Algorithms and Data Structures 2015/16
Coursework 2

Issue date: Tuesday, 23rd February 2016

The deadline for this coursework is 4pm on Tuesday 15th March, 2016. Please submit your solutions electronically via `submit`. *This is worth 50% of the coursework for A&DS.*

note: *While it is ok to discuss the concepts of the coursework with your colleagues, and to help others with their understanding, it is not ok to copy work from others (or from textbooks or websites), or to compare or debug each other's solutions. So remember this. I don't expect you will find helpful literature for this coursework online; if you do, however, you must cite any references of material that helped you with your work.*

In this coursework, we consider the problem of "counting acyclic orientations". Your mission is to understand, prove some simple facts, implement and experiment with a suite of algorithms related to this problem. First I give some definitions.

Definition 1. *For any given (simple) undirected graph $G = (V, E)$, an* orientation $\vec{G}$ *of G is any directed graph $\vec{G} = (V, \vec{E})$ such that $|\vec{E}| = |E|$ and such that for every $(u, v) \in E$, exactly* one *of the arcs $(u \to v)$ and $(v \to u)$ belongs to $\vec{E}$.*

Definition 2. *For any directed graph $\vec{G} = (V, \vec{E})$, we say that $\vec{G}$ is* acyclic *if there is no directed cycle in $\vec{G}$.*

Such a graph is sometimes called a directed acyclic graph (DAG).

Definition 3. *For any simple* undirected *graph $G = (V, E)$, the set of* acyclic orientations (AOs) *of G, denoted $AO(G)$, is the set of all orientations $\vec{G}$ of G which are acyclic.*

We will consider a group of Algorithms related to *acyclic orientations* in this coursework. The various parts of the coursework will contribute to our developing an understanding of the problem of *counting the number of acyclic orientations* of undirected graphs. This is a well-known $\sharp P$-complete ("hard to count") problem when we consider an arbitrary given graph as input. We will consider the problem when the input graph is drawn from the *random model* $\mathcal{G}_{n,p}$: in this case, the situation for counting (at least the *expected* number of AOs) is better.

There will be a mixture of *theoretical* and *implementation* work for this coursework. The implementation will be done by completing the `acyclic.java` template file.

For the implementation part, we will adopt the convention that whether *directed* or *undirected*, all graphs have the vertex set $V = \{0, \ldots, n-1\}$ for some n, and that their edges are represented as an *adjacency matrix* of type `boolean`. For a *directed graph* (or *digraph*) $\vec{G} = (\{0, \ldots, n-1\}, \vec{E})$, the adjacency matrix `adjG` should be defined as:

$$\texttt{adjG[i][j]} = \begin{cases} \texttt{true} & \text{if } (i \to j) \in \vec{E} \\ \texttt{false} & \text{otherwise} \end{cases} .$$

For an *undirected graph* $G = (\{1, \ldots, n\}, E)$, we have a small difference in defining `adjG`:

$$\texttt{adjG[i][j]} = \begin{cases} \texttt{true} & \text{if } (i,j) \in E \text{ or } (j,i) \in E \\ \texttt{false} & \text{otherwise} \end{cases} .$$

Details of the 6 tasks (T1)-(T6) of this coursework are given in the following sections, together with some advice for each task.

1 Testing Acyclicity

A number of polynomial-time Algorithms exist for the problem of testing whether a given directed graph $\vec{G}$ is acyclic or not. We will consider a particular one of these, the "sink elimination" algorithm. The Algorithm depends on the following well-known fact:

Lemma 4. *Any $\vec{G} = (V, \vec{E})$ which is an* acyclic orientation *will always have at least one* sink *(a vertex $t \in V$ for which there is no* outgoing *arc $t \to \cdot$).*

Proof. Proof is by contradiction. Suppose that there is *no* sink in $\vec{G}$. Then choose any starting vertex $u \in V$ from $\vec{G}$ and construct a directed path iteratively - at every step choosing an unused outgoing arc from our current vertex. In the situation where every vertex has at least one outgoing arc, this process will only terminate when we arrive at a vertex which we have previously visited. This gives a cycle. Hence if $\vec{G}$ is acyclic we are guaranteed there is a sink. □

The interesting thing is that the Lemma can be exploited to give the following "acyclicity-testing" algorithm:

Algorithm isDAG($\vec{G} = (V, \vec{E})$)

