**Algorithm Design and Analysis**

**Homework #5**

**Due: 2:20pm, Monday, December 23, 2013**

TA email: ada@csie.ntu.edu.tw

**=== Homework submission instructions ===**

- For Problem 1, commit your source code to the SVN server (katrina.csie.ntu.edu.tw). You should create a new folder "`hw5`" and put the source code file in it. The filename of the source code should be "`main.c`"; you will get some penalties in your grade if your submission does not follow the naming rule.
  You have to login to judgegirl system (http://katrina.csie.ntu.edu.tw/judgegirl) to make it judge your program.
- You need to submit your `report of Problem 1` and your `answer to other Problems` via the SVN server (electronic copy, all in a single pdf file named "`hw5_[student ID].pdf`") or to the TA at the beginning of class on the due date (hard copy).
- For hard copies
  - Please use A4 paper.
  - Please write down `your name` and `the page number` on every page of your homework.
  - Please do not use paper with things printed on the other side.
    You are encouraged to write down your answers on both sides of the paper.
  - Please don't use staples to bind you papers.
    The TA will provide you paperclips when you hand in your homework.
  - Please take your homework out of the L-folder when you hand it in, or you might lose your L-folder.

  You may get some penalties in your grade if you do not follow the above rules.
- Discussions with others are encouraged. However, you should write down your solutions by your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem.
- **NO LATE SUBMISSION IS ALLOWED** for the homework submission in hard copies - no score will be given for the part that is submitted after the deadline. For submissions via SVN (including the programming assignment and electronic copies of the writing problems), up to one day of delay is allowed; however, the score of the part that is submitted after the deadline will get some penalties according to the following rule (the time will be in seconds):

$$\text{LATE SCORE} = \text{ORIGINAL SCORE} \times (1 - DelayTime/86400)$$

## *Problem* 1. Travelling salesman problem

(30%)

As time goes by, you have understood the meaning and hardness of NP-complete problems. In this homework, you have to tackle a programming assignment with a very different format. First, you have to calculate the time complexity. Then, given the computational power of a computer, you will calculate the consumed time for solving an NP-hard problem. Subsequently, you will find that it is not possible to find the exact optimal solution in real life.

Here comes the actual problem: given a list of cities and the distances between each pair of cities, what is the "shortest" possible route that visits each city exactly once and returns to the origin city? Note that the route is in fact a cycle.

### Calculation of time complexity (3%)

If you have stored the distance between every two cities in a table, what is the time complexity you need to find the optimal solution of travelling salesman problem?

### Calculation of time needed to solve the problem (2%)

Given the distance table between every two cities, suppose that in our computer an addition operation (the '+' operation) takes $2 \times 10^{-12}$ second, or 2 picoseconds, and comparison takes no time (0 second). How many years does it need to calculate the optimal solution for 40 cities?

### Coding (25%)

Since NP problems are not solvable within a reasonable time, it is necessary to use some other method to approximate the exact optimal solution. So we want to introduce the heuristic algorithm to you: "Heuristic refers to experience-based techniques for problem solving, learning, and discovery that give a solution which is not guaranteed to be optimal. Where the exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution via mental shortcuts to ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, stereotyping, or common sense." - Wikipedia

And there is a method used in Artificial Intelligence called "Genetic algorithm," which is designed to simulate the process in natural systems for evolution. Hopefully, it can be used to solve the Travelling salesman problem:

`http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html`

As we mentioned in the lectures, in your future working or learning environments, it is always necessary for you to have the ability to quickly learn and implement an existing algorithm developed by others. So we decided to give you the opportunity to develop this ability. You will have to study the given on-line material of the Genetic algorithm and learn it by yourself. Please implement the given Genetics algorithm to solve the Travelling salesman problem.

**Note:**

1. You have to confirm that the children inherited by the crossover are indeed a valid cycle.
2. There is a high probability that your output is not the same as the sample output. There is no need to worry about this since you will pass the test on judgegirl as long as your output is within a certain margin of the result generated by the TA's program.

**Input format:**

The first line contains the total number of cities $N$, $N \leq 50$. The following $N$ lines contain the X and the Y coordinates of each city, separated by a space character. The given coordinates are all floating point numbers.

**Output format:** Please output the solution of the cycle using your heuristic method.

**Sample input:** 10

1.0 1.0

2.0 2.0

3.0 3.0

8.0 3.0

2.0 9.0

1.0 10.0

9.0 3.0

-13.0 94.0

-81.0 34.0

3.0 4.0

**Sample output:**

7 8 9 6 5 3 2 1 10 4

***Problem* 2.** Birds in a graph (20%)

Let us revisit the problem "birds in the tree" in HW2, but this time birds may live in a general undirected graph.

The detailed problem description is as follows. Given an undirected graph $G = (V, E)$, where $|V| = n$, there are many birds living on its vertices. Each vertex has its own capacity $c_i$, $i \in V$, which limits the number of birds on the vertex. These birds have a strange habit. The bird stays peace with the other birds which live on the same vertex. But If there is any other bird that lives on the vertex which is directly connected to (i.e., that has a edge to) the vertex where it lives, they will start a fight. The goal is to find the maximum number of birds in the given graph.

