Машина с топки

Мадина е изобретила машина с топки, за да забавлява участниците на ИОИ. Вътрешната структура на машината е следната:

- Машината се състои от N върха, индексирани от 0 до $N-1$. Връх $N-1$ е **коренът**.
- За всеки връх u за $0 \leq u < N-1$, **родителят** на връх u е връх $P[u]$, където $P[u] > u$, а връх u е **дете** на $P[u]$. Коренът няма родител. Връх без деца е **листо**.

Вие **не** знаете N или родителя $P[u]$ за никой връх u. Вместо това, Мадина Ви казва M, броя листа, и това, че индексите на листата са $0, 1, \ldots, M-1$. Вашата задача е да откриете вътрешната структура на машината като я оперирате.

Всеки връх на машината може да съдържа най-много една **топка**, а всяка топка има **стойност**, която е неотрицателно цяло число. Казваме, че връх има стойност x, ако съдържа топка със стойност x. Връх е **зает**, ако съдържа топка; иначе е **свободен**. В началото, всички върхове са свободни.

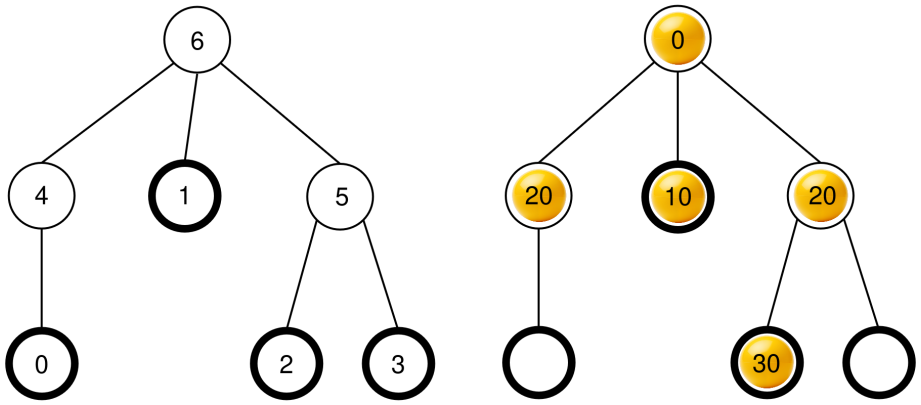
Можете да извършвате два вида операции:

- **insert**(U, X): пробвате да вкарате нова топка със стойност X в листо U.
    - Ако листо U е заето, операцията връща `false` без топката да бъде вкарана.
    - Ако листо U е свободно, топката бива поставена във връх U. След това тя многократно се качва в родителя на текущия си връх, стига той да е свободен. Това движение спира, когато топката достигне корена или връх, чийто родител е зает. Операцията връща `true`.
- **collect()**: ако коренът е свободен (което означава, че машината е празна), `collect` връща празен масив. В противен случай, рекурсивната процедура `traverse`, описана долу, събира в масив S стойностите на всички топки текущо в машината. В началото, масивът S е празен и се вика `traverse(N - 1)`.

```
traverse(u):
    добавяме стойността на топката във връх u към края на S
    нека c[u] да бъде списъкът от заети деца на u
    сортираме c[u] в ненамаляващ ред спрямо стойностите
      на топките в тези върхове (ако множество върхове има
      еднакви стойности, те може да бъдат в който и да е ред)
    за всяко v в c[u]:
      traverse(v)
```

След като тази процедура приключи, **всички топки се премахват** от машината и `collect` връща S.

Например, разгледайте машина с $N = 7$ върха и родителен масив $P = [4, 6, 5, 5, 6, 6]$. Картинката в ляво показва празната машина с индекси на върховете, а картинката в дясно показва възможна конфигурация на топките след някаква поредица `insert` операции (тази поредица е дадена в секцията с примера).



Когато `collect()` бъде извикана, коренът (връх 6) е зает, т.е. S е инициализран с $S = [\,]$ и е извикана `traverse(6)`.

`traverse(6)`: връх 6 има стойност 0; добавя се към S и се получава $S = [0]$. Децата на връх 6 са 4, 1, и 5, всички от които са заети. Техните стойности са 20, 10, и 20, съответно. Сортират се в ненамаляващ ред и се получава $c[6] = [1, 5, 4]$ (забележете, че $c[6] = [1, 4, 5]$ също би било валидно, защото върхове 4 и 5 имат еднакви стойности). Процедурата тогава извиква `traverse` на всеки връх в $c[6]$ в този ред.

