3SAT $\leq_P$ Clique $\leq_P$ Vertex-Cover

Due: March 7, 2024, Thursday, 11:55 PM **50 points (45 on programs, 5 on report)**

Discuss with your classmates and find two partners to form a team of 3 students for this programming assignment. Each team should designate a *corresponding student* who will be in charge of submitting the report and preparing the programs on our Linux server, where your programs will be compiled and tested. Also, the corresponding student should answer my questions and deliver my comments to the team. After the team is formed, use Canvas (under People/Group) to join the team.

Programming tasks and team members' major responsibilities: Let the team members be students A, B, and C. The major programming responsibility are divided as follows but not limited to them. Every method developed by any team member can be shared within the team. Note that, the following description is not about the program's input and output, but the tasks of some major methods. Keep the input and output formats consistent will simplify communication cost between team members' methods.

- **Student A**: Solve $\varphi \in$ 3SAT by reduction 3SAT $\leq_P$ Clique.

  Student A should develop a method that will take a 3CNF φ and decide if $\varphi \in$ 3SAT. If $\varphi \in$ 3SAT, the method should return an assignment that satisfies φ. However, student A should not solve the problem directly. Instead, he/she should reduce φ in polynomial time to a corresponding Clique problem (i.e., a graph and $k \in N$) for student B to solve and use the results (some k-clique, if exists) to answer the original question. In short, student A should implement the reduction for 3SAT $\leq_P$ Clique.

- **Student B**: Solve does $G : (V, E)$ have a k-clique by reduction Clique $\leq_P$ Vertex-Cover.

  Student B should develop a method that will take an undirected graph $G : (V, E)$ and an integer k and return a k-clique of G, if exists. Same as student A, student B should not solve the problem directly. Instead, he/she should reduce the input graph $G : (V, E)$ and k in polynomial time to corresponding k-vertex cover problem (i.e., another a graph $G : (V', E')$ and k') for student C to solve. If a k'-vertex cover of G' exists, use it to answer the original problem. In short, student B should implement the reduction for Clique $\leq_P$ Vertex-Cover.

- **Student C**: Solve does $G : (V, E)$ have a k-vertex cover.

  Student C should develop a method that will take an undirected graph $G : (V, E)$ and an integer k and return a k-vertex cover of G, if such k-vertex cover exists. A common technique for this problem is backtracking unless you have a better method, which is highly unlikely. Note that, while we do not know if all NP problems can be solved in polynomial time but we do know all NP problem can be solved in polynomial space, also known as PSpace (i.e. use only polynomial amount of memory, which is a superset of NP). Thus, you should not generate all possible subsets of V and check one by one, since it will use exponential space.

- In addition to the major tasks described above, there are some minor but necessary methods to implement, such as methods to parse the text files into your internal data structures, to take care of the required output formats, to upload and test the programs on our Linux server, and so on. Also, each team is required to submit a report. You should reach an agreement to distribute the workload fairly. For example, student B's main task is rather easy, he/she may take of more chores.

Input file formats: There are two text files your programs need to read, one contains undirected graphs and the other contains boolean formulas in 3CNF. You can use any data structures (including standard Java packages) in your program to handle graphs and boolean formulas in 3CNFs once they are read in from the files (or created by your programs). You don't have to output any files. The two input files are described as follows.

- **3CNF file:** `cnfs2024.txt`

  In this text file, each line represents a 3CNF, where the positive numbers 1, 2, . . . represent boolean variables a_1, a_2, . . . and the negative numbers $-1, -2, \ldots$ represent negation of boolean variables, $\neg a_1, \neg a_2, \ldots$. Since each line is a 3CNF, the numbers should be grouped 3 by 3, and each group is a clause. For example,

  `1 3 -2 3 6 4 -4 2 1` $\Longrightarrow$ $(a_1 \vee a_3 \vee \neg a_2) \wedge (a_3 \vee a_6 \vee a_4) \wedge (\neg a_4 \vee a_2 \vee a_1)$.

  You may use the largest index to indicate the number of variables in the 3CNF. For example, the number of variables in the formula above is 6. It should not be a problem if some variables with smaller indices are missing from the formula.

