Design do Brinquedo

Nome do Problema	*Design* do Brinquedo
Arquivo de Entrada	Tarefa Interativa
Arquivo de Saída	Tarefa Interativa
Limite de tempo	1 segundo
Limite de memória	256 megabytes

Você está trabalhando para uma empresa que faz o *design* de brinquedos. Um novo brinquedo que está sendo criado funciona assim: Há n pregos, numerados de 1 a n, que estão pregados em uma caixa. Alguns pares de pregos são conectados por fios dentro da caixa. (Em outras palavras, os pregos e os fios formam um grafo não direcionado, onde os pregos são os vértices e os fios são as arestas). Os fios não são visíveis da parte de fora e a única maneira de descobrir algo sobre eles é usar um **verificador** nos pregos: Podemos escolher dois pregos i e j tal que $i \neq j$ e o verificador dirá se esses dois pregos estão conectados dentro da caixa, seja direta ou indiretamente. (Assim, o testador diz se existe um caminho entre esses pregos no grafo).

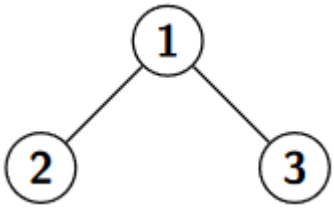
Chamaremos o conjunto de conexões dentro da caixa de ***design*** do brinquedo.

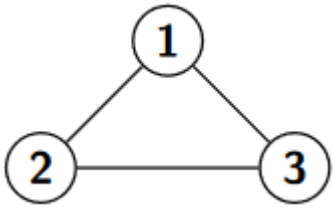
Você está usando um *software* especializado para consultar e criar esses *designs*. Este *software* funciona desta forma: Ele começa com algum *design* do brinquedo que nós denotaremos como "*design* 0". Ele não mostra as conexões dentro da caixa para este *design*. Ao invés disso, você pode realizar repetidamente a seguinte operação de três passos:

1. Você escolhe um *design* de número a e dois pregos de números i e j tal que $i \neq j$.
2. O *software* te diz o que aconteceria se utilizássemos o verificador nesses dois pregos. Em outras palavras, ele diz se os pregos i e j estão (direta ou indiretamente) conectados no *design* a.
3. Além disso, se os pregos não foram direta ou indiretamente conectados no *design* a, então é criado um novo *design* que tem todas as conexões do *design* a mais uma conexão direta adicional entre i e j. Este *design* recebe o próximo número de *design* disponível. (Assim, o primeiro *design* criado desta forma terá o número 1, depois o número 2, e assim por diante). Note que isso não altera o design a, apenas cria um novo design que tem a conexão adicional.

Seu objetivo é aprender o máximo possível sobre o *design* 0, utilizando esta operação.

Note que nem sempre é possível determinar o conjunto exato de conexões para o *design* 0, pois não há como distinguir conexões diretas e indiretas. Por exemplo, considere os dois *designs* a seguir com $n = 3$:





O verificador relataria qualquer par de pregos como conectados para ambos os *designs*, portanto, não poderemos distingui-los usando o *software* descrito acima.

Seu objetivo é determinar qualquer *design* equivalente ao *design* 0. Dois desenhos são **equivalentes** se o verificador informar o mesmo resultado em ambos os *designs* para todos os pares de pregos.

Implementação

Este é um problema interativo. Você deve implementar uma função

```
void ToyDesign(int n, int max_ops);
```

que determina um *design* que é *equivalente* ao *design* 0. Sua implementação deve atingir este objetivo chamando duas funções descritas abaixo. A primeira função que você pode chamar é:

```
int Connected(int a, int i, int j);
```

onde $1 \leq i, j \leq n$, $i \neq j$, $a \geq 0$ e a não deve exceder o número de *designs* criados até agora. Se os pregos i e j estiverem (direta ou indiretamente) conectados no *design* a, então ela retornará a. Caso contrário, ela retornará o número de *designs* criados até agora mais um, que se torna o número atribuído ao novo *design* que tem todas as conexões do *design* a mais a conexão entre i e j. A função `Connected` pode ser chamada `max_ops` vezes.

Quando seu programa terminar de fazer as operações `Connected`, ele deve descrever um *design* que seja equivalente ao *design* 0. Para descrever um *design*, o programa deve chamar:

```
void DescribeDesign(std::vector<std::pair<int,int>> result);
```

O parâmetro `result` é um *vector* de *pairs* de inteiros que descreve as conexões diretas entre os pregos. Cada par corresponde a uma conexão e deve conter os dois números dos pregos da conexão. Deve haver no máximo uma conexão direta entre cada par (não ordenado) de pregos e

nenhuma conexão direta entre um prego e ele mesmo. Chamar esta função termina a execução do seu programa.

Restrições

- $2 \leq n \leq 200$

Pontuação

- Subtarefa 1 (10 pontos): $n \leq 200$, *max_ops* $= 20\,000$
- Subtarefa 2 (20 pontos): $n \leq 8$, *max_ops* $= 20$
- Subtarefa 3 (35 pontos): $n \leq 200$, *max_ops* $= 2\,000$
- Subtarefa 4 (35 pontos): $n \leq 200$, *max_ops* $= 1\,350$

Exemplo de Interação

Ação da competidora	Ação do corretor	Explicação
	`ToyDesign(4, 20)`	Há 4 pregos no brinquedo. Você precisa determinar qualquer *design* que seja equivalente ao *design* 0, chamando `Connected` no máximo 20 vezes.
`Connected(0, 1, 2)`	`Retorna 1.`	Os pregos 1 e 2 não são conectados direta ou indiretamente no *design* 0. O novo *design* 1 é criado.
`Connected(1, 3, 2)`	`Retorna 2.`	Os pregos 3 e 2 não estão ligados direta ou indiretamente no *design* 1. O novo *design* 2 é criado.
`Connected(0, 3, 4)`	`Retorna 0.`	Os pregos 3 e 4 são conectados direta ou indiretamente no *design* 0. Não é criado nenhum *design* novo.
`DescribeDesign({{3, 4}})`	-	Descrevemos um *design* que tem apenas uma conexão: Os pregos 3 e 4.

Corretor Exemplo

O corretor exemplo fornecido, `grader.cpp`, no anexo da tarefa `ToyDesign.zip`, lê a entrada da entrada padrão no seguinte formato:

- A primeira linha contém o número de pregos, n, o número de conexões diretas, m e *max_ops*.
- As m linhas seguintes contêm conexões diretas como pares de pregos.

O corretor exemplo lê a entrada e chama a função `ToyDesign` na solução da usuária. O corretor imprimirá uma das seguintes mensagens, com base no comportamento de sua solução:

- "Wrong answer: Number of operations exceeds the limit.", se o número de chamadas de `Connected` for maior que *max_ops*
- "Wrong answer: Wrong design id.", se o parâmetro a de uma chamada de `Connected` é o número de um *design* que não existe no momento em que a chamada foi feita.
- "Wrong answer: Incorrect design.", se o *design* descrito por `DescribeDesign` não é equivalente ao *design* 0.
- "OK!" se o *design* descrito por `DescribeDesign` é equivalente ao *design* 0.

Para compilar o corretor exemplo com a sua solução, você pode usar o seguinte comando no terminal de comando:

```
g++ -std=gnu++11 -O2 -o solution grader.cpp solution.cpp
```

onde `solution.cpp` é seu arquivo de solução a ser submetido no CMS. Para executar o programa com a entrada exemplo fornecida no anexo, digite o seguinte comando a partir do terminal de comando:

```
./solution < input.txt
```