Dodatne naloge pri predmetu Programiranje 2

1 Razbijanje števil

Naloga

Napišite program, ki prebere pozitivni celi števili n in m, nato pa po vrsti izpiše posamezne dele (zaporedja števk) števila n, pri čemer je dolžina vsakega dela določena s pripadajočo števko v številu m. Dolžina prvega dela je tako enaka prvi števki števila m, dolžina drugega dela je enaka drugi števki števila m itd.

Nalogo rešite zgolj z operacijami nad celimi števili. Uporaba realnoštevilskih operacij, nizov, tabel, seznamov, vektorjev ipd. bo kaznovana s prepolovitvijo doseženega števila točk.

Vhod

Na vhodu sta podani celi števili $n \in [1, 10^{18}]$ in $m \in [1, 10^{18}]$, ločeni s presledkom. Vsota števk števila m je enaka številu števk števila n. Število m ne vsebuje nobene ničle.

Izhod

Na izhodu izpišite toliko vrstic, kolikor je števk števila m. V prvi vrstici izpišite začetnih a_1 števk števila n (pri čemer je a_1 prva števka števila m), v drugi sledečih a_2 števk števila n (pri čemer je a_2 druga števka števila m) itd.

Testni primer J1

Vhod:

```
362903157 2313
```

Izhod:

```
36
290
3
157
```

Testni primer J2

Vhod:

```
123456789087654321 72315
```

Izhod:

```
1234567
89
87
6
54321
```

Tretji del števila n je zapisan kot `87`, ne kot `087`, saj začetno ničlo pri izpisu celih števil izpuščamo.

2 Kodiranje čet

Naloga

Kodiranje čet (angl. run-length encoding) je tehnika stiskanja, ki se obnese, kadar niz, ki ga želimo stisniti, vsebuje dolga zaporedja enakih znakov. Obstaja več različic tega postopka, za potrebe te naloge pa se bomo ravnali po sledečih pravilih:

- V vhodnem nizu po vrsti obravnavamo zaporedja enakih znakov. V nizu `AAAAABCCC`, denimo, najprej obravnavamo zaporedje `AAAAA`, nato `B`, nazadnje pa še `CCC`.
- Zaporedje k znakov `#` (za $k \geqslant 1$) predstavimo s sosledjem `##`k`#`. Zaporedje `#` torej zapišemo kot `##1#`, zaporedje `##` kot `##2#`, zaporedje `###` kot `##3#` itd.
- Naj bo c katerikoli znak razen `#`. Zaporedje do štirih znakov c prepišemo v nespremenjeni obliki, zaporedje $k \geqslant 5$ znakov c pa predstavimo s sosledjem `#`ck`#`. Zaporedje `AAAA` potemtakem zapišemo kot `AAAA`, zaporedje `AAAAA` pa kot `#A5#`.

Navedena pravila enolično določajo *kodiranje* nizov, obratna pravila pa *dekodiranje* — pretvorbo kodiranih nizov nazaj v izvirne.

Napišite program, ki prebere ukaz u (1: kodiraj; 2: dekodiraj) in niz in izpiše rezultat kodiranja oz. dekodiranja podanega niza.

Vhod

Vhod je sestavljen iz ene same vrstice, ta pa vsebuje ukaz (število $u \in \{1, 2\}$), presledek, niz znakov iz množice {`'A'`, ..., `'Z'`, `'0'`, ..., `'9'`, `'#'`} in znak za skok v naslednjo vrstico (`'\n'`).

Noben vhodni ali izhodni niz ne vsebuje več kot 10^6 znakov. V testnih primerih z $u = 2$ je vhodni niz veljaven rezultat kodiranja izhodnega niza, zato se bo postopek dekodiranja vedno pravilno iztekel.

V testnih primerih J1–J6 in S1–S25 velja $u = 1$. V primerih J1–J4 in S1–S17 vhodni niz ne vsebuje nobenega znaka `#`. V primerih J1–J2 in S1–S8 vhodni niz ne vsebuje nobenega zaporedja z več kot 9 enakimi znaki.

V testnih primerih J7–J12 in S26–S50 velja $u = 2$. V primerih J7–J10 in S26–S42 izhodni niz ne vsebuje nobenega znaka `#`. V primerih J7–J8 in S26–S33 izhodni niz ne vsebuje nobenega zaporedja z več kot 9 enakimi znaki.

Izhod

V primeru $u = 1$ izpišite rezultat kodiranja, v primeru $u = 2$ pa rezultat dekodiranja vhodnega niza.

Testni primer J5

Vhod:

```
1 B7BBB7BBBBB7#777777777777777####777######7
```

Izhod:

```
B7BBB7#B5#7##1##715###4#777##6#7
```

Testni primer J11

Vhod:

```
2 B7BBB7#B5#7##1##715###4#777##6#7
```

Izhod:

```
B7BBB7BBBBB7#777777777777777####777######7
```

Napotki

Znake vhodnega niza lahko berete s funkcijo `getchar`. V C-ju so znaki predstavljeni z njihovimi ASCII-kodami, zato jih lahko obravnavate kar kot števila tipa `int`. Če želite znak izpisati kot znak, uporabite določilo `%c` v funkciji `printf`, določilo `%d` pa izpiše ASCII-kodo znaka.

Za lažje razumevanje si oglejmo primer. Predpostavimo, da vhod vsebuje niz `P2`:

```
int prvi = getchar();
int drugi = getchar();
printf("%d | %d\n", prvi, drugi);    // 80 | 50
printf("%c | %c\n", prvi, drugi);    // P | 2
if (prvi >= 'A' && prvi <= 'Z') {
    printf("%c je %d. črka angleške abecede\n", prvi, prvi - 'A' + 1);
}
```

3 Snake

Naloga

Naloga temelji na videoigri Snake, v kateri glavno vlogo igra »kača«, ki se premika po igralni površini (mreži kvadratnih celic) in pri tem golta pojavljajoče se predmete. Kača je sestavljena iz zaporedja členov, ki se nahajajo na sosednjih celicah na igralni površini.

Prvi člen kače se imenuje *glava*, *dolžina* kače pa je skupno število členov, iz katerih je kača sestavljena. Igre je konec, ko se glava kače zaleti v rob mreže ali sama vase.

Napišite program, ki izračuna dolžino kače in položaj njene glave po podanem številu korakov izvajanja igre. Igro bomo igrali po sledečih pravilih (ta se nekoliko razlikujejo od originalnih):

- **Igralna površina:**
  - Igralna površina je mreža kvadratnih celic, ki se v obeh dimenzijah razteza od koordinate -10^6 do koordinate 10^6. Koordinate x naraščajo v desno, koordinate y pa navzgor.
- **Kača:**
  - Kačo na začetku sestavlja le glava, ki zaseda celico na položaju (0, 0). Dolžina kače je na začetku igre potemtakem enaka 1.
  - Kača se na začetku giblje v smeri navzgor.
  - V vsakem koraku se glava kače premakne za eno celico v smeri gibanja, preostali členi kače pa se premaknejo na celice, ki so jih pred izvedbo koraka zasedali njihovi predhodniki.
  - Če se na koncu izvajanja koraka glava kače nahaja na isti celici kot nek drug člen kače (kača se zaleti sama vase), se igra takoj zaključi.
- **Predmeti:**
  - Na igralni površini se lahko nahaja poljubno število predmetov velikosti ene celice. Predmeti so na igralni površini prisotni od začetka do konca igre, ne glede na to, ali jih kača prečka.
  - Predmet se nikoli ne nahaja na položaju (0, 0).
  - *Predmet tipa 1 (sadež)*: ko glava kače zapusti predmet tipa 1, se kača na repu podaljša za eno celico. Na primer, če se v celici (1, 4) nahaja predmet tipa 1, kača pa (gledano od repa do glave) zavzema celice (1, 2), (1, 3) in (1, 4) (torej je predmet ravnokar »pojedla«), bo v naslednjem koraku zavzemala celice (1, 2), (1, 3), (1, 4) in (1, 5). Če na položaju (1, 4) ne bi bilo ničesar, bi kača v naslednjem koraku zavzemala celice (1, 3), (1, 4) in (1, 5).
  - *Predmet tipa 2 (zavij levo)*: ko glava kače doseže predmet tipa 2, se smer gibanja kače spremeni za 90 stopinj v levo.
  - *Predmet tipa 3 (zavij desno)*: ko glava kače doseže predmet tipa 3, se smer gibanja kače spremeni za 90 stopinj v desno.

