Վերջին ընթրիքը

Լեոնարդոն շատ էր ծանրաբեռնված, երբ աշխատում էր *Վերջին ընթրիքի*՝ իր ամենահայտնի որմնանկարի վրա. ամենօրյա իր առաջին գործերից մեկն էր լինում որոշել, թե օրվա ընթացքում ինչ որմնաներկ է պետք օգտագործել: Նրան շատ ներկեր էին պետք գալիս, բայց իր փայտամածի վրա նա կարող էր սահմանափակ թվով ներկեր պահել: Նրա օգնականի պարտականությունների մեջ, բացի այլ բաներից, մտնում էր մագլցել փայտամած նրան ներկ հասցնելու համար և ապա ներկը ներքև իջեցնել և դնել գետնին դրված դարակում:

Այս խնդրում պահանջվում է գրել երկու առանձին ծրագրեր օգնականի համար: Առաջին ծրագիրը ստանալով Լեոնարդոյի հրահանգները (գույների հաջորդականությունը, որ Լեոնարդոյին պետք կգա ողջ օրվա ընթացքում), պետք է ստեղծի, *հուշում* կոչվող, *կարճ* պլան: Օրվա ընթացքում Լեոնարդոյի պահանջները կատարելու ժամանակ օգնականին Լեոնարդոի ապագա պահանջները հասանելի չեն լինի, հասանելի կլինի միայն ձեր առաջին ծրագրի տված հուշումը: Երկրորդ ծրագիրը ստանալով հուշումը, ապա պետք է ստանա և մշակի Լեոնարդոյի պահանջները օնլայն եղանակով (այսինքն մեկ առ մեկ): Այս ծրագիրը պետք է հասկանա հուշման իմաստը և այն օգտագործի օպտիմալ ընտրություն կատարելու համար: Ամեն ինչ առավել մանրամասն բացատրված է ստորև:

Դարակի և փայլտամածի միջև ներկի տեղափոխումը

Մենք կդիտարկենք պարզեցված սցենար: Ենթադրենք ներկերի քանակը N է, նրանք համարակալված են 0-ից N - 1 թվերով, և ամեն օր Լեոնարդոն ճիշտ N անգամ իր օգնականից ներկ է ուզում: Դիցուք C-ն Լեոնարդոյին անհրաժեշտ N ներկերի հաջորդականությունն է: Այսինքն, C-ն 0-ից N-1 սահմաններում գտնվող թվերի հաջորդականություն է: Նկատենք, որ որոշ գույներ կարող չլինել C-ում, իսկ որոշ գույներ կարող են մի քանի անգամ հանդիպել:

Փայտամածը միշտ լիքն է և պարունակում է K ներկ, որտեղ K < N: Սկզբում փայտամածը պարունակում է 0-ից K - 1 ներկերը:

Օգնականը Լեոնարդոյի պահանջները կատարում է մեկ առ մեկ: Եթե պահանջվող որմնաներկը *արդեն փայտամածի վրա է*, օգնականը հանգստանում է: Հակառակ դեպքում նա վերցնում է պահանջվող ներկը դարակից և տանում է փայտամած: Իհարկե, փայտամածի վրա նոր ներկի համար տեղ չկա, և օգնականը պետք է ընտրի այնտեղի ներկերից մեկը և

ետ տանի դարակ:

Լեոնարդոյի օպտիմալ ստրատեգիան

Օգնականը ցանկանում է, որքան հնարավոր է շատ անգամ, հանգստանալ: Թե քանի անգամ նա կհանգստանա, կախված է նրանից, թե ինքը իր աշխատանքի ընթացքում ինչ ընտրություններ են անում: Ավելի ստույգ, կախված նրանից, թե օգնականը ինչ գույնի ներկ է փայտամածից տանում, ապագայում տարբեր ընտրության առաջ կարող է կանգնել: Լեոնարդոն բացատրում է նրան, թե ինչպես է նա կարող հասնել իր նպատակին իմանալով C-ն: Փայտամածից հեռացնելու համար գույնի լավագույն ընտրությունը կարելի է կատարել ընթացիկ պահին փայտամածի վրա գտնվող գույների և C-ում մնացած հարցումների հիման վրա: Փայտամածի գույներից ընտրությունը պետք է կատարել հետևյալ կանոնների հիման վրա.