1. (5%) This problem is an optimization problem. Please define a corresponding decision problem. Furthermore, show that *if* your decision problem belongs to $P$, this problem (i.e., the original optimization problem) can be solved in polynomial time.
2. (5%) Show that your decision problem belongs to $NP$.
3. (7%) Show that 3-CNF-SAT is polynomial-time reducible to your decision problem. (When you are working on this problem, we might not have yet covered this part in the lecture. Please study the slides and the related textbook sections before you work on this problem.)
4. (3%) Show that your decision problem is NP-complete. You can use the fact that 3-CNF-SAT is a known NP-complete problem.

***Problem* 3.** MCS (15%)

Show that the Maximum Common Subgraph problem is NP-Complete.

MAXIMUM COMMON SUBGRAPH

Input: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

Question: Find the largest graph $H$ such that $H$ is an induced subgraph of both $G_1$ and $G_2$. $H$ is an induced subgraph of $G$ if it has exactly the edges that appear in $G$ over the vertex set of $H$.

To show that the problem $L \in NP$-Complete, we can show that that $L \in NP$ and $L \in NP$-hard.

1. (5%) Show that $L \in NP$.
2. (10%) Show that the Maximum Clique Problem can be reduced into Maximum Common Subgraph problem. Maximum Clique Problem is that for a graph $G = (V,\ E)$, find the maximal complete subgraph in it, that is, we need to find the maximal subgraph that each vertex is edged with all other vertices in the subgraph. In your proof, you can assume that you know that the Maximum Clique Problem is $NP$-Complete, and use it to prove that the Maximum Common Subgraph problem $\in NP$-hard.

***Problem*** 4. (15%)

Show that any language in NP can be decided by an algorithm running in time $2^{O(n^k)}$ for some constant $k$.

***Problem*** 5. Vertex Cover(20%)

Given an undirected simple graph, $G = (V, E)$, a subset of vertices $V_C \subseteq V$ is a *vertex cover* of $G$ if and only if for each edge $(u, v) \in E$, either or both of $u$ and $v$ is in $V_C$.

A *minimum vertex cover* of an undirected simple graph, $G$, is a vertex cover of $G$ with minimum cardinality (i.e., the size of the set).

1. (4%) Find a minimum vertex cover of the graph in Figure 1.

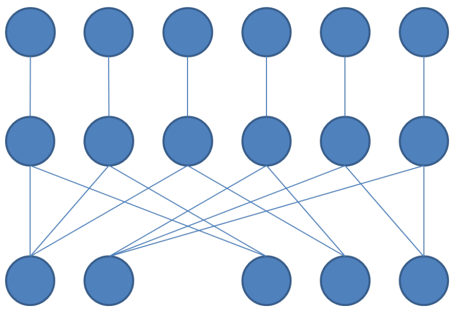

Figure 1: an undirected simple graph

definition of $k$-VERTEX COVER problem and VERTEX COVER problem:

$k$-VERTEX COVER: Given a positive integer, $k$, and an undirected simple graph, $G$, is there a vertex cover of $G$ with cardinality less than or equals to $k$?

VERTEX COVER: Given a undirected simple graph, $G$, find a minimum vertex cover of $G$.

$k$-VERTEX COVER is an NP-Complete problem. (See `http://www.csc.kth.se/utbildning/kth/kurser/DD1352/adk12/alvie/vertexCoverNPCompleteness.pdf` for the proof.)

If we can find a *minimum vertex cover* of the given graph, we can solve $k$-VERTEX COVER. It means that $k$-VERTEX COVER is reducible to VERTEX COVER and VERTEX COVER is therefore in NP-hard (i.e., problems that are "at least as hard as the hardest problems in NP". See `http://en.wikipedia.org/wiki/NP-hard` for more information).

Because we don't know how to solve VERTEX COVER in polynomial time, we want to design a polynomial time algorithm that can find a "small enough" vertex cover.

2. (8%) Now we purpose a greedy algorithm:

   (a) Let $C = \emptyset$

   (b) Find the vertex $v$ with greatest degree

   (c) Add $v$ into $C$

(d) Remove all edges connected to $v$

(e) Repeat (b)(c)(d) until all edges are removed.

(f) Output $C$

Let $V_C^*$ be a minimum vertex cover, $V_C$ be a vertex cover constructed by the greedy algorithm. Is there a constant number $k$ such that $k * |V_C^*|$ would always be an upper bound of $|V_C|$? If yes, please give the number $k$ and a proof. If not, please give a counterexample.
(Hint: Figure 1)

3. (8%) Now we have another algorithm:

   (a) Let $C = \emptyset$

   (b) Find a edge $(u, v)$

   (c) Add $u$ and $v$ into $C$

   (d) Remove all edges connected to $u$

   (e) Remove all edges connected to $v$

   (f) Repeat (b)(c)(d)(e) until all edges are removed.

   (g) output $C$

   Let $V_C^*$ be a minimum vertex cover, $V_C$ be a vertex cover constructed by the greedy algorithm. Is there a constant number $k$ such that $k * |V_C^*|$ would always be an upper bound of $|V_C|$? If yes, please give the number $k$ and a proof. If not, please give a counterexample.

***Problem*** 6. (15%)

Please choose a user interface that you don't like, and redesign its interface with paper prototyping (as we discussed in the lecture). The user interface can be the one for a software program (an smartphone app, a website, etc.), or for an electronic appliance (the control panel for the air conditioner, or the remote for the TV). Test your prototype with 3 different users, one of which must not be a person with computer science or electrical engineering background.