

Task C23. მონოპოლია 2

⌚ 1 sec. 📦 256 MB

მას შემდეგ, რაც დენის 2021 წელს მონოპოლისთვის იდეალური სათამაშო რუკა შეუქმენით, მას კიდევ ერთხელ სჭირდება თქვენი დახმარება! მონოპოლის სათამაშო რუკა შედგება N ცალი სივრცისგან (გადანომრილია 1-დან N -მდე) და მათ შორის M ცალი მიმართული (ორიენტირებული) კავშირისგან, სადაც $M = N - 1$. ეს მიმართული კავშირები აკმაყოფილებს შემდეგ პირობებს: არცერთი მათგანი არ აკავშირებს სივრცეს საკუთარ თავთან და სივრცეების არცერთ წყვილს შორის არ არის ერთზე მეტი კავშირი. ასევე, კავშირებს აქვთ თვისება, რომ სივრცე 1-დან ყველა სხვა სივრცემდე მიღწევა შესაძლებელია.

განმარტება: გვაქვს 1-დან N -მდე რიცხვების პერმუტაცია. ასეთ პერმუტაციას ვუწოდებთ სივრცეების *სწორ* განლაგებას იმ შემთხვევაში, თუ შემდეგი პირობა სრულდება: ყველა კავშირისთვის სივრცე i -დან სივრცე j -მდე, პერმუტაციაში i აუცილებლად უფრო ადრე უნდა გვხვდებოდეს, ვიდრე j .

დენის პრობლემა ის არის, რომ მან მონოპოლის სათამაშო რუკა დაკარგა. საბედნიეროდ, მისმა მეგობარმა ბობიმ რუკა დაიმახსოვრა, მაგრამ მან გართობა გადაწყვიტა, სანამ დენის კავშირებს გაუმხელდა. თავდაპირველად, ბობი ეუბნება დენის, რომ ერთ-ერთი *სწორი* პერმუტაცია $1, 2, \dots, N$ არის. შემდეგ, ბობი უპასუხებს დენის კითხვებს იმის შესახებ, თუ რამდენად *სწორია* რომელიმე მოცემული პერმუტაცია (დამატებითი დეტალებისთვის იხილეთ იმპლემენტაციის სექცია). ჩვენს გმირს სჭირდება თქვენი დახმარება, რათა კავშირების პოვნისთვის შეიმუშავოს სტრატეგია მინიმალური რაოდენობის კითხვებით.

ამოცანა

დაწერეთ პროგრამა **monopoly2**, რომელიც ბობისგან უცნობ კავშირებს მინიმალური რაოდენობის კითხვებით დაადგენს. პროგრამა უნდა შეიცავდეს ფუნქციას `find_connections`, რომელიც ჟიურის პროგრამასთან ერთად დაკომპილირდება (ბობის როლის შესასრულებლად).

იმპლემენტაცია

თქვენს ფუნქცია `find_connections`-ს შემდეგი ფორმატი უნდა ჰქონდეს:

```
std::vector<std::pair<int, int>> find_connections (int N);
```

ჟიურის პროგრამა ამ ფუნქციას ერთხელ გამოიძახებს ერთი პარამეტრით – სივრცეების რაოდენობით. ფუნქციამ უნდა დააბრუნოს წყვილთა სია, რომლებიც აღწერს ნაპოვნ კავშირებს სივრცეებს შორის. წყვილების მიმდევრობას მნიშვნელობა არ აქვს.

ფუნქციას, რომელიც ბობის კითხვებს უსვამს, შემდეგი ფორმატი აქვს:

```
std::vector<bool> check (std::vector<int> p);
```

პარამეტრი p არის რიცხვების პერმუტაცია 1-დან N -მდე, რომელიც წარმოადგენს შეკითხვის პერმუტაციას. ფუნქცია აბრუნებს `bool` ტიპის ვექტორს b სიგრძით N , სადაც $b_i = 1$ ($0 \leq i < N$) მხოლოდ იმ შემთხვევაში, თუ შემდეგიდან ერთ-ერთი პირობა სრულდება:

- არსებობს კავშირი სივრცე p_j -დან სივრცე p_i -მდე, სადაც $i < j$;
- არსებობს სივრცე p_j , რომლისთვისაც $b_j = 1$ და კავშირების საშუალებით p_j -დან

p_i -მდე მისვლა შეგიძლია.

