საიდუმლო სერობა

ლეონარდო ძალიან აქტიური იყო "საიდუმლო სერობა"–ზე მუშაობისას, რომელიც მის ერთ-ერთ ყველაზე ცნობილ ფრესკას წარმოადგენს. მისთვის დღის ერთ–ერთი პირველი ამოცანა იყო – გადაეწყვიტა რომელი ფერის საღებავები გამოეყენებინა დღის განმავლობაში. მას ბევრი საღებავები სჭირდებოდა, მაგრამ მხოლოდ გარკვეული რაოდენობის საღებავები შეეძლო ჰქონოდა თავის ხარაჩოებზე ("საიდუმლო სერობა" კედელზეა დახატული). მისი თანაშემწის მოვალეობაში შედიოდა მიეტანა ისინი ხარაჩოზე, აგრეთვე დაეწყო ისინი უკან ხარაჩოდან იატაკზე.

ამ ამოცანაში თქვენ თანაშემწისათვის უნდა დაწეროთ ორი განსხვავებული პროგრამა. პირველი მათგანი მიიღებს ლეონარდოს ინსტრუქციებს (საღებავების მიმდევრობას, რომლებიც ლეონარდოს დასჭირდება დღის განმავლობაში) და შექმნის ბიტების "მოკლე" სტრიქონს, რომელსაც "რეკომენდაცია" ვუწოდოთ. დღის განმავლობაში, ლეონარდოს მოთხოვნათა შესრულებისას ასისტენტს არ შეუძლია მისი სამომავლო მოთხოვნათა გაგება. მას შეუძლია მხოლოდ თქვენი პროგრამის მიერ გენერირებული რეკომენდაციების გამოყენება. მეორე პროგრამა მიიღებს თქვენს რეკომენდაციას და ლეონარდოს მოთხოვნებს რეალურ დროში (ანუ ყოველ ჯერზე – თითოს). მეორე პროგრამამ უნდა იცოდეს, რა არის რეკომენდაცია და გამოიყენოს ის ოპტიმალური ქმედებისათვის. ქვემოთ ეს ახსნილია უფრო დეტალურად.

საღებავების გადატანა იატაკსა და ხარაჩოს შორის

ჩვენ განვიხილავთ გამარტივებულ სცენარს. წარმოიდგინეთ, რომ არის N ცალი საღებავი, რომლებიც დანომრილია 0–დან N-მდე და ლეონარდო ყოველდღე თავის თანაშემწეს საღებავის მიტანას სთხოვს N-ჯერ. ვთქვათ C არის საღებავების მიმდევრობა, რომლებიც მოითხოვა ლეონარდომ. C შეგვიძლია განვიხილოთ როგორც N ცალი რიცხვი (0-დან N-1-ის ჩათვლით), შევნიშნავთ, რომ ზოგიერთი საღებავი შეიძლება არ შედიოდეს C -ში, ხოლო ზოგიერთი შეიძლება რამდენჯერმე შედიოდეს.

ხარაჩო მუდამ სავსეა და შეიცავს ზუსტად K ცალ საღებავს, ამასთან K < N. თავიდან ხარაჩო შეიცავს 0-დან K - 1-მდე ჩათვლით საღებავებს.

თანშემწე ლეონარდოს მოთხოვნებს ასრულებს სათითაოდ. თუკი საღებავი, რომელიც მოითხოვა ლეონარდომ არის ხარაჩოზე, მას შეუძლია დაისვენოს. წინააღმდეგ შემთხვევაში მან უნდა მიუტანოს ლეონარდოს მოთხოვნილი საღებავი. თუ არ არის ადგილი ხარაჩოზე, მან უნდა აიღოს ნებისმიერი

საღებავი და დააბრუნოს იატაკზე, ხოლო მის ადგილზე დადოს მის მიერ მიტანილი საღებავი.

