Poslednja večerja

Leonardo je bil med slikanjem Poslednje večerje, svoje najslavnejše stenske slikarije, izjemno aktiven: sivi vsakdan je barvito začel z izbiranjem temper, ki jih je nameraval tisti dan uporabljati. Umetnik je potreboval veliko barv, fizični svet pa mu je na žalost dal le končno velik gradbeni oder. Leonardov pomočnik je bil med drugim zadolžen za pomembno opravilo plezanja na oder, da bi slikarju prinesel potrebne barve, in sestopanja, da bi nepotrebne barve vrnil na pravo polico.

Pri tej nalogi boš pomočniku v pomoč napisal dva ločena programa. Prvi program bo sprejel Leonardova navodila (zaporedje barv, ki bodo ta dan potrebne) in ustvaril *kratek* niz bitov. Pomočnik tekom dneva, med obdelavo umetnikovih zahtevkov, ne bo imel dostopa do njegovih prihodnjih zahtev, ampak le do niza, ki ga bo naredil tvoj prvi program. Drugi program bo ta niz sprejel, potem pa sprejemal Leonardove zahtevke ter jih sproti obdeloval (t.j. enega za drugim). Ta program bo moral razumeti pomen niza in ga uporabiti pri sprejemanju optimalnih odločitev. Vse je bolj natančno razloženo v nadaljevanju.

Premikanje barv med odrom in polico

Omejili se bomo na malce lažji scenarij. Imamo N barv, označenih s številkami od 0 do N - 1, Leonardo pa vsak dan od pomočnika natanko N-krat zahteva novo barvo. Naj bo C zaporedje teh N zahtevkov; lahko si ga zamislimo kot zaporedje N števil, vsako med 0 in N - 1, vključujoče. Pazi na to, da v tem zaporedju ne nastopajo nujno vse barve, in da se kakšna barva lahko pojavi več kot enkrat.

Oder je seveda ves čas poln, na njem je K izmed N možnih barv, kjer $K < N$. Na začetku so na odru barve od 0 do K - 1, vključujoče.

Pomočnik umetnikove zahtevke obdeluje le po enega naenkrat. Vsakokrat, ko je zahtevana barva *že na odru*, lahko pomočnik počiva. Sicer mora zahtevano barvo vzeti s police in jo odnesti na oder. Ker je le-ta seveda poln, mora zaradi nove barve najprej izbrati eno izmed barv, ki so že na odru, in jo vrniti na polico.

Leonardova optimalna strategija

Pomočnik je, kot vsak pristen delavec, seveda izredno len, in si želi čim več počitka. Število počitkov je odvisno od njegovih odločitev tekom delovnega dne. Natančneje, vsakič ko mora odstraniti barvo z odra, lahko različne izbire vodijo do različnih izidov v prihodnosti. Leonardo mu razloži, kako se lahko s poznavanjem zaporedja C čim bolj približa svojemu cilju: ko zahtevane barve ni na odru, je najbolje ponovitev tega neljubega dogodka odmakniti čim dlje v prihodnost. To lahko doseže tako, da preveri, katere barve so trenutno na odru ter katere so v preostanku zaporedja C. Katero barvo naj odstrani z odra, se potem odloči takole:

- Če je na odru barva, ki je slikar ne bo več potreboval, naj odstrani to.
- Sicer naj odstrani tisto, *ki bo zahtevana najpozneje*. (T.j. za vsako izmed barv na odru najdemo prvi prihodnji zahtevek zanjo. Z odra je potem potrebno umakniti tisto, ki bo potrebovana najkasneje.)

Dokazati je mogoče, da Leonardova strategija vodi v največjo količino pomočnikovega počivanja.

1. primer

Naj bo N = 4, torej imamo 4 barve (oštevilčene z 0 do 3) in 4 zahtevke. Naj bo zaporedje zahtevkov C = (2, 0, 3, 0). Predpostavi, da je K = 2, torej ima Leonardo oder, ki je dovolj prostoren le za ušivi dve barvi. Kot povedano zgoraj, sta na začetku na odru barvi 0 in 1. Stanje na odru bomo označili takole: [0, 1]. En izmed mogočih načinov, kako bi pomočnik obdeloval zahtevke, je sledeči.

