Problem 1 - ♈ Clique (Programming) (13 points)

Problem Description

♈ loves complicated things. Hence, in the graph theory, cliques are ♈'s favorite since they are the most complex structure within all simple graphs. One day, ♈ found a new interesting game called "click clique" on a gaming website. The rule of the game is described as follows.
When the game starts, you are given a simple undirected graph $G = (V, E)$. For each move, you can click a clique C of G, and it will remove those vertices and edges from the graph. That is, the graph will become the induced subgraph of $V \backslash C$ after the move. You can make arbitrary number of moves until the graph is empty. When the game ends (i.e. the graph becomes empty), you will receive stars according to the number of moves you make. Furthermore, you can receive full 3 stars only when you make the minimum number of moves. As a clique lover, ♈ can't bear to miss the full stars in the game. Thus, ♈ comes and asks you for help. Given a simple graph, please tell ♈ an optimal way to finish the game.

Input

The first line of the input contains an integer T indicating the number of testcases. For each testcase, the first line contains two integers N, M indicating the number of vertices and edges in the graph, respectively. The following M lines each contains two integers u, v describing the edges of the graph.

- $1 \leq T \leq 1000$
- $1 \leq N \leq 80$
- $0 \leq M \leq \frac{N(N-1)}{2}$
- $0 \leq u, v < N$
- $1 \leq C \leq 800$ where C is the number of maximal clique in the graph.
- It is guaranteed that the given graph is simple.
- Each test group will include its previous test groups.

Test Group 0 (0 %)

- Sample Input

Test Group 1 (10 %)

- $n \leq 7$
- $C \leq 800$

Test Group 2 (20 %)

- $n \leq 14$
- $1 \leq C \leq 100$

Test Group 3 (30 %)

- $T \leq 10$
- $n \leq 50$
- $C \leq 500$

Test Group 4 (40 %)

- $T \leq 3$

Output

For each testcase, the first line output an integer R, which is the minimum number of moves $\mathcal{P}$ needs to make.
Then each of the following R lines output the vertices that $\mathcal{P}$ should click in each move in the format of "$k\ v_1\ v_2\ \ldots\ v_k$", where k is the number of vertices in this move and $\{v_i\}$ are the indices of these vertices. You can output $\{v_i\}$ in any order.
Note that the summation of k in each testcase should be equal to N. If there are multiple optimal solutions, you can output any of them.

Sample Input 1

```
1
6 6
0 1
0 2
1 2
1 3
3 4
4 5
```

Sample Output 1

```
3
3 0 1 2
1 3
2 4 5
```

Sample Input 2

```
2
3 3
1 2
0 2
0 1
3 0
```

Sample Output 2

```
1
3 2 0 1
3
1 2
1 0
1 1
```

Sample Input 3

```
2
4 2
1 2
0 2
4 6
0 1
1 2
2 3
0 3
0 2
1 3
```

Sample Output 3

```
3
2 0 2
1 1
1 3
1
4 1 2 0 3
```

Notes

A clique is defined as a complete subgraph. That is, a set $C \subseteq V$ is a clique if and only if $\forall u, v \in C$ and $u \neq v$, $(u, v) \in E$.

A maximal clique is a clique that cannot be extended by adding any other vertex. That is, there is no proper superset of C which is also a clique. Formally speaking, a clique C is maximal if and only if $\nexists C \subset S \subseteq V$ such that S is a clique.

Hints

1. `std::bitset` is a good data structure and it might be useful in this problem.
2. You may want to use (integer) linear programming to solve this problem. Please see the following section to use the provided linear programming solver. (Of course, you are not forced to use it if you want to implement it by yourself.)

3. You are welcome to use any heuristic method to solve this problem. Happy hacking.

Linear Programming Solver

The GLPK (GNU Linear Programming Kit)[1] package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. You can include the GLPK library (`<glpk.h>`) and use it in this problem. See GLPK Manual[2] for more information about the usage.
There is also a packed version of the GLPK tools provided for you to use it easier. The full header file can be downloaded here[3]. You can include it by `#include "ypglpk.hpp"` in your solution.
There are three functions defined in the namespace "ypglpk". You can follow the instruction below to use them. All the std::vector below are 0-based. Note that if the parameters violate the restriction, the behavior is undefined.

```
std::pair<double,std::vector<double>> linear_programming(
const std::vector<std::vector<double>> &A,
const std::vector<double> &b, const std::vector<double> &c);
```

- The function `linear_programming(A,b,c)` is used to solve the standard linear programming problem.
- Parameters: $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$ where n is the number of structural variables and m is the number of constraints.
- Targets: Maximize $c \cdot x$ subject to $Ax \leq b, x \in \mathbb{R}^n$.
- Return value: pair of (optimal value $c \cdot x^*$, optimal vector x^*). If the solution is infeasible or unbounded, the return value is (negative infinity `-ypglpk::INF`, empty vector `vector<double>()`).

```
std::pair<double,std::vector<double>> mixed_integer_linear_programming(
const std::vector<std::vector<double>> &A, const std::vector<double> &b,
const std::vector<double> &c, const std::vector<bool> &isint);
```

