Toy Design

Oppgavenavn	ToyDesign
Inputfil	Interaktiv oppgave
Outputfil	Interaktiv oppgave
Tidsbegrensning	1 sekund
Minnebegrensning	256 megabytes

Du arbeider for et selskap som designer leker. En ny leke som lages fungerer som følgende: Det er n kontaktplugger (engelsk: "pins"), nummerert fra 1 til n, stikkende ut av en boks. Noen par av kontaktplugger er koblet sammen med ledninger inni boksen. (Med andre ord danner kontaktpluggende og ledningene en graf, hvor kontaktpluggene er nodene og ledningene kantene i grafen.) Ledningene er ikke synlige fra utsiden, og den eneste måten å finne ut noe om dem er å benytte en **tester** på kontaktpluggene: Vi kan velge to kontaktplugger i og j slik at $i \neq j$, og testeren vil fortelle oss om disse kontaktpluggene er koblet sammen direkte eller indirekte inne i boksen. (Testeren forteller oss altså om det eksisterer en sti mellom disse kontaktpluggene i grafen.)

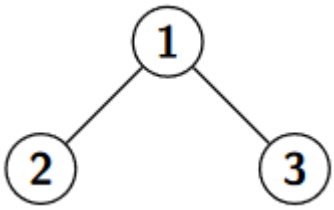
Vi kaller mengden koblinger inni boksen lekens **design**.

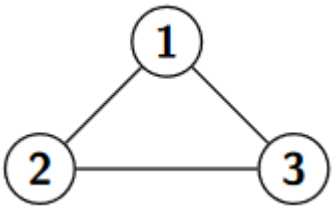
Du benytter spesialisert programvare for å lese og lage slike design. Programvaren fungerer slik: Den begynner med et spesifikt design av en leke som vi kaller "design 0". Den viser deg ikke koblingene inni boksen, men lar deg flere ganger utføre følgende trestegsoperasjon:

1. Du velger et designnummer a og to kontaktplugger i og j slik at $i \neq j$.
2. Programvaren forteller deg hva som ville skjedd om vi benyttet en tester på disse to kontaktpluggene. Med andre ord forteller den deg om kontaktplugg i og j er (direkte eller indirekte) koblet i design a.
3. Til slutt, hvis kontaktpluggene ikke var koblet hverken direkte eller indirekte i design a, lager programvaren et nytt design som har alle kontaktene til design a pluss en direkte kobling mellom i og j. Dette designet blir nå tildelt neste tilgjengelige designnummer. (Det første designet laget på denne måten vil ha designnmmer 1, så 2, osv.). Merk at dette ikke endrer design a, det bare lager et nytt design med den nye koblingen.

Målet ditt er å lære så mye som mulig om design 0 ved å bruke denne operasjonen.

Merk at det ikke alltid er mulig å bestemme nøyaktig hvilke koblinger som eksisterer for design 0, da det ikke er mulig å skille direkte og indirekte koblinger. Betrakt for eksempel følgende to design med $n = 3$:





Testeren ville rappotert at ethvert par av kontaktplugger er koblet for begge desingene, og man kan dermed ikke skille dem ved hjelp av programvaren beskrevet over.

Målet ditt er å finne et design som er ekvivalent til design 0. Vi sier at to design er **ekvivalente** om testeren rappoterer samme resultat i begge designene for ethvert par av kontaktplugger.

Implementasjon

Dette er en interaktiv oppgave. Du må implementere en funksjon

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

som bestemmer et design som er *ekvivalent* til design 0. Implementasjonen din skal oppnå dette målet ved å kalle to funksjoner som er beskrevet under. Den første funksjonen du kan kalle er:

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

Hvor $1 \leq i, j \leq n$, $i \neq j$, $a \geq 0$, og a ikke må overstige antallet design laget så langt. Hvis kontaktpluggene i og j er (direkte eller indirekte) koblet, vil funksjonen returnere a. Hvis ikke vil den returnere antallet design laget så langt pluss en, som blir nummeret tilordnet deisnget som har alle koblingene fra a pluss en kobling mellom i og j. Funksjonen `Connected` kan bare bli kalt maksimalt `max_ops` ganger.

Når programmet ditt er ferdig med `Connected`-operasjonene, skal det beskrive et deisgn ekvivalent til design 0. For å beskrive et design, skal programme kalle:

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

Parameteret `result` er en vector av par av heltall som beksriver de direkte koblingene mellom koblingspunktene. Hvert par korresponderer til en kobling og skal inneholde nummeret til de to kontaktpluggene som er koblet. Det skal ikke være mer enn én kobling mellom hvert (uordnet) par av kontaktplugger, og ingen direkte koblinger mellom en kontaktplugg og seg selv. Et kall til denne funksjonen terminerer programmet ditt.

Begrensninger

- $2 \leq n \leq 200$

Uttelling

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

Eksempelinteraksjon

Deltakerhandling	Motpart ("Grader") handling	Forklaring
	`ToyDesign(4, 5)`	Det er 4 kontaktplugger i leken. Du må bestemmet et design som er ekvivalent til design 0 ved å kalle `Connected` ikke mer enn 5 ganger.
`Connect(0, 1, 2)`	`Returns 1.`	Kontaktplugger 1 og 2 er ikke koblet direkte eller indirekte i design 0. Et nytt design, design 1, blir laget.
`Connected(1, 3, 2)`	`Returns 2.`	Kontaktplugger 3 og 2 er ikke koblet direkte eller indirekte i design 1. Et nytt design, design 2, blir laget.
`Connected(0, 3, 4)`	`Returns 0.`	Kontaktplugg 3 og 4 er koblet direkte eller indirekte i design 0. Intet nytt design lages.
`DescribeDesign({{3, 4}})`	-	Vi beskriver et design med én kobling: Kontaktplugg 3 og 4.

Test-grader

Det er gitt en *grader* for å sjekke løsninger i oppgavevedlegget `ToyDesign.zip`. Denne leser input fra standard input i følgende format:

- Første linje inneholder antallet noder, n, antallet kanter, m og *max_ops*.

- Følgende m linjer inneholder kanter som tupler av noder.

Graderen leser input og kaller `ToyDesign` i brukerens løsning. For å kompilere graderen sammen med din egen løsning kan du benytte følgende kommando i terminalen:

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

hvor `solution.cpp` er løsningen din slik den ville blitt sendt til CMS. For å kjøre programmet med testinputet gitt i vedlegget, skriv følgende kommando fra terminalen:

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