodnih datotekah so vrstice oblike

```
tip čas število,
```

kjer je tip bodisi `v` (vnos) bodisi `p` (poizvedba), čas in število pa sta naravni števili, ki ne presegata $10^6$. Če gre za vnos, si morate par (`čas`, `število`) učinkovito zapomniti, sicer pa morate z do sedaj shranjenimi vrednostmi odgovoriti na poizvedbo.

Število vnosov v posamezni podnalogi ne presega $10^5$. Število poizvedb ne presega $10^4$. Odgovori $Z(t, n)$ ne presegajo $10^7$.

Lahko se zgodi, da si več ljudi izbere isto najljubše število in da se ob istem času zgodi več stvari (npr. vnos in poizvedba), kot je razvidno iz primerov na koncu. Časi vnosov naraščajo (a morda ne strogo), za čase, na katere se nanašajo poizvedbe, pa (pri nalogi B) omejitev ni.

# Naloga

V datoteki `resitev.h` vas že čaka razred `Resitev` z javno dostopno metodo `int obdelaj(const string & ukaz)`. Implementirajte jo. Kot lahko razberete iz `main.cpp`, je `ukaz` preprosto ena od vrstic iz vhodne datoteke. Njen izhod naj bo $Z(t, n)$, če je tip ukaza `p`, sicer pa je izhod poljuben. Če želite, lahko razširite razred z ustreznimi polji in metodami.

Datoteke `main.cpp` ne spreminjajte, lahko pa jo uporabite za preverjanje rešitev. Za povsem pravilno se šteje rešitev, ki je dovolj hitra in daje pravilne odgovore.

Opazili boste, da se čas izvajanja določi ob zagonu datoteke `main.exe`, saj imamo različno hitre računalnike[1].

Končno verzijo kode oddajte na učilnici (samo izvorno kodo), ker jo bomo izvajalci predmeta še enkrat preverili.

[1]Na 7-letnem prenosniku se vsak od primerov izvede v malo manj kot 200 ms (z uradno rešitvijo), dovoljen čas pa je približno 500 ms.

# Primer

Poleg glavnih nalog sta tu še primera vhoda za obe težavnosti (A0 in B0):

`A0.in`:

```
v 5 50
v 6 60
v 7 70
v 8 80
v 9 90
p 9 90
p 10 91
p 10 89
```

`A0.out`:

```
60
64
58
```

`B0.in`:

```
v 2 50
v 4 60
v 6 70
v 8 80
p 9 90
v 9 90
p 9 90
p 8 90
```

`B0.out`:

```
100
60
100
```

Ko računamo odgovor na prvi poizvedbo v B0 (`p 9 90`), upoštevamo prve štiri vnose, medtem ko jih pri drugi poizvedbi (prav tako `p 9 90`) upoštevamo pet (zato se odgovora razlikujeta). Pri tretji poizvedbi ponovno upoštevamo le prve štiri. Prvi in tretji odgovor sta zato enaka $100 = |90-80|+|90-70|+|90-60|+|90-50|$, drugi pa $60 = |90-90|+|90-80|+|90-70|+|90-60|$.

# Namig

Ker časi vnosov novih odgovorov ne padajo, se da uporabiti prilagoditev splošnega poldinamičnega principa, ki bo deloval tudi za brskanje po zgodovini. Del trika je v tem, da (statične) strukture na danem nivoju nikoli ne izbrišemo, ampak vedno dodamo le novo:

| | dodaj $x_1$ | dodaj $x_2$ | dodaj $x_3$ | dodaj $x_4$ | ... |
|---|---|---|---|---|---|
| nivo 0 | $[x_1]$ | $[x_1]$ | $[x_1]$, $[x_3]$ | $[x_1]$, $[x_3]$ | ... |
| nivo 1 | | $[x_1, x_2]$ | $[x_1, x_2]$ | $[x_1, x_2]$ | ... |
| nivo 2 | | | | $[x_1, x_2, x_3, x_4]$ | ... |
| $\vdots$ | | | | | |

kjer $[\dots]$ označuje statično strukturo (ne nujno seznam). Treba je še ugotoviti, ali se poraba prostora s tem ne poveča preveč, katere strukture so relevantne ob danem času $t$ ipd.