COS 126 **General Computer Science** **Fall 2008**

Exam 2 Solutions

1. Creating Data Types

```
public class Button {

        private boolean pushed;        //could be int or string

        public Button() { pushed = false; }

        public void push() { pushed = true; }

        public void release() { pushed = false; }

        public void toggle() { pushed = !pushed; }

        public boolean isPushed() { return pushed; }

 }
```

2. Queue and Stack

Fill in the blanks in the following block of client code that reverses the items in a `QueueOfInts` `q` by using an intermediate `StackOfInts s`.

```
StackOfInts s = new StackOfInts();
while(!q.isEmpty()) {

    int x = q.___dequeue___();

    __s__.push(x);
}

while(___!s.isEmpty()___) {

    int x = __s__.pop();

    q.___enqueue___(x);
}
```

3. ADT and Linked Structures

a.

```
public boolean isFirstCousin(Person p) {
    return (this.mom.isSibling(p.mom) || this.mom.isSibling(p.dad) ||
            this.dad.isSibling(p.mom) || this.dad.isSibling(p.dad));
}
```

b. Straightforward method:

```
public static Person birth(String babyname, Person mom, Person dad) {
   Person mom = this;
   if (mom.spouse != dad) throw new RuntimeException("Adultery detected!");

   Person baby = new Person(babyname, mom, dad);
   Children temp = mom.kids;
   mom.kids = new Children();
   mom.kids.child = baby;
   mom.kids.siblings = temp;
   dad.kids = mom.kids;
   return baby;
}
```

But since `mom` and `dad` have the same `kids` we can shorten the code to this:

```
public static Person birth(String babyname, Person mom, Person dad) {
   Person mom = this;
   if (mom.spouse != dad) throw new RuntimeException("Adultery detected!");

   Person baby = new Person(babyname, mom, dad);
   mom.kids = new Children();
   mom.kids.child = baby;
   mom.kids.siblings = dad.kids;        //same as temp above
   dad.kids = mom.kids;
   return baby;
}
```

4. Regular Expressions and DFA's

a. strings that start and end with **a** (including just **a**)
   - Regular expression: **a|a(a|b|c)*a**
   - DFA: **E**

b. strings containing at most one **a**
   - Regular expression: **(b|c)*|(b|c)*a(b|c)***
   - DFA: **B**

c. strings containing at least two **a**'s
   - Regular expression: **(b|c)*(a(b|c)*a)(a|b|c)***
   - DFA: **A**

d. strings containing an even number of **a**'s
   - Regular expression: **(b|c)*(a(b|c)*a)*(b|c)***
   - DFA: **D**

5. Universality and Computability

a. FALSE

b. TRUE

c. FALSE

d. unlimited memory

e. ii. Any computational problem that can be solved by a physically harnessable process of this universe can be solved by a Turing Machine.

6. Intractability

a. Big deal, I wrote a program that finds an optimal solution for TSP in COS 126, and its running time had order of growth only N^2.
DISAGREE. My TSP program was pretty slick but had no chance of finding optimal solutions for all TSP problems. It just used some heuristics to make the route shorter.

b. If you indeed have such an algorithm, then P = NP. (Can we split the million dollar prize?)
AGREE. Actually, you did this on company time, right? So maybe I get all of the prize? After all, if *on company time* you came up with a polynomial-time algorithm for TSP, then, since TSP is NP-complete, we have a way, via problem reductions, to get polynomial-time algorithms for every problem in NP.

c. Wow, that algorithm could be used to solve the Halting Problem.
DISAGREE. The infamous Halting Problem was proven to be unsolvable long ago.

d. Cool, that algorithm could be used to break the RSA cryptosystem that encrypts credit card numbers transmitted on the Internet, since RSA depends on factoring and factoring is in NP.

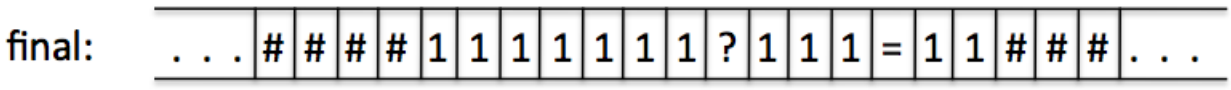
AGREE. See above. Factoring is in NP, so if P = NP then your stupendous result proves that via one or more reductions, we can get a polynomial-time algorithm for Factoring.

e. That is impossible. Your algorithm runs in Polynomial time, TSP is in NP, and we know that P != NP. Sounds like a contradiction to me.

DISAGREE. We only really really strongly *believe* that P != NP. It is wildly unlikely but not impossible that your algorithm is correct. No contradiction.

7. Turing Machines

a. Here's the final tape:



b. the (unary) integer quotient of `m` and `n`

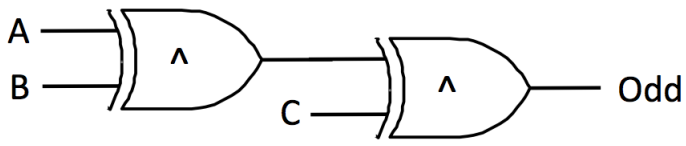
8. Circuits

a.

A	B	C	Odd
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

b. $A'B'C + A'BC' + AB'C' + ABC$

c. $(A \wedge B) \wedge C$



d. means that not every Boolean function (or combinational circuit) can be expressed using XORs alone