# Algorithm Design and Analysis
# Homework #6

Due: 11:59am January 07, 2013

## Homework submission instructions

- Submit your programming assignment (problem 1) to the Judgegirl System (http://katrina.csie.ntu.edu.tw/judgegirl/). Also, use SVN to commit your report (named "report.pdf") in the same folder of your code.
- Submit the answer of the writing problems by one of the following ways:
  1. Submit an electronic copy to the CEIBA system before the deadline; or
  2. Submit a hard copy right before the class on the day of the deadline.

  If you submit by both ways, the TA will randomly select only one of them for grading.
- Please make sure that you write down your name and school ID in the header of your documents. Otherwise, the TA will not grade them.
- If you choose to submit the answers of the writing problems via CEIBA, please combine the answers of all writing problems into only one file in the pdf format, with the file name in the format of "hw6 [ID].pdf" (e.g. "hw6 b99902010.pdf"). Otherwise, the TA will not grade them.
- What should be included in the report:
  1. Explain how your program works in detail.
  2. Derive the time complexity of your program and briefly explain why. No need for formal proof.
  3. The reference of your report.
- If you are stuck with any problem, you are welcome to ask the TAs via e-mails or in person during the office hour. The TA may discuss with you and give you some useful hints. :D

## Problem 1 – Queen

### Description

There is an N*N chessboard and N queens. Some queens have already been placed on the chessboard and you cannot change their positions. In this problem, you need to find out how many ways are there for you to place other queens on the chessboard so that no two queens can attack each other. A queen can attack others in positions that are in the same row, the same column, or the same diagonal line.

For example:
(The positions that can be attacked by the queen 'Q' are marked as 'x')

```
. . . x . x . x
. . . . x x x .
x x x x x Q x x
. . . . x x x .
. . . x . x . x
. . x . . x . .
. x . . . x . .
x . . . . x . .
```

A solution for N=8:

```
. . . Q . . . .
. . . . . . Q .
. . Q . . . . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .
Q . . . . . . .
. . . . . Q . .
```

### Input

The first line contains two integers, N and P, indicating the size of the chessboard and the number of the queens that have already been placed on the chessboard (4 <= N <= 19). Each of the following P lines contains two integers, $R_i$ and $C_i$, which means that there is a queen placed at the junction of the ($R_i$+1)-th row and the ($C_i$+1)-th column. No two queens given in the input can attack each other.

## Output

Output the number of ways to place other (N-P) queens in a single line. The answer will not be too large. The TA's program can solve every test case within 5 seconds in JudgeGirl.

## Sample Input

```
7 1
6 3
```

## Sample Output

```
6
```

## Hint

1. You must solve the problem with your program (not by hands). For example, if you only have instructions to print out the solutions in your program, you will get zero point for this problem (for both judge results & the report).
2. You are encouraged to use OpenMP directives to implement a multithread algorithm to solve this problem. (compile with '-fopenmp' option in gcc)
3. There are 4 cores in the judge machine.
4. '#pragma omp parallel for' is enough to get full points. (Some useful clause: private, reduction)
5. The TA will upload a set of slides to explain how to use OpenMP. In the next lecture, we will also spend 10-15 minutes to cover the basics.

## Grade

Judge result (10%)
Report & code (10%)

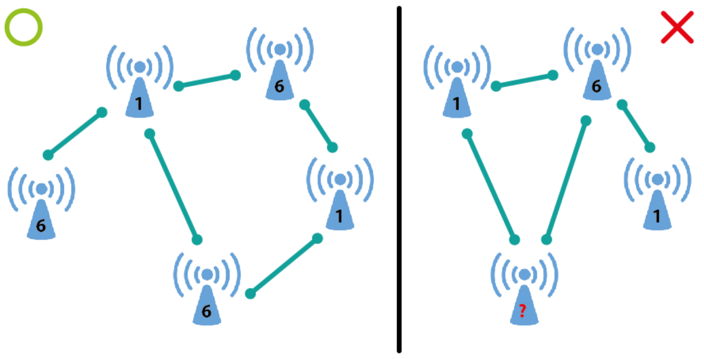
## Problem 2 – 2 is easy, 3 is hard, and how about 6?

The CSIE network administrators need to deploy many WiFi APs (access points) in the CSIE building to provide wireless Internet connections throughout the building.