ლეონარდოს ოპტიმალური სტრატეგია

თანაშემწეს უნდა რაც შეძლება ბევრჯერ დაისვენოს. ლეონარდოს მოთხოვნების რაოდენობა, რომლის დროსაც თანაშემწეს შეუძლია დაისვენოს, დამოკიდებულია თვითონ თანაშემწის მოქმედებაზე. ლეონარდომ აუხსნა, რომ მას შეუძლია ამის მიღწევა, რადგან წინასწარ იცის მოთხოვნათა C მიმდევრობა. თუკი საღებავი, რომელიც მოითხოვა ლეონარდომ არ არის ხარაჩოზე და რადგან ამ შემთხვევაში ის ვერ დაისვენებს, უკეთესი იქნება შემდეგი ასეთი მომენტი რაც შეიძლება გვიან დადგეს. ამისათვის საჭიროა მან გამოიკვლიოს:

- მიმდინარე საღებავების მიმდევრობა,რომელიც არის ხარაჩოზე და
- C მიმდევრობაში დარჩენილი მოთხოვნები.

საღებავი ხარაჩოდან ასაღებად ამოირჩევა, თუკი მხატვარი საერთოდ არ ითხოვს მას დარჩენილ მოთხოვნებში, ანდა მოითხოვება ხარაჩოზე არსებულ საღებავებს შორის ყველაზე გვიან.

მაგალითი 1

ვთქვათ, N = 4, ე,ი. გვაქვს 4 საღებავი (გადანომრილი 0-დან 3-ის ჩათვლით) და 4 მოთხოვნა. ვთქვათ მოთხოვნათა მიმდევრობა C = (2, 0, 3, 0). აგრეთვე, K = 2. ეს ნიშნავს, რომ ლეონარდოს აქვს ხარაჩო, რომელიც იტევს 2 საღებავს. როგორც ზემოთ ითქვა, თავიდან ხარაჩოზე იქნება 0 და 1 საღებავები. ხარაჩოზე მყოფი საღებავები ჩავწეროთ შემდეგი სახით: [0, 1]. თანაშემწეს შეუძლია დაამუშაოს მოთხოვნები შემდეგნაირად:

- პირველად მოთხოვნილი საღებავი (ნომერი 2) არ არის ხარაჩოზე. თანაშემწე მიიტანს მას და გადაწყვეტს აიღოს 1. ხარაჩოზე საღებავების ნაკრები ასეთი იქნება [0, 2].

- შემდეგი მოთხოვნილი საღებავი (ნომერი 0) უკვე არის ხარაჩოზე, თანაშემწეს შეუძლია დაისვენოს.

- იმისათვის, რომ თანაშემწემ შეასრულოს შემდეგი მოთხოვნა (ნომერი 3), იგი იღებს 0 საღებავს და ხარაჩოზე საღებავების ნაკრები გახდება [3, 2].

- და ბოლოს მოთხოვნილი საღებავი (ნომერი 0) რადგან არ არის ხარაჩოზე, მან უნდა აიღოს იგი იატაკიდან და ხარაჩოზე მიიტანოს, ხოლო ხარაჩოდან აიღებს 2 ნომერ საღებავს. ხარაჩოზე ნაკრები გახდება [3, 0].

შევნიშნოთ, რომ თანაშემწე არ მიჰყვა ლეონარდოს ოპტიმალურ სტრატეგიას. ოპტიმალური იქნებოდა, თუკი მესამე მოთხოვნის შესრულებისას აიღებდა ნომერ 2 საღებავს, ამით თანაშემწე დაისვენებდა ბოლო ბიჯზე.

თანაშემწის სტრატეგია, როცა მისი მეხსიერება შეზღუდულია

დილით თანაშემწემ სთხოვა ლეონარდოს ქაღალდზე დაეწერა C მითხოვნათა მიმდევრობა, რომ ეპოვნა ოპტიმალური მიმდევრობა და მიჰყოლოდა მას, მაგრამ ლეონარდოს სურს საიდუმლოდ შეინახოს თავისი სამუშაო ტექნიკა და ამიტომ უარი უთხრა ჩანაწერის გადაცემაზე, მხოლოდ ნება დართო წაეკითხა ქაღალდზე დაწერილი მოთხოვნათა მიმდევრობა C და დაემახსოვრებინა იგი.