1. $n \leftarrow |V|$.
2. *existssink* $\leftarrow$ `true`
3. **while**(*existssink*) **do**
4. check for an un-eliminated sink, and assign *sink* this value
5. **if** (we found an un-eliminated sink) **then**
6. **for** ($v \in V \setminus \{sink\}$)
7. remove $v \to sink$ from the graph
8. **else** *existssink* $\leftarrow$ `false`
9. **od**
10. **if** (our graph is non-empty) **then**
11. **return** `false`
12. **else return** `true`

The key observation for isDAG is that no directed cycle can possibly include any sink node. Hence in a directed graph which has some sink, the acyclicity status of $\vec{G}$ does not change after we delete the incoming arcs into the sink from $\vec{G}$. By applying this reduction iteratively, we either end up with a graph with no arcs (in which case we have found the graph to be acyclic) or else we end up with a subgraph which has no sink at all (in which case the subgraph, and hence the original graph also, does contain a cycle).

The Algorithm is $O(n^3)$ if implemented naively - there are more efficient ways of implementing it, but the naive implementation is fine for this coursework.

(T1) Write (in Java) a method `isDAG` to test, for a given directed graph $\vec{G}$, whether $\vec{G}$ is acyclic or not. The method should conform to the following type *[10 marks]*

```
public static boolean isDAG (boolean digraph[][])
```

and should be implemented within the file `acyclic.java`.

2 Generating Random Orientations

It will sometimes be the case that we will want to start with an undirected graph $G = (V, E)$, and try out various (random) orientations of the edges of G. In particular, we will want to do this for task (T6). For this task, we assume the *uniform distribution*, with all orientations being generated with the same probability (which is $2^{-|E|}$).

(T2) Write (in Java) a method `uniformOrient` to generate a *uniform random orientation* $\vec{G}$ of a given simple undirected graph $G = (V, E)$ and return this $\vec{G}$ as its result. The method should conform to the following type *[0 marks]*

```
public static boolean[][] uniformOrient(boolean[][] graph)
```

and should be implemented within the file `acyclic.java`.

There are 0 marks going for this - however, you will need the method later!

3 Erdős-Rényi graphs

Since we will be studying the count of acyclic orientations on randomly generated graphs, we present our formal model:

Definition 5. *The Erdős-Rényi model $\mathcal{G}_{n,p}$ of random graphs is parametrized by the number of vertices n, and an edge addition probability $p \in [0, 1]$.*

We generate a undirected simple graph $G = (V, E)$ from $\mathcal{G}_{n,p}$ by setting $V = \{0, \ldots, n-1\}$. To construct the edge set E, we consider each (i, j) pair $0 \leq i < n-1$, $i < j \leq n-1$ independently, and we add the undirected edge (i, j) to E with probability p (omitting it with probability $1 - p$).

Observe that we are guaranteed to create a simple graph because we only consider (i, j) edges for $i \neq j$, and we consider each (i, j) exactly once. Hence no loops or parallel edges can be added.

Another observation is that the particular number of edges added will vary depending on what happens as the graph is generated. The *expected* number of edges, written $\mathbb{E}_{n,p}[|E|]$ has the particular value $p \cdot \binom{n}{2}$. Also observe that for a particular "number of edges" m, all simple graphs with that number of edges have the same probability (which is $p^m(1-p)^{\binom{n}{2}-m}$) in the Erdős-Rényi model for n, p.

Here is the next task:

(T3) Write (in Java) a method `erdosRenyi` to generate a random undirected graph G according to the Erdős-Rényi random graph model $\mathcal{G}_{n,p}$, and return G as its result. The method should conform to the following type *[5 marks]*

```
public static boolean[][] erdosRenyi(int n, double p)
```

and should be implemented within the file `acyclic.java`.

4 The Robinson-Stanley recurrence

We now present a recurrence which was proved independently by two famous combinatorics researchers (Robert Robinson, and Richard Stanley) in the early 1970s. We need one more definition first:

Definition 6. *Let $n, m \in \mathbb{N}$. The number of different* directed acyclic graphs (DAGs) *on n vertices with m arcs, is denoted by $A_{n,m}$.*

We will never actually try to compute the $A_{n,m}$ values; the definition is only necessary for the proof I ask you to do.