- Prva zahtevana barva (številka 2) ni na odru. Pomočnik jo nanj postavi in se odloči, da bo z odra odstranil barvo 1. Trenutni oder je potem [0, 2].
- Naslednja zahtevana barva (številka 0) je že na odru, zato pomočnik počiva.
- Pri tretjem zahtevku (številka 3) se pomočnik odloči odstraniti barvo 0 in trenutni oder je [3, 2].
- Zadnjo zahtevano barvo (številko 0) je potrebno vzeti s police in jo odnesti na oder. Pomočnik se odloči odstraniti barvo 2, oder pa postane [3, 0].

Pozoren bodi na to, da se v zgornjem primeru pomočnik ni držal Leonardove optimalne strategije. Ta bi bila, če bi pri tretjem koraku odstranil barvo 2 in bi tako pri zadnjem koraku lahko spet počival.

Pomočnikova strategija, če ima omejen spomin

Pomočnik vsako jutro prosi umetnika, naj mu na papir napiše zaporedje C, da bo lahko naštudiral optimalno strategijo in ji sledil. Ker pa je Leonardo popolnoma obseden s tem, da bi njegove skrivne tehnike ostale skrivnost, pomočniku ne pusti, da bi papir obdržal. Nejevoljno mu dovoli le, da si zaporedje C prebere in skuša zapomniti.

Na žalost pa ima pomočnik nedopustno slab spomin in si lahko zapomni le M bitov. V splošnem mu to lahko prepreči obnovo celotnega zaporedja C, zato pomočnik potrebuje brihten način računanja takega zaporedja bitov, kakršnega si bo dejansko zapomnil. Temu zaporedju bomo rekli *pomočniško zaporedje* in ga označili z A.

2. primer

Zjutraj lahko pomočnik vzame Leonardov papir z zaporedjem C, ga prebere in sprejme vse potrebne odločitve. Ena od stvari, ki jih lahko naredi je, da si ogleda stanje odra po vsakem zahtevku. Npr. če uporablja (ne optimalno) strategijo, opisano v 1. primeru, bi stanja bila [0, 2], [0, 2], [3, 2], [3, 0]. (Spomni se, da začetno stanje [0, 1] pozna že vnaprej.)

Sedaj predpostavi, da je M = 16, torej si pomočnik lahko zapomni kvečjemu 16 bitov informacij.

Ker je N = 4, lahko vsako barvo zapišemo z dvema bitoma, torej je 16 bitov dovolj za hrambo celotnega zaporedja stanj odra. Tako pomočnik pride do naslednjega pomočniškega zaporedja: A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

Pozneje tisti dan lahko pomočnik svoje zaporedje dešifrira in si z njim pomaga pri odločanju.

(Seveda si lahko pri M = 16 pomočnik namesto zaporedja stanj zapomni kar celo zaporedje C, s samo 8. od 16 bitov, ki so na voljo. V tem primeru smo samo poskušali ilustrirati, da obstajajo tudi druge možnosti, ne da bi pri tem izdali kako dobro rešitev.)

Naloga

Napisati moraš *dva ločena programa* v istem programskem jeziku. Tadva programa bosta izvedena zaporedoma in med izvajanjem ne bosta mogla komunicirati en z drugim.

Prvi program je tisti, ki ga bo pomočnik uporabljal zjutraj. Sprejme zaporedje C, izračunati pa mora pomočniško zaporedje A.

Drugi program je tisti, ki ga bo pomočnik uporabljal čez dan. Ta sprejme pomočniško zaporedje A, zatem pa mora obdelati Leonardovo zaporedje zahtevkov. Pazi na to, da bo to zaporedje programu na voljo le po en zahtevek naenkrat, vsak zahtevek pa je potrebno obdelati preden dobi naslednjega.

Natančneje, v prvem programu moraš implementirati eno samo funkcijo, `ComputeAdvice`(C, N, K, M), ki dobi za vhodne podatke zaporedje C, sestavljeno iz N celih števil (vsako izmed 0, …, N - 1), število barv na odru K in število bitov, ki si so na voljo pomočniku, M. Ta program mora izračunati pomočniško zaporedje A, sestavljeno iz kvečjemu M bitov. To zaporedje A potem sporoči okolju tako, da po vrsti za vsak bit pokliče sledečo funkcijo:

- `WriteAdvice`(B) — dodaj bit B na konec trenutnega pomočniškega zaporedja A. (To funkcijo lahko pokličeš največ M-krat.)