- Եթե փայտամածի վրա կա այնպիսի գույն, որն էլ ապագայում չի օգտագործվելու, օգնականը պետք է այդ գույնը հանից փայտամածից:
- Հակառակ դեպքում փայտամածից պետք է հեռացնել այն գույնը, որը *որքան հնարավոր է, ավելի ուշ պետք կգա ապագայում*: (Այսինքն, փայտամածի գույներից յուրաքանչյուրի համար գտնենք նրա առաջին պետք լինելը ապագայում: Դարակ է վերադարձվում այն գույնը, որը մյուսների համեմատ ավելի ուշ պետք կգա):

Կարելի է ապացուցել, որ Լեոնարդոյի ստրատեգիայի միջոցով օգնականը առավելագույնս հանգստանալու հնարավորություն կունենա:

Օրինակ 1

Դիցուք N = 4, այսինքն ունենք 4 գույն (համարակալված 0-ից 3 թվերով) և 4 հարցում: Դիցուք հարցումների հերթականությունը այսպիսին է C = (2, 0, 3, 0): Նաև ենթադրենք, որ K = 2: Այսինքն Լեոնարդոյի փայտամածը ամեն անգամ կարող է միայն 2 ներկ պահել: Ինչպես ասված է վերևում, սկզբում փայտամածի վրա գտնվում են 0 և 1 ներկերը: Փայտամածի պարունակությունը կգրենք հետևյալ կերպ. [0, 1]: Օգնականը հարցումները կարող է կատարել, օրինակ, հետևյալ կերպ.

- Առաջին որմնաներկը (համար 2) փայտամածի վրա չէ: Օգնականը այն տանում է այնտեղ և որոշում է փայտամածից հանել 1 ներկը: Փայտամածի ընթացիկ վիճակը կդառնա [0, 2]:

- Հաջորդ պահանջվող գույնը (համար 0) արդեն փայտամածի վրա է, այնպես որ օգնականը հանգստանում է:

- Երրորդ հարցման համար (համար 3), օգնականը հեռացնում է 0 գույնը, փայտամածը դարձնելով [3, 2]:

- Վերջապես, վերջին պահանջվող գույնը (համար 0) պետք է դարակից տանել փայտամած: Օգնականը որոշում է հեռացնել 2 գույնը, և փայտամածը հիմա դառնում է [3, 0]:

Նկատենք, որ այս օրինակում օգնականը օպտիմալ կերպով չի սպասարկում Լեոնարդոյին: Օպտիմալ ստրատեգիայի դեպքում երրորդ քայլին պետք է հանել 2 գույնը և, այդպիսով, հանգստանալ վերջին քայլին:

Օգնականի ստրատեգիան, երբ նրա հիշողությունը սահմանափակ է

Առավոտյան օգնականը Լեոնարդոյին խնդրում է գրել C հաջորդականությունը թղթի վրա, որպեսզի նա կարողանա գտնել օպտիմալ ստրատեգիա և դրանով շարժվել: Սակայն, Լեոնարդոն մտահոգված է, որ իր աշխատելու տեխնիկական գաղտնի մնա, և նա արգելում է օգնականին թուղթ ունենալ: Նա միայն թույլատրում է օգնականին կարդալ C-ի և փորձել հիշել այն:

Դժբախտաբար, օգնականի հիշողությունը շատ վատ է: Նա կարող է հիշել մինչև M բիթ: Դրա պատճառով նա ընդունակ չէ վերականգնել ամբողջ C հաջորդականությունը: Բայց օգնականը կարող է խելացի ձևով հաշվել բիթերի այն հաջորդականությունը, որ պետք է հիշել: Այդ հաջորդականությունը անվանենք *հուշող հաջորդականություն*, և մենք նշանակենք A-ով:

Օրինակ 2

Առավոտյան օգնականը կարող է վերցնել Լեոնարդոյի թուղթը, որի վրա գրված է C հաջորդականությունը, կարդալ հաջորդականությունը, և կատարել անհրաժեշտ ընտրություն: Օրինակ, ընտրությունը կարող կատարվել հետևյալում. ամեն հարցումից հետո ստուգել փայտամածի պարունակությունը: Օրինակ 1-ի (ոչ օպտիմալ) ստրատեգիայի դեպքում փայտամածի վիճակների հաջորդականությունը այսպիսին է.[0, 2], [0, 2], [3, 2], [3, 0]: (Հիշենք, նա գիտի, որ փայտամածի սկզբնական պարունակությոնը [0, 1] է):

