**Теория на игрите**

Ние всички обичаме да играем. Особено игри, в които може да се победи. И сме най-доволни, когато сме победители. Но в много от игрите победата се постига с предварително обмисляне начина на игра, т.е. трябва да имаме ***стратегия за победа***.

1. **Алгоритмични игри**

***Задача 1. Пулове в кръг***

Играят двама, всеки по ред. Има N пула, подредени в кръг. Да се направи ход, означава да се вземат един или два съседни пула от няколко, които са разположени един до друг по кръга. Победител е онзи, който съумее да вземе последния пул.

***Решение:***

След какъвто и да е първи ход кръгът от N пула ще се окаже разкъсан и ще остане верижка от N1= N – P1 пула.

Ако N1 е нечетно, то вторият играч трябва да вземе средния пул, образувайки две къси верижки с еднакъв брой пулове N2= (N1-1)/2.

Ако N1 е четно, то вторият играч трябва да вземе двата средни пула, образувайки две къси верижки с еднакъв брой пулове N2= N3= (N1-2)/2.

По-нататък вторият играч действа, като при изпълнение на всеки ход се обръща към симетричната верижка и взима от нея толкова пула и така разположени, както е направил това съперникът му в симетричната верижка в предишния ход (запазва разположението на пуловете с централна симетрия).

**Победител е** **винаги** играчът, който влиза в играта втори.

**Въпроси:**

1. Ще се промени ли негубещата стратегия, ако при един ход могат да се взимат 1, 2 или 3 съседни пула?

**Отговор:** Не

1. Изследвайте играта двойник (победител е онзи, който успее да накара съперникът да вземе последния пул).

**Отговор**:

* 1. Съперникът започва първи
  2. Ответният ход е: вземете един или два пула, така че да се образуват две верижки и количеството на пуловете в едната от тях да бъде с едно повече от количеството на пуловете в другата верижка.
  3. Вторият играч действа по симетрия.

***Задача 2. Фомина-1***

Има N групи, съдържащи съответно m1, m2, …, mk предмета. Играят двама. Задачата на всеки от играчите е при всеки пореден ход да разбие **една произволна** група на две по-малки. Ходовете се изпълняват поредно дотогава, докато във всички групи не остане по един предмет. Победител е онзи, който съумее да изпълни последния ход.

***Решение:***

Началната ситуация може да се разглежда като ситуация, възникнала от разбиването на по-рано съществуваща група, съдържаща М= предмета на N групи, ход след ход. Ясно е, че за образуването на N групи от една обща са необходими (N-1) разбивания. Тъй като целта на играта е образуването на M групи, всяка от които да съдържа по един предмет, то ще са необходими (М-1) хода. Следователно, ако вече има N групи и за тяхното образуване са използвани (N-1) хода, до края на играта остават да се направят още (М-1) – (N-1)=М-N хода. Нека R=M-N. Тогава:

1. Ако R е нечетно, в играта трябва да се влезна пръв, ако е четно - втори.
2. При всеки пореден вход разбивайте произволна група на две по-малки.

Броят на разбиванията е .

**Въпроси:**

1. Какъв ще бъде алгоритмът, ако в играта участват 3 души?

**Отговор**: Нека P=R % 3. Ако Р е 1, влизайте в играта пръв, ако Р е 2 – втори, ако Р е 0 – трети.

1. Изследвайте играта двойник (губи онзи, който изпълни последното разбиване).

**Отговор**: Ако R е нечетно – втори, иначе – първи в играта.

***Задача 3. Фомина-2***

Има N групи, съдържащи съответно m1, m2, …, mk предмета. Играят двама. Задачата на всеки от играчите е при всеки пореден ход да разбие **всяка** група, състояща се от повече от един предмет, на две по-малки. Ходовете се изпълняват поредно дотогава, докато във всички групи не остане по един предмет. Победител е онзи, който съумее да изпълни последния ход.

***Решение:***

Всички групи са разположени на разбиване на части едновременно. Следователно играта ще продължава дотогава, докато не завърши разбиването на най-голямата група и на образуващите се от нея групи, т.е. количеството групи не влияе на алгоритъма, а влияе количеството предмети в най-голямата група (М) при всяко разбиване. Да разгледаме дървото на играта в няколко примера:

1. М=3

2, 1

3

1, 1, 1

Следователно, започващият играч губи.

7

1, 6

2, 5

3, 4

1,1,5

1,2,4

1,3,3

1,1,1,4

1,1,2,3

1,2,2,2

1,1,1,4

1,1,2,3

1,1,1,1,3

1,1,1,2,2

1,1,1,1,1,2

1,1,1,1,1,1,1

1,1,1,1,3

1,1,1,2,2

1,1,1,1,1,2

1,1,1,1,1,1,1

1,1,1,1,1,2

1,1,1,1,1,1,1

1,1,1,1,1,1,1

1. М=7

Забелязва се, че при внимателна игра на втория играч, първият играч ще загуби (дебелите стрелки).

1. М=15… Започващият играч отново губи

6

1. М=6

3, 3

2, 4

1, 5

1,1,4

1,1,2,2

1,2,3

1,1,1,3

1,1,2,2

1,1,1,1,1,1

1,1,1,1,2

1,1,1,3

1,1,1,1,1,1

1,1,1,1,2

1,1,1,1,1,1,1

Първият играч печели!

**Хипотеза:** Ако М=2к-1, к=1, 2, 3, …, то започващият играта пръв губи.

**Доказателство (чрез математическа индукция):**

1. При к=2 и к=3 е вярно
2. Да допуснем, че е вярно и при к=р, т.е. при М=2р-1 първият играч губи.
3. Ще докажем, че при к=р+1 първият играч отново губи:

Мр+1=2р+1-1=2.2р-1=2.(2р-1)+1=2Мр+1

Оттук следва, че ако най-голямата група съдържа Мр+1 предмета, започващият играта не може за един ход да направи разбиване на тази група на две групи съдържащи по Мр предмета, или на две групи, от които едната съдържа Мр предмета, а другата – по-малък брой предмети. С други думи, първият играч не може да постави съперника пред необходимостта да разбие най-голямата група от (2р-1) предмета, т.е. той ще загуби.

Използвайки вече доказаната хипотеза, то може да се направи алгоритъм на играта. Трябва да се намери групата съдържаща най-голям брой предмети (М). Ако М=2к-1, то трябва да влезете в играта като втори играч. В противен случай трябва да направите ход, така че след него в най-голямата група предмети количеството да е от вида 2к-1, а всички останали групи могат да се разбият произволно.