We now present the Robinson (independently Stanley) theorem which defines a polynomial $A_n(x)$ in terms of the various $A_{n,m}$ values, and proves an elegant recurrence for $A_n(x)$.

Theorem 7 (Robinson, Stanley)**.** *Define*

$$A_n(x) = \sum_{m=0}^{\binom{n}{2}} A_{n,m} x^m.$$

(the co-efficients of $A_n(x)$ are the counts of DAGs with m arcs for various m).

Then

$$A_n(x) \quad = \quad \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} (1+x)^{i(n-i)} A_{n-i}(x).$$

We will *not* be proving the correctness of this recurrence in this coursework, as it is difficult. However, we will be proving that it can be *used* to compute the *expected number of AOs* of a random graph G, when G is drawn according to the Erdős-Rényi model.

Here is a reminder of how the expected number of AOs for $\mathcal{G}_{n,p}$ is defined:

$$\mathbb{E}_{n,p}[|AO(G)|] = \sum_{G=(V,E),|V|=n} \mathbb{P}r_{n,p}[G] \cdot |AO(G)|.$$

The expression $\mathbb{P}r_{n,p}[G]$ denotes the probability that graph G is generated by $\mathcal{G}_{n,p}$.

Now here is the next task:

(T4) Prove that the Robinson-Stanley polynomial $A_n(x)$, when evaluated at $x = \frac{p}{1-p}$ for any $p \in (0,1)$, satisfies the following equality: *[10 marks]*

$$A_n\left(\frac{p}{1-p}\right) = (1-p)^{-\binom{n}{2}} \times \mathbb{E}_{n,p}[|AO(G)|].$$

You will not need to use the result of Theorem 7 for this part. You will not need to use *induction*, or *proof by contradiction*, just manipulation and simplification of equations. The proof will only be about 5-6 lines long if done right.

Your proof should be submitted in a file called `task4.docx`, `task4.doc` or (preferably) `task4.pdf`.

5 Dynamic Programming for $\mathbb{E}_{n,p}[|AO(G)|]$

You are now asked to *use* the facts from Section 4 to design, analyse and implement an $O(n^2)$-time (or at a minimum an $O(n^3)$-time) Dynamic Programming algorithm to evaluate $\mathbb{E}_{n,p}[|AO(G)|]$ exactly.

The two main facts that you will need to exploit are:

- The relationship

$$\mathbb{E}_{n,p}[|AO(G)|] = (1-p)^{\binom{n}{2}} A_n\left(\frac{p}{1-p}\right)$$

  which was proven for task (T4).

- The recurrence of the Robinson/Stanley theorem:

$$A_n(x) \;=\; \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} (1+x)^{i(n-i)} A_{n-i}(x).$$

1. Develop an $O(n^2)$-time *dynamic programming* algorithm to evaluate $\mathbb{E}_{n,p}[|AO(G)|]$ exactly for given n, given $p \in (0,1)$. Write pseudocode for your algorithm, and justify the (n^2) running-time in detail. *[10 marks]*

   If you can only design/justify an $O(n^3)$ implementation, then present and discuss that one (you will be marked out of 7 instead).

   This should be submitted in a file called `task5.docx`, `task5.doc` or (preferably) `task5.pdf`

2. Implement your algorithm as the method `expErdosRenyi` in Java. *[5 marks]*

   The method should conform to the following type

   ```
   public static double expErdosRenyi(int n, double p)
   ```

   and should be coded within `acyclic.java`.

   If you want to test the results of your implementation, one small example is that for $n = 3$, and any p, the expectation should be $(1-p)^3 + 6p$.

6 Experimental estimation for $|AO(G)|$ in Erdős-Rényi

Assuming that task (T5) has been completed successfully, you now have a working method which allows you to exactly evaluate the expected number of acyclic orientations $\mathbb{E}_{n,p}[|AO(G)|]$ in the Erdős-Rényi model for any $n \in \mathbb{N}, p \in (0,1)$.

In this final section we will combine the methods of Tasks (T1), (T2), (T3) to develop an understanding of how well (or badly) "concentrated" the typical number of AOs in $\mathcal{G}_{n,p}$ is around its expected value.