Vhod

V prvi vrstici je podano število predmetov na igralni površini ($n \in [0, 10^3]$). V naslednjih n vrsticah so zapisani podatki o posameznih predmetih. Vsak predmet je opisan s koordinato $x \in [-10^6, 10^6]$, koordinato $y \in [-10^6, 10^6]$ in tipom ($t \in \{1, 2, 3\}$). Podatki o predmetu so med seboj ločeni s presledkom. V zadnji vrstici je zapisano število korakov izvajanja ($k \in [0, 10^3]$). Vsa števila na vhodu so cela.

V primerih J1–J5 in S1–S25 nastopajo samo predmeti tipa 2 in tipa 3.

Izhod

Izpišite dolžino kače ter koordinati x in y njene glave po podanem številu korakov izvajanja. Izpisana števila naj bodo med seboj ločena s presledkom.

Če se kača zaleti sama vase, potem namesto njene dolžine izpišite vrednost 0.

Testni primer J7

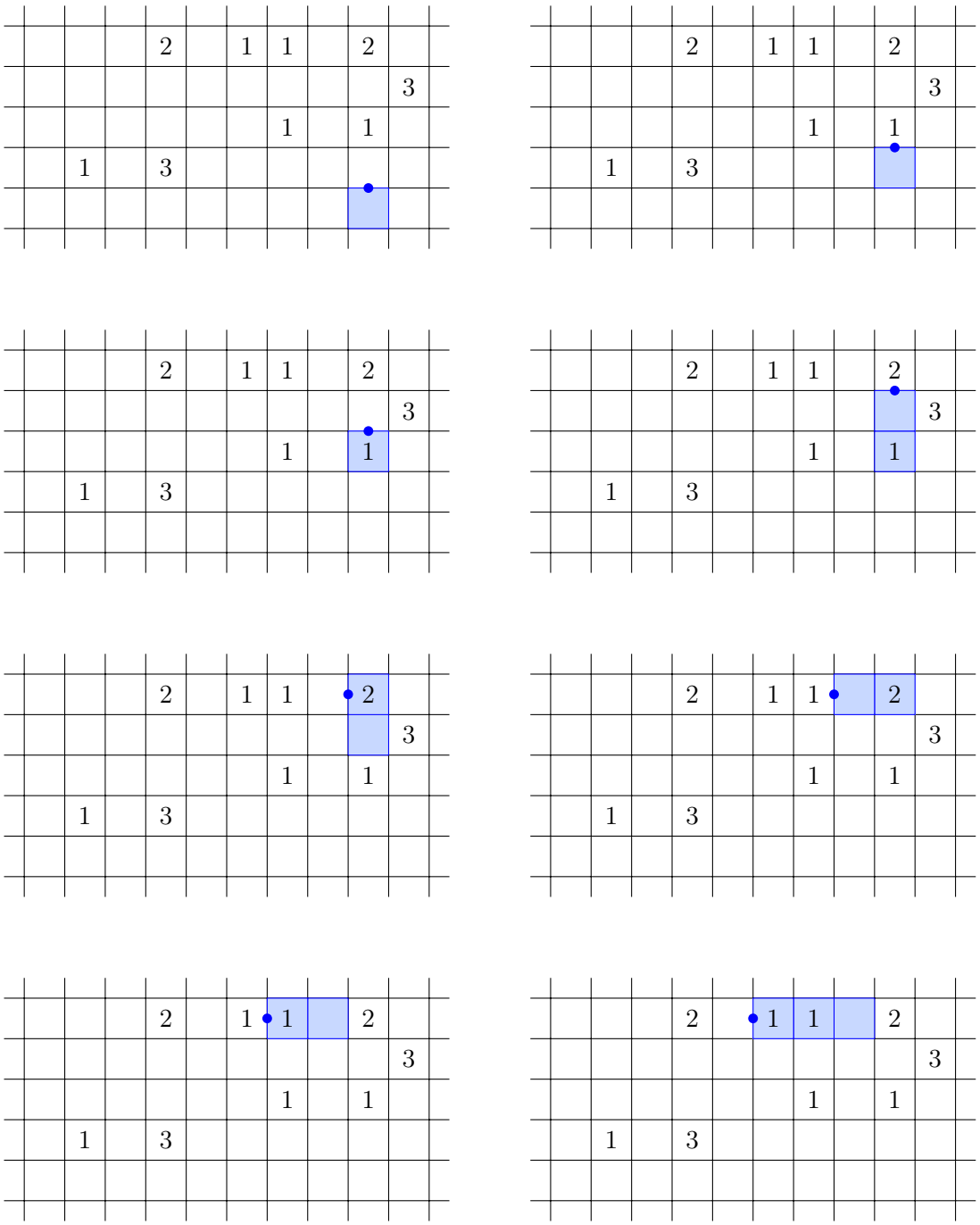
Vhod:

```
9
-5 4 2
-3 4 1
-2 4 1
0 4 2
1 3 3
-2 2 1
0 2 1
-7 1 1
-5 1 3
15
```

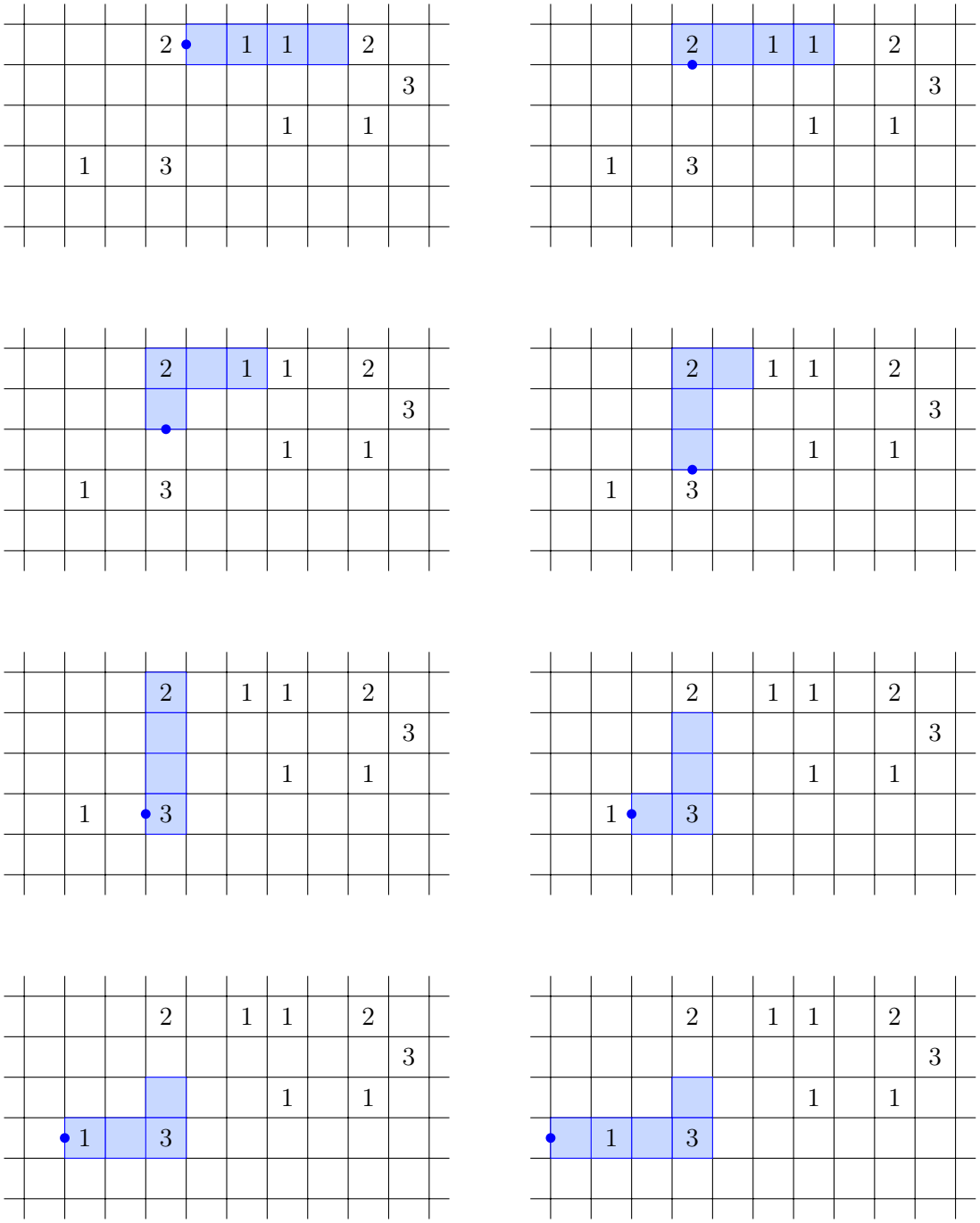
Izhod:

```
5 -8 1
```

Sliki 1 in 2 prikazujeta stanje igralne površine na začetku in po posameznih korakih izvajanja.