***Задача 4. Кибритена кутия***

В началото на играта в кибритена кутия има някакво количество клечки. Двама играчи подред взимат от кутийката произволно количество клечки без да превишават половината от съдържащите се в нея. Губи онзи игра, който вземе последната клечка.

***Решение:***

Ако броят клечки в началото е М=2к-1=2(2к-1-1)+1, понеже М div 2=2к-1-1, то след взимането на клечки от първия играч в кутийката ще останат повече от (2к-1-1) клечки, но не повече от (2к-2) клечки. Тогава вторият играч взима толкова клечки, че в кутийката да останат (2к-1-1) клечки. Така алгоритмът продължава, докато останат 3 клечки и е на ред първия играч. Вторият играч печели.

Ако броят клечки в началото не е 2к-1, то първият играч взима толкова клечки, че да останат 2r-1 клечки и отново се прилага съшият алгоритъм.

Интересен е алгоритъма на **играта двойник** (победител е онзи, който вземе последната клечка). Нека се разгледат няколко частни случая:

n=1 – Първият играч печели (печеливша позиция)

n=2 – Вторият играч печели (губеща позиция)

n=3 - Първият играч печели (печеливша позиция), защото ще остави две клечки, т.е. ще прати вторият играч в губеща позиция

n=4 - Първият играч печели (печеливша позиция), защото ще остави две клечки, т.е. ще прати вторият играч в губеща позиция

n=5 – Вторият играч печели (губеща позиция), защото първият играч може да вземе 1 или 2 клечки, но и при двата случая изпраща втория играч в печеливша позиция (остават 4 или 3 клечки)

n=6 - Първият играч печели (печеливша позиция), защото ще остави пет клечки, т.е. ще прати вторият играч в губеща позиция

n=7, 8, 9, 10 - Първият играч печели (печеливша позиция), защото ще остави пет клечки, т.е. ще прати вторият играч в губеща позиция

n=11 - Вторият играч печели (губеща позиция)

и т.н.

Следователно, първият играч губи, когато броят на клечките е 2, 5, 11, 23, 47, …

Да намерим обща формула за този брой:

a1=2

ak=2ak-1+1

Следователно:

a2=22+1, a3=2(22+1)+1=23+3, a4=2(23+3)+1=24+7, a5=25+15, т.е.

ак=2к+2к-1-1=3.2к-1-1

Вече може да се напише и алгоритъма: Ако броят клечки в началото е 3.2к-1, то в играта се влиза втори и се взимат толкова клечки, че да останат 3.2r-1, иначе се започва първи и отново се прилага същата стъпка на взимане.

**Въпрос:**

Направете алгоритъм за играта двойник на играта Фомина-2.

Отговор: Същият алгоритъм като в горната игра, но се гледа в максималната група да остават 3.2к-1 предмета.

1. **Игри с печеливши и губещи позиции**

* Една позиция е **печеливша (означава се с 1)** за играча на ход, ако има поне един ход, който води от нея в губеща позиция, т.е. може да се прати съперника в губеща позиция.
* Една позиция е **губеща (означава се с 0)** за този, който е на ход, ако всички позиции, в които може да се отиде от нея се печеливши позиции.

pos[i]=(pos[k1] & pos[k2] & … & pos[kn]) ^ 1

означения:

i – текуща позиция

k1, k2, …, kn – позициите, в които може да се отиде от текущата позиция

***Печелившите позиции имат следните свойства:***

* Краят на играта е печеливша позиция
* За един ход не може да се попадне от една печеливша позиция в друга
* От всяка позиция, която не е печеливша, може с един ход да се попадне в печеливша позиция

***Примери за игри:***

1. Върху шахматна дъска в полето а1 е поставен топ. Всеки ход представлява придвижване на топа произволен брой полета надясно или нагоре. Печели онзи, който постави топа в полето h8.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8** |  |  |  |  |  |  |  |  |
| **7** |  |  |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |
| **1** | ♠ |  |  |  |  |  |  |  |
|  | **a** | **b** | **c** | **d** | **e** | **f** | **g** | **H** |

**Решение:**

В играта побеждава втория играч със следната стратегия: с всеки ход той трябва да се стреми да върне топа на диагонала а1-h8. Това е възможно, зашото първият играч всеки път трябва да направи ход, с който премества топа извън диагонала, а вторият играч има възможност да върне топа на този диагонал. Тъй като h8 лежи на въпросния диагонал, то вторият играч ще успее да постави топа там.

1. Върху шахматна дъска в полето а1 е поставен цар. Всеки ход представлява придвижване на царя едно поле надясно или нагоре, или по диагонал в посока „надясно-нагоре”. Печели онзи, който постави царя в полето h8.

**Решение:**

Печелившите позиции в тази игра са полетата с две четни координати. Ще проверим трите свойства за печеливши позиции. Първото свойство е очевидно. Второто следва от това, че всеки ход увеличава с 1 поне една от координатите. За да проверим третото свойство, нека царят е поле с координати, от които поне едно от числата е нечетно. Ако е нечетна първата координата, то придвижваме царя едно поле надясно; ако е нечетна втората координата, придвижваме царя едно поле нагоре; ако са нечетни и двете координати, придвижваме царя по диагонала. Понеже първоначалната позиции не е печеливша, играта може да спечели първия играч.

***Задача 1. НОИ1, 2009, Задача B3. КАМЪНИ***

Двама приятели играят следната игра. На масата има купчина, съдържаща M камъка в началото. Двамата се редуват да правят ходове, като играчът, който е на ход взема от купчината няколко камъка, спазвайки следните правила:

1. Ако количеството останали на масата камъни се дели на 3, то могат да бъдат взети 1 или 2 камъка;

2. Ако количеството останали на масата камъни при деление на 3 дава остатък 1, то могат да бъдат взети 1 или 3 камъка;

3. Ако количеството останали на масата камъни при деление на 3 дава остатък 2, то могат да бъдат взети 1, 2 или 3 камъка;

Печели играчът, който вземе последния камък.

Двамата приятели играят серия от N игри.

Напишете програма **stones**, която за всеки начален брой камъни в купчината определя кой от играчите (играещ първи или втори ход) може да спечели при правилна игра.

**Вход**

От първия ред на стандартния вход се въвежда едно цяло положително число N, задаващо броя на игрите в серията;

От втория ред се въвеждат N на брой цели положителни числа, разделени с интервали, задаващи началния брой камъни в купчината за всяка игра от серията.

**Изход**

На един ред на стандартния изход се извежда символен низ, съдържащ N символа, като в позиция с номер *i* от този низ се извежда 1, ако в игра с номер *i* от серията може да спечели играещият първи ход и 2, ако в игра с номер *i* от серията може да спечели играещият втори ход.