This is not an easy task; if the distance between two APs is smaller than the transmission range, then these two APs would interfere with each other. Although the administrators can avoid this problem by configuring the APs to use different channels, but the number of non-overlapping channels which can be utilized by the AP is limited. For example, when IEEE 802.11b/g/n is used, there are only 3 non-overlapping channels (channel 1, 6, and 11).

We define the problem as the following. Given a number of APs, the number of channels *c*, and the interfering relations of them (whether two APs would interfere with each other), can the network administrator find a way to allocate channels to APs so that no two APs would interfere with each other?

Example: Two channels {1, 6} are available (*c*=2). The line between two APs implies that they are too close and would interfere with each other.



1) Assume that channel 1 is reserved for research purposes in the CSIE building. Thus, only channel 6 and channel 11 are available (*c*=2).
   a) (5%) Provide an algorithm to solve this problem in polynomial time.
   b) (5%) Analyze the time complexity of your algorithm. Then explain why this problem belongs to the complexity class P.
2) Now, the network administrators decide to also utilize channel 1 (*c*=3). Prove that the problem becomes a NP-Complete problem.

We can use a reduction from 3-CNF-SAT problem, which is already proven to be a NP-Complete problem in the lecture. The following outlines the reduction algorithm.

The three channels {1, 6, 11} can represent the 3 states of a variable, {True, False, Other}, respectively. Given a formula $\varphi$ of $m$ clauses on $n$ variables $x_1, x_2, \dots, x_n$, we can construct a AP network as follows:

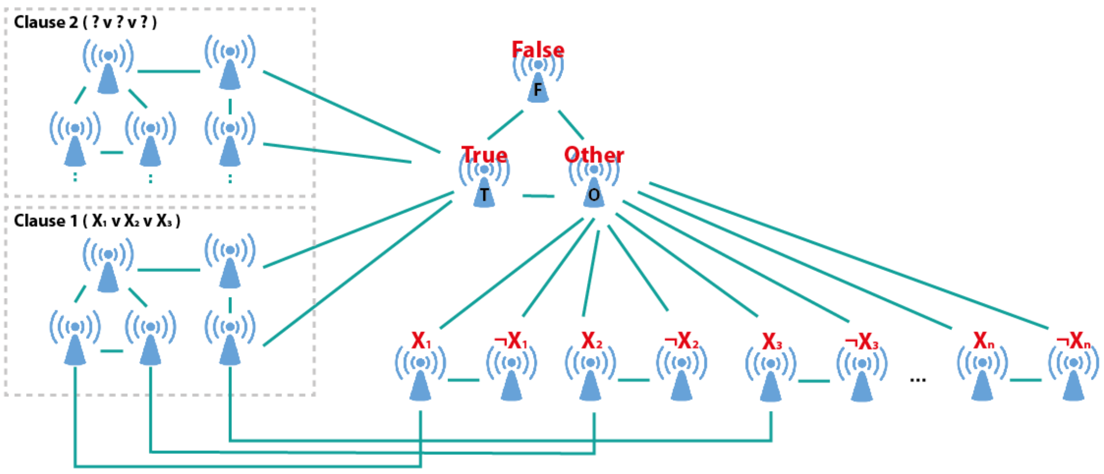
Step 1: Create 3 special APs with respectively the channels {True, False, Other}; these 3 special APs form a triangle (they interfere with each of the two other APs).

Step 2: Create APs that represents each variable x and its negation.

Step 2a: APs representing $x_i$ and its negation $\neg x_i$, for all i = 1 to n, interfere with each other.

Step 2b: Link all 2n APs created so far to the special AP that uses the channel "Other".

Step 3: Create 5 APs for each clause, and the interfering relations are as follows.



a) (3%) Show that this reduction can be done in polynomial time.

b) (15%) Now prove that $\varphi$ is satisfiable if and only if you can solve the 3-channel APs problem of the reduced instance. You can then show that this 3-channel APs problem is in NP-hard.

c) (2%) Show that this problem is in NP and NP-Complete

3) If now the APs use IEEE 802.11a instead and 6 channels, {36, 40, 44, 48, 149, 153} are available ($c = 6$). Prove that the 6-channel APs problem is also a NP-Complete problem.

a) (8%) Show that this problem is in NP-hard.

