Svētais vakarēdiens

Strādājot pie sava slavenākā sienas gleznojuma "Svētais vakarēdiens", Leonardo bija ļoti aktīvs: katru dienu viens no pirmajiem uzdevumiem bija noteikt, kādas krāsas būs nepieciešamas līdz darbadienas beigām. Viņam var būt nepieciešams daudz krāsu, bet vienlaicīgi uz sastatnēm iespējams novietot ierobežotu krāsu skaitu. Asistenta pienākumos ietilpst rāpšanās uz sastatnēm ar nepieciešamo krāsu un lieko krāsu nonešana un nolikšana uz grīdas novietotā plauktā.

Šajā uzdevumā jums nepieciešams uzrakstīt divas atsevišķas programmas, kas palīdzētu asistentam. Pirmā programma saņemtu Leonardo instrukcijas (viņam dienas garumā gleznošanai nepieciešamo krāsu virkni) un izveidotu *īsu* bitu virkni, ko sauksim par *padomdevēju virkni*. Dienas garumā izpildot Leonardo pieprasījumus, asistentam nebūs piekļuves vēlāk nepieciešamo krāsu sarakstam, toties būs pieejama pirmās programmas izveidotā padomdevēja virkne. Otrā programma saņems šo padomdevēju virkni un izmantos to Leonardo pieprasījumu apstrādes laikā tiešsaistes režīmā (t.i., pēc kārtas pa vienam). Šai programmai, izvēloties optimālu stratēģiju, jāvar izmantot padomdevēja virkne. Tālāk ir dots detalizēts uzdevuma apraksts.

Krāsu pārvietošana starp plauktu un sastatnēm

Aplūkosim šādu vienkāršotu scenāriju. Pieņemsim, ka ir N krāsas, kas sanumurētas pēc kārtas ar skaitļiem no 0 līdz N - 1, un katru dienu Leonardo vēršas pie asistenta pēc jaunas krāsas tieši N reizes. Ar C apzīmēsim šo N jauno krāsu pieprasījuma virkni. Tādejādi mēs varam domāt par C kā N skaitļu (katrs robežās no 0 līdz N - 1, ieskaitot) virkni. Ievērojiet, ka dažas krāsas virknē C var neparādīties nemaz, bet citas var parādīties vairākkārt.

Sastatnes visu laiku ir pilnas un uz tām atrodas K no iespējamām N krāsām (K < N). Sākumā uz sastatnēm atrodas krāsas ar numuriem no 0 līdz K-1 ieskaitot.

Asistents izpilda Leonardo pieprasījumus pa vienam. Ja pieprasītā krāsa *jau atrodas uz sastatnēm*, tad asistents var neko nedarīt un atpūsties. Pretējā gadījumā viņam nepieciešamā krāsa jāpaņem no plaukta un jāuznes uz sastatnēm. Tā kā uz sastatnēm brīvas vietas nav, tad asistentam nepieciešams izvēlēties kādu no uz sastatnēm esošajām krāsām, kura jānogādā lejā un jānoliek plauktā.

Leonardo optimālā stratēģija

Asistents vēlas atpūsties cik daudz vien iespējams. Pieprasījumu skaits, kuru laikā iespējams atpūsties, ir atkarīgs no procesa laikā izdarītajām izvēlēm. Precīzāk, katru reizi, kad kāda krāsa no sastatnēm jānes lejā, pieņemtais lēmums ietekmē situāciju nākotnē. Leonardo asistentam ir izskaidrojis, kā, zinot virkni C, asistents var sasniegt mērķi. Lai noteiktu labāko izvēli, nepieciešams zināt šobrīd uz sastatnēm esošo krāsu komplektu, kā arī virknē C atlikušos pieprasījumus. Starp uz sastatnēm esošajām krāsām zemē nonesamā jāizvēlas, izmantojot šādus likumus:

- Ja uz sastatnēm atrodas krāsa, kas līdz dienas beigām vairs nebūs nepieciešama, asistentam no sastatnēm jānones tā;
- Citādi, jānones tā krāsa, kas *būs nepieciešama vistālāk nākotnē* - t.i., katrai uz sastatnēm esošajām krāsām mēs atrodam tuvāko tās parādīšanos pasūtījumu virknē C. No sastatnēm jānones tā, kas būs nepieciešama visvēlāk.

Iespējams pierādīt, ka, lietojot Leonardo stratēģiju, asistentam būs iespēja atpūsties visilgāk.