**Ограничения**

2 <= ***N*** <= 100, 1 <= ***М*** <= 1000000

**ПРИМЕР**

**Вход Изход**

3  121

2 3 5

**Обяснение на примера:**

В първата игра от серията първият играч печели, тъй като той направо може да вземе двата камъка от купчината.

***Решение:***

Да разгледаме няколко частни случая:

* + 1. Ако М=1, то М % 3=1, следователно може да се вземе 1 камък и първият играч печели. Това е печеливша позиция.
    2. Ако М=2, то М % 2=2, следователно може да се вземат 2 камъка и първият играч печели. Това е печеливша позиция.
    3. Ако М=3, то М % 3=0, следователно може да се вземат 1 или 2 камъка, което води до оставането на 2 или 1 камъка, съответно. Но позициите с М=1 и М=2 са печеливши, следователно позицията М=3 е губеща, т.е. първият играч губи.
    4. Ако М=4, то М % 3=1, т.е. може да се вземе 1 или 3 камъка, което води съответно до позиции с М=3 или М=1. Една от тези позиции е губеща (М=3), следователно позицията М=4 е печеливша за първия играч, защото ще вземе 1 камък и ще прати вторият играч в губеща позиция!

и т.н.

Ето таблица със печелившите и губещите позиции при първите няколко случаи:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |

Нека означим с pos[i] стойността на позиция I, т.е. ако има I предмета пред играча на ход. Тогава pos[i] е 0 или 1. Как се пресмята pos[i]?

Има няколко частни случая:

pos[1]=1, pos[2]=1, pos[3]=0

При i>3, формулите са следните:

1. pos[i]=(pos[i-1] & pos[i-3]) ^1, ако i % 3=1
2. pos[i]=(pos[i-1] & pos[i-2] & pos[i-3]) ^1, ако i % 3=2
3. pos[i]=(pos[i-1] & pos[i-2]) ^1, ако i % 3=0

Забелязва се, че ако позицията е кратна на 3, то тя е губеща, във всички останали случаи тя е печеливша. Това наблюдение (може ли да го докажете?) води до още по-просто решение:

pos[i]=0, ако i % 3=0, иначе pos[i]=1.

***Задача 2. ЗМС, 2004, Задача B3. ИГРА***

Еднакви бонбони са разделени на две купчинки, съдържащи съответно А и B броя. Двама души играят с редуващи се ходове игра, при която на всеки ход играчът трябва да избере една от купчинките и да я раздели на две части. Другата купчинка се отделя настрана и играта продължава с двете новополучени купчинки. Разбира се, новополучените купчинки може да не са с равен брой бонбони, но всяка от тях трябва да съдържа поне един бонбон.

Играта завършва, когато двете новополучени купчинки съдържат по един бонбон – не могат да се правят повече ходове. Печели този, който е направил последния ход.

Напишете програма GAME, която въвежда броя на бонбоните А и B на двете купчинки преди поредния ход на някой от играчите. Програмата трябва да определи, дали играчът, който е на ход винаги загубва играта, ако противникът играе оптимално (т.е. възможно най-добре за себе си, за да спечели) или може винаги да спечели – без значение как играе другия играч. Ако играчът, който е на ход ще загуби при оптимална игра на другия, изведете числото 0; ако обаче играчът, който е на ход, може да спечели, изведете броя бонбони C и D, които ще има в двете новополучени купчинки след неговия ход, така че този ход да му гарантира при подходящи негови ходове по-нататък победа в играта, без значение как ще играе противникът.

Според условията на играта числата C и D са по-големи или равни на 1 и C + D е равно или на A, или на B. Понеже ще са възможни няколко различни хода, от които се интересуваме, търсим тези от тях, за които C + D е възможно най-малко и от всички такива ходове искаме да изберем този, за който C е възможно най-малко.

Програмата чете входните данни от стандартния вход – на единствен ред са записани целите числа A и B, разделени с един интервал (1 < A < 3001, 1 < B < 3001).

Програмата трябва да запише резултата на стандартния изход. На един ред трябва да се изведе или 0 (ако първият играч ще загуби при оптимална игра на противника), или целите числа C и D, разделени с един интервал (ако първият играч може винаги да спечели. **ПРИМЕРИ**

**Вход Вход**

4 2 3 3

**Изход Изход**

1 1 0

***Решение:***

Разглеждат се позиции спрямо броя на елементите в едната купчинка. Ако тази купчинка може да се раздели на две части, поне една от които винаги е печеливша позиция, то тази позиция е губеща, защото противникът ще избере печелившата купчинка, за да продължи. Ако броят на елементите в разглежданата купчинка може да се раздели на две части, всяка от които е губеща позиция, то тази позиция е печеливша.

1) 1 предмет – губеща позиция, т.е. pos[1]=0

2) 2 предмета 1 – губеща позиция

1 – губеща позиция

Следователно pos[2]=1, т.е. е печеливша позиция.

3) 3 предмета 1 – губеща позиция

2 – печеливша позиция

Следователно pos[3]=0, т.е. е губеща позиция.

4) 4 предмета 1 – губеща позиция

3 – губеща позиция

Следователно pos[4]=1, т.е. е печеливша позиция.

5) 5 предмета 1 – губеща позиция

4 – печеливша позиция

или

5 предмета 2 – печеливша позиция

3 – губеща позиция

Следователно pos[5]=0, т.е. е губеща позиция.

и т.н.

Ето таблица със печелившите и губещите позиции при първите няколко случаи:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Забелязва се, че когато предметите са нечетен брой, то двете купчинки на които се разпадат са такива, че винаги едната е с четен брой, а другата с нечетен брой предмети, т.е. индуктивно се предполага, че едната е губеща позиция, а другата е печеливша позиция. Следователно противникът ще избере печелившата позиция (купчинката с четен брой предмети) и тази позиция е губеща позиция за играча на ход.

Аналогично, ако броят е четен, то може да се разделят на две купчинки, всяка от които с нечетен брой елементи, т.е. губещи позиции. Следователно тази позиция е печеливша за играча на ход.