საუბედუროდ თანაშემწეს ცუდი მეხსიერება აქვს, მას შეუძლია დაიმახსოვროს ინფორმაციის არაუმეტეს M ბიტისა. ამის გამო მას ყოველთვის არ შეუძლია მოთხოვნათა მიმდევრობის მთლიანი დამახსოვრება. ამიტომ მან უნდა მოიფიქროს რაიმე ჭკვიანური გზა ბიტების იმ მიმდევრობის გამოსათვლელად, რომელსაც ის დაიხსომებს. ვუწოდოთ ამ მიმდევრობას *რეკომენდაცია* და აღვნიშნოთ A-თი.

მაგალითი 2

დილით თანაშემწე იღებს ლეონარდოს მოთხოვნათა C მიმდევრობას, კითხულობს მას და აკეთებს საჭირო გათვლებს. მაგალითად, მას შეუძლია თვალყური მიადევნოს ხარაჩოზე საღებავების მდგომარეობას ყოველი მოთხოვნის შემდეგ, მაგალითი 1-ის სტრატეგიის (არაოპტიმალური) შემთხვევაში ხარაჩოს მდგომარეობათა მიმდევრობა იქნება [0, 2], [0, 2], [3, 2], [3, 0]. (შევნიშნოთ, რომ ხარაჩოს საწყისი მდგომარეობა მან იცის [0, 1].)

დავუშვათ M = 16. ეს ნიშნავს, რომ მას შეუძლია 16 ბიტი ინფორმაციის დამახსოვრება.რადგანაც N = 4, საღებავის ნომრის დამახსოვრებისათვის საჭიროა 2 ბიტი. ამის გამო 16 ბიტი საკმარისია ზემოთ აღწერილი ხარაჩოს მდგომარეობათა მიმდევრობის დამახსოვრებისათვის. მაშასადამე, თანაშემწე გამოთვლის შემდეგ რეკომენდაციას : A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

დღის განმავლობაში თანაშემწეს შეუძლია რეკომენდაციის გაშიფვრით გააკეთოს ასაღები საღებავების არჩევანი C მიმდევრობის არ ცოდნის შემთხვევაშიც კი.

(რა თქმა უნდა, M = 16-თვის თანაშემწეს შეუძლია უბრალოდ დაიხსომოს მიმდევრობა C და გამოიყენოს მხოლოდ 8 ბიტი შესაძლებელი 16–დან. ამ მაგალითით ჩვენ გვინდოდა გვეჩვენებინა, რომ მას შეიძლება ჰქონდეს სხვა ვარიანტებიც, რაც არ მოგვცემს რაიმე კარგ შედეგს).

ამოცანის დასმა

თქვენ უნდა დაწეროთ *ორი სხვადასხვა პროგრამა* დაპროგრამების ერთი და იგივე ენაზე. ეს პროგრამები გაშვებული იქნება რიგ-რიგობით და შესრულების დროს არ ექნებათ ერთმანეთთან კავშირის საშუალება.

პირველი პროგრამა გამოყენებული იქნება თანაშემწის მიერ დილით. ეს პროგრამა მიიღებს მიმდევრობა C-ს და გამოთვლის A რეკომენდაციას.

მეორე პროგრამა გამოყენებული იქნება თანაშემწის მიერ დღის განმავლობაში. ეს პროგრამა მიიღებს A რეკომენდაციას და დაამუშავებს ლეონარდოს მოთხოვნათა მიმდევრობა C-ს. შევნიშნოთ, რომ C მიმდევრობის ელემენტები გადაეცემა პროგრამას სათითაოდ და ყოველი მოთხოვნა დამუშავებული უნდა იქნეს მომდევნო ელემენტის მიღებამდე.

უფრო ზუსტად, თქვენმა პირველმა პროგრამამ რეალიზება უნდა გაუკეთოს `ComputeAdvice`(C, N, K, M) პროცედურას, რომელიც არგუმენტებად მიიღებს C-ს, ანუ N ცალ მთელ რიცხვს (ყოველი მათგანი 0, …, N - 1 შუალედშია), ხარაჩოზე საღებავების K რაოდენობას, ბიტების M რაოდენობას, რომელიც შეუძლია დაიხსომოს თანაშემწემ. პროგრამამ უნდა გამოთვალოს A რეკომენდაცია, რომელიც შედგება არაუმეტეს M ბიტისა. პროგრამამ A მიმდევრობა სისტემას უნდა გადასცეს შემდეგი პროცედურის თანმიმდევრობითი გამოძახებით A-ს ყოველი ბიტისათვის:

- `WriteAdvice`(B) — დავუმატოთ მიმდინარე A მიმდევრობის ბოლოში ბიტი B (ეს პროცედურა შეგიძლიათ გამოიძახოთ არა უმეტეს M -ჯერ).