- **Graph file:** `graphs2024.txt`

  In this text file, there are a sequence of graphs. A graph $G : (V, E)$ with $|V| = n$ is stored in $(n+1)$ lines. For each graph, the first line contains a number n to indicate the size of V. The following n lines represent the *adjacency matrix* m of G (i.e., m is an $n \times n$ two dimensional array). If the entry $m[i][j] = 1$, that means $(i, j) \in E$; if $m[i][j] = 0$ means $(i, j) \notin E$.

  One can see that all matrices in this file are symmetric, i.e., for every $0 \leq i, j < n$ we have $m[i][j] = m[j][i]$, which mean undirected. In the context of undirected graphs, if $m[i][j] = m[j][i] = 1$, we count as one edge in E. Also note that, the diagonal of the matrix is set to 1, i.e. $m[i][i] = 1$ as we consider every vertex links to itself but we do not count it as an edges, thus (v, v) does not contribute to the edge count, $|E|$. The file is ended by an empty graph, i.e., $n = 0$.

What to do and program output format:

1. Use our terminal server to login your Linux account. Let $\sim$ denote your home directory. Make directory `~/IT328/npc/`. All files related to this assignment should be prepared in this directory. Note that, Unix is case sensitive, name your directory and programs precisely.

2. From my `/home/ad.ilstu.edu/cli2/Public/IT328/npc/`, copy `graphs2024.txt` and `cnfs2024.txt` to your `~/IT328/npc/`.

3. Develop three independent Java programs: `find3SAT.java`, `findClique.java`, and `findVCover.java` described in the following. Students are free to create other programs or classes if that helps but the

three main programs need to be named as required. All programs will be compiled and tested on our Linux server using Unix command line.

Note: This doesn't mean students A, B and C have to develop their programs independently. You should develop all program together, because each program needs other's methods. For example, student A's program (`find3SAT.java`) needs student B's method to solve Clique problem, and student B's program (`findClique.java`) needs student C's method to solve Vertex Cover problem.

- `find3SAT.java`
  This program reads in every 3CNF in the inputs file and for each 3CNF φ, determines if φ is satisfiable, and if it is, prints an assignment that satisfies φ. Your program will be compiled and run with an input file argument as shown in the following.

```
javac find3SAT.java
java find3SAT cnfs2024.txt
```

  The output will be shown on the screen in the following format:

```
* Solve 3CNF in cnfs2024.txt:  (reduced to k-clique) *
X means can be either T or F

............

3CNF No.3:  [n=4 c=3] ==> Clique:  [V=9 E=25 k=3]
In 0 ms, find solution [1:F 2:T 3:X 4:F]
( 2|-1|-1)∧(-3|-2|-4)∧( 4|-3|-1)
( T| T| T)∧( X| F| T)∧( F| X| T)
............

3CNF No.7:[n=3 c=6] ==> Clique:  [V=18 E=110 k=6]
In 2 ms, find no solution!  Random assignment [1:F 2:T 3:T]
( 1|-2| 3)∧(-2| 2| 1)∧( 2| 2| 1)∧(-3|-3|-1)∧(-2|-3|-3)∧(-1|-1| 3) ==>
( F| F| T)∧( F| T| F)∧( T| T| F)∧( F| F| T)∧( F| F| F)∧( T| T| T)

3CNF No.8:[n=3 c=7] ==> Clique:  [V=21 E=157 k=7]
In 0 ms, find solution:[1:T 2:T 3:F]
( 2| 1|-1)∧( 3| 3| 2)∧( 1| 2|-2)∧( 2|-3| 1)∧( 3| 1| 3)∧(-2| 1|-3)∧(-2|-1|-3) ==>
( T| T| F)∧( F| F| T)∧( T| T| F)∧( T| T| T)∧( F| T| F)∧( F| T| T)∧( F| F| T)

............
***
```