Ето и програмната реализация:

#include<iostream>

using namespace std;

int a, b;

int main()

{

cin>>a>>b;

if (a % 2 == 1)

if (b % 2 == 1) cout<<0<<endl;

else cout<<1<<” “<<b-1<<endl;

else

if (b % 2 ==1) cout<<1<<” “<<a-1<<endl;

else cout<<1<<” “<<min(a, b)-1<<endl;

return 0;

}

***Задача 3. JBOI, 2008, Ден 1, Задача 1. КУЛИ ОТ МОНЕТИ***

Асен и Боян играят следната игра: Те избират две различни, цели положителни числа *K* и *L*, и започват игра с кула от *N* монети. Асен винаги играе първи, Боян – втори, след това – отново Асен, после Боян и т.н. Момчето, което е на ход може да вземе 1*, K* или *L* монети от кулата. Победител е момчето, което взема последната монета (монети). След като играли дълго време Асен забелязал, че има случаи, в които той може да спечели, независимо от това как играе Боян. И обратното, има случаи, в които Боян може да спечели, независимо от това как играе Асен. Така, преди началото на всяка игра, Асен е нетърпелив да узнае от кой вид е тя. Напишете програма **coins**, която помага на Асен да предскаже резултата за дадени *K, L* и *N*.

**Вход**

Входът описва *m* игри.

Първият ред на стандартния вход съдържа целите числа *K, L* и *m*, 1< *K* < *L* <10, 3 < *m* < 50. Вторият ред на стандартния вход съдържа *m* цели числа *N*1*, N*2*, …, Nm,* 1<=Ni<=1000000, *i* =1, 2, ..., *m*, представящи броя монети във всяка от *m*-те кули.

**Изход**

Стандартния изход съдържа низ с дължина *m*, съставен от буквите *A* и *B.* Ако Асен побеждава в *i*-тата игра (независимо от това как играе противника му), *i*-тата буква от низа трябва да бъде *A*. Когато Боян побеждава в *i*-тата игра (независимо от това как играе Асен), *i*-тата буква от низа трябва да бъде *B*.

**Пример**

**Вход**

2 3 5

3 12 113 25714 88888

**Изход**

ABAAB

***Решение:***

Определят се печеливши и губещи позиции в масив с големина max{N1, N2, …, Nm}. В началото печеливши са позиции 1, К и L. От позиция 2 до края позициите се определят по следните правила:

1. pos[i]=(pos[i-1] & pos[i-K] & pos[i-L]) ^1, ако i>max(K, L).
2. pos[i]=pos[i-1] ^1, ако i<min(K, L).
3. pos[i]=(pos[i-1]&pos[i-min(K, L)])^1, ако min(K, L)<i<max(K, L).

Накрая, ако Ni е печеливша позиция, то се извежда А, иначе се извежда В.

Ето как изглежда таблицата с част от позициите за дадения пример:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

***Задача 4. ПТ,2005 C2. На две***

Двама души започват игра с N цели клечки по следните правила:

Играчите се редуват на всеки ход;

Един ход се състои в следното: играчът, чийто ред е, или разчупва някоя цяла клечка на две части, или взема два компактни елемента;

Печели този, който направи последен ход.

С това всичко е казано, но ще обърнем внимание на следните подробности:

- играчът на ход прави само едно от двете действия – или взема, или чупи (ако има какво);

- никое парче не може повече да се чупи – тази операция е допустима само върху цяла клечка;

- парчетата и целите клечки са наречени с общото име “компактни елементи” при вземане.

Ако, например, има четири клечки, първият играч може да направи само едно от следните действия:

- да вземе две клечки: остават два компактни елемента и всеки от тях може да се чупи на следващ ход;

- да счупи една клечка на две: стават пет компактни елемента, три от които (целите клечки) още могат да се чупят;

Ако преди хода Ви има останала само една част от клечка, Вие вече сте загубили, поради невъзможност да направите ход: частта не може повече да се чупи, а един компактен елемент не може да се взема.

Напишете програма TWO, която определя дали първият играч може да спечели при зададено начално количество клечки, колкото и добре да играе вторият (казваме още “първият играч има печеливша стратегия”). Освен това определете с какъв първи ход може да стане това. Ако има повече от една възможност, изберете коя да е от тях.

От стандартното входно устройство коректно се въвежда един ред с естественото число N<=500.

Изведете на стандартното изходно устройство един ред с:

- числото 0, ако първият играч не може да спечели;

- числото 2, ако печелившият първи ход е “вземане на два компактни елемента”;

- числото -2, ако печелившият първи ход е “чупене на цяла клечка на две”.

**Пример:**

Вход:

4

Изход:

-2

***Решение:***

Ако М е броят на парчетата клечки, а N е броят на целите клечки, то ходът (M, N) зависи от резултатите на ходовете:

1. (M-2, N) – взимане на две парчета
2. (M, N-2) – взимане на две парчета
3. (M-1, N-1) – взимане на две парчета
4. (M+2, N-1) – чупене на едно цяло

Очевидно е, че ако записваме позициите в двумерен масив (2\*N x N), то трябва да го обхождаме по стълбове, тъй като е възможно парчетата да се увеличат, но за целите части това не е така.

Освен това с увеличаване на целите части с 1 (стълб), то броят на парчетата (редовете) трябва да намаляват с 2.

За отговор ще гледаме клетка (0,N) и откъде е дошла.

Нулевият стълб се попълва отделно!

Пример за N=5:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** | 0 | 0 | 1 | 1 | 1 | **1** |
| **1** | 0 | 1 | 1 | 1 | 1 |  |
| **2** | 1 | 1 | 1 | 0 | 0 |  |
| **3** | 1 | 1 | 0 | 0 |  |  |
| **4** | 0 | 0 | 1 | 1 |  |  |
| **5** | 0 | 1 | 1 |  |  |  |
| **6** | 1 | 1 | 1 |  |  |  |
| **7** | 1 | 1 |  |  |  |  |
| **8** | 0 | 0 |  |  |  |  |
| **9** | 0 |  |  |  |  |  |
| **10** | 1 |  |  |  |  |  |

Ето и програмната реализация:

#include<iostream>

using namespace std;

int n, pos[1001][501], m;

int main()

{

cin>>n;

m=2\*n;

pos[2][0]=1;

for (int i=3; i<=m; i++)

pos[i][0]=pos[i-2][0]^1;

for (int j=1; j<=n; j++, m-=2)

for (int i=0; i<=m; i++)

{

pos[i][j]=1;

if (j>=2) pos[i][j]&=pos[i][j-2];

if (i>=2) pos[i][j]&=pos[i-2][j];

if (i>=1) pos[i][j]&=pos[i-1][j-1];

pos[i][j]&=pos[i+2][j-1];

pos[i][j]^=1;

}

if (pos[0][n]==1)

if (pos[0][n-2]==0) cout<<2<<endl;

else cout<<-2<<endl;

else cout<<0<<endl;

return 0;

}

***Задача 5. НОИ3, 2007, С1. НЕ КАТО МЕН!***

Асен и Боби играят играта „Не като мен!”: сядат пред купчинка от N предмета и, редувайки се, всеки взима на своя ход не по-малко от Р предмета и не повече от Q предмета. Губи този, който е на ход, но не може да го изиграе, спазвайки правилата. Името на играта се оправдава от това, че никой ***няма право да повтори последния ход*** на опонента си. За да е честно, игрите винаги се играят по две.