V drugem programu moraš implementirati eno samo funkcijo, `Assist`(A, N, K, R). Vhodni podatki so pomočniško zaporedje A, celi števili N in K kot povedano zgoraj in R, dejanska dolžina zaporedja A v bitih (R ≤ M). Ta funkcija pomočniku izvede tvojo predlagano strategijo tako, da uporablja dani funkciji:

- `GetRequest`() — vrne naslednjo barvo, ki jo zahteva Leonardo. (O prihodnjih zahtevkih ne razkrije nobenih informacij.)

- `PutBack`(T) — z odra odstrani barvo T in jo vrne na polico. To funkcijo lahko pokličeš le s tako barvo T, ki je trenutno na odru.

Med izvajanjem mora tvoja funkcija `Assist` poklicati funkcijo `GetRequest` natanko N-krat; s tem zaporedoma dobi Leonardove zahtevke, ki naj jih sproti obdeluje. Če barve, ki jo funkcija `GetRequest` vrne, *ni* na odru, *moraš* poklicati tudi `PutBack`(T), s svojo izbiro T. Sicer `PutBack` *ne smeš* klicati. Če se tvoj program tega ne bo držal, se to šteje kot napaka in program bo neslavno ustavljen. Spomni se, da so na odru na začetku barve od 0 do K - 1, vključujoče.

Posamezen testni primer bo sprejet kot rešen, če se bosta tvoji funkciji držali vseh danih omejitev in bo število klicev funkcije `PutBack` *natanko enako* številu klicev, ki bi ga dobili, če bi uporabljali Leonardovo optimalno strategijo. Če je strategij s tem številom klicev `PutBack` več, lahko tvoj program uporabi katerokoli. (T.j. ni potrebno slediti Leonardovi strategiji, če obstaja še kakšna enako dobra strategija.)

3. primer

Če nadaljujemo z 2. primerom, predpostavi, da si v `ComputeAdvice` naračunal A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Da bi to sporočil sistemu, bi izvedel sledeče zaporedje klicev: `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`.

Po tem se izvede tvoja druga funkcija, `Assist`, ki dobi zgornje zaporedje A in vrednosti N = 4, K = 2 in R = 16. Pri tem bi morala izvesti natanko N = 4 klicev funkcije `GetRequest`, po nekaterih izmed teh klicev pa mora poklicati še `PutBack`(T), s primerno izbranim T.

Spodnja tabela prikazuje zaporedje klicev, ki ustrezajo (neoptimalnim) izbiram iz 1. primera. Pomišljaj pomeni, da ni bilo klica `PutBack`.

`GetRequest()`	Akcija
2	`PutBack`(1)
0	-
3	`PutBack`(0)
0	`PutBack`(2)

1. podnaloga [8 točk]

- $N \leq 5\,000$.
- Porabiš lahko največ M = 65 000 bitov.

2. podnaloga [9 točk]

- $N \leq 100\,000$.
- Porabiš lahko največ M = 2 000 000 bitov.

3. podnaloga [9 točk]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Porabiš lahko največ M = 1 500 000 bitov.

4. podnaloga [35 točk]

- $N \leq 5\,000$.
- Porabiš lahko največ M = 10 000 bitov.

5. podnaloga [do 39 točk]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Porabiš lahko največ M = 1 800 000 bitov.

Točkovanje pri tej podnalogi je odvisno od dolžine pomočniškega zaporedja, ki ga bo tvoj program sporočil okolju. Naj bo R_{max} maksimum (čez vse testne primere) dolžin pomočniških zaporedij, ki jih naračuna tvoja funkcija `ComputeAdvice`. Potem boš dobil točke po naslednjem postopku:

- 39 točk, če $R_{max} \leq 200\,000$;
- 39 $(1\,800\,000 - R_{max}) / 1\,600\,000$ točk, če $200\,000 < R_{max} < 1\,800\,000$;
- 0 točk, če $R_{max} \geq 1\,800\,000$.

Podrobnosti implementacije

Oddati moraš natanko dve datoteki, *v istem programskem jeziku*.

Prva datoteka naj bo imenovana `advisor.c`, `advisor.cpp` ali `advisor.pas`. V njej naj bo implementacija funkcije `ComputeAdvice`, kot je opisano zgoraj, ki lahko kliče funkcijo `WriteAdvice`. Druga datoteka naj se imenuje `assistant.c`, `assistant.cpp` ali `assistant.pas`. V njej mora biti implementacija funkcije `Assist`, kot je opisano zgoraj, ki lahko kliče funkciji `GetRequest` in `PutBack`.