  The output indicates that every 3CNF in the file is reduced to a clique problem. Take 3CNF No. 3 as an example. The output should show that the 3CNF has 4 boolean variables and 3 clauses, and is reduced to a graph $G(V, E)$ with $|V| = 9, |E| = 25$ and determine the size of the clique $k = 4$. Since a k-clique exists, the next line show how much time in million seconds your program used to find the clique and use it to find the assignment that satisfies the 3CNF, which is `[1:F 2:T 3:X 4:F]`, i.e., $a_1 = false$, $a_2 = true$, $a_4 = false$, and `X` for a_3 means it can be either *true* or *false*. The next line shows the 3CNF using parenthesis to group clauses and `|` for logical **or** and $\wedge$ for logical **and**. Then, use another line to show the evaluation of each literal using the same assignment. For another example, 3CNF No. 7, since its corresponding graph does not have a k-clique, it is *unsatisfiable*, i.e., there is no solution. In this case, select a random assignment, for example `[1:F 2:T 3:T]` and evaluate the 3CNF and expect at least one clause is *false*, i.e., `( F| F| F)`.

- `findClique.java`
  This program finds and prints a maximum clique of each graph in the input file as follows:

  ```
  javac findClique.java
  java findClique graphs2024.txt
  ```

  The output will be shown on the screen in the following format:

  ```
  * Max Cliques in graphs in graphs2024.txt (reduced to K-Vertex Cover) *
     (|V|,|E|) (size, ms used) Cliques
  G1 ( 5, 8) (size=4 ms=0) {0,1,3,4}
  G2 ( 5, 7) (size=3 ms=0) {0,1,4}
  G3 (10, 21) (size=5 ms=0) {1,4,6,7,9}
  ............
  G50 (125,3882) (size=9 ms=289) {0,6,18,24,68,85,90,91,110}
  ***
  ```

  Using `G1` as an example, `(5, 8)` indicates the number of vertices (`|V|`=5) and the number of edges (`|E|`=8), followed by the size the maximum clique, milliseconds needed the program ran, and the 4-clique.

- `findVCover.java`
  This program finds and prints a minimum vertex cover of each graph in the input file as follows:

  ```
  javac findVCover.java
  java findVCover graphs2024.txt
  ```

  The output will be shown on the screen in the following format:

  ```
  * A Minimum Vertex Cover of every graph in graphs2024.txt *
     (|V|,|E|) (size, ms used) Vertex Cover
  G1 ( 5, 8) (size=3 ms=0) {0,3,4}
  G2 ( 5, 7) (size=2 ms=0) {1,4}
  G3 (10, 21) (size=6 ms=0) {0,1,5,6,7,9}
  ............
  ............
  G50 (125,3882) (size=115 ms=435) {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,19,20,21,2
  ***
  ```

  The output format is same `findClique.java`.

Prepare your programs in the Linux server and submission:

1. When the project is done, make sure all programs are put in the directory `~/IT328/npc/` of corresponding student's Linux account. Then, select a secret name, say "`peekapoo`" as an example (you

should chose your own, but no more than 8 characters and no space and special symbols, and this secret name should not be shared with anyone except the team members and the instructor as it will be used as a secret directory to store your programs.) Run the following bash script program from corresponding student's Unix account:

```
bash /home/ad.ilstu.edu/cli2/Public/IT328/npc/submitnpc.sh peekapoo
```

Note that your programs have to be in the required directory `~/IT328/npc/` as described in step 1 in order to let this script program correctly copy your programs into the secret directory, where your programs will be tested. If you modify your programs, you have to run `submitnpc.sh` again. Don't compile and run your programs from the directory where `submitnpc.sh` copies to. Also, unix is case sensitive. Name you directory and program files correctly.

2. **Report.** Each team has to write up a report and submit a copy of the report through corresponding student's Canvas account. The report should include:

    - A cover page that contains assignment number, team members' names and indicate who is the corresponding student, ULID (not student ID) and the name of the secret directory.
    - A brief summary of the methods, algorithms and data structures, and the difficulties, if any, the project has faced and how to solve them.
    - Do not include the program code and the output in the report.

    The score is based on the correctness and documentation of your program, 50 points (45 on programs, 5 on report).