Очевидно всяка игра е определена от тройка числа {**N, P, Q**}. Да видим как би протекла една игра {11, 2, 3}. Да считаме, че сега Асен играе пръв. Ако Асен вземе 2, Боби може да вземе само 3 (не може да повтори хода на Асен) и остават 6. Ако Асен вземе сега 3, Боби взема 2 и печели, защото остава 1, което Асен няма право да вземе. Останалите възможности се разглеждат аналогично. В този случай както и да играе Асен, Боби винаги може да спечели.

Напишете програма ***notalike***, която за дадена двойка игри определя дали първият на ход може да спечели и ако да – колко **най-много** предмета може да вземе на първия си ход. От стандартния вход се въвеждат два реда, на всеки от които е описана една игра: естествените числа **N, P** и **Q** в тоя ред, разделени с интервал. Числото **N** не надвишава 1000, а 1≤**P≤Q**≤20. За всяка от игрите запишете на стандартния изход по един ред с едно неотрицателно число: нула, ако вторият играч може да спечели, както и да играе първият, или максималния брой предмети, които първият играч може да вземе на първия си ход по пътя към победата.

**ПРИМЕР**

**Вход Изход**

11 2 3 0

21 1 9 5

***Решение:***

Нека pos[i][j] да показва каква е позицията (печеливша или губеща), ако играчът на ход има i предмета и преди това другият играч е взел j предмета. Тогава трябва да се разглеждат всички позиции pos[i-x][x], за които p<=x<=q и x≠j.

Ето как изглежда таблицата с позициите за втория пример:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **1** | **0** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **2** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **3** | 1 | 1 | **0** | 1 | 1 | 1 | 1 | 1 | 1 |
| **4** | 1 | 1 | 1 | **0** | 1 | 1 | 1 | 1 | 1 |
| **5** | 1 | 1 | 1 | 1 | **0** | 1 | 1 | 1 | 1 |
| **6** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **7** | 1 | 1 | 1 | 1 | 1 | 1 | **0** | 1 | 1 |
| **8** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **9** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** |
| **10** | 1 | 1 | 1 | 1 | **0** | 1 | 1 | 1 | 1 |
| **11** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **12** | **0** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **13** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| **14** | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |
| **15** | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |
| **16** | 1 | 1 | 1 | 1 | **0** |  |  |  |  |
| **17** | 1 | 1 | 1 | 1 |  |  |  |  |  |
| **18** | 1 | 1 | 1 |  |  |  |  |  |  |
| **19** | 1 | 1 |  |  |  |  |  |  |  |
| **20** | 1 |  |  |  |  |  |  |  |  |
| **21** |  |  |  |  |  |  |  |  |  |

Накрая се търси последната губеща позиция по диагонала pos[n-x][x], което отговаря на първия ход, който ще направи първия играч, за да прати втория играч в губеща позиция. Ако няма такъв ход, то първият играч не може да спечели играта!

Ето и програмната реализация:

#include<iostream>

using namespace std;

int n, p, q, pos[1001][501];

void scan()

{

for (int i=0; i<n; i++)

for (int j=p; j<=q; j++)

if (i<p) pos[i][j]=1;

else pos[i][j]=0;

for (int i=p; i<n; i++)

for (int j=p; j<=q; j++)

{

pos[i][j]=1;

for (int x=p; x<=q; x++)

if (x!=j && x<=i)

pos[i][j]&=pos[i-x][x];

pos[i][j]^=1;

}

int ans=0;

for (int i=p; i<=q; i++)

if (pos[n-i][i]==0) ans=i;

cout<<ans<<endl;

}

int main()

{

for (int i=1; i<=2; i++)

{

cin>>n>>p>>q;

scan();

}

return 0;

}

***Задача 6. НОИ2, 2009, Задача B1. ЧЕТНОТО ПЕЧЕЛИ***

Както всеки ден, Ваньо е пред компютъра. Играе една простичка игра и се ядосва, че все губи срещу „една щайга чипове”! Помогнете му, в името на Естествения Интелект! Ето правилата на играта. Има ***N*** обекта, където ***N*** е нечетно естествено число, не по-малко от 3. Играещите се редуват, като всеки взема по няколко обекта на ход (винаги трябва да се вземе поне един обект, но не повече от ***K***). В един момент обектите свършват. Всеки вижда какво е събрал. Печели този, който е събрал четен брой обекти. Просто, нали?

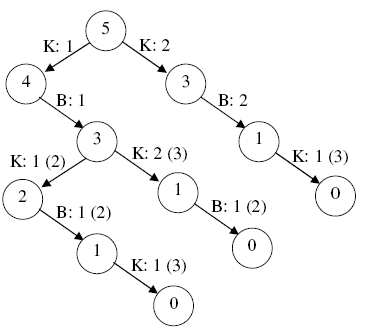
Преди самата игра, компютърът избира ***K***, след това Ваньо избира ***N***. Играта започва и компютърът играе пръв. Напишете програма **even**, която при избрано от компютъра число ***K*** да подскаже на Ваньо как да избере ***N*** така, че да има шанс срещу перфектно играещия компютър-противник!

**Вход**

От стандартния вход се въвежда един ред, на който е записано числото ***K***, избрано от компютъра, като ***K*** > 1 и има не повече от 15 цифри.

**Изход**

Запишете на стандартния изход един ред с 10 ***различни*** стойности за ***N***, разделени с интервал нечетни естествени числа, по-големи от 3, но с не повече от 18 цифри, при които Ваньо би имал шанс (ако играе умно) срещу компютъра, който никога не допуска грешки.

**Пример**

**Вход:**

2

**Изход:**

5 13 17 29 33 21 2221 101 37 25

***Обяснение:***

Ще разгледаме подробно как би протекла играта при *N*=5, така че

Ваньо да спечели. Първият ход е на компютъра, който може да вземе 1 или 2, съответно да останат 4 или 3 обекта. В зависимост от хода на компютъра, Ваньо може да играе умно, в случая – да вземе толкова, колкото е взел компютърът. В скобки са дадени натрупаните до момента обекти на всеки играч. Както се вижда, Ваньо може така да играе, че накрая винаги да има два обекта, независимо от ходовете на компютъра, а следователно – и да спечели.