Slika 1: *Začetno stanje igralne površine in stanje po korakih 1–7 za testni primer J7.*



Slika 2: *Stanje igralne površine korakih 8–15 za testni primer J7.*

Testni primer J9

Vhod:

```
7
0 1 1
0 2 1
0 3 1
0 4 1
0 5 3
1 5 3
1 4 3
100
```

Izhod:

```
0 0 4
```

V tem primeru se kača že po 8 korakih zaleti sama vase.

4 Nadomestni znaki

Naloga

Napišite program, ki prebere vzorčni niz, število n in n testnih nizov, nato pa izpiše število testnih nizov, ki se ujemajo z vzorčnim. Vzorčni niz lahko vsebuje nadomestne znake (angl. wildcards) `?` in `*`. Vprašaj lahko nadomesti poljuben znak, zvezdica pa poljubno (lahko tudi prazno) zaporedje znakov. Na primer, z vzorcem `p?t` se ujemajo nizi `pat`, `pet`, `pot`, `prt`, `ptt` ipd., ne pa nizi `poet`, `povest`, `pt` ipd. Z vzorcem `p*t` pa se ujemajo vsi našteti nizi.

Vhod

V prvi vrstici je zapisan vzorčni niz, v drugi pa celo število $n \in [0, 10^3]$. V naslednjih n vrsticah so zapisani posamezni testni nizi. Vsak niz (vzorčni in testni) vsebuje vsaj en in kvečjemu 100 znakov. Testni nizi so sestavljeni iz malih črk angleške abecede, vzorčni pa lahko poleg tega vsebujejo še vprašaje in zvezdice.

Vzorčni nizi v testnih primerih J1–J3 in S1–S15 vsebujejo po en vprašaj in nobene zvezdice, v primerih J4–J6 in S16–S30 po eno zvezdico in nobenega vprašaja, v primerih J7–J8 in S31–S40 poljubno mnogo zvezdic in nobenega vprašaja, v primerih J9–J11 in S41–S50 pa poljubno mnogo zvezdic in vprašajev.

Izhod

Izpišite število testnih nizov, ki se ujemajo z vzorčnim.

Testni primer J9

Vhod:

```
a*bc**d?
10
abcd
abcdd
abcde
abcdef
abcddf
aaabbbcccddd
axyzbcdp
axbcyde
abcabcabcabcdecdecdecde
abeceda
```

Izhod:

```
7
```

Z vzorčnim nizom `a*bc**d?` se ujemajo vsi testni nizi razen `abcd`, `abcdef` in `abeceda`.

5 Leksikalni analizator

Naloga

Napišite program, ki prebere sintaktično pravilen program v jeziku C in po vrsti izpiše podatke o njegovih leksikalnih elementih. Vhodni program lahko vsebuje sledeče leksikalne elemente:[1]

Niz: zaporedje znakov med dvema dvojnima narekovajema (`"`). Niz se nikoli ne razteza čez več kot eno vrstico. V tej nalogi lahko nizi vsebujejo znake z ASCII-kodami od 32 do vključno 126 z izjemo znakov `"` in `\`. Niz obravnavamo kot celoto; njegove vsebine ne analiziramo.

Število: celo število z intervala $[0, 10^9]$.

Rezerviranka: beseda `char`, `else`, `for`, `if`, `int`, `return` ali `while`.

Ime: zaporedje velikih in malih črk angleške abecede, števk in podčrtajev, ki se *ne* prične s števko in ki ni rezerviranka.

Operator: eden od znakov oz. zaporedij znakov `+`, `-`, `*`, `/`, `=`, `>`, `<`, `+=`, `-=`, `*=`, `/=`, `==`, `>=` in `<=`.

Ločilo: eden od znakov `(`, `)`, `[`, `]`, `{`, `}`, `;` in `,`.

Vhodni program lahko vsebuje tudi vrstične in bločne komentarje. Vrstični komentar se prične z zaporedjem `//` in zaključi s koncem vrstice, bločni pa se prične z zaporedjem `/*` in konča z zaporedjem `*/`. Vaš program naj komentarje enostavno preskoči, prav tako pa naj ignorira (nebistvene) presledke, tabulatorje in znake za skok v naslednjo vrstico.

[1] Omejili se bomo na razmeroma majhno podmnožico jezika C.

Vhod

Na vhodu je zapisan sintaktično pravilen program v jeziku C, ki vsebuje zgolj zgoraj navedene leksikalne elemente. Nobena vrstica ni daljša od 10^4 znakov.

V testnih primerih J1–J7 in S1–S35 vhodni programi ne vsebujejo nobenega komentarja. Testni primeri J1–J3 in S1–S15 ne vsebujejo nobenega niza. V primerih J1–J5 in S1–S25 velja še sledeče:

- vsa imena so sestavljena iz ene same črke;
- nastopajo le operatorji `+`, `-`, `*`, `/`, `=`, `>` in `<`;
- vsa števila so sestavljena iz ene same števke.

Izhod

Po vrsti izpišite podatke o leksikalnih elementih programa. Za vsak element izpišite vrstico oblike *vrsta*`[`*vsebina*`]`, kjer je *vrsta* vrsta leksikalnega elementa (`niz`, `stevilo`, `rezerviranka`, `ime`, `operator` ali `locilo`), *vsebina* pa zaporedje znakov, ki tvori leksikalni element. Začetni in končni dvojni narekovaj ne pripadata vsebini niza.

Testni primer J8

Vhod:

```
int main() {
    int a= 3, b,c, ploscina;  // spremenljivke tipa int
    int b7D_81e=-124;
    while (ploscina >7){
        a*=b*-c/ploscina>=42;
        char* niz = "danes je lep dan";
        /*
         * to je komentar
         * "niz znotraj komentarja sploh ni niz"
         */ int fuj666;
        char*t="//komentar/*znotraj niza*/sploh ni komentar";
    }
    return 0;
}
```

Izhod:

```
rezerviranka[int]
ime[main]
locilo[(]
locilo[)]
locilo[{]
rezerviranka[int]
ime[a]
operator[=]
stevilo[3]
locilo[,]
```

```
ime[b]
locilo[,]
ime[c]
locilo[,]
ime[ploscina]
locilo[;]
rezerviranka[int]
ime[b7D_81e]
operator[=]
operator[-]
stevilo[124]
locilo[;]
rezerviranka[while]
locilo[(]
ime[ploscina]
operator[>]
stevilo[7]
locilo[)]
locilo[{]
ime[a]
operator[*=]
ime[b]
operator[*]
operator[-]
ime[c]
operator[/]
ime[ploscina]
operator[>=]
stevilo[42]
locilo[;]
rezerviranka[char]
operator[*]
ime[niz]
operator[=]
niz[danes je lep dan]
locilo[;]
rezerviranka[int]
ime[fuj666]
locilo[;]
rezerviranka[char]
operator[*]
ime[t]
operator[=]
niz[//komentar/*znotraj niza*/sploh ni komentar]
locilo[;]
locilo[}]
rezerviranka[return]
stevilo[0]
locilo[;]
locilo[}]
```

Napotek

Morda vam bo koristila funkcija `ungetc`. Če jo boste uporabili, jo pokličite takole:

`ungetc(`*znak*`, stdin);`

6 Knjižnica

Naloga

Knjižnica ima C članov in K različnih knjižnih naslovov. Tako člani kot naslovi so označeni z zaporednimi številkami od 1 naprej. Knjižnica premore po P izvodov vsakega knjižnega naslova. Član si lahko vsak izvod izposodi za največ D dni. Če se ob vrnitvi izvoda izkaže, da je rok za njegovo vrnitev prekoračil, mora plačati zamudnino v višini Z krat število dni zamude.