Suppose instead that we were trying to *estimate* $\mathbb{E}_{n,p}[|AO(G)|]$ by random sampling. To examine how we should approach this, it makes sense to expand out $\mathbb{E}_{n,p}[|AO(G)|]$. We can write

$$\mathbb{E}_{n,p}[|AO(G)|] \quad = \quad \sum_{G=(V,E),|V|=n} \mathbb{P}r_{n,p}[G] \cdot |AO(G)|,$$

where $\mathbb{P}r_{n,p}[G]$ is the (well-understood) probability of generating G in $\mathcal{G}_{n,p}$. The equation above suggests that we could *estimate* $\mathbb{E}_{n,p}[|AO(G)|]$ by generating a number of graphs (k, say), and taking the average of the $|AO(G)|$ value over those k graphs. This would work fine *if we were able to evaluate* $|AO(G)|$ *efficiently when given a particular graph* G. However, we know this is a ♯P-complete problem, and hence we do not expect there is any polynomial time algorithm for evaluating $|AO(G)|$ for a given G.[1]

[1] Of course we could code up the naive (exponential-time) algorithm which generates *each of* the $2^{|E|}$ orientations one-at-a-time, and then checks each one for acyclicity. However, it would be prohibitively slow even for smallish n and $|E|$.

Hence we will also need to take a "random sampling" approach to calculating $|AO(G)|$ for each of our k particular graphs G. We know that

$$|AO(G)| \;=\; \sum_{\vec{G}} \mathbb{I}_{\vec{G} \text{ is acyclic}},$$

where $\vec{G}$ is taken over all $2^{|E|}$ orientations of G, and $\mathbb{I}_{\vec{G} \text{ is acyclic}}$ is an "indicator variable" returning the value 1 when $\vec{G}$ is acyclic and 0 otherwise. The size of the set of contributing $\vec{G}$ graphs is $2^{|E|}$ - however let us instead estimate $AO(G)$ by drawing 200 different *uniform random orientations* $\vec{G}_1, \ldots, \vec{G}_{200}$ of G (we can use `uniformOrient` for this) and then *estimate* $|AO(G)|$ by the value

$$\widehat{ao(G)} = 2^{|E|} \frac{1}{200} \sum_{i=1}^{200} \mathbb{I}_{\vec{G}_i \text{ is acyclic}}.$$

Note that the value of $\mathbb{I}_{\vec{G} \text{ is acyclic}}$ for any particular directed graph $\vec{G}$ can be determined with our `isDAG` method.

This will allow us to obtain an *estimate* $\widehat{ao(G)}$ of $|AO(G)|$ for each of our k sampled graphs G. Then by taking the average of these k values, we will get an estimate of $\mathbb{E}_{n,p}[|AO(G)|]$.

Your tasks on this section are as follows:

(T6) 1. Follow the discussion above to implement a method `estimErdosRenyi` which takes a natural number n, a probability $p \in (0,1)$, and a "required number of samples k", and implements the process above (with k samples of graphs from $\mathcal{G}_{n,p}$, and 200 samples of orientations for each of these k graphs). The `double` value returned should be the estimate of $\mathbb{E}_{n,p}[|AO(G)|]$. *[5 marks]*

   Your method should conform to the following type

   ```
   public static double estimErdosRenyi (int n, double p, int k)
   ```

   and should be coded inside `acyclic.java`

   2. Use your methods to run `estimErdosRenyi` and `expErdosRenyi` on the same values of n and p (for n growing, and for a few different p values). You should use a few values of k. *[5 marks]*

   Write a short report presenting values of n, p (and k) used in your experiments, and discussing the quality of the value returned by `estimErdosRenyi`.

   Your report should be named `task6.txt`, `task6.docx`, `task6.pdf` or `task6.txt`

Please turn over

7 Submitting your work

Download the file `acyclic.java` from the course webpage. This file contains declarations for the methods you are required to write.

Implement *all of your methods* within `acyclic.java`, available from the course webpage.

Write the theoretical material (and results) for tasks 4, 5 and 6 in files called `task4.???`, `task5.???` `task6.???` respectively. Then submit as follows:

```
submit ads 2 acyclic.java task4.???  task5.???  task6.???
```

(if you have extra files, please also include them.)

The **DEADLINE** is 4pm, Tuesday, March 15, 2016.

Warning: Before submitting, please do "`more acyclic.java`" (and use `acroread` or `ooffice` to view your `task?.???` files) from your current directory, to check that you have the right versions to hand (the rules are "what is marked, is what is submitted").

Mary Cryan, 15th February, 2016