

# GEM 1501 Problem Solving With Computers

## Lecture 11:

## Probabilistic Algorithms

Martin Henz

# Summary of Previous Lecture

- Parallelism allows to speed up computation
- New complexity classes that take parallelism in to account
- Problems of synchronization

# Appendix to last lecture: Semaphores

- Programming languages for concurrent programming have constructs for synchronization
- Example: Semaphore
  - $S$  is shared integer variable
  - $request(S)$  decrements  $S$  if it is positive and waits otherwise
  - $release(S)$  increments  $S$ .
  - Both actions are atomic (cannot be interrupted).

# Solving Mutual Exclusion Problems with Semaphores

Semaphore Shower

Subroutine takeAShower

```
    request(Shower);  
    enter and take a shower  
    release(Shower)
```

# Dining Philosophers with Semaphores?

First attempt:

```
Semaphore F0,F1,F2,F3,F4
```

```
Subroutine philosopher_P2
```

```
    think;
```

```
    request(F2);
```

```
    request(F3);
```

```
    eat;
```

```
    release(F3);
```

```
    release(F2);
```

# Problem: Deadlock!

- This problem cannot be handled with semaphores (or any other mechanism), where each philosopher can only communicate with his forks.
- Centralized control is needed

# Dining Philosophers (Centralized Control)

Semaphore Manager

Array Fork[5]

Subroutine philosopher(i)

    think;

    request(Manager);

    if Fork[i] = ontable and Fork[(i+1) mod 5] = ontable

    then Fork[i] := i; Fork[(i+1) mod 5] := i;

        release(Manager);

        eat;

        Fork[i] := ontable; Fork[(i+1) mod 5] := ontable;

    else release(Manager);

        philosopher(i)

# Probabilistic Algorithms

- Probabilistic algorithms make use of random choices
- “Tossing a coin”
- Used in:
  - concurrency (Dining Philosophers)
  - generating large prime numbers
  - cryptography

# Dining Philosophers

1. do the following forever:
  - (a) think until hungry;
  - (b) toss a coin to choose left or right;
  - (c) wait until fork in the chosen direction is free, lift it;
  - (d) if other fork is not available:
    - i. put down fork;
    - ii. go to 1b
  - (e) otherwise lift other fork; eat;
  - (f) put down both forks;

# Randomization for Problem Solving

- Many intractable problems have probabilistic solutions
- Such solution guarantee with a certain probability to find the answer.
- Example: Generating large prime numbers

# Generating Large Prime Numbers

- Let's say we need new prime numbers with 100 digits.
- Observation: The number of primes less than  $N$  is  $O(N/\log N)$ .
- About one in every 300 numbers with 100 digits is a prime number.
- If we had a way of testing the primality of numbers, we could proceed as follows:
  - Make a random number with 100 digits
  - Test it for primality, and if yes, you are done

# Update on Primality

- Book (second edition) says that primality testing is unsolved
- In 2002, Prof. Manindra Agarwal (IIT Kanpur) and two of his students, Nitin Saxena and Neeraj Kayal gave a polynomial algorithm ( $O((\log n)^{12} f(\log \log n))$  where  $f$  is a polynomial)
- Unfortunately, the algorithm is not fast enough (yet) to work for 100 digit numbers.

# Testing for Primality

- There is an exact algorithm that runs in  $O(N^{O(\log \log N)})$
- There are very fast probabilistic algorithms.
  - They indicate primality of large numbers with very low chance of being wrong.
  - Based on witnesses; numbers that allow us to state with a probability of  $1/2$  that  $N$  is prime.

# Other Applications

- Searching in strings
- Traveling salesman
- Satisfiability problems

# Encrypting and Decrypting Data

Can we encode data such that:

- the receiver can be sure that it comes from the sender
- the receiver cannot send messages pretending to be the sender

# The Usual Way

Sender invents a secret “key” and gives it to the receiver.

Sender:  $H = Encr(M)$

Receiver:  $M = Decr(H)$

# Public Key Cryptography

- Consider locks that can be closed without a key, but opened only with a key.
- Everyone has such a lock and a secret key for his/her lock.
- B can send a message to A by placing it in a box and locking it with A's lock.

# Details

- Of course we must have  $Decr_A(Encr_A(M)) = M$ .
- We also require  $Encr_A(Decr_A(M)) = M$ .
- The most important is that having  $Encr$  does not allow to deduce  $Decr$ .

# RSA Cryptosystem

- Based on easy primality testing,...
- ... and hard factoring!
- The secret key consists of a pair of large prime numbers  $P$  and  $Q$
- The public key consists of  $P \times Q$
- In PSA, the public is a bit more involved.

# The Essence of RSA

- It is hard to factor a given large number.
- Factoring is believed not to be in P.
- We are making use of intractability!
- If we would have a fast factoring algorithm, RSA would come apart!

# Next Week

- Artificial Intelligence
- Summary of course