***Решение:***

Означавам с (четно, m) състояние в играта, при което играчът на ход има взети **четен** брой предмети и са останали невзети още m предмети.

Аналогично (нечетно, m) е състояние в играта, при което играчът на ход има взети **нечетен** брой предмети и са останали невзети още m предмети.

Ако се направи ход, при който се взимат p предмети (1<=р<=к), то от едно състояние на играта се преминава в друго състояние по следните правила:

1. **Ако m е четно число:**
   1. **(четно, m) –> взимат се р предмета -> (нечетно, m-p)**

т.е., ако играчът А има в себе си четен брой предмети и са останали четен брой нераздадени предмети, то играчът В има в себе си нечетен брой предмети (защото в началото N е нечетно) и след хода на А ще дойде хода на В с нечетен брой предмети в себе си и (m-p) останали.

* 1. **(нечетно, m) –> взимат се р предмета -> (четно, m-p)**

Разсъжденията са аналогични.

1. **Ако m е нечетно число:**
   1. **(четно, m) –> взимат се р предмета -> (четно, m-p)**
   2. **(нечетно, m) –> взимат се р предмета -> (нечетно, m-p)**

Попълва се таблица за всяко състояние, започвайки от началните (всъщност те са крайни за играта):

(четно, 0) е печеливша позиция.

(нечетно, 0) е губеща позиция.

За задачата се интересуваме при кои m (четно, m) е губеща позиция, защото в началото играчът А (компютърът) има 0 предмети в себе си (о е четно) и искаме компютърът да загуби.

Разглеждаме няколко примера:

1. k=2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **m** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| нечетно | 0 | 1 | 1 | 0 | 0 | 1 | 1 |  |  |  |
| четно | 1 | 0 | 1 | 1 | 1 | 0 | 1 |  |  |  |

Забелязва се цикличност през 4, т.е. губещи позиции сa:

m=1, 5, 9,…, т.е. m=4q+1

1. k=3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **m** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| нечетно | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| четно | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

Забелязва се цикличност през 8, т.е. губещи позиции сa:

m=1, 4, 9, 12,…, т.е. m=8q+1 и m=8q+4

1. k=4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **m** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| нечетно | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| четно | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

Забелязва се цикличност през 6, т.е. губещи позиции сa:

m=1, 7, 13,…, т.е. m=6q+1

1. k=4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **m** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| нечетно | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| четно | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

Забелязва се цикличност през 12, т.е. губещи позиции сa:

m=1, 6, 13, 18,…, т.е. m=12q+1 и m=12q+6

От тези примери се виждат общите зависимости:

1. **Ако k е четно, то губещи позиции са m=(k+2)\*q+1, за q=0, 1, 2,…**
2. **Ако k е нечетно, то губещи позиции са m=2(k+1)\*q+1 и m=2(k+1)\*q+(к+1), за q=0, 1, 2,…**

Ето и програмната реализация:

#include<iostream>

using namespace std;

long long k;

int main()

{

cin>>k;

if (k % 2 == 1)

for (int i=1; i<=10; i++)

{

cout<<2\*(k+1)\*i+1;

if (i<10) cout<<” “;

else cout<<endl;

}

else

for (int i=1; i<=10; i++)

{

cout<<(k+2)\*i+1;

if (i<10) cout<<” “;

else cout<<endl;

}

return 0;

}

***Задача 7. ПТ, 2011, Задача C1. ИГРА***

Дадена е купчина от *S* бонбона. Двама играят следната игра – един след друг вземат бонбони от купчината, като играч номер 1 има право на всеки ход да вземе или *а*1 или *a*2, или *a*3, …, или *am* бонбона, а играч номер 2 – да вземе *b*1 или *b*2, или *b*3, …, или *bn* бонбона. Играта губи този, който при пореден свой ход не може да вземе бонбони, защото купчината е вече празна или защото няма толкова бонбони в купчината, колкото му е позволено да вземе. Когато единият играч загубва играта, другия играч печели. Напишете програма **game**, която определя за даден играч, дали той може да спечели играта, ако играе възможно най-добре при всякакъв начин на игра от противника.

**Вход**

На първия ред е записана стойността на *S* и номер на играч (1 или 2), за който програмата ще изведе дали този играч може да спечели играта. Следва ред със стойността на *m* и ред със стойностите *а*1, *a*2, *a*3, ..., *am*. Следващ ред съдържа стойността на *n* и последният ред е със стойностите *b*1, *b*2, *b*3, ..., *bn*.

**Изход**

Програмата трябва да изведе числото 1, ако съответния играч може да спечели играта. В противен случай, програмата трябва да изведе числото 0.

**Ограничения**

1 ≤ *S* ≤ 1 000 000

1 ≤ *m* ≤ 200

1 ≤ *n* ≤ 200

Стойностите на *а*1, *a*2, *a*3, ..., *am*, и *b*1, *b*2, *b*3, ..., *bn* са в диапазона от 1 до 1000.

**Пример**

**Вход**

6 1

2

2 3

1

1

**Изход**

1

***Решение:***

В програмата се подържа масив pos[g][k], където за играч с номер g (g=0,1) и за позиция k (k=0,1,2…,S) се записва стойност true или false, в зависимост от това, дали играчът g, намиращ се пред купчинка с k бонбона може да спечели или не може.

Oчевидно pos[g][0]=false.

Пресмятането на стойностите pos[g][k] се извършва последователно за k=1,2,3,…S, като в pos[g][k] записваме false, когато при всички позволени ходове, играчът g ще представи на противника (който е с номер g–1) само печеливши позиции, т.е. ако вече е пресметнато, че pos[g–1][v]=true за всяко такова v<k, което може да се получи от k с вземане на позволен брой бонбони.

1. **Игра на Ним**

В началото на играта има К групи от предмети, съдържащи съответно m1, m2, …, mk предмети. Играят двама. При поредния ход всеки играч може да вземе от **една група** произволно количество предмети (даже всичките). Победител е онзи, който успее последен да вземе всички останали предмети.

***Алгоритъм:***

Нека групите са 3 и във всяка от тях има съответно а, b и c предмета.

***Теорема:*** **Ако a^b^c=0, то тогава (a, b, c) е губеща позиция.**

*Доказателство:* Трябва да се докаже, че от една губеща позиция не може да се отиде в друга губеща позиция.

Щом a^b^c=0, то c=a^b.

Нека от третата група се вземат с1 предмета. Да допуснем, че тази новополучена позиция (a, b, c-c1) е губеща. Тогава a^b^(c-c1)=0. Следователно c-c1=a^b, но a^b=с и тогава с1=0, което води до противоречие.