(Hint: Give a reduction from 3CH-APs to 6CH-APs)

b) (2%) Show that this problem is in NP and NP-Complete.

## Problem 3 – Multithreading reductions and prefix computations

A ⊗-reduction of an array x[1 . ..n], where ⊗ is an associative operator, is the value

$$y = x[1] \otimes x[2] \otimes \ldots \otimes x[n].$$

The following procedure computes the ⊗-reduction of a subarray x[i . . j] serially.

```
REDUCE($x, i, j$)
1  $y = x[i]$
2  for $k = i + 1$ to $j$
3      $y = y \otimes x[k]$
4  return $y$
```

1) (5%) Use nested parallelism to implement a multithreaded algorithm P-REDUCE, which performs the same function with $\Theta(n)$ work and $\Theta(lg\ n)$ span. Write your algorithm in pseudo-code and analyze your algorithm.

A related problem is that of computing a $\otimes$-prefix computation, sometimes called a $\otimes$-scan, on an array $x[1\,..\,n]$, where $\otimes$ is once again an associative operator. The $\otimes$-scan produces the array $y[1\,..\,n]$ given by

$$\begin{aligned} y[1] &= x[1]\,, \\ y[2] &= x[1] \otimes x[2]\,, \\ y[3] &= x[1] \otimes x[2] \otimes x[3]\,, \\ &\vdots \\ y[n] &= x[1] \otimes x[2] \otimes x[3] \otimes \cdots \otimes x[n]\,, \end{aligned}$$

that is, all prefixes of the array x "summed" using the $\otimes$ operator. The following serial procedure SCAN performs a $\otimes$-prefix computation:

```
SCAN(x)
1  n = x.length
2  let y[1..n] be a new array
3  y[1] = x[1]
4  for i = 2 to n
5      y[i] = y[i - 1] ⊗ x[i]
6  return y
```

2) (5%) If we simply change the "for loop" to a "parallel for loop" to create a multithreading version of this algorithm, the output would be wrong. Please explain the reason briefly and give an example.

The following procedure P-SCAN-1 performs the $\otimes$-prefix computation in parallel, albeit inefficiently:

```
P-SCAN-1(x)
1  n = x.length
2  let y[1..n] be a new array
3  P-SCAN-1-AUX(x, y, 1, n)
4  return y
```

```
P-SCAN-1-AUX(x, y, i, j)
1  parallel for l = i to j
2      y[l] = P-REDUCE(x, 1, l)
```

3) (5%) Analyze the work, span, and parallelism of P-SCAN-1.

By using nested parallelism, we can obtain a more efficient $\otimes$-prefix computation:

```
P-SCAN-2(x)
1  n = x.length
2  let y[1..n] be a new array
3  P-SCAN-2-AUX(x, y, 1, n)
4  return y
```

```
P-SCAN-2-AUX(x, y, i, j)
1  if i == j
2      y[i] = x[i]
3  else k = ⌊(i + j)/2⌋
4      spawn P-SCAN-2-AUX(x, y, i, k)
5      P-SCAN-2-AUX(x, y, k + 1, j)
6      sync
7      parallel for l = k + 1 to j
8          y[l] = y[k] ⊗ y[l]
```

4) (5%) Argue that P-SCAN-2 is correct, and analyze its work, span, and

parallelism.

## Problem 4 – All-Pairs Shortest Paths

Please read ch25.1 to ch25.2 of the textbook beforehand. The textbook states that the Floyd-Warshall algorithm would fail if the graph contains any negative cycle.

```
FLOYD-WARSHALL(W)
n = W.rows
$D^{(0)} = W$
for $k = 1$ to $n$
    let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
    for $i = 1$ to $n$
        for $j = 1$ to $n$
            $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$
```

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty , \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty . \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} , \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} . \end{cases}$$

1) (10%) How can we use the output of the Floyd-Warshall algorithm to detect the presence of a negative-weight cycle? Please explain. (You can give an example)
2) (10%) Describe how to modify the Floyd-Warshall algorithm, so that your algorithm can either return the matrix of shortest-path distances or return a negative cycle (the vertices in cycle). Your algorithm can arbitrarily choose one negative cycle if the graph contains more than one.