გაითვალისწინეთ, რომ მნიშვნელობების ერთმნიშვნელოვნობა გარანტირებულია კავშირების თვისებების გამო.

თუ ფუნქციისათვის გადაცემული პერმუტაცია არ არის ვალიდური პერმუტაცია 1-დან N -მდე, ტესტისათვის მიიღებთ პასუხს Wrong answer. ფუნქცია იყენებს $O(N)$ დროს. ფუნქციის გამოძახება შეგიძლიათ მაქსიმუმ $\frac{10^8}{N}$ -ჯერ, სხვა შემთხვევაში მიიღებთ ვერდიქტს Wrong answer.

თქვენს პროგრამაში **monopoly2** იმპლემენტირებული უნდა იყოს ფუნქცია `find_connections`. ასევე, პროგრამა შეიძლება შეიცავდეს სხვა კოდს, ფუნქციებს ან გლობალურ ცვლადებს, მაგრამ არ უნდა შეიცავდეს `main` ფუნქციას. ასევე არ უნდა გამოიყენოთ სტანდარტული შეტანის და გამოტანის ოპერაციები. თქვენს პროგრამაში აუცილებლად უნდა შედიოდეს ჰედერის ფაილი `monopoly2.h`, შემდეგი პრეპროცესორის ინსტრუქციის საშუალებით:

```
#include "monopoly2.h"
```

შეზღუდვები

- ♣ $1 \leq N \leq 1\,000$;
- ♣ $M = N - 1$.

ქვეამოცანები

XVI INTERNATIONAL ADVANCED TOURNAMENT IN INFORMATICS
SHUMEN 2025

ქვეამოცანა	ქულები	საჭირო ქვეამოცანები	N	სხვა შეზღუდვები
0	0	—	—	რუკა მაგალითიდან.
1	5	—	$\leq 1\,000$	1, 2, ..., N ერთადერთი სწორი სივრცეების განლაგებაა.
2	6	0	≤ 6	—
3	17	1	$\leq 1\,000$	1, 2, ..., N -ის მიღება ასევე შეიძლება შემდეგნაირად: ვიწყებთ სივრცე 1-დან, შემდეგ ვწერთ ყველა იმ სივრცეს, რომელსაც აქვს პირდაპირი კავშირი სივრცე 1-თან, შემდეგ ვწერთ ყველა იმ სივრცეს, რომელსაც აქვს პირდაპირი კავშირი მეორე დამატებულ სივრცესთან და ასე შემდეგ.
4	13	0, 2	≤ 300	—
5	59	0 – 4	$\leq 1\,000$	თუ დავიწყებთ სივრცე 1-დან და კავშირებს გავეყვებით, 25-ზე მეტ კავშირს ვერ გამოვიყენებთ.

ქვეამოცანა სწორად შესრულებულად ითვლება მხოლოდ იმ შემთხვევაში, თუ თქვენი პროგრამა სწორად მუშაობს ამ ქვეამოცანის ყველა ტესტზე და ყველა საჭირო წინა ქვეამოცანაზე, მიღებული ქულა კი უდრის მინიმალური ტესტის ქულის (მის და საჭირო ქვეამოცანების ტესტებში) ნამრავლს ქვეამოცანის ქულაზე.

შეფასება

ყველა ტესტს აქვს ქულა, რომელიც არის ათწილადი რიცხვი 0-დან 1-ის ჩათვლით. თუ ტესტს აქვს დადებითი ქულა, იგი ითვლება წარმატებულად. ტესტის ქულა არის დადებითი, თუ თქვენი პროგრამა წარმატებით აღმოაჩენს კავშირებს სივრცეებს შორის.