მეორე პროგრამაში თქვენ რეალიზება უნდა გაუკეთოთ `Assist`(A, N, K, R) პროცედურას. ამ პროცედურას გადაეცემა A რეკომენდაცია, ზემოთ აღწერილი მთელი რიცხვები N და K და A რეკომენდაციის R სიგრძე ბიტებში ($R \leq M$). ეს ფუნქცია უნდა ახორციელებდეს თქვენს მიერ თანაშემწისათვის შეთავაზებულ სტრატეგიას და იყენებდეს შემდეგ ფუნქციებს:

- `GetRequest`() —აბრუნებს ლეონარდოს მიერ მოთხოვნილ მორიგ საღებავს (ინფორმაცია სამომავლო მოთხოვნებზე არ მჟაღავნდება.)

- `PutBack`(T) — გადაიტანს T საღებავს ხარაჩოდან უკან – იატაკზე. ამ ფუნქციის გამოძახება მხოლოდ მაშინ არის შესაძლებელი, თუ T საღებავი ხარაჩოზე იმყოფება.

თქვენმა პროცედურამ `Assist` უნდა გამოიძახოს პროცედურა `GetRequest` ზუსტად N-ჯერ და და ყოველი გამოძახებისას უნდა მიიღოს ლეონარდოს რიგით მომდევნო მოთხოვნა. `GetRequest` პროცედურის ყოველი გამოძახების შემდეგ, თუ ლეონარდოს მიერ მოთხოვნილი საღებავი ხარაჩოზე არ იმყოფება, მაშინ გამოძახებული უნდა იქნას `PutBack`(T) პროცედურა თქვენს მიერ შერჩეული T-თი. წინააღმდეგ შემთხვევაში პროცედურა `PutBack` არ უნდა გამოიძახოთ. თუ ამას ისე არ გააკეთებთ, როგორც აღწერილია, მაშინ ეს შეფასებული იქნება, როგორც შეცდომა და თქვენი პროგრამის მუშაობის დამთავრებას გამოიწვევს. ყურადღება მიაქციეთ იმ ფაქტს, რომ თავდაპირველად ხარაჩოზე აწყვია საღებავები ნომრებით 0-დან (K - 1)-მდე (ჩათვლით).

ტესტი წარმატებით გავლილად ჩაითვლება, თუ თქვენმა ორივე პროცედურამ დააკმაყოფილა ზემოთ აღწერილი შეზღუდვები და `PutBack` პროცედურის გამოძახებათა რაოდენობა ზუსტად იმდენია, რამდენიც

საჭიროა ლეონარდოს მიერ აღწერილი ოპტიმალური სტრატეგიის გამოყენების დროს. თუ ასეთი ოპტიმალური სტრატეგია რამდენიმეა, თქვენმა პროგრამამ შეიძლება გამოიყენოს ნებისმიერი მათგანი. ანუ, სავალდებულოა არაა ზუსტად მიყვეთ ლეონარდოს სტრატეგიას, თუკი არსებობს სხვა ოპტიმალური სტრატეგია.

მაგალითი 3

ისევ განვიხილოთ მე-2 მაგალითი. დავუშვათ, რომ თქვენმა პროცედურამ `ComputeAdvice` დააბრუნა A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0). სისტემასთან ურთიერთქმედებისათვის თქვენ უნდა გააკეთოთ გამოძახებათა შემდეგი მიმდევრობა: `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)`.