- The function `mixed_integer_linear_programming(A,b,c,isint)` is used to solve the mixed integer linear programming problem, which can restrict some structural variables to be integers.
- Parameters: $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n, \text{isint} \in \{true, false\}^n$ where n is the number of structural variables and m is the number of constraints.
- Targets: Maximize $c \cdot x$ subject to $Ax \leq b, x \in \mathbb{R}^n$, and $\forall i$ with $\text{isint}_i = true, x_i \in \mathbb{Z}$.
- Return value: pair of (optimal value $c \cdot x^*$, optimal vector x^*). If the solution is infeasible or unbounded, the return value is (negative infinity `-ypglpk::INF`, empty vector `vector<double>()`).

```
void set_output(bool output);
```

[1] https://www.gnu.org/software/glpk/
[2] http://most.ccib.rutgers.edu/glpk.pdf
[3] http://www.csie.ntu.edu.tw/~giver/ADA/hw4/ypglpk.hpp

- The function `set_output(output)` is used to set whether GLPK to output verbose information about the (MI)LP solver.
- The default setting is `output=false`. You may enable it to debug your code with more information about the constraints and the solver. **However, please do not set it to `true` when you submit to the judge; otherwise, you will certainly get `Wrong Answer` verdict due to the unexpected output.**

Local Testing

Here[4] is a simple example code to demonstrate how to use the provided header file. It should be able to be compiled and run successfully on the judge (although you will receive `Wrong Answer`).
The installed GLPK version on the judge system is 5.0. (The version of the GLPK package does not affect the correctness. It only affects the performance in some cases since the later version might have better presolver and branching strategy.)
For CSIE students, you are highly recommended to run the program on the CSIE workstation, as there is already an installed GLPK 5.0 package for you. You can simply put your source code and the given header file together and compile them with the command below Compilation.
Otherwise, to compile and run the GLPK library successfully, please refer to the Appendix GLPK Installation to install it.

Compilation

After having an environment with installed GLPK package, you can compile the provided codes (or your source code) by adding the additional linker argument `-lglpk` to the compilation command. **Note that the order of the arguments is essential. The linker argument should be put after the source code.** For example (assume that the filename of the source code is "example.cpp"):

```
g++ -std=c++17 -O2 -oexample example.cpp -lglpk -Wall
```

Demonstration

You can try to compile and run the provided example program to ensure there is no problem with the package. Below is a script to download the provided files, compile them, and run the program. (Note that the compilation command and the execution command might differ if you built the package from source by yourself. See the Appendix GLPK Installation for more information.)

```
curl -o example.cpp 'https://www.csie.ntu.edu.tw/~giver/ADA/hw4/example.cpp'
curl -o ypglpk.hpp 'https://www.csie.ntu.edu.tw/~giver/ADA/hw4/ypglpk.hpp'
g++ -std=c++17 -O2 -oexample example.cpp -lglpk -Wall
./example
```

And the expected output of the program execution should be like figure 1.

[4] http://www.csie.ntu.edu.tw/~giver/ADA/hw4/example.cpp

```
giver@GiverArchLinux /t/tmp.7DFD7aBTUt> curl -o example.cpp 'https://www.csie.ntu.edu.tw/~giver/ADA/hw4/example.cpp'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   976  100   976    0     0   3728      0 --:--:-- --:--:-- --:--:--  3725
giver@GiverArchLinux /t/tmp.7DFD7aBTUt> curl -o ypglpk.hpp 'https://www.csie.ntu.edu.tw/~giver/ADA/hw4/ypglpk.hpp'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2898  100  2898    0     0  55511      0 --:--:-- --:--:-- --:--:-- 55730
giver@GiverArchLinux /t/tmp.7DFD7aBTUt> g++ -std=c++17 -O2 -oexample example.cpp -lglpk -Wall
giver@GiverArchLinux /t/tmp.7DFD7aBTUt> ./example
GLPK Simplex Optimizer 5.0
4 rows, 3 columns, 10 non-zeros
Preprocessing...
4 rows, 3 columns, 10 non-zeros
Scaling...
 A: min|aij| =  1.000e+00  max|aij| =  3.000e+00  ratio =  3.000e+00
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part is 4
*     0: obj =  -0.000000000e+00 inf =   0.000e+00 (3)
*     3: obj =   1.273823529e+01 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
LP: max=12.7382 with x=3.63529 y=0.941176 z=-2.88824
GLPK Integer Optimizer 5.0
4 rows, 3 columns, 10 non-zeros
2 integer variables, none of which are binary
Preprocessing...
4 rows, 3 columns, 10 non-zeros
2 integer variables, none of which are binary
Scaling...
 A: min|aij| =  1.000e+00  max|aij| =  3.000e+00  ratio =  3.000e+00
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part is 4
Solving LP relaxation...
GLPK Simplex Optimizer 5.0
4 rows, 3 columns, 10 non-zeros
*     0: obj =  -0.000000000e+00 inf =   0.000e+00 (3)
*     3: obj =   1.273823529e+01 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
Integer optimization begins...
Long-step dual simplex will be used
+     3: mip =     not found yet <=              +inf        (1; 0)
+     6: >>>>>   1.130000000e+01 <=   1.130000000e+01   0.0% (3; 0)
+     6: mip =   1.130000000e+01 <=     tree is empty   0.0% (0; 5)
INTEGER OPTIMAL SOLUTION FOUND
MILP: max=11.3 with x=1.3 y=0 z=-4
```

Sample output of the example.cpp program