- **`traverse(1)`**: връх 1 има стойност 10; добавя се към S и се получава $S = [0, 10]$. Връх 1 няма заети деца, така че това извикване приключва.
- **`traverse(5)`**: връх 5 има стойност 20; добавя се към S и се получава $S = [0, 10, 20]$. Връх 5 има едно заето дете, връх 2, т.е. $c[5] = [2]$ и процедурата извиква `traverse(2)`.
    - **`traverse(2)`**: връх 2 има стойност 30; добавя се към S и се получава $S = [0, 10, 20, 30]$. Връх 2 няма заети деца, така че това извикване приключва.

- Екзекуцията се завръща в `traverse(5)`, което е процесирало всички деца в $c[5]$, така че това извикване приключва.

- **`traverse(4)`**: връх 4 има стойност 20; добавя се към S и се получава $S = [0, 10, 20, 30, 20]$. Връх 4 няма заети деца, така че това извикване приключва.

Всички извиквания са приключени и няма повече върхове за процесиране. Накрая, всяка топка е премахната от машината и `collect` връща масива $S = [0, 10, 20, 30, 20]$.

Вашата задача е да откриете броя върхове N и да дадете родителен масив $R = [R[0], R[1], \ldots, R[N-2]]$, който описва структурата на машината, но с потенциално различна номерация на върховете, които не са листа или корена. Формално, даден масив R се счита за верен, ако е възможно да се даде уникален номер $L[u]$, където $0 \leq L[u] < N$, за всеки връх u на машината, така че:

- $L[u] = u$ за всички $0 \leq u < M$ и $u = N - 1$, и
- $L[P[u]] = R[L[u]]$ за всички $0 \leq u < N - 1$.

Не е нужно да е изпълнено $R[u] > u$. Машината **трябва да бъде празна**, когато връщате решението си.

Нека K да бъде броят на изпълнените `collect` операции, а B да бъде максималната стойност на някоя топка сред всички `insert` извиквания. Нека $C = K + B$. Тогава C **не трябва да надвишава** 1000**.** Вашият резултат на някои подзадачи зависи от стойността на C.

Детайли по имплементацията

Трябва да имплементирате следната функция:

```
std::vector<int> find_structure(int M)
```

- M: броят на листата в машината.
- Функцията се извиква точно веднъж на тест.
- Функцията трябва да върне масив $R = [R[0], R[1], \ldots, R[N-2]]$ с дължина $N - 1$, който е верен родителен масив.

За да ползвате машината, Вашата програма може да извиква следните две функции.

Първата функция е:

```
bool insert(int U, int X)
```

- U: индексът на листото, където да се вкара топката. Условието $0 \leq U < M$ трябва да е изпълнено.
- X: целочислената стойност на топката. Условието $0 \leq X \leq 1000$ трябва да е изпълнено.
- Функцията връща `true`, ако топката е успешно вкарана, или `false`, ако листото U вече е било заето.
- Функцията може да бъде извиквана най-много $500\,000$ пъти.

Втората функция е:

```
std::vector<int> collect()
```

- Събира стойностите на вкараните топки, изпразва машината, и връща събрания масив S.

Ако в който и да е момент по време на екзекуцията на програма Ви стойността на C надвиши 1000, решението Ви ще получи резултат `Output isn't correct: Too many resources used`.

Структурата на машината е фиксирана преди `find_structure` да бъде извикана. Грейдърът е **детерминистичен** в смисъла, че ако бъде изпълнен два пъти, като двете изпълнения направят еднакви поредици операции, извикванията на `collect` ще връщат еднакви масиви.

Ограничения

- $2 \leq N \leq 1000$
- $1 \leq M \leq 200$
- $M < N$

Подзадачи

Подзадача	Точки	Допълнителни ограничения
1	5	Коренът има точно M деца.
2	10	$M \leq 3$
3	25	$N \leq 200$, $M \leq 45$
4	60	Няма допълнителни ограничения.

В подзадачи 3 и 4, точките Ви ще зависят от C както следва.

