Case Study: PageRank Algorithm

Outline

1. Random Surfer Model
2. Input Format
3. Transition Matrix
4. Simulating a Random Surfer
5. Markov Chain

Random Surfer Model

The random surfer model is a simple model of the world-wide web (www) that has proven to be a particularly useful approach to understand some of its properties.

The www is modeled as a fixed set of pages, with each page containing a fixed set of links (called hyperlinks), and each link is a reference to some other page; such an abstraction is called a graph in mathematics.

We are interested in what happens to a person (a random surfer) who randomly moves from page to page according to the 90-10 rule: 90% of the time the random surfer clicks a random link on the current page and 10% of the time the random surfer goes directly to a random page by typing a page name into the address bar.

Page rank is the probability that the random surfer lands on a page.

Input Format

A tiny web of five pages

and a file tiny.txt that encodes it

```
$ more tiny.txt
5
0 1
1 2 1 2
1 3 1 3
1 4
2 3
3 0
4 0 4 2
```
The transition matrix $P$ for a www with $N$ pages is an $N$-by-$N$ matrix such that the entry $P_{ij}$, i.e., the entry in row $i$ and column $j$, is the probability that the random surfer moves to page $j$ when on page $i$.

For example, a transition matrix for our tiny five-page web is

$$P = \begin{bmatrix}
0.02 & 0.92 & 0.02 & 0.02 & 0.02 \\
0.02 & 0.38 & 0.38 & 0.20 & 0.10 \\
0.02 & 0.02 & 0.92 & 0.02 & 0.02 \\
0.92 & 0.02 & 0.02 & 0.02 & 0.02 \\
0.47 & 0.02 & 0.47 & 0.02 & 0.02
\end{bmatrix}$$
Markov Chain

A Markov chain is a random process that describes behavior similar to the random surfer.

An useful property of Markov chains is that of mixing, which states that the random surfer could start anywhere, since the probability that the surfer eventually winds up on any particular page is the same for all starting pages.

Using the power method we can obtain the probability \( P \), fortunately, because of the mixing property, we can make do with relatively inexpensive vector-matrix multiplications.

\[
\begin{align*}
\text{Using the power method we can obtain the probability } & P \text{, that the random surfer will move from page } i \text{ to page } j \text{ in two moves by squaring the transition matrix } P \\
\quad & P^2_{ij} = P_i \cdot P_j \\
\text{For example, for our tiny web, we have} & \\
\quad & P^2_{ij} = \begin{bmatrix} 0.02 & 0.92 & 0.02 & 0.02 \\ 0.02 & 0.92 & 0.02 & 0.02 \\ 0.92 & 0.02 & 0.02 & 0.02 \\ 0.47 & 0.47 & 0.02 & 0.02 \end{bmatrix} \cdot \begin{bmatrix} 0.02 & 0.92 & 0.02 & 0.02 \\ 0.02 & 0.92 & 0.02 & 0.02 \\ 0.92 & 0.02 & 0.02 & 0.02 \\ 0.47 & 0.47 & 0.02 & 0.02 \end{bmatrix} \\
\quad & = \begin{bmatrix} 0.05 & 0.04 & 0.36 & 0.37 & 0.19 \\ 0.45 & 0.04 & 0.12 & 0.07 & 0.92 \\ 0.86 & 0.04 & 0.04 & 0.05 & 0.92 \\ 0.05 & 0.85 & 0.04 & 0.05 & 0.92 \\ 0.05 & 0.44 & 0.04 & 0.45 & 0.92 \end{bmatrix} \\
\end{align*}
\]

Now, multiplying the above vector by the transition matrix \( P \) again gives the vector

\[
\begin{bmatrix} 0.02 & 0.92 & 0.02 & 0.02 \\ 0.02 & 0.92 & 0.02 & 0.02 \\ 0.92 & 0.02 & 0.02 & 0.02 \\ 0.47 & 0.47 & 0.02 & 0.02 \end{bmatrix} = \begin{bmatrix} 0.05 & 0.04 & 0.36 & 0.37 & 0.19 \\ 0.45 & 0.04 & 0.12 & 0.07 & 0.92 \\ 0.86 & 0.04 & 0.04 & 0.05 & 0.92 \\ 0.05 & 0.85 & 0.04 & 0.05 & 0.92 \\ 0.05 & 0.44 & 0.04 & 0.45 & 0.92 \end{bmatrix}
\]

which contains the probabilities that the surfer winds up on each of the pages after two steps.

The vector giving the probabilities that the surfer winds up on each of the pages after \( t \) steps is the product of the corresponding vector for \( t - 1 \) steps and the transition matrix \( P \).

```python
import stdio
import sys

moves = int(sys.argv[1])
n = stdio.readInt()

probs = stdarray.create2D(n, n, 0.0)
for i in range(n):
    for j in range(n):
        probs[i][j] = stdio.readInt()

ranks = stdarray.create1D(n, 0.0)
ranks[0] = 1.0

for i in range(moves):
    newRanks = stdarray.create2D(n, n, 0.0)
    for j in range(n):
        for k in range(n):
            newRanks[j] += ranks[k] * probs[k][j]
    ranks = newRanks

stdio.writeln("%8.5f", rank)
stdio.writeln()
```

Markov Chain

In general, the probability that the random surfer will move from page \( i \) to page \( j \) in \( t \) moves is obtained by computing \( P^t \).

Unfortunately, matrix-matrix multiplication is expensive.

Fortunately, because of the mixing property, we can make do with relatively inexpensive vector-matrix multiplications.

For example, with our tiny web, if we start with the vector

\[
\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}
\]

specifying that the random surfer starts on page 0, multiplying the vector by the transition matrix \( P \) gives the vector

\[
\begin{bmatrix} 0.02 & 0.92 & 0.02 & 0.02 & 0.02 \end{bmatrix} \cdot \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} = \begin{bmatrix} 0.02 & 0.92 & 0.02 & 0.02 & 0.02 \end{bmatrix}
\]

specifying the probabilities that the surfer winds up on each of the pages after \( t \) steps.

```python
$ python3 transition.py < tiny.txt | python3 markov.py 1
0.27303 0.26573 0.14618 0.24723 0.06783

$ python3 transition.py < tiny.txt | python3 markov.py 2
0.27303 0.26573 0.14618 0.24723 0.06783

$ python3 transition.py < tiny.txt | python3 markov.py 4
0.27303 0.26573 0.14618 0.24723 0.06783
```

Markov Chain

```python
import stdio
import sys

moves = int(sys.argv[1])
n = stdio.readInt()

probs = stdarray.create2D(n, n, 0.0)
for i in range(n):
    for j in range(n):
        probs[i][j] = stdio.readInt()

ranks = stdarray.create1D(n, 0.0)
ranks[0] = 1.0

for i in range(moves):
    newRanks = stdarray.create2D(n, n, 0.0)
    for j in range(n):
        for k in range(n):
            newRanks[j] += ranks[k] * probs[k][j]
    ranks = newRanks

stdio.writeln("%8.5f", rank)
stdio.writeln()
```