თქვენი მეორე პროცედურა `Assist` გაშვებისას მიიღებს ზემოთაღწერილ A მიმდევრობას და მნიშვნელობებს: N = 4, K = 2 და R = 16. პროცედურამ `Assist` უნდა მოახდინოს `GetRequest` პროცედურის ზუსტად N = 4 გამოძახება. გარდა ამისა, თქვენ გჭირდებათ `PutBack`(T) პროცედურის `Assist` პროცედურით გამოძახება T-ს შესაბამისი მნიშვნელობების შერჩევით.

ქვემოთ მოყვანილ ცხრილში ნაჩვენებია გამოძახებათა მიმდევრობა, რომლებიც შეესაბამება თანაშემწის არაოპტიმალურ არჩევებს 1-ლ მაგალითში. ტირე აღნიშნავს, რომ `PutBack` პროცედურის გამოძახება არ მოხდა.

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

ქვეამოცანა 1 [8 ქულა]

- $N \leq 5\,000$.
- თქვენ შეგიძლიათ გამოიყენოთ არაუმეტეს M = 65 000 ბიტისა.

ქვეამოცანა 2 [9 ქულა]

- $N \leq 100\,000$.
- თქვენ შეგიძლიათ გამოიყენოთ არაუმეტეს M = 2 000 000 ბიტისა.

ქვეამოცანა 3 [9 ქულა]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- თქვენ შეგიძლიათ გამოიყენოთ არაუმეტეს $M = 1\,500\,000$ ბიტისა.

ქვეამოცანა 4 [35 ქულა]

- $N \leq 5\,000$.
- თქვენ შეგიძლიათ გამოიყენოთ არაუმეტეს $M = 10\,000$ ბიტისა.

ქვეამოცანა 5 [39 ქულამდე]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- თქვენ შეგიძლიათ გამოიყენოთ არაუმეტეს $M = 1\,800\,000$ ბიტისა.

ქულები ამ ქვეამოცანაში დამოკიდებულია თქვენი პროგრამის მიერ შედგენილი რეკომენდაციის R სიგრძეზე. კერძოდ, თუ R_{max} არის მაქსიმალური (ყველა ტესტში) სიგრძე `ComputeAdvice` ფუნქციის მიერ შექმნილი რეკომენდაციისა, თქვენი ქულა ტოლი იქნება

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

რეალიზაციის დეტალები

თქვენ შესამოწმებლად უნდა გაგზავნოთ ზუსტად ორი ფაილი, რომლებიც შეიცავენ ზემოთ აღწერილ ფუნქციებს. ორივე ფაილი დაწერილი უნდა იყოს პროგრამირების ერთსა და იმავე ენაზე.

პირველ ფაილს უნდა ერქვას `advisor.c`, `advisor.cpp` ან `advisor.pas`. ეს ფაილი უნდა შეიცავდეს `ComputeAdvice` ფუნქციის რეალიზაციას და უნდა იძახებდეს `WriteAdvice` ფუნქციას. მეორე ფაილს უნდა ერქვას `assistant.c`, `assistant.cpp` ან `assistant.pas`. ეს ფაილი უნდა შეიცავდეს `Assist` აფუნქციის რეალიზაციას და უნდა იძახებდეს `GetRequest` და `PutBack` ფუნქციებს.

ქვემოთ მოცემულია საჭირო პროცედურის (ფუნქციის) პროტოტიპი

C/C++ პროგრამებისათვის

```
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 პროგრამებისათვის