Интересен е въпроса как от една печеливша позиция може да се отиде в губеща позиция, което е необходимо за играта.

Нека a^b^c=х, х≠0. Искаме да намалим някое от числата а, b и c, така че х да стане 0. Ако намалим с и новополучената стойност е с1<=с, то трябва a^b^c1=0, следователно с1=a^b. Но тъй като a^b може да е по-голямо от с, то трябва да проверим дали a^b<=с, или a^c<=b, или b^c<=a и това ще покаже от коя от трите групи ще взимаме предмети.

**Пример:**

a=10, b=5, c=8. Имаме позиция (10, 5, 8) и 10^5^8=7, т.е. не е губеща позиция.

Тогава a^b=15>c, b^c=13>a, a^c=2<b. Следователно ще взимаме от втората група 3 предмета, така че да останат 2 предмета. Тогава новополучената позиция е (10, 2, 8), която е губеща, защото 10^2^8=0.

Сега остава да се разгледа обшият случай с К групи, съдържащи съответно m1, m2, …, mk предмети. Тогава ако m1^m2^…mk=0, то това е губеща позиция и вторият играч винаги може да спечели. В противен случай се правят всички комбинации от (К-1) елемента, които са точно К на брой и се търси такава комбинация, за която M=mi1^mi2^…mik-1<=mik. Тогава от групата ik се взимат mik-M предмета, за да останат точно М и да се получи губеща позиция. Най-лесна реализация на този алгоритъм е да се пресметне х= m1^m2^…mk и итеративно да се проверява дали x^mi<=mi, за всяко i=1, 2, …, k.

***Задача 1. НШИ, Бургас, 20-27 Септември, 2008, Контролно състезание1, Задачи за 8 клас, Задача Заек3***

Както навярно знаете, истинското богатство за зайците това са морковите. Нека имаме два заека (за яснота ще ги кръстим заек едно и заек две) и N+1 на брой кашона с моркови. Всеки един кашон е номериран с число от 0 до N и в него може да има, а може и да няма моркови. Кашонът с номер 0 е празен. Всеки един от двата заека иска да изяде всички моркови. След като не стигнали до съгласие, кой да изяде морковите, те решили да играят на игра.

Победителя от играта изяжда всички моркови. Правилата й са прости. Заек едно и заек две се редуват на ходове. Този от тях, които е на ход взема един морков от кашон и го премества в някои от левите съседни кашони. След това е на ход другия заек. Играта приключва, когато всички моркови се намират в кашона с номер 0. Заекът, които е сложил последен морков в кашона с номер 0 печели.

Напишете програма **cbunny**, която по зададено разположение на морковите в кашоните да определи, кой от зайците ще спечели играта, ако играят оптимално.

Заек едно е първи на ход.

**Вход**

От първия ред на стандартния вход се прочита едно число N – броя на кашоните без нулевия. От следващия ред се прочитат N числа описващи броя на морковите. Първото число описва, колко моркова има в първия кашон, второто число, колко моркова има във втория кашон и т.н.. В един кашон има не повече от 10000 моркова.

**Изход**

На стандартния изход отпечатайте резултата във следния формат: ако

заек едно ще спечели играта отпечатайте две числа разделени с интервал,

първото от които показва номера на кашона от който ще вземе морков, а второто номера на кашона в който ще го постави. Ако има няколко възможни печеливши хода, отпечатате този от тях, в които се взема морков от кашона с най-малък номер. Ако заек две печели отпечатайте “LOSER”

**Ограничения:**

0 < N < 10001

**Примерен вход: Примерен вход:**

5 4

1 0 3 5 2 1 1 1 2

**Примерен изход: Примерен изход:**

4 2 LOSER

***Решение:***

Играта NIMBLE:

<http://www.cut-the-knot.org/Curriculum/Games/Nimble.shtml>

Ясно е, че кашоните, в които има четен брой предмети не ни интересуват, тъй като при тях двамата играчи играят симетрично и винаги печели вторият на ход.

Тогава целта е да останат само кашони с четен брой моркови и да е ред на втория заек, тъй като това е губеща позиция.

Нека разгледаме позициите на кашоните, в които има нечетен брой моркови.

За първия пример те са a=1, b=3, c=4.

Искаме тези позиции да станат 0, т.е. да се пренесат морковите в кашон 0 и да е на ход втория заек. Тъй като всеки път можем да местим само наляво, то а, b и с намаляват. Това означава, че можем да променяме само едно от тях (намаляваме) и искаме да извършим последния ход, когато а=b=с=0. Но това е точно играта на Ним!

Следователно, ако a^b^c=0, то първият играч винаги губи. Иначе първият ход се прави по описания в играта на Ним начин.

Ето и програмната реализация:

#include<iostream>

using namespace std;

int n, pos[10001],x;

int main()

{

scanf(“%d”,&n);

for (int i=1; i<=n; i++)

{

scanf(“%d”,a+i);

if (a[i] % 2 == 1) x^=i;

}

if (x==0) printf(“LOSER\n”);

else

for (int i=1; i<=n; i++)

if (a[i] % 2 == 1)

if (i>x^i)

{

printf(“%d %d\n”, i, x^i);

break;

}

return 0;

}

***Задача 2. COCI, 2010-2011, Contest 4, Task HRPA***

Mirko and Slavko’s favourite pastime is competing against each other in mathematical games. This time they took a heap of **N** pebbles and settled on the following rules:

1. Mirko is the first to play, then Slavko, then Mirko again, then Slavko and so on;

2. Mirko can take any number of pebbles (between 1 and **N**, inclusive) from the heap during his first move;

3. In each of the following turns the current player must take at least 1 pebble and is allowed to take **at most double the amount of pebbles taken during the previous turn** by the other player; naturally, one cannot take more pebbles than the remaining amount in the heap;

4. The player who takes the last pebble is the winner.

Both Mirko and Slavko play optimally (if it is possible for one player to beat the other, that player will always win). We need to find the **minimum** number of pebbles that Mirko must take during his first turn such that he is guaranteed to win the game.

**INPUT**

The first and only line of input contains the positive integer **N** (2 ≤ **N** ≤ 1015), the number of pebbles in the starting heap.

**OUTPUT**

The first and only line of output must contain the required minimum number of pebbles that Mirko needs to remove during his first turn.

**SAMPLE TESTS**

**input output**

4 1

**input output**

7 2

**input output**

8 8

**First sample description:**

Mirko has 4 possibilities to choose from: he can take 1, 2, 3, or 4 pebbles from the heap. If he takes all 4 pebbles he will naturally win, but that is not the minimum solution. We need to check the remaining alternatives. If Mirko takes only one pebble, Slavko is left with a heap of 3, but he can take at most 2.