Napišite program, ki v skladu z navodili v nadaljevanju bere ukaze in izpisuje pripadajoče rezultate.

Vhod

V prvi vrstici so navedena cela števila $C \in [1, 100]$, $K \in [1, 100]$, $P \in [1, 100]$, $D \in [0, 10^4]$, $Z \in [0, 10^3]$ in $V \in [1, 10^4]$, ločena s presledkom. Vsaka od naslednjih V vrstic zavzema eno od sledečih oblik:

- `+` *datum* *c* *k*

  Član z zaporedno številko c si na dan *datum* poskuša izposoditi izvod knjižnega naslova z zaporedno številko k. Če ima knjižnica na voljo vsaj en izvod naslova k in če član c nima v izposoji nobenega izvoda tega naslova, potem izposoja uspe, vaš program pa naj izpiše znak `D`. V nasprotnem primeru ukaz nima učinka, program pa naj izpiše znak `N`.

  Izpis naj se zaključi s skokom v naslednjo vrstico. To velja tudi za ostale ukaze.

  Datum je zapisan v obliki `DD.MM.LLLL` (npr. `03.04.2017` za 3. april 2017).

- `-` *datum* *c* *k*

  Če ima član c v izposoji izvod knjižnega naslova k, ga na dan *datum* vrne v knjižnico, program pa naj izpiše zamudnino za ta izvod. V nasprotnem primeru naj program izpiše znak `N`.

- `A` *c*

  Program naj izpiše skupno število izvodov, ki jih ima član c trenutno v izposoji.

- `B` *k*

  Program naj izpiše skupno število izvodov knjižnega naslova k, ki so trenutno v izposoji.

- `C` *c*

Program naj izpiše zaporedne številke vseh knjižnih naslovov, ki jih ima član c trenutno v izposoji, in pripadajoče roke vrnitve. Pari naslov-rok naj bodo izpisani v obliki

/naslov1:*rok1*/*naslov2*:*rok2*/.../

pri čemer so posamezni roki izpisani v obliki `DD.MM.LLLL`. Seznam naj bo urejen po naraščajočih zaporednih številkah naslovov. Če član c nima ničesar v izposoji, naj program izpiše samo znak `/`.

- `D` k

  Program naj izpiše zaporedne številke vseh članov, ki imajo v izposoji izvod naslova k, in pripadajoče roke vrnitve. Pari član-rok naj bodo izpisani v obliki

  /član1:*rok1*/*član2*:*rok2*/.../

  pri čemer so posamezni roki izpisani v obliki `DD.MM.LLLL`. Seznam naj bo urejen po naraščajočih zaporednih številkah članov. Če trenutno ni noben izvod naslova k v izposoji, naj program izpiše samo znak `/`.

V vseh ukazih velja $c \in [1, C]$ in $k \in [1, K]$. Ukazi so urejeni po naraščajočih datumih. Vsi datumi pripadajo intervalu med 1. januarjem 1901 in 31. decembrom 2017. (Znotraj tega intervala so prestopna vsa leta, ki so deljiva s 4.)

Sledeča tabela prikazuje ukaze, ki lahko nastopajo v posameznih testnih primerih, in morebitne dodatne omejitve:

Testni primeri	Možni ukazi	Dodatne omejitve
J1–J2, S1–S10	`+`	$Z = 0$
J3–J4, S11–S20	`+`, `-`	$Z = 0$
J5–J6, S21–S30	`+`, `-`, `A`, `B`	$Z = 0$
J7–J8, S31–S40	`+`, `-`, `A`, `B`, `C`, `D`	$D \leqslant 28$, vsi datumi med 01.01.2017 in 01.12.2017
J9–J10, S41–S50	`+`, `-`, `A`, `B`, `C`, `D`	(brez)

Za oceno do 60% točk se vam z datumi torej ne bo treba ukvarjati.

Izhod

Izhod sestavljajo vsi izpisi, ki jih od programa zahtevajo ukazi.

Testni primer ...

... je zaradi preglednosti prikazan na naslednji strani.

Testni primer J9

Vhod:

```
3 4 2 10 6 27
+ 10.01.2016 3 4
+ 20.01.2016 2 3
- 25.01.2016 3 2
- 27.01.2016 3 4
+ 31.01.2016 1 4
- 10.02.2016 1 4
+ 15.02.2016 1 3
+ 15.02.2016 3 3
+ 18.02.2016 2 3
+ 29.02.2016 1 1
A 1
A 2
A 3
B 1
B 2
B 3
B 4
+ 17.03.2016 2 2
- 01.01.2017 1 1
+ 23.04.2017 3 2
C 1
C 2
C 3
D 1
D 2
D 3
D 4
```

Izhod:

```
D
D
N
42
D
0
D
N
N
D
2
1
0
1
0
2
0
D
1782
D
/3:25.02.2016/
/2:27.03.2016/3:30.01.2016/
/2:03.05.2017/
/
/2:27.03.2016/3:03.05.2017/
/1:25.02.2016/2:30.01.2016/
/
```

7 Drevo

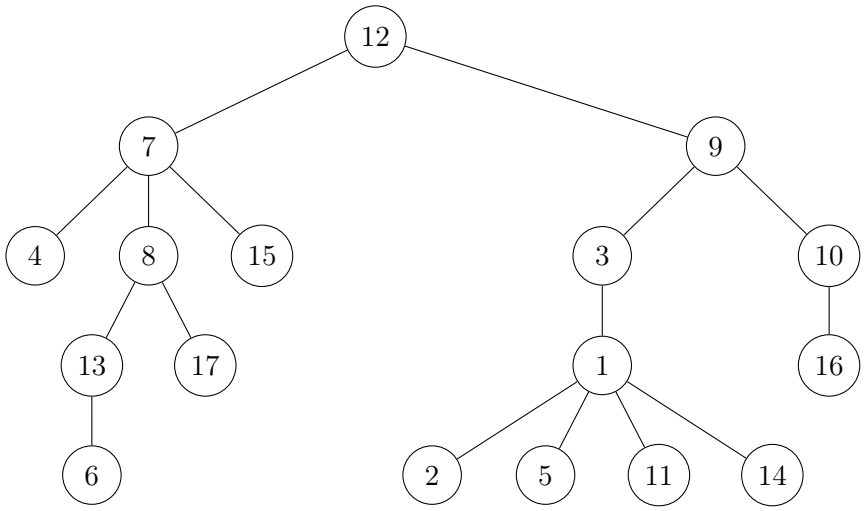
Naloga

Napišite program, ki za podano drevo z n vozlišči, označenimi s številkami od 1 do n, izpolni vse podane ukaze. Primeri v nadaljevanju se nanašajo na drevo na sliki 3.

Vhod

Vsa števila na vhodu so cela, števila v isti vrstici pa so med seboj ločena s presledkom.

V prvi vrstici je zapisano število $n \in [1, 1000]$, v naslednjih n vrsticah pa je podana zgradba drevesa: v i-ti od teh vrstic je najprej zapisano število otrok vozlišča i, nato pa sledijo številke posameznih otrok vozlišča i. Sledi vrstica s številom $k \in [1, 10]$, v naslednjih k vrsticah pa so zapisani posamezni ukazi. Vsak ukaz je sestavljen iz številke ukaza ($u \in \{1, \ldots, 6\}$) in parametra ($m \in [1, n]$). Parameter ukaza je številka enega od vozlišč drevesa.