Подзадача 3 (25 точки)

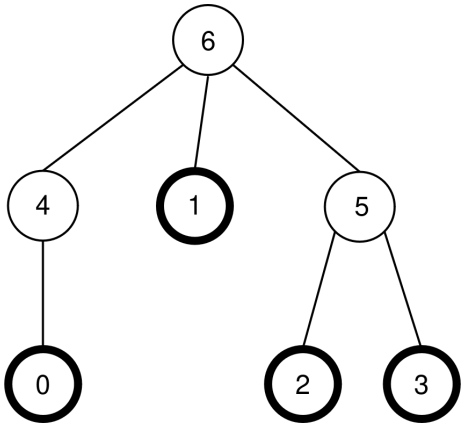
Случай	Точки
$1000 < C$	0
$45 < C \leq 1000$	13
$C \leq 45$	25

Подзадача 4 (60 точки)

Случай	Точки
$1000 < C$	0
$200 < C \leq 1000$	7
$71 < C \leq 200$	$47 - \frac{C}{5}$
$44 < C \leq 71$	$104 - C$
$C \leq 44$	60

Пример

Разгледайте същата машина както в условието, т.е. $N = 7$, $M = 4$, и $P = [4, 6, 5, 5, 6, 6]$. Машината е показана отново в следната картинка:



Грейдърът прави следното извикване:

```
find_structure(4)
```

Функцията може да направи следната поредица извиквания:

1. **`insert(0, 0)`**: листо 0 е свободно, така че топка със стойност 0 се поставя в листо 0. Връх $P[0] = 4$ е свободен, така че топката се мести към връх 4. Връх $P[4] = 6$ е

свободен, така че топката се мести към връх 6. Тъй като връх 6 е коренът, движението спира. Извикването връща `true`.

2. **`insert(3, 20)`**: листо 3 е свободно, така че топка със стойност 20 се поставя в листо 3. Връх $P[3] = 5$ е свободен, така че топката се мести към връх 5. Тъй като връх $P[5] = 6$ е зает, движението спира. Извикването връща `true`.
3. **`insert(1, 10)`**: листо 1 е свободно, така че топка със стойност 10 се поставя в листо 1. Тъй като връх $P[1] = 6$ е зает, топката остава във връх 1. Извикването връща `true`.
4. **`insert(2, 30)`**: листо 2 е свободно, така че топка със стойност 30 се поставя в листо 2. Тъй като връх $P[2] = 5$ е зает, топката остава във връх 2. Извикването връща `true`.
5. **`insert(1, 25)`**: листо 1 е заето, така че топката не се поставя в листо 1. Извикването връща `false`.
6. **`insert(0, 20)`**: листо 0 е свободно, така че топка със стойност 20 се поставя в листо 0. Връх $P[0] = 4$ е свободен, така че топката се мести към връх 4. Тъй като връх $P[4] = 6$ е зает, движението спира. Извикването връща `true`.

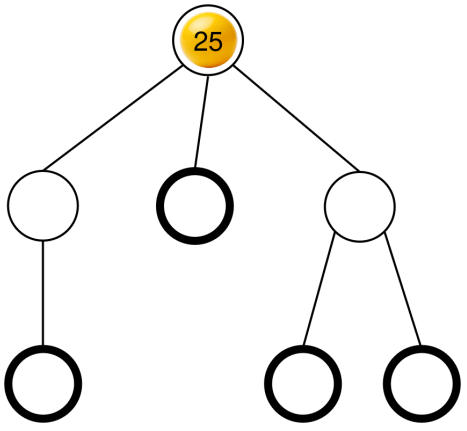
Крайната конфигурация на топките е тази вече показана в картинката в условието.

След това, функцията може да извика:

```
collect()
```

Екзекуцията на това извивкане е вече обяснена в условието; то връща $S = [0, 10, 20, 30, 20]$. След това, машината е отново празна.

После, функцията може да извика **`insert(2,25)`**. Листо 2 е свободно, така че топка със стойност 25 се поставя в листо 2. След това, топката се мести към връх 5, а после към връх 6, където спира. Извикването връща `true`. Резултатната конфигурация от топки е показана на следната картинка.



Накрая, функцията може да извика **`collect()`**, което връща $S = [25]$ и изпразва машината.

Има два верни родителни масива, които `find_structure` може да върне: $R = P = [4, 6, 5, 5, 6, 6]$ (което съответства на номерацията $L = [0, 1, 2, 3, 4, 5, 6]$), и

$R = [5, 6, 4, 4, 6, 6]$ (което съответства на номерацията $L = [0, 1, 2, 3, 5, 4, 6]$). В този пример, $K = 2$ и $B = 30$, т.е. $C = 32$.

Локален грейдър

Входен формат:

```
N M
P[0] P[1] P[2] ... P[N-2]
```

Изходен формат:

```
K B C
Q
R[0] R[1] R[2] ... R[Q-1]
```

Тук, Q е дължината на масива R, върнат от `find_structure`.