Այժմ ենթադրենք, որ M = 16, այսինքն օգնականը կարող է հիշել մինչև 16 բիթ ինֆորմացիա: Քանի որ N = 4, յուրաքանչյուր գույն կարող ենք պահել 2 բիթում: Հետևաբար 16 բիթը բավական է փայտամածի վիճակների վերևի հաջորդականությունը պահելու համար: Այսպիսով, օգնականը հաշվում է հետևյալ հուշող հաջորդականությունը. A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0):

Այդ օրն ավելի ուշ օգնականը կարող է ապակոդավորել այս հուշող հաջորդականությունը և այն օգտագործել իր ընտրությունները կատարելու համար:

(Իհարկե M = 16 հիշողությունով օգնականը կարող է հիշել ամբողջ C հաջորդականությունը 16-ի փոխարեն օգտագործելով միայն 8 բիթ: Այս օրինակում մենք միայն ուզում ենք ցուցադրել, որ նա կարող այլ մոտեցումներ ևս ունենալ):

Խնդիրը

Դուք պետք է գրեք *երկու առանձին ծրագիր* միևնույն ծրագրավորման լեզվով: Այս ծրագրերը պետք է հերթով կատարվեն՝ առանց կատարման ընթացքում իրար հետ կապ պահելու:

Առաջին ծրագիրը պետք է օգնականը օգտագործի առավոտյան: Այս ծրագիրը տրված C հաջորդականության հիման վրա պետք է կառուցի A հուշող հաջորդականությունը:

Երկրորդ ծրագիրը օգնականը պետք է օգտագործի օրվա ընթացքում: Այս ծրագիրը պետք է ստանա A հուշող հաջորդականությունը և հետո պետք է մշակի Լեոնարդոյի հարցումների C հաջորդականությունը: Նկատենք, որ C հաջորդականությունը ծրագրին պետք է տրվի հարցում առ հարցում, և յուրաքանչյուր հարցում պետք է մշակվի նախքան հաջորդի ստացումը:

Ավելի ստույգ, առաջին ծրագրում դուք պետք է իրականացնեք `ComputeAdvice`(C, N, K, M) ֆունկցիան, որտեղ C-ն ամբողջ թվերի (յուրաքանչյուրը 0, …, N - 1 տիրույթից) N երկարության զանգված է, K-ն փայտամածում գույների քանակն է, M-ը հիշողության բիթերի քանակն է: Այս ծրագիրը պետք է կառուցի M բիթերից կազմված A հուշող հաջորդականությունը: Ծրագիրը A հաջորդականությունը համակարգին փոխանցելու համար, A-ի յուրաքանչյուր բիթի համար, պահպանելով կարգը, պետք է կանչի հետևյալ ֆունկցիան.

- `WriteAdvice`(B) — ավելացնում է B բիթը ընթացիկ հուշող A հաջորդականությանը: (Այս ֆունկցիան կարող եք կանչել առավելագույնը M անգամ):

Երկրորդ ծրագրում պետք է իրականացնեք `Assist`(A, N, K, R) ֆունկցիան: Այստեղ A-ն հուշող հաջորդականությունն է: N-ը և K-ն սահմանված են վերևում: R-ը A-ի բիթերի փաստացի քանակն է ($R \leq M$): Այս ֆունկցիան պետք է կատարի օգնականի համար ձեր առաջարկած ստրատեգիան, օգտագործելով հետևյալ ֆունկցիաները, որոնք ձեր տրամադրության տակ են.

- `GetRequest`() — վերադարձնում է Լեոնարդոյի պահանջված հաջորդ գույնը: (Ապագա հարցումների վերաբերյալ ոչ մի ինֆորմացիա չի տրվում):

- `PutBack`(T) — T գույնը փայտամածից տանում է դարակ: Դուք այս ֆունկցիան կարող եք կանչել միայն այն դեպքում, երբ T-ն ընթացիկ պահին փայտամածի գույներից մեկն է:

Կատարվելուց ձեր `Assist` ֆունկցիան պետք է կանչի `GetRequest` ֆունկցիան ճիշտ N անգամ, ամեն անգամ ստանալով Լեոնարդոյի հերթական հարցումը: `GetRequest`-ի յուրաքանչյուր կանչից հետո, եթե պահանջվող գույնը փայտամածի վրա *չէ*, դուք *պետք է* կանչեք նաև

`PutBack(T)` ֆունկցիան ձեր ընտրած T-ով: Հակառակ դեպքում *չպետք է* կանչեք `PutBack` ֆունկցիան: Եթե ձեր ծրագիրը այս կերպ չաշխատի, դա կհամարվի սխալ, և ձեր ծրագրի աշխատանքը կդադարեցվի: Հիշեք, խնդրեմ, որ սկզբում փայտամածը պարունակում է 0-ից K - 1 գույները, նեառյալ:

Թեստը կհամարվի լուծված, եթե ձեր երկու ֆունկցիաները բավարարում են խնդրի պահանջներին և `PutBack`-ի կանչերի ընդհանուր քանակը *ճիշտ հավասար է* Լեոնարդոյի օպտիմալ ստրատեգիայի դեպքում համապատասխան քանակին: Նկատենք, որ եթե կա մի քանի ստրատեգիա, որոնք նույն քանակով են կանչում `PutBack`, ֆունկցիան, ձեր ծրագիրը կարող է նրանցից ցանկացածը կատարել: (այսինքն, չի պահանջվում անպայման հետևել Լեոնարդոյի ստրատեգիային, եթե կա նույնքան լավ այլ ստրատեգիա):

Օրինակ 3

Շարունակելով օրինակ 2-ը, ենթադրենք `ComputeAdvice`-ում կառուցել եք A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0). Այդ տվյալները համակարգին փոխանցելու համար պետք է անել կանչերի հետևյալ հաջորդականությունը. `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)`.

Ապա ձեր երկրորդ՝ `Assist` ֆունկցիան կարող է կատարվել, ստանալով վերևի A հաջորդականությունը, N = 4, K = 2 և R = 16 արժեքները: `Assist` ֆունկցիան պետք է անի `GetRequest` ֆունկցիայի ճիշտ N = 4 կանչ: Նաև այդ կանչերի արանքում `Assist` ֆունկցիան պետք է կանչի `PutBack(T)`-ն թույլատրելի ձևով ընտրելով T-ն:

Ստորև բերված աղյուսակը ցուցադրում է օրինակ 1-ի ոչ օպտիմալ ընտրություններին համապատասխանող կանչերի հաջորդականությունը: Գծիկը նշանակում է, որ `PutBack`-ի կանչ չկա:

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

Ենթախնդիր 1 [8 միավոր]

- $N \leq 5\,000$.
- Դուք կարող եք օգտագործել առավելագույնը M = 65 000 բիթ:

Ենթախնդիր 2 [9 միավոր]

- $N \leq 100\,000$.
- Դուք կարող եք օգտագործել առավելագույնը $M = 2\,000\,000$ բիթ:

Ենթախնդիր 3 [9 միավոր]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Դուք կարող եք օգտագործել առավելագույնը $M = 1\,500\,000$ բիթ:

Ենթախնդիր 4 [35 միավոր]

- $N \leq 5\,000$.
- Դուք կարող եք օգտագործել առավելագույնը $M = 10\,000$ բիթ:

Ենթախնդիր 5 [մինչև 39 միավոր]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Դուք կարող եք օգտագործել առավելագույնը $M = 1\,800\,000$ բիթ:

Այս ենթախնդրի համար ձեր միավորը կախված է ձեր ծրագրի կազմած հուշող հաջորդականության R երկարությունից: Ավելի ստույգ, եթե R_{max}-ը ձ ե ր `ComputeAdvice` ֆունկցիայի տված հուշող հաջորդականության մաքսիմում երկարությունն է (ըստ բոլոր թեստերի), ապա ձեր միավորը կլինի.

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

Իրականացման մանրամասներ

Դուք պետք է submit անեք ճիշտ երկու ֆայլ գրված *միևնույն ծրագրավորման լեզվով*:

Առաջին ֆայլը կոչվում է `advisor.c`, `advisor.cpp` կամ `advisor.pas`: Այս ֆայլում պետք է իրականացնել `ComputeAdvice` ֆունկցիան ինչպես

նկարագրված է վերևում, կանչելով `WriteAdvice` ֆունկցիան: Երկրորդ ֆայլը կոչվում է `assistant.c`, `assistant.cpp` կամ `assistant.pas`: Այս ֆայլում պետք է իրականացնել `Assist` ֆունկցիան ինչպես նկարագրված է վերևում, կանչելով `GetRequest` և `PutBack` ֆունկցիաները:

Ստորև բերված են բոլոր ֆունկցիաների սիգնատուրները:

C/C++ ծրագրեր

```
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 ծրագրեր

```
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;
```

Այս ծրագրերը պետք է իրենց պահեն այնպես, ինչպես նկարագրված է վերևում: Իհարկե, դուք կարող եք ներքին օգտագործման համար այլ ֆունկցիաներ նույնպես իրականացնել: C/C++ ծրագրերում ձեր ներքին ֆունկցիաները պետք է հայտարարել `static`, որպեսզի գրեյդերը դրանք իրար կապակցի: Կամ էլ պիտի, պարզապես, խուսափել նույն անունով երկու ֆունկցիա (ամեն ծրագրում մի հատ) հայտարարելուց: Ձեր ծրագրերը ստանդարտ մուտք/ելքում, ինչպես նաև որևէ ֆայլում, գրել կարդալու գործողություններ չպիտի կատարեն:

Ծրագիրը գրելիս դուք նաև պետք է հետևեք հետևյալ հրահանգներին (մրցույթի ձեր միջավայրի շաբլոնները բավարարում են ստորև թվարկված պահանջներին):

C/C++ ծրագրեր

Ձեր լուծման սկզբում դուք պետքէ ավելացնեք `advisor.h` և `assistant.h` տողերը, համապատասխանաբար advisor-ում և assistant-ում: Դրա համար ձեր ծրագրում պետք է ավելացվի.

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

կամ

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

Այս երկու՝`advisor.h` և `assistant.h` ֆայլերը կլինեն ձեր ֆոլդերում ինչպես նաև մրցույթի վեբ-ինտերֆեյսում: Ձեզ նաև կտրամադրվի (նույն կերպով) ձեր լուծումը կոմպիլացնելու և թեստավորելու համար կոդ և սկրիպտ: Մասնավորապես ձեր լուծումը այդ սկրիպտները պարունակող ֆոլդերը պատճենելուց հետո դուք կարող եք աշխատացնել `compile_c.sh` կամ `compile_cpp.sh` սկրիպտը (նայած թե ձեր կոդի լեզուն որն է):

Pascal ծրագրեր

Դուք կարող եք օգտագործել `advisorlib` և `assistantlib` յունիթները, համապատասխանաբար, advisor-ում և assistant-ում: Դրա համար ձեր կոդում պետք է ընդգրկել հետևյալ տողը.

```
uses advisorlib;
```

կամ

```
uses assistantlib;
```

Ձեր ֆոլդերում նաև վեբ ինտերֆեյսում կարող եք գտնել `advisorlib.pas` և `assistantlib.pas` ֆայլերը: Ձեզ նաև կտրվի (նույն եղանակով) ձեր լուծումը կոմպիլացնելու և աշխատացնելու համար կոդ և սկրիպտներ: Մասնավորապես, ձեր լուծումը այս սկրիպտները պարունակող ֆոլդեր պատճենելուց հետո կարող եք աշխատացնել `compile_pas.sh`.սկրիպտը:

Գրեյդերի օրինակ

Գրեյդերի օրինակն ընդունում է տվյալները հետևյալ ֆորմատով.

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

Գրեյդերը սկզբում կանչելու է `ComputeAdvice` ֆունկցիային: Արդյունքում գեներացվելու է `advice.txt` ֆայլը, որը պարունակում է հուշող հաջորդականության բիթերի արժեքները՝ իրարից մեկ բացակով անջատված, իսկ վերջում դրվելու է 2:

Ապա այն կկատարի ձեր `Assist` ֆունկցիան և գեներացնի ելքը, որտեղ յուրաքանչյուր տող կամ `"R [number]"` տեսքի է, կամ `"P [number]"` տեսքի: Առաջին տիպի տողերը նշանակում են `GetRequest()`-ի կանչերը և ստացված պատասխանները: Երկրորդ տիպի կանչերը ցույց են տալիս `PutBack()` ֆունկցիայի կանչերը և ետ տանելու համար ընտրված գույները: Ելքը ավարտվում է `"E"` տեսքի տողով:

Նկատեք, խնդրեմ, որ պաշտոնական գրեյդերով ձեր ծրագրի աշխատելու

ժամանակը փոքր ինչ կարող է տարբերվել ձեր լոկալ համակարգչում աշխատանքի ժամանակից: Այս տարբերությունը էական չպիտի լինի: Այնուհանդերձ, դուք կարող եք օգտագործել թեստավորման ինտերֆեյսը համոզվելու համար, որ ձեր ծրագիրը տեղավորվում է ժամանակային սահմանափակման մեջ:

Ժամանակի և հիշողության սահմանափակումները

- Ժամանակի սահմանափակումը. 7 վայրկյան:
- Հիշողության սահմանափակումը. 256 MiB.