Slika 3: *Primer drevesa.*

Izhod

Program naj po vrsti izpolni vse podane ukaze. Vsak ukaz naj se izvrši na poddrevesu, katerega koren je podan s parametrom ukaza. Ukazi imajo sledeči pomen:

- **Ukaz 1** [J1–J2, S1–S9]: Izpiše število vozlišč v podanem poddrevesu. V primeru na sliki 3 je poddrevo s korenom 12 (tj. celotno drevo) sestavljeno iz 17 vozlišč, poddrevo s korenom 9 iz devetih, poddrevo s korenom 10 pa iz dveh vozlišč.
- **Ukaz 2** [J3–J4, S10–S17]: Izpiše vozlišče z največjo številko v podanem poddrevesu. V poddrevesu s korenom 12 ima največje vozlišče številko 17, v poddrevesih s korenoma 9 in 10 pa 16.
- **Ukaz 3** [J5–J6, S18–S25]: Izpiše višino podanega poddrevesa, tj. razdaljo od korena do najbolj oddaljenega vozlišča. Poddrevo s korenom 12 ima višino 4, poddrevo s korenom 9 ima višino 3, poddrevo s korenom 10 pa 1.
- **Ukaz 4** [J7–J8, S26–S34]: Izpiše število vozlišč na nivoju h v podanem poddrevesu, kjer je h višina poddrevesa. (Koren se nahaja na nivoju 0, njegovi otroci na nivoju 1 itd.) Poddrevo s korenom 12 ima pet takih vozlišč (2, 5, 6, 11, 14), poddrevo s korenom 9 ima štiri taka vozlišča (2, 5, 11, 14), poddrevo s korenom 10 pa eno (16).
- **Ukaz 5** [J9–J10, S35–S42]: Izpiše poddrevo po zgledu sledečih primerov. Poddrevo s korenom 12 naj se izpiše kot ...

  ```
  12[7[4, 8[13[6], 17], 15], 9[3[1[2, 5, 11, 14]], 10[16]]]
  ```

  ... poddrevo s korenom 9 kot

  ```
  9[3[1[2, 5, 11, 14]], 10[16]]
  ```

  ... poddrevo s korenom 10 pa kot

  ```
  10[16]
  ```

- **Ukaz 6** [J11–J12, S43–S50]: Nariše poddrevo po zgledu sledečih primerov. Poddrevo s korenom 12 naj se nariše kot ...

  ```
  12
  +-- 7
  ```

```
|   +-- 4
|   +-- 8
|   |   +-- 13
|   |   |   +-- 6
|   |   +-- 17
|   +-- 15
+-- 9
    +-- 3
    |   +-- 1
    |       +-- 2
    |       +-- 5
    |       +-- 11
    |       +-- 14
    +-- 10
        +-- 16
```

... poddrevo s korenom 9 kot

```
9
+-- 3
|   +-- 1
|       +-- 2
|       +-- 5
|       +-- 11
|       +-- 14
+-- 10
    +-- 16
```

... poddrevo s korenom 10 pa kot

```
10
+-- 16
```

Testni primer J1

Vhod:

```
17
4 2 5 11 14
0
1 1
0
0
0
3 4 8 15
2 13 17
2 3 10
1 16
0
2 7 9
1 6
0
0
```

```
0
0
3
1 12
1 9
1 10
```

Izhod:

```
17
9
2
```

8 Politična nasprotja

Naloga

Na politično konferenco je povabljenih l levičarjev, d desničarjev in c centristov. Organizatorji jih morajo razmestiti na $l + d + c$ zaporedno postavljenih sedežev, in to tako, da levičar in desničar nikjer ne bosta soseda. Napišite program, ki prebere števila l, d in c in izpiše število vseh sedežnih redov, ki ustrezajo opisanemu pogoju.

Vhod

Na vhodu so podana cela števila l, d in c, ločena s presledkom.

V primerih J1–J4 in S1–S20 velja $l \in [0, 10]$, $d = 0$ in $c \in [0, 5]$.

V primerih J5–J7 in S21–S35 velja $l \in [0, 10]$, $d \in [0, 10]$ in $c \in [0, 5]$.

V primerih J8–J10 in S36–S50 velja $l \in [0, 20]$, $d \in [0, 20]$ in $c \in [0, 20]$.

Izhod

Izpišite samo število možnih razporeditev. To število je v vseh testnih primerih manjše od 2^{63}.

Testni primer J5

Vhod:

```
2 3 2
```

Izhod:

```
15
```

V tem primeru so možne sledeče razporeditve:

```
LLCDDDC
LLCDDCD
LLCDCDD
LLCCDDD
LCLCDDD
LCDDDCL
DDDCLLC
DDDCLCL
DDDCCLL
DDCLLCD
DDCDCLL
DCLLCDD
DCDDCLL
CLLCDDD
CDDDCLL
```

Testni primer J6

Vhod:

```
2 3 1
```

Izhod:

```
2
```

V tem primeru sta možni le razporeditvi `LLCDDD` in `DDDCLL`.

Če želite še več ...

Če je vaš program dovolj hiter za vseh deset testnih primerov, poskusite napisati različico, ki bo v eni sekundi zmlela primer $l = d = c = 100$. (Rezultat bo sicer močno presegal omejitev tipa `long`, vendar pa se glede tega ne vznemirjajte; pretvarjajte se, kot da tip `long` ni omejen.)

Morda boste morali (memoizirano) rekurzivno rešitev predelati v iterativno. Kako se računajo vrednosti v memoizacijski tabeli?

9 Tovarna

Naloga

V tovarni dela m delavcev in n strojev. Vsak delavec lahko upravlja kvečjemu en stroj in vsak stroj lahko upravlja kvečjemu en delavec. Delavci se med seboj lahko razlikujejo po usposobljenosti za delo s posameznimi stroji; morda je delavec i na stroju j učinkovitejši od delavca i', ta pa je od njega učinkovitejši na stroju j'. Za vsak par delavca i in stroja j tako poznamo *produktivnost* p_{ij}, ki pove, koliko izdelkov na dan lahko delavec i opravi na stroju j.

Napišite program, ki prebere števila m, n in p_{ij} (za vsak par $i \in \{1, \ldots, m\}$ in $j \in \{1, \ldots, n\}$) in izpiše največjo možno skupno produktivnost, torej maksimalno število izdelkov, ki jih je mogoče proizvesti na dan.

Vhod

Vsa števila na vhodu so cela, števila v isti vrstici pa so med seboj ločena s presledkom.

V prvi vrstici sta podani števili m in n, v naslednjih m vrsticah pa števila $p_{ij} \in [0, 1000]$; i-ta od teh vrstic (za vsak $i \in \{1, \ldots, m\}$) po vrsti vsebuje števila p_{i1}, p_{i2}, $\ldots$, p_{in}.

V testnih primerih J1–J4 in S1–S20 velja $m \in [1, 6]$ in $n \in [1, 6]$.

V testnih primerih J5–J7 in S21–S35 velja $m \in [1, 9]$ in $n \in [1, 9]$.

V testnih primerih J8–J10 in S36–S50 velja $m \in [1, 18]$ in $n \in [1, 18]$.

V testnih primerih J1–J3, J5 in J8 ter S1–S15, S21–S25 in S36–S40 velja $m = n$.

Izhod

Izpišite največjo možno skupno produktivnost.

Testni primer J4

Vhod:

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

Izhod:

```
11
```

Optimalno produktivnost dosežemo tako, da delavcu 0 dodelimo stroj 3, delavcu 1 stroj 0, delavcu 2 pa stroj 1.

10 Križarjenje

Naloga

Vodstvo pomorske družbe je kapitanu in njegovi ladji določilo načrt križarjenja, ki ga sestavlja zaporedje posameznih plovb med pari pristanišč, in mu ga skupaj s potnimi nalogi za posamezne plovbe tudi predala. Žal je kapitan načrt križarjenja izgubil, ostali pa so mu potni nalogi v naključnem vrstnem redu. Sedaj vas ponižno prosi, da rekonstruirate načrt križarjenja.

Napišite program, ki prebere podatke o posameznih plovbah in izpiše, kje naj bi se v skladu z načrtom križarjenja ladja nahajala v posameznih trenutkih in koliko potnikov naj bi bilo tedaj predvidoma na ladji.

Vhod

Na vhodu so najprej zapisani podatki o posameznih plovbah, in to ne nujno v zaporedju, kot si sledijo v križarjenju. Vsaka plovba je opisana v svoji vrstici v obliki

začetek konec trajanje vkrcanja izkrcanja

pri čemer *začetek* in *konec* (niza dolžine od 1 do 100, sestavljena iz črk angleške abecede in podčrtajev) predstavljata začetno in končno pristanišče plovbe, *trajanje* (celo število z intervala $[1, 10^6]$) podaja trajanje plovbe v dnevih, *vkrcanja* in *izkrcanja* (celi števili z intervala $[0, 10^6]$) pa število potnikov, ki se vkrcajo na začetku prvega oziroma izkrcajo ob koncu zadnjega dne plovbe. Število plovb znaša med 1 in 1000.