Sledijo podpisi vseh funkcij.

C/C++ programi

```
void ComputeAdvice(int *C, int N, int K, int M);
void WriteAdvice(unsigned char a);
```

```
void Assist(unsigned char *A, int N, int K, int R);
void PutBack(int T);
int GetRequest();
```

Pascal programi

```
procedure ComputeAdvice(var C : array of LongInt; N, K, M : LongInt);
procedure WriteAdvice(a : Byte);
```

```
procedure Assist(var A : array of Byte; N, K, R : LongInt);
procedure PutBack(T : LongInt);
function GetRequest : LongInt;
```

Te funkcije se morajo obnašati, kot je opisano zgoraj. Kot ponavadi lahko v lastno zadovoljstvo in uporabo ustvarjaš tudi raznolike druge funkcije, ki pa morajo v C/C++ programih biti statično deklarirane (ključna beseda static pred tipom v deklaraciji), saj jih bo primer ocenjevalca povezal (*linking*) skupaj. Lahko pa se vsem tegobam izogneš tako, da preprosto nimaš dveh funkcij z istim imenom v različnih datotekah (vsako v enem od programov). Oddaje na noben način ne smejo uporabljati standardnega vhoda/izhoda, niti katerihkoli drugih datotek.

Med programiranjem tudi pazi na naslednjih nekaj navodil (predloge v tvojem tekmovalnem okolju jim že zadoščajo).

C/C++ programi

Na začetku rešitve moraš vključiti (*include*) datoteki `advisor.h` (v svetovalnem programu, ki računa pomočniško zaporedje) in `assistant.h` (v implementaciji pomočnika). To narediš z naslednjo vrstico kode:

```
#include "advisor.h"
```

ali

```
#include "assistant.h"
```

Datoteki `advisor.h` in `assistant.h` bosta na voljo v mapi v tvojem tekmovalnem okolju in tudi v spletnem vmesniku tekmovanja. Poleg tega ti bodo (na ista načina) na voljo tudi koda in skripte za prevajanje in testiranje tvoje rešitve. Natančneje, ko skopiraš svojo kodo v mapo s temi skriptami, lahko tam poženeš `compile_c.sh` ali `compile_cpp.sh` (odvisno od jezika tvoje kode).

Pascal programi

Uporabiti moraš knjižnici `advisorlib` (v svetovalnem programu, ki računa pomočniško zaporedje) in `assistantlib` (v implementaciji pomočnika). To narediš z naslednjo vrstico kode:

```
uses advisorlib;
```

ali

```
uses assistantlib;
```

Datoteki `advisorlib.pas` in `assistantlib.pas` bosta na voljo v mapi v tvojem tekmovalnem okolju in tudi v spletnem vmesniku tekmovanja. Poleg tega ti bodo (na ista načina) na voljo tudi koda in skripte za prevajanje in testiranje tvoje rešitve. Natančneje, ko skopiraš svojo kodo v mapo s temi skriptami, lahko tam poženeš `compile_pas.sh`.

Primer ocenjevalca

Ocenjevalec bo sprejemal vhodne podatke v sledeči obliki:

- vrstica 1: N, K, M;
- vrstice 2, …, N + 1: C[i].

Ocenjevalec najprej izvede funkcijo `ComputeAdvice`. To ustvari datoteko `advice.txt`, v kateri so posamezni biti pomočniškega zaporedja, ločeni s presledki in končani s številko 2.

Zatem izvede tvojo `Assist` funkcijo in ustvari izhod, v katerem je vsaka vrstica oblike "`R [številka]`" ali "`P [številka]`". Vrstice prve oblike označujejo klice `GetRequest()` in njene odgovore. Vrstice druge oblike predstavljajo klice `PutBack`() in barve, ki so bile izbrane za odstranitev z odra. Izhod se zaključi z vrstico oblike "`E`".

Upoštevaj, da se lahko čas izvajanja tvojega programa na uradnem ocenjevalcu malce razlikuje od časa na tvojem računalniku. Ta razlika naj ne bi bila pomembna. Vseeno si vabljen, da uporabiš vmesnik za testiranje in se s tem prepričaš, da rešitev teče znotraj omejitev.

Omejitve časa in porabe spomina

- Omejitev časa: 7 sekund.
- Omejitev spomina: 256 MiB.