1. piemērs

Ja N = 4, tad ir četras krāsas, kas sanumurētas ar skaitļiem no 0 līdz 3 un četri pieprasījumi. Pieņemsim, ka pieprasījumu virkne C ir (2, 0, 3, 0) un K = 2. Tātad, uz Leonardo sastatnēm katrā brīdī var novietot divas krāsas un, kā aprakstīts iepriekš, sākumā uz sastatnēm atrodas krāsas ar numuriem 0 un 1. Pierakstīsim uz sastatnēm esošo krāsu numurus kā [0, 1]. Viens variants, kā asistents var izpildīt pieprasījumus, ir šāds:

Pirmā pieprasītā krāsa (nr. 2) neatrodas uz sastatnēm. Asistents to uznes un izvēlas nonest krāsu nr. 1. Uz sastatnēm atrodas [0, 2].

- Nākamā pieprasītā krāsa (nr. 0) jau atrodas uz sastatnēm, tāpēc asistents var atpūsties.

- Uznesot trešo pieprasīto krāsu (nr. 3), asistents izvēlas nonest krāsu nr.0. Uz sastatnēm atrodas [3, 2].

- Visbeidzot, beidzamā pieprasītā krāsa (nr. 0) jānogādā no plaukta uz sastatnēm. Asistents izvēlas nonest krāsu nr. 2 un uz sastatnēm tagad atrodas [3,0].

Ievērojiet, ka šajā piemērā asistents nevadījās pēc Leonardo optimālās stratēģijas. Pielietojot optimālo stratēģiju, krāsa nr. 2 tiktu nonesta trešajā solī un asistents varētu atpūsties beidzamā pieprasījuma laikā.

Asistenta stratēģija ierobežota atmiņas apjoma apstākļos

No rīta asistents lūdz Leonardo uzrakstīt virkni C uz papīra, lai būtu iespējams pielietot optimālo stratēģiju. Leonardo ir apsēsts ar sava darba tehnikas turēšanu slepenībā un atsakās atdot asistentam papīru ar uzrakstīto virkni C. Viņš atļauj asistentam izlasīt C un mēģināt to atcerēties.

Diemžēl asistenta atmiņa ir gauži slikta. Viņš var atcerēties ne vairāk kā M bitus. Vispārīgā gadījumā tas var neļaut asistentam rekonstruēt visu virkni C. Tādejādi, asistentam nepieciešams izdomāt kādu gudru algoritmu, kā izskaitļot bitu virkni, kuru viņam atcerēties. Viņš šo virkni sauks par "padomdevēju virkni" un apzīmēsim šo virkni ar A.

2. piemērs

No rīta asistents drīkst paņemt papīru ar Leonardo uzrakstīto virkni C, izlasīt virkni un izvēlēties kā rīkoties turpmāk. Piemēram, viņš var izvēlēties pierakstīt situāciju uz sastatnēm pēc katra pieprasījuma apstrādes. Piemēram, lietojot šo pieeju 1. piemērā aprakstītajai (neoptimālajai) darbību virknei, situācijas uz sastatnēm izmaiņa tiktu pierakstīta šādi: [0, 2], [0, 2], [3, 2], [3, 0]. (Atcerieties, ka asistentam ir zināms sākotnējais krāsu novietojums uz sastatnēm [0, 1], tāpēc to pierakstīt nav nepieciešams.)

Tagad pieņemsim, ka M = 16, tādejādi asistents ir spējīgs atcerēties 16 bitus informācijas. Tā kā N = 4, katras krāsas numura saglabāšanai pietiek ar 2 bitiem. Tādejādi 16 biti ir pietiekami situācijas uz sastatnēm izmaiņu pierakstam. Tādejādi asistenta padomdevēja virkne A būtu šāda: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Vēlāk asistents var atkodēt šo padomdevēju virkni un izmantot to, lai izlemtu kā rīkoties.

(protams, gadījumam M = 16 asistents var izvēlēties atcerēties visu sākotnējo virkni C, izmantojot tikai astoņus no pieejamajiem 16 bitiem. Ar šo piemēru tika mēģināts ilustrēt, ka asistentam ir iespējamas dažādas vienlīdz labas atcerēšanās stratēģijas).

Uzdevums

Jums jāuzraksta *divas atsevišķas programmas* vienā un tajā pašā programmēšanas valodā. Šīs programmas tiks izpildītas pēc kārtas, bez iespējas tām komunicēt savā starpā.

Pirmo programmu izpildīs asistents no rīta. Šai programmai tiks padota pieprasījumu virkne C un tai jāaprēķina un jāizvada padomdevēja virkne A.

Otro programmu asistents izmantos dienas laikā. Programmai tiks padota padomdevēja virkne A un tad tai būs jāapstrādā Leonardo pieprasījumu virkne C. Ievērojiet, ka virknes C krāsu pieprasījumi programmai tiks padoti pēc kārtas pa vienam un nākamais pieprasījums tiks padots tikai pēc iepriekšējā pieprasījuma apstrādes.