```
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;
```

ეს ფუნქციები უნდა მოქმედებდნენ ისე, როგორც ეს ზემოთაა აღწერილი. ცხადია, თქვენ შეგიძლიათ გამოიყენოთ სხვა ფუნქციები შიგა მოხმარებისათვის. C/C++ პროგრამებისათვის შიგა ფუნქციები გამოცხადებული უნდა იყვნენ, როგორც `static`, რადგან ისინი ერთ ფაილში იქნებიან კომპილირებული. ამით თქვენ მარტივად აიცილებთ თავიდან ფუნქციათა ერთნაირი სახელების პრობლემას. თქვენი პროგრამა არ უნდა ურთიერთქმედებდეს სხვა ფაილებთან და სტანდარტულ შეტანა/გამოტანასთან.

თქვენი ამოხსნის პროგრამირებისას უნდა შეასრულოთ შემდეგი ინსტრუქციები (შაბლონები, რომლებსაც იპოვით თქვენს კომპიუტერზე, უკვე აკმაყოფილებენ ქვემოთ მითითებულ მოთხოვნებს):

C/C++ პროგრამებისათვის

ქვენი ამოხსნის დასაწყისში უნდა ჩაურთოთ ფაილები `advisor.h` და `assistant.h`, შესაბამისად advisor–სა და assistant–ში. ეს განხორციელდება თქვენს კოდში შემდეგი სტრიქონების დამატებით:

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

ან

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

ორი ფაილი – `advisor.h` და `assistant.h` ჩაწერილი იქნება თქვენი კომპიუტერის სამუშაო გარემოში. ეს ფაილები თქვენ აგრეთვე შეიძლება ჩამოტვირთოთ ვებ–ინტერფეისიდან. სკრიპტები თქვენი ამოხსნის კომპილაციისა და გაშვებისათვის თქვენთვის ანალოგიური გზით იქნება ხელმისაწვდომი. კერძოდ, თქვენ უნდა გაუშვათ `compile_c.sh` ან

`compile_cpp.sh` (გამოყენებული ენის მიხედვით) იმ ფოლდერში, სადაც თქვენი ამოხსნაა მოთავსებული.

Pascal პროგრამებისათვის

თქვენი ამოხსნის დასაწყისში უნდა ჩაურთოთ მოდულები `advisorlib` და `assistantlib`, შესაბამისად advisor–სა და assistant–ში. ეს განხორციელდება თქვენს კოდში შემდეგი სტრიქონების დამატებით:

```
uses advisorlib;
```

ან

```
uses assistantlib;
```

ორი ფაილი – `advisorlib.pas` და `assistantlib.pas` ჩაწერილი იქნება თქვენი კომპიუტერის სამუშაო გარემოში. ეს ფაილები თქვენ აგრეთვე შეიძლება ჩამოტვირთოთ ვებ–ინტერფეისიდან. სკრიპტები თქვენი ამოხსნის კომპილაციისა და გაშვებისათვის თქვენთვის ანალოგიური გზით იქნება ხელმისაწვდომი. კერძოდ, თქვენ უნდა გაუშვათ `compile_pas.sh` იმ ფოლდერში, სადაც თქვენი ამოხსნაა მოთავსებული.

შემმოწმებელი მოდულის (grader) მაგალითი

შემმოწმებელი მოდული მონაცემებს მიიღებს შემდეგი ფორმატით:

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

შემმოწმებელი მოდული თავდაპირველად გამოიძახებს ფუნქციას `ComputeAdvice`. ამის შედეგად გენერირდება ფაილი `advice.txt`, რომელშიც ჩაწერილი იქნება ჰარით გაყოფილი რეკომენდაციის ბიტები. ფაილი დამთავრდება ციფრით 2.

ამის შემდეგ ამუშავდება თქვენი ფუნქცია `Assist`, რომელიც მოახდენს გამოსატანი მონაცემების გენერაციას. გამოსატანი მონაცემების ყოველი სტრიქონი იქნება ან `"R [number]"` ფორმატის, ან – `"P [number]"` ფორმატის. პირველი ტიპის სტრიქონები განსაზღვრავენ `GetRequest()` ფუნქციის გამოძახებას და პასუხების მიღებას. მეორე ტიპის სტრიქონები განსაზღვრავენ `PutBack()` ფუნქციის გამოძახებას და საღებავებს, რომელთა აღებაც გადაწყვიტეთ ხარაჩოდან. გამოტანა დასრულდება `"E"` სიმბოლოთი.

მიაქციეთ ყურადღება, რომ ოფიციალური შემმოწმებელი მოდულის და თქვენი ლოკალური კომპიუტერის მუშაობის დრო შეიძლება უმნიშვნელოდ განსხვავდებოდეს. თქვენ შეგიძლიათ გამოიყენოთ სერვერზე ტესტირების ინტერფეისი, რათა თქვენი კოდი გაუშვათ ოფიციალურ შემმოწმებელ მოდულზე.

დროის და მეხსიერების ლიმიტი

- დროის ლიმიტი: 7 წამი.
- მეხსიერების ლიმიტი: 256 მბ.