Podatkom o posameznih plovbah sledi vrstica, ki vsebuje zgolj znak `?`, nato pa sledi n (najmanj 1 in največ 100) vrstic, od katerih vsaka vsebuje celo število z intervala $[1, 10^9]$. Število v i-ti vrstici (označimo ga z d_i) predstavlja zaporedno številko dneva, za katerega nas zanima položaj ladje in število potnikov na njej. Križarjenje se prične na dan z zaporedno številko 1.

Vsako pristanišče (razen izvornega in ciljnega pristanišča križarjenja) se natanko enkrat pojavi kot začetno in natanko enkrat kot končno pristanišče neke plovbe. Izvorno in ciljno pristanišče križarjenja pa se pojavita samo po enkrat (izvorno kot začetno, ciljno pa kot končno pristanišče neke plovbe).

V testnih primerih J1–J4 in S1–S20 si plovbe na vhodu sledijo v pravilnem vrstnem redu ($A \to B$, $B \to C$, $C \to D$, ...).

Izhod

Izpišite n vrstic, pri čemer naj ima i-ta vrstica sledečo obliko:

$začetek_i$ $konec_i$ $štPotnikov_i$

V gornjem zapisu predstavljata $začetek_i$ in $konec_i$ začetno in končno pristanišče plovbe, ki naj bi po načrtu potekala v dnevu z zaporedno številko d_i, $štPotnikov_i$ pa število potnikov, ki naj bi bili tedaj na ladji.

Če je v dnevu d_i križarjenje že končano, naj i-ta vrstica vsebuje samo niz `NAPAKA`. Ta scenarij je možen samo v testnih primerih J8–J10 in S36–S50.

Testni primer J8

Vhod:

```
Barcelona New_York 27 81 195
Koper Split 1 108 54
Split Barcelona 8 120 60
?
1
9
36
37
25
2
```

Izhod:

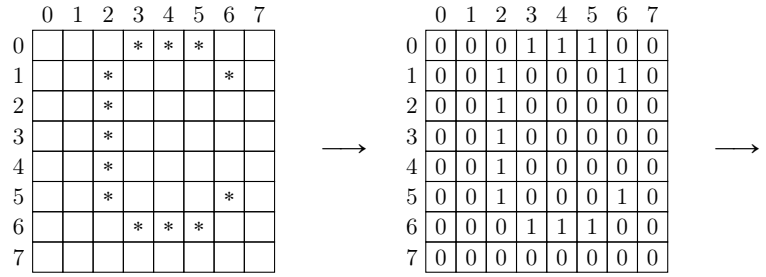
```
Koper Split 108
Split Barcelona 174
Barcelona New_York 195
NAPAKA
Barcelona New_York 195
Split Barcelona 174
```

V tem primeru ladja potuje po trasi Koper – Split – Barcelona – New York. Na relaciji Koper – Split potuje v dnevu 1, na relaciji Split – Barcelona potuje od vključno dneva 2 do vključno dneva 9, na relaciji Barcelona – New York pa od vključno dneva 10 do vključno dneva 36.

11 Dvojiške slike

Naloga

Dvojiško sliko velikosti 8×8 lahko predstavimo s 64-bitnim nepredznačenim številom. Elemente slike prepišemo po vrsticah, nato pa dobljeno 64-bitno dvojiško število pretvorimo v desetiški sistem. Na primer:



$$00011100\ 00100010\ 00100000\ 00100000\ 00100000\ 00100010\ 00011100\ 00000000_{(2)}$$
$$= 2\,027\,218\,104\,620\,293\,120_{(10)}$$

Napišite program, ki prebere ukaz u, števili m in n ter imeni vhodne in izhodne datoteke, nato pa izvrši podani ukaz. V primeru $u = 1$ naj program pretvori sliko velikosti $8m \times 8n$ (višina krat širina) v matriko $m \times n$ 64-bitnih nepredznačenih števil. Podslika velikosti 8×8, ki se v vhodni sliki nahaja na i-tem mestu po višini in j-tem mestu po širini (za $i \in \{1, \ldots, m\}$ in $j \in \{1, \ldots, n\}$), naj se pretvori v število v i-ti vrstici in j-tem stolpcu izhodne matrike. V primeru $u = 2$ pa naj vaš program izvrši obratno pretvorbo.

Program naj dvojiško sliko (v primeru $u = 1$) oziroma matriko števil (v primeru $u = 2$) prebere iz podane vhodne datoteke, rezultat pretvorbe pa naj izpiše v podano izhodno datoteko.

Vhod

V prvi vrstici vhoda je zapisano število $u \in \{1, 2\}$, v drugi pa celi števili $m \in [1, 10]$ in $n \in [1, 10]$, ločeni s presledkom. V tretji vrstici je zapisano ime vhodne datoteke, v četrti pa ime izhodne datoteke. Imeni sta niza največ 20 znakov, sestavljena iz črk angleške abecede, števk in pik.

Vhodne datoteke, ki pripadajo primerom z ukazom 1, vsebujejo $8m$ vrstic po $8n$ znakov, pri čemer je vsak znak bodisi zvezdica (*), ki predstavlja enico, bodisi presledek, ki predstavlja ničlo. Vhodne datoteke, ki pripadajo primerom z ukazom 2, pa vsebujejo m vrstic po n celih števil z intervala $[0, 2^{64} - 1]$, ločenih s presledkom.

V testnih primerih J1–J5 in S1–S25 velja $u = 1$, v primerih J6–J10 in S26–S50 pa $u = 2$. V primerih J1–J2, S1–S10, J6–J7 in S26–S35 velja $m = n = 1$.

Izhod

Program naj na standardni izhod ne izpiše ničesar. V primeru $u = 1$ naj v podano izhodno datoteko izpiše m vrstic s po n celimi števili, ločenimi s presledkom, ki tvorijo izhodno številsko matriko. V primeru $u = 2$ pa naj v podano izhodno datoteko izpiše $8m$ vrstic s po $8n$ znaki, ki tvorijo izhodno dvojiško sliko (zvezdica predstavlja enico, presledek pa ničlo).

Testni primer J3

Vhod:

```
1
2 3
vhod03.txt
rezultat03.txt
```

V tem primeru je vhodna datoteka videti takole . . .[2]

```
-- ***    ****     *** -
--*   *   *   *   *   *-
--*   *   *   *   *    -
--*   *   ****    *    -
--*****   *   *   *    -
--*   *   *   *   *   *-
--*   *   ****     *** -
------------------------
--  *      ***     *** -
-- **     *   *   *   *-
--  *         *       *-
--  *       **     *** -
--  *      *          *-
--  *     *       *   *-
-- ***    *****    *** -
------------------------
```

. . . izhodna pa takole:

```
2027220312736801280 4333063433149889536 2027218104620293120
583224982331988992 2027185033103556096 2027185101588274176
```

[2]Zavoljo večje preglednosti so nekateri presledki prikazani z znaki -.

Testni primer J8

Vhod:

```
2
2 3
vhod08.txt
rezultat08.txt
```

V tem primeru sta vhodna in izhodna datoteka videti takole:

```
2027220312736801280 4333063433149889536 2027218104620293120
583224982331988992 2027185033103556096 2027185101588274176
```

```
  ***    ****     ***
 *   *   *   *   *   *
 *   *   *   *   *
 *   *   ****    *
 *****   *   *   *
 *   *   *   *   *   *
 *   *   ****     ***

   *      ***     ***
  **     *   *   *   *
   *         *       *
   *       **     ***
   *      *          *
   *     *       *   *
  ***    *****    ***
```

12 Huffmanovo kodiranje

Naloga

Huffmanovo kodiranje je tehnika stiskanja datotek, pri kateri pogostejše znake zapišemo z manj, redkejše pa z več biti. Na ta način lahko prihranimo precej pomnilnega prostora, četudi tehnike, ki temeljijo na kodiranju zaporedij znakov in so del standardnih orodij za stiskanje (npr. programa `zip`), v splošnem dosegajo boljše rezultate.

Napišite program, ki prebere ukaz (`1`: kodiraj; `2`: dekodiraj) ter ime vhodne in izhodne datoteke, nato pa ustvari izhodno datoteko in vanjo zapiše (de)kodirano vsebino vhodne datoteke. Ravnajte se po navodilih v nadaljevanju.

Tekoči primer

Predpostavili bomo, da kodiramo datoteko s sledečo vsebino:

```
HOKUSPOKUS
```

V nasprotju z običajno prakso pri besedilnih datotekah se naša datoteka *ne* zaključi z znakom za skok v novo vrstico.