თუ cnt არის ფუნქცია $check$ -ის გამოძახების რაოდენობა კონკრეტული ტესტისთვის, მაშინ ტესტის ქულა ითვლება შემდეგნაირად:

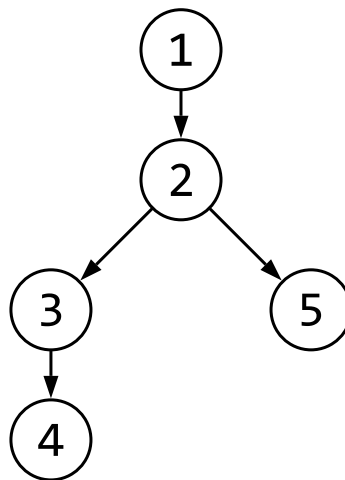
- ყველა ქვეამოცანისთვის: თუ $cnt > \frac{10^8}{N}$, მიღებული ქულა უდრის 0-ს.
- ქვეამოცანები 0, 1, and 2.
 - თუ $cnt \leq \frac{10^8}{N}$, მიღებული ქულა უდრის 1-ს.
- ქვეამოცანა 3.
 - თუ $cnt \leq 10\,000$, მიღებული ქულა უდრის 1-ს.
 - თუ $10\,000 < cnt \leq 15\,000$, მიღებული ქულა უდრის $\min(\frac{1000}{cnt-9000}, 1)$ -ს.
 - თუ $15\,000 < cnt \leq \frac{10^8}{N}$, მიღებული ქულა უდრის 0.1-ს.
- ქვეამოცანა 4.
 - თუ $cnt \leq 50\,000$, მიღებული ქულა უდრის 1-ს.
 - თუ $50\,000 < cnt \leq 200\,000$, მიღებული ქულა უდრის $\min(\frac{25000}{cnt-25000}, 1)$ -ს.
 - თუ $200\,000 < cnt \leq \frac{10^8}{N}$, მიღებული ქულა უდრის 0.1-ს.
- ქვეამოცანა 5.

XVI INTERNATIONAL ADVANCED TOURNAMENT IN INFORMATICS SHUMEN 2025

- თუ $cnt \leq 170$, მიღებული ქულა უდრის 1-ს.
- თუ $170 < cnt \leq 1000$, მიღებული ქულა უდრის $\min((\frac{170+3000}{cnt+3000})^{\frac{3000}{x}}, 1)$ -ს.
- If $1000 < cnt \leq 20\,000$, მიღებული ქულა უდრის $\min((\frac{170}{cnt})^{0.4}, 0.5)$ -ს.
- თუ $20\,000 < cnt \leq \frac{10^8}{N}$, მიღებული ქულა უდრის 0.1-ს.

მაგალითი

გვაქვს რუკის შემდეგი ილუსტრაცია 5 სივრცით და 4 კავშირით:



თქვენი პროგრამის მოქმედებები	ჟიურის მოქმედებები და პასუხები
	find_connections(5)
check({2, 3, 4, 5, 1})	return {1, 1, 1, 1, 0}
check({1, 5, 2, 3, 4})	return {0, 1, 0, 0, 0}
check({1, 4, 3, 2, 5})	return {0, 1, 1, 0, 0}
return {{1, 2}, {2, 3}, {3, 4}, {2, 5}}	

ლოკალური ტესტირება

ლოკალური ტესტირებისთვის შეგიძლიათ გამოიყენოთ შემდეგი ფაილები: monopoly2.h, Lgrader.cpp, სანიმუშო ფაილი monopoly2.cpp თქვენი პროგრამისთვის და ფაილი სანიმუშო კომუნიკაციის რუკით. როდესაც ეს ფაილები ერთსა და იმავე ფოლდერშია, შეგიძლიათ ერთად დააკომპილიროთ თქვენი პროგრამა monopoly2.cpp და Lgrader.cpp. ეს შექმნის პროგრამას, რომელიც თქვენი ფუნქციის სისწორეს შეამოწმებს.

პროგრამას დასჭირდება შემდეგი რიცხვების წაკითხვა სტანდარტული შეტანიდან:

- პირველ ხაზზე: ერთი მთელი დადებითი რიცხვი N – სივრცეების რაოდენობა;
- შემდეგ $N-1$ ხაზზე: ორი დადებითი მთელი რიცხვი, რომლებიც მიმართულ კავშირს აღწერს.

თუ კომუნიკაციის პროტოკოლს არ მიჰყვებით, მიიღებთ შესაბამის შეცდომის შეტყობინებას. სხვა შემთხვევაში, თუ პროგრამა წარმატებით შესრულდება, მიიღებთ შეტყობინებას "Correctly found connections."