Precīzāk, pirmajā programmā jums jāimplementē procedūra `ComputeAdvice`(C, N, K, M), kuras parametri ir masīvs C, kurā ir doti N veseli skaitļi (katrs robežās 0, …, N - 1), krāsu uz sastatnēm skaits K un pieejamo bitu skaits M, kurus atļauts izmantot padomdevējas virknes veidošanai. Šai programmai jāizskaitļo padomdevēja virkne A, kas sastāv no ne vairāk kā M bitiem. Pēc tam virkne A ir jāpaziņo sistēmai, noraidot katru bitu pa vienam pēc kārtas šādā veidā:

- `WriteAdvice`(B) — pievienot bitu B līdz šim izveidotajai virknei A. (Šo procedūru drīkst izsaukt ne vairāk kā M reizes.)

Otrajā programmā jums jāimplementē viena procedūra `Assist`(A, N, K, R). Šīs programmas ievaddati ir padomdevēja virkne A, iepriekš definēti naturāli skaitļi N un K, kā arī bitu skaits R virknē A (R ≤ M). Šai procedūrai jārealizē jūsu piedāvātā asistenta stratēģija, izmantojot šādas pieejamās funkcijas un procedūras:

- `GetRequest`() — atgriež nākamo Leonardo pieprasīto krāsu. (Informācija par tālāk sekojošiem pieprasījumiem netiek atklāta.)
- `PutBack`(T) — nonest krāsu ar numuru T no sastatnēm un nolikt to plauktā. Šo procedūru drīkst izsaukt tikai tādiem krāsu numuriem T, kas šajā brīdī atrodas uz sastatnēm.

Izpildes laikā jūsu procedūrai `Assist` jāizsauc `GetRequest` tieši N reizes, katrā no tām saņemot vienu Leonardo pieprasījumu pareizā secībā. Pēc katra `GetRequest` izsaukuma, ja krāsa ar funkcijas atdoto numuru *neatrodas* uz sastatnēm, jūsu programmai *jāizsauc* arī procedūra `PutBack`(T) ar izvēlēto T vērtību. Pretējā gadījumā, jūs *nedrīkstat* izsaukt `PutBack`. Šo

noteikumu neizpilde tiks uzskatīta par kļūdu un jūsu programmas darbība tiks pārtraukta. Atcerieties, ka sākumā uz sastatnēm atrodas krāsas ar numuriem no 0 līdz K - 1, ieskaitot.

Atsevišķs tests tiks uzskatīts par atrisinātu, ja abas jūsu uzrakstītās procedūras precīzi sekos noteiktajiem likumiem un kopējais `PutBack` izsaukumu skaits *precīzi sakrīt* ar to, kādu nodrošina Leonardo optimālā stratēģija. Ievērojiet, ka, ja eksistē vairākas vienlīdz labas stratēģijas, kas nodrošina tādu pašu `PutBack` izsaukumu skaitu, tad jūsu programma drīkst realizēt jebkuru no tām (t.i. netiek pieprasīta sekošana Leonardo stratēģijai, ja eksistē cita tikpat laba startēģija).

3. piemērs

Turpinot 2.piemēru, pieņemsim, ka `ComputeAdvice` jūs izskaitļojāt A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Lai paziņotu A sistēmai, jāveic šādi procedūru izsaukumi:
`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`).

Jūsu otrajai procedūrai `Assist` jātiek izpildītai, saņemot iepriekš izveidoto padomdevēja virkni A, un vērtības N = 4, K = 2, un R = 16. Procedūrai `Assist` tad jāveic tieši N = 4 `GetRequest` izsaukumi. Pēc dažiem no šiem izsaukumiem procedūrai `Assist` jāizsauc arī `PutBack`(T) ar atbilstošo T vērtību.

Zemāk redzamajā tabulā dota izsaukumu virkne, kas atbilst (neoptimālajām) 1. piemērā aprakstītajām darbībām. Defise nozīmē, ka nav `PutBack` izsaukuma.

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

1. apakšuzdevums [8 punkti]

- $N \leq 5\,000$.
- Drīkst izmantot ne vairāk kā M = 65 000 bitus.

2. apakšuzdevums [9 punkti]

- $N \leq 100\,000$.
- Drīkst izmantot ne vairāk kā M = 2 000 000 bitus.

3. apakšuzdevums [9 punkti]

- $N \leq 100\,000$.

- $K \leq 25\,000$.
- Drīkst izmantot ne vairāk kā M = 1 500 000 bitus.

4. apakšuzdevums [35 punkti]

- $N \leq 5\,000$.
- Drīkst izmantot ne vairāk kā M = 10 000 bitus.

5. apakšuzdevums [līdz 39 punktiem]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Drīkst izmantot ne vairāk kā M = 1 800 000 bitus.