Množica znakov

Za potrebe tega besedila bomo množice znakov zapisovali kot zaporedja znakov, urejena po abecedi. Na primer, množico {`P`, `H`, `K`} bomo zapisali kot `HKP`. Rekli bomo, da je množica A *lepša* od množice B, če prvi znak zapisa množice A po abecedi sodi pred prvi znak zapisa množice B.[3] Na primer, množica `HKP` je lepša od množice `IJ`, ker znak `H` sodi pred znak `I`. Primera, ko sta prva znaka enaka, nam ni treba obravnavati, ker se pri Huffmanovem kodiranju nikoli ne pojavi.

Gradnja Huffmanovega drevesa

Huffmanovo kodiranje temelji na *Huffmanovem drevesu*, ki ga zgradimo takole:

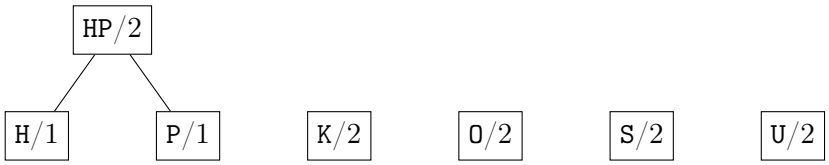
1. Za vsak znak določimo število njegovih pojavitev v vhodni datoteki. Upoštevamo samo znake, ki v datoteki dejansko nastopajo; ostale ignoriramo.
2. Za vsak znak, ki nastopa v vhodni datoteki, izdelamo samostojno drevo, ki vsebuje eno samo vozlišče (to vozlišče je potemtakem tudi koren drevesa). Drevesu pripišemo dva podatka: (1) množico pripadajočih znakov in (2) težo. Množica pripadajočih znakov za začetno drevo vsebuje samo znak, na podlagi katerega smo drevo izdelali, teža drevesa pa je enaka številu pojavitev tega znaka v vhodni datoteki.
3. Dokler ne ostane eno samo drevo, ponavljamo sledeče:
   - Poiščemo drevo z najmanjšo težo. Če je takih dreves več, izberemo tisto, ki ima najlepšo množico znakov.
   - Še enkrat poiščemo drevo z najmanjšo težo, pri čemer seveda ne upoštevamo drevesa, izbranega v prejšnjem koraku. Če je takih dreves več, tudi tokrat izberemo tisto, ki ima najlepšo množico znakov.
   - Tisto izmed dveh izbranih dreves, ki ima lepšo množico znakov, proglasimo za drevo L, tisto z gršo množico znakov pa za drevo R.
   - Izdelamo drevo, sestavljeno iz korena, drevesa L in drevesa R. Drevo L je levo, drevo R pa desno poddrevo korena. Množica znakov za novo drevo je unija množic znakov dreves L in R, težo pa izračunamo kot vsoto tež dreves L in R. Dreves L in R v nadaljevanju ne obravnavamo več.

V našem primeru se znaka `H` in `P` pojavita po enkrat, znaki `K`, `O`, `S` in `U` pa po dvakrat. Začetni nabor dreves potemtakem izgleda takole:[4]

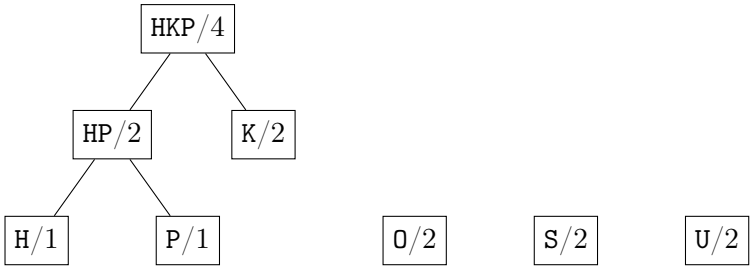
V prvem koraku drevo z množico `H` postane drevo L, drevo z množico `P` pa drevo R. Na podlagi teh dreves izdelamo novo, originalni drevesi pa izločimo iz nadaljnje obravnave. Dobimo sledeči nabor dreves:

[3] Nimamo nikakršnih predsodkov do znakov ob koncu abecede, le nek pridevnik si je bilo treba izmisliti ...

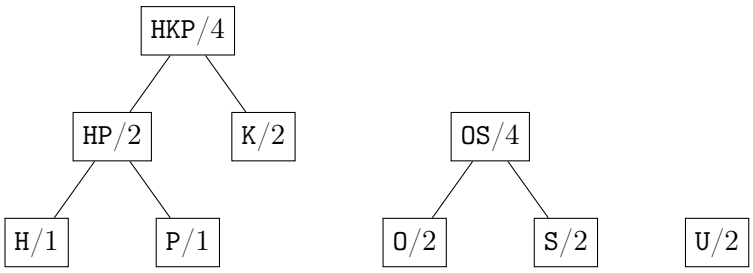
[4] Drevesa so v prikazu urejena po padajoči lepoti njihovih množic znakov.



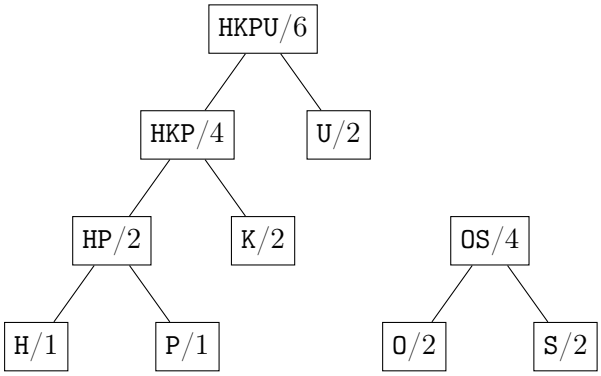
V naslednjem koraku združimo drevesi $L = \texttt{HP}$ in $R = \texttt{K}$:



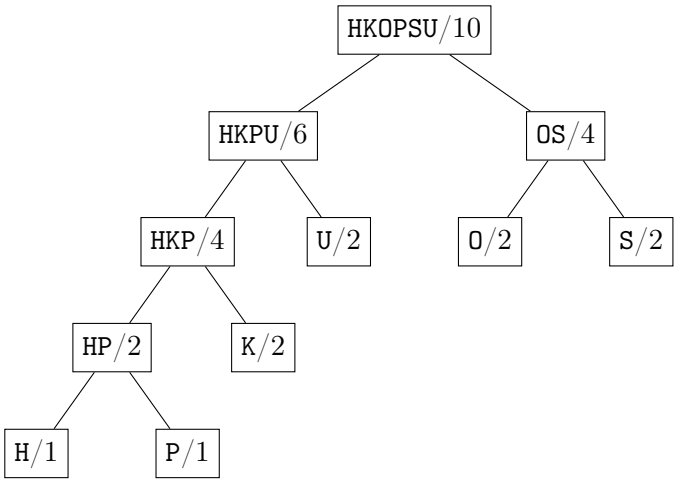
Združimo še drevesi $L = \texttt{O}$ in $R = \texttt{S}$:



Drevo z najmanjšo težo je sedaj `U`, drevo z drugo najmanjšo težo pa `HKP` (to je lepše od alternative `OS`). Ker je množica `HKP` lepša od `U`, proglasimo drevo `HKP` za L, drevo `U` pa za R. Po združitvi dobimo sledeče:



Nazadnje združimo še $L = \texttt{HKPU}$ in $R = \texttt{OS}$:



Ko Huffmanovo drevo zgradimo do konca, ohranimo le množice znakov za liste (to so dejansko kar posamezni znaki), na teže vozlišč in množice znakov za notranja vozlišča pa lahko pozabimo.

Zapis in rekonstrukcija Huffmanovega drevesa

Huffmanovo drevo potrebujemo tako za kodiranje kot za dekodiranje, zato ga moramo pri kodiranju zapisati v izhodno datoteko, pri dekodiranju pa prebrati iz vhodne. Huffmanovo drevo zapišemo takole:

- Koren drevesa proglasimo za trenutno vozlišče.
- Če je trenutno vozlišče list (vozlišče brez poddreves), zapišemo bit 1, potem pa še 8-bitno ASCII-kodo znaka v tem listu.[5]
- Če trenutno vozlišče ni list, zapišemo bit 0, potem pa rekurzivno zapišemo še obe poddrevesi trenutnega vozlišča.