Slavko cannot take all pebbles, but Mirko will be able to take all remaining pebbles during his next turn, winning the game. We conclude that 1 is the minimum solution for this test case.

***Решение:***

Играта Фибоначиев ним: Играят двама. Един след друг избират предмети от обща група с n предмети. Влизащият пръв в играта може да вземе произволен брой предмети, но не всичките. От втория ход нататък на всеки от играчите се разрешава да вземе произволно количество предмети р (1<=р<=n), където n<=2n1 (n1 е количеството предмети, които съперникът е взел при предишния ход). Победител е онзи, който вземе всички останали предмети.

***Теорема***: Всяко естествено число може да се представи като сума от числа на Фибоначи, разположени в низходящ ред, по единствен начин.

Пример: 20=13+5+2, 67=55+8+3+1

Тогава алгоритмът на задачата Фибоначиев ним е следния: Ако броят на предметите в началото не е число на Фибоначи, то се прави ход, като броят се представя като сума от числа на Фибоначи и се взима най-малкия (Фибоначиев) брой предмети. В противен случай трябва да се влезе в играта като втори играч, за да се спечели.

Да се върнем към нашата задача. Ето таблица с печеливши и губещи позиции:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **1** | 1 |  |  |  |  |  |  |  |  |
| **2** | 0 | 1 |  |  |  |  |  |  |  |
| **3** | 0 | 0 | 1 |  |  |  |  |  |  |
| **4** | 1 | 0 | 0 | 1 |  |  |  |  |  |
| **5** | 0 | 0 | 0 | 0 | 1 |  |  |  |  |
| **6** | 1 | 0 | 0 | 0 | 0 | 1 |  |  |  |
| **7** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |  |  |
| **8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| **9** | 1 | 0 | 0 |  |  |  |  |  |  |

Ето и таблица с първите отговори:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **ans** | **n** | **ans** | **n** | **Ans** | **n** | **ans** |
| **1** | 1 | **12** | 1 | **23** | 2 | **34** | 34 |
| **2** | 2 | **13** | 13 | **24** | 3 | **35** | 1 |
| **3** | 3 | **14** | 1 | **25** | 1 | **36** | 2 |
| **4** | 1 | **15** | 2 | **26** | 5 | **37** | 3 |
| **5** | 5 | **16** | 3 | **27** | 1 | **38** | 1 |
| **6** | 1 | **17** | 1 | **28** | … | **39** | 5 |
| **7** | 2 | **18** | 5 | **29** | … | **40** | 1 |
| **8** | 8 | **19** | 1 | **30** | … | **41** | 2 |
| **9** | 1 | **20** | 2 | **31** | … | **42** | 8 |
| **10** | 2 | **21** | 21 | **32** | … | **43** | … |
| **11** | 3 | **22** | 1 | **33** | … | **44** | … |

Забелязва се, че при n=1, 2, 3, 5, 8, 13, 21, 34, 55, …, т.е. числа на Фибоначи, отговорът е n и че отговорът е винаги числа на Фибоначи. Още:

35=**1**+34

36=**2**+34

37=**3**+34

38=**1**+3+34

39=**5**+34

40=**1**+5+34

41=**2**+5+34

42=**8**+34

Следователно, отговорът е най-малкото число в това представяне!

Ето и програмната реализация:

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

vector<long long> v;

long long n, br, x;

int main()

{

cin>>n;

v.push\_back(1);

v.push\_back(1);

br=1;

while (1)

{

br++;

x=v[br-1]+v[br-2];

if (x>n) break;

v.push\_back(x);

}

br--;

while (n != v[br])

{

if (n>v[br]) n-=v[br];

br--;

}

cout<<n<<endl;

return 0;

}

1. **Още една задача: ТЕРОРИСТИ**

В държава има няколко аерогари и полети между тях. Възможно е да се долети от една аерогара до друга. Възможно е това да стане с прехвърляне на междинни аерогари, но между две произволни аерогари съществува единствен маршрут за прелитане.

Двама терористи играят следната игра. Правят ход един след друг. Играта се състои от следните операции. Играчът на ход минира аерогарата, на която се намира, избира полет и излита заедно със своя колега. След излитането, той взривява аерогарата и всички полети до и от тази аерогара се отменят. След приземяването в друга аерогара, на ход е другия играч. Играта загубва този, който не може да направи ход. Напишете програма GAME, която по зададен начален списък с полетите и аерогарата на която се намират терористите, определя кой ще победи, ако всеки играч се придържа към оптималната стратегия.

**Входни данни:**

Първият ред съдържа две цели числа: N – броя на аерогарите (1<=N<=1000) и К – номера на първоначалната аерогара.

Следващите (N-1) реда съдържат по две цели числа – номерата на аерогарите, между които има полет (всички полети са двупосочни и се споменават само веднъж). ОТ всяка аерогара има не повече от 20 полета.

**Изходни данни:**

Единственият ред трябва да съдържа 0, ако първият играч губи играта, или номера на аерогарата, до която трябва да долети, ако първият играч печели играта. Ако има няколко печеливши аерогари, то да се изведе тази с минимален номер.

Пример:

Вход Изход

4 3 2

3 2

3 1

1 4

***Решение:***

Очевидно, графът образуван от аерогарите и полетите до тях е дърво.

Също толкова очевидно е, че листата са губещи позиции за играча на ход.

ПП

ПП

ГП

ГП

Дървото трябва да се обходи чрез рекурсия (DFS), и позициите ще се изчисляват на връщане от рекурсията.

Ето и програмната реализация:

#include<iostream>

#include<vector>

using namespace std;

int n, k, used[1024], pos[1024];

vector<int> v[1024];

void dfs(int x)

{

pos[x]=1;

used[x]=1;

for (int j=0; j<a[x].size(); j++)

if (used[a[x][j]]==0)

{

dfs(a[x][j]);

pos[x]&=pos[a[x][j]];

}

pos[x]^=1;

}

int main()

{

scanf(“%d%d”, &n, &k);

for (int i=1; i<n; i++)

{

int x, y;

scanf(“%d%d”, &x, &y);

a[x].push\_back(y);

a[y].push\_back(x);

}

dfs(k);

if (pos[k]==0) printf(“0\n”);

else

{

int ans=1024;

for (int i=0; i<a[k].size(); i++)

if (pos[a[k][i]]==0)

ans=min(ans,a[k][i]);

printf(“%d\n”, ans);

}

return 0;

}