Punktu skaits par šo uzdevumu ir atkarīgs no padomdevēja virknes garuma R. Precīzāk, ja R_{max} ir maksimālais (starp visiem testiem) padomdevēja virknes garums, ko ir izveidojusi jūsu procedūra `ComputeAdvice`, punktu skaits tiks aprēķināts kā:

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

Implementācijas detaļas

Jums jāiesūta tieši divi faili *vienā programmēšanas valodā*.

Pirmā faila nosaukumam jābūt `advisor.c`, `advisor.cpp` vai `advisor.pas`. Šim failam jāimplementē iepriekš aprakstītā procedūra `ComputeAdvice` un tai jāizsauc procedūra `WriteAdvice`. Otram failam jābūt `assistant.c`, `assistant.cpp` vai `assistant.pas`. Šajā failā jābūt implementētai iepriekš aprakstītajai procedūrai `Assist` un jāizsauc procedūras `GetRequest` un `PutBack`.

Procedūru signatūras ir šādas:

C/C++ programmām

```
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 programmām

```
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;
```

Šīm procedūrām ir jādarbojas iepriekš aprakstītajā veidā. Jūs drīkstat implementēt arī citas procedūras un funkcijas iekšējai lietošanai. C/C++ programmās šādām iekšējām procedūrām un funkcijām jābūt deklarētām kā `static`, lai paraugvērtētājs varētu tās salinkot. Vai arī dažādos moduļos izvairieties no funkciju ar vienādiem vārdiem lietošanas. Jūsu iesūtītās procedūras nedrīkst izmantot standarta ievadu un izvadu, kā arī jebkuru ārēju failu.

Programmējot savu risinājumu, jums jāievēro vēl daži nosacījumi (risinājumu šablonos, kas pieejami sacensību vidē, šie nosacījumi ir ievēroti).

C/C++ programmām

Sava risinājuma sākumā jums padomdevēja un asistenta programmā jāiekļauj, attiecīgi, faili `advisor.h` vai `assistant.h`. To panāk, iekļaujot programmas izejas kodā rindu:

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

vai

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

Faili `advisor.h` un `assistant.h` būs pieejami katalogā sacensību vidē, kā arī būs pieejami izmantojot sacensību tīmekļa saskarni. Šādā pat veidā būs pieejams jūsu risinājuma testēšanai nepieciešamie skripti un programmu moduļi. Precīzāk, pēc sava risinājuma iekopēšanas katalogā, kur atrodas šie skripti, jums jāpalaiž `compile_c.sh` vai `compile_cpp.sh` (atkarībā no valodas, kurā rakstīta jūsu programma).

Pascal programmām

Sava risinājuma sākumā jums padomdevēja un asistenta programmā jāiekļauj, attiecīgi, moduļi `advisorlib` vai `assistantlib`. To panāk, iekļaujot programmas izejas kodā rindu:

```
uses advisorlib;
```

vai

```
uses assistantlib;
```

Faili `advisorlib.pas` un `assistantlib.pas` būs pieejami katalogā sacensību vidē, kā

arī būs pieejami izmantojot sacensību tīmekļa saskarni. Šādā pat veidā būs pieejams jūsu risinājuma testēšanai nepieciešamie skripti un programmu moduļi. Precīzāk, pēc sava risinājuma iekopēšanas katalogā, kur atrodas šie skripti, jums jāpalaiž `compile_pas.sh`.

Paraugvērtētājs

Paraugvērtētājs akceptē šādi formatētus ievaddatus:

- 1. rindā: N, K, M;
- Katrā no nākamajām N rindām: C[i].

Paraugvērtētājs vispirms izpildīs procedūru `ComputeAdvice`. Tās rezultātā tiks izveidots fails `advice.txt`, kas saturēs padomdevēja virknes bitus, atdalītus ar tukšumzīmēm. Virknes beigas apzīmēs cipars 2.

Tālāk paraugvērtētājs izpildīs jūsu procedūru `Assist` un ģenerēs izvaddatus, kur katra rinda ir vai nu formā "`R [number]`", vai arī formā "`P [number]`". Pirmā tipa rindas apzīmē procedūras `GetRequest()` izsaukumus un saņemtās atbildes darbības. Otrā tipa virknes atbilst procedūras `PutBack`() izsaukumiem un to krāsu numuriem, kuras izvēlētas nonešanai no sastatnēm un nolikšanai plauktā. Izvaddatu beigās atrodas rinda "`E`".

Ievērojiet, ka oficiālajā vērtētājā izpildes laiks var mazliet atšķirties no izpildes laika uz jūsu lokālā datora. Šīm atšķirībām nebūtu jābūt būtiskām, tomēr ieteicams izmantot testēšanas saskarni, lai pārliecinātos, ka jūsu risinājums iekļaujas atvēlētajā laikā.

Izpildes laika un atmiņas ierobežojumi

- Izpildes laika ierobežojums: 7 seconds.
- Atmiņas ierobežojums: 256 MiB.