Naše drevo bi torej zapisali takole:

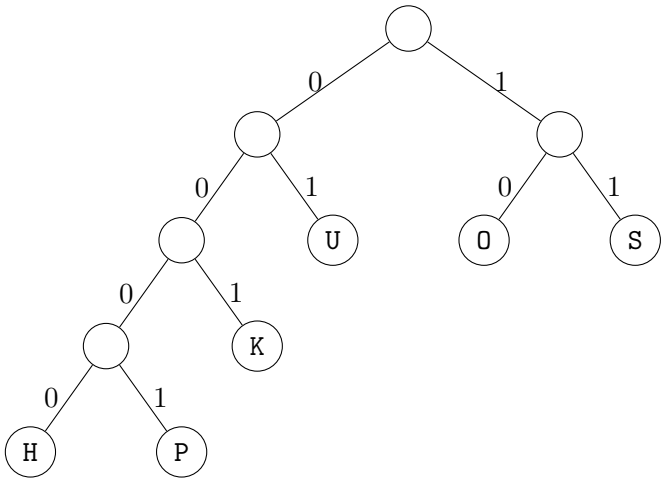
$$00001\underbrace{01001000}_{\texttt{H}}1\underbrace{01010000}_{\texttt{P}}1\underbrace{01001011}_{\texttt{K}}1\underbrace{01010101}_{\texttt{U}}01\underbrace{01001111}_{\texttt{O}}1\underbrace{01010011}_{\texttt{S}}$$

Kako pa drevo preberemo in rekonstruiramo? Pričnemo s prvim bitom zapisa. Če je bit enak 1, vemo, da gre za list, zato preberemo še naslednjih 8 bitov, ki določajo ASCII-kodo znaka v tem listu. Če je bit enak 0, pa vemo, da gre za notranje vozlišče, zato rekurzivno preberemo levo poddrevo in nato še desno.

Kodiranje in dekodiranje

Ko zgradimo Huffmanovo drevo, vse njegove leve veje (povezave od posameznih vozlišč do njihovih levih poddreves) označimo z 0, vse desne veje pa z 1:

[5] `printf("%d", znak);`



Kodo posameznega znaka sedaj pridobimo tako, da po vrsti preberemo vse oznake povezav od korena do tega znaka. V našem primeru ima znak `H` kodo `0000`, znak `P` kodo `0001`, znak `K` kodo `001`, znak `U` kodo `01`, znak `O` kodo `10`, znak `S` pa kodo `11`. Kodo celotnega niza sestavimo iz kod posameznih znakov v nizu. Niz `HOKUSPOKUS` ima torej takšno kodo:

$$\underbrace{0000}_{\texttt{H}}\ \underbrace{10}_{\texttt{O}}\ \underbrace{001}_{\texttt{K}}\ \underbrace{01}_{\texttt{U}}\ \underbrace{11}_{\texttt{S}}\ \underbrace{0001}_{\texttt{P}}\ \underbrace{10}_{\texttt{O}}\ \underbrace{001}_{\texttt{K}}\ \underbrace{01}_{\texttt{U}}\ \underbrace{11}_{\texttt{S}}$$

Če želimo podano kodo dekodirati (rekonstruirati izvorni niz), se postavimo v koren Huffmanovega drevesa in ga proglasimo za »trenutno vozlišče«, nato pa pričnemo z branjem bitov, ki sestavljajo kodo niza. Ko preberemo bit 0, se pomaknemo v levega otroka trenutnega vozlišča, ko preberemo bit 1, pa v desnega. Ko prispemo do lista, zapišemo pripadajoči znak, nato pa se zopet prestavimo v koren Huffmanovega drevesa in postopek ponovimo.

Format zakodirane datoteke

Pri kodiranju bomo v izhodno datoteko po vrsti zapisali sledeče:

- zapis Huffmanovega drevesa;
- število bitov, ki sestavljajo kodo vhodnega niza (32-bitno število, zapisano od najbolj levega do najbolj desnega bita);
- kodo vhodnega niza;
- toliko bitov 0, da bo skupno število bitov v izhodni datoteki deljivo z 8.

Zaključne ničle potrebujemo v primeru, ko skupno število bitov ni deljivo z 8. Datoteka lahko namreč vsebuje le celo število bajtov, saj je bajt najmanjša zapisljiva enota tako v pomnilniku kot na trdem disku. Iz istega razloga hranimo tudi dolžino kodiranega niza, saj sicer ne vemo, kje se prične zaporedje zaključnih ničel.

V našem primeru je kodirani niz sestavljen iz 26 bitov, število 26 pa v 32-bitnem zapisu izgleda takole:

```
00000000 00000000 00000000 00011010
```

Datoteka z nizom `HOKUSPOKUS` se torej pretvori v izhodno datoteko s sledečo bitno vsebino:

```
00001010010001010100001010010111010101010100111110101001100000000000000000
00000000000110100000100010111000110001011100 0
```

Z rdečo je označen zapis Huffmanovega drevesa, z zeleno dolžina kodiranega niza, z modro kodirani niz, z vijolično pa ničle, ki jih je treba dodati, da bo skupno število bitov v izhodni datoteki deljivo z 8.

Delo z biti

Pri tej nalogi boste na veliko delali s posameznimi biti. Verjetno vam bodo prišli prav sledeči bitni operatorji (v C-ju predpona `0b` označuje, da gre za dvojiško število):

Operator	Pomen	Primer
`&`	bitni *in*	`(0b01001011 & 0b11010010) == 0b01000010`
`|`	bitni *ali*	`(0b01001011 | 0b11010010) == 0b11011011`
`<<`	pomik v levo za n mest	`(0b101 << 4) == 0b1010000`
`>>`	pomik v desno za n mest	`(0b1011011 >> 4) == 0b101`

Pri pomiku v levo se na desno stran doda n ničel, pri pomiku v desno pa se odstrani zadnjih n bitov. Vrednost posameznega bita v bajtu lahko potemtakem pridobite s kombinacijo desnega pomika in operatorja `&`. Če želite določeni bit v bajtu nastaviti na 1, pa uporabite kombinacijo levega pomika in operatorja `|`.

Upoštevajte, da relacijski operatorji (`<`, `==` itd.) vežejo močneje od bitnih, zato je izraz `a & b == c` enakovreden izrazu `a & (b == c)`, ne pa, kot bi morda pričakovali, izrazu `(a & b) == c`.

Vhod

Standardni vhod je sestavljen iz treh vrstic. Prva vsebuje ukaz $u \in \{1, 2\}$, druga ime vhodne datoteke, tretja pa ime izhodne datoteke. Ime datoteke je sestavljeno iz največ 50 znakov, vsebuje pa lahko kvečjemu črke angleške abecede, števke, podčrtaje in pike.

Če je ukaz enak 1, naj vaš program podano vhodno datoteko kodira s Huffmanovim postopkom in rezultat zapiše v podano izhodno datoteko. (Če bi nastalo izhodno datoteko dekodirali, bi morali dobiti kopijo vhodne datoteke.)

Če je ukaz enak 2, naj vaš program podano vhodno datoteko dekodira in rezultat zapiše v podano izhodno datoteko. (Če bi nastalo izhodno datoteko kodirali, bi morali dobiti kopijo vhodne datoteke.)

V testnih primerih J1–J6 in S1–S25 velja $u = 1$, v primerih J7–J12 in S26–S50 pa $u = 2$.

Nobena vhodna ali izhodna datoteka ni večja od $2 \cdot 10^6$ bajtov. Vsaka nekodirana datoteka vsebuje najmanj dva različna znaka. To pomeni, da bo postopek gradnje Huffmanovega drevesa imel vsaj en korak, Huffmanovo drevo pa vsaj eno notranje vozlišče.

Izhod

Na standardni izhod ne izpišite ničesar.

Testni primer J1

Vhod:

```
1
vhod01.txt
rezultat01.txt.bin
```

Vsebina datoteke `vhod01.txt` (tekstovno):

```
HOKUSPOKUS
```

Vsebina datoteke `vhod01.txt` (dvojiško):

```
01001000 01001111 01001011 01010101 01010011 01010000 01001111 01001011
01010101 01010011
```

Vsebina datoteke `rezultat01.txt.bin` (dvojiško):

```
00001010 01000101 01000010 10010111 01010101 01010011 11101010 01100000
00000000 00000000 00000011 01000001 00010111 00011000 10111000
```