Solutions to TopCoder
SRM 236

Programming Puzzles and Competitions
CIS 4900 / 5920
Spring 2009
SRM 236 (Division II)

- MassiveNumbers (250-point problem)
- BusinessTasks (500-point problem)
- ComputerExpert (1000-point problem)
MassiveNumbers:
Problem Statement

• Given $x$, $y$, $a$, and $b$, which is bigger: $x^a$ or $y^b$?
MassiveNumbers

• In the problem statement, you are given a hint:
  If $i \cdot \log(x) \mathbin{R} j \cdot \log(y)$, then $x^i \mathbin{R} y^j$, where $\mathbin{R}$ is one of \{<, >, ≥, ≤, =\}. 
MassiveNumbers

- $x^i \leq y^j$
  - $\log(x^i) \leq \log(y^j)$
  - $i \cdot \log(x) \leq j \cdot \log(y)$
- e.g. $x^5 = y^6$
  - $\log(x^5) = \log(y^6)$
  - $5 \cdot \log(x) = 6 \cdot \log(y)$
MassiveNumbers: Computing the result

```c
int x, i, y, j; //must be initialized  
if( i * log(x) == j * log(y) )  
    return numberA;     //argument 1 or 2  
else if( i * log(x) > j * log(y) )  
    return numberA;     //argument 1  
else  
    return numberB;     //argument 2
```

Remember the problem: $x^i \Rightarrow y^j$
MassiveNumbers: Parsing the input

• **C++**
  - `class istringstream;`

• **Java**
  - `String.split(String regex);`
  - `Integer.parseInt(String);`

Remember the problem: $x^i \overset{?}{=} y^j$
MassiveNumbers: Parsing the input in C++

```cpp
int x, i, y, j;
char temp;
istringstream inA(numberA);
inA >> x >> temp >> i;  //temp == '^'
istringstream inB(numberB);
inB >> y >> temp >> j;  //temp == '^'
```

Remember the problem: $x^i \oplus y^j$
MassiveNumbers: Parsing the input in C++

• Sample run:

```cpp
string numberA = "2^100";
istringstream inA(numberA);

inA >> x >> temp >> i;
```

Remember the problem: $x^i \ ? \ y^j$
MassiveNumbers: Parsing the input in C++

• Sample run:
  string numberA = “2^100”;
  istreamstream inA(numberA);

  (inA >> x) >> temp >> i;

Remember the problem: $x^i \neq y^j$
MassiveNumbers: Parsing the input in C++

• Sample run:

string numberA = "2^100";
istringstream inA(numberA);

x = 2
inA >> temp >> i;

Remember the problem: $x^i \ ? \ y^j$
MassiveNumbers: Parsing the input in C++

• Sample run:
  
  string numberA = "2^100";
  istringstream inA(numberA);

  (inA >> temp) >> i;

  x = 2;

Remember the problem: \( x^i \ ? \ y^j \)
MassiveNumbers: Parsing the input in C++

- Sample run:
  string numberA = “2^100”;
  istream inA(numberA);

  temp = ‘^’
  inA >> i;  
  x = 2;

Remember the problem: $x^i \div y^j$
MassiveNumbers: Parsing the input in C++

- Sample run:
  ```
  string numberA = "2^100";
  istringstream inA(numberA);
  (inA >> i);
  x = 2;
  temp = '^';
  ```

Remember the problem: $x^i \neq y^j$
MassiveNumbers: Parsing the input in C++

- Sample run:

  string numberA = "2^100";
  istringstream inA(numberA);

  y = 100
  inA;
  x = 2;
  temp = '^';

Remember the problem: $x^i \text{ ? } y^j$
MassiveNumbers: Parsing the input in C++

- Sample run:
  
  ```cpp
  string numberA = "2^100";
  istringstream inA(numberA);
  ```

  ```cpp
  x = 2;
  temp = '^';
  y = 100;
  ```

  Remember the problem: \( x^i \ ? \ y^j \)
MassiveNumbers: Parsing the input in C++

int x, i, y, j;
char temp;
istringstream inA(numberA);
inA >> x >> temp >> i;
istringstream inB(numberB);
inB >> y >> temp >> j;

Remember the problem: $x^i \ ? \ y^j$
MassiveNumbers: Parsing the input in Java

String[] partsA = numberA.split("\^\^");
int x = Integer.parseInt(partsA[0]);
int i = Integer.parseInt(partsA[1]);
String[] partsB = numberB.split("\^\^");
int y = Integer.parseInt(partsB[0]);
int j = Integer.parseInt(partsB[1]);

Remember the problem: $x^i \, ? \, y^j$
MassiveNumbers: Returning the result

• This part has already been taken care of
• Recall:
  ```cpp
  int x, i, y, j; //must be initialized
  if( i * log(x) == j * log(y) )
      return numberA; //argument 1 or 2
  else if( i * log(x) > j * log(y) )
      return numberA; //argument 1
  else
      return numberB; //argument 2
  ```

Remember the problem: $x^i \ ? \ y^j$
MassiveNumbers: Putting it all together (C++)

```cpp
#include <string>
#include <sstream>
#include <cmath>
using namespace std;

const double ERROR = 1E-7;

class MassiveNumbers {
  public: string getLargest(const string, const string);
};

string MassiveNumbers::getLargest(const string numberA, const string numberB) {
  int x, i, y, j;
  char temp;
  istringstream inA(numberA);
  istringstream inB(numberB);
  inA >> x >> temp >> i;
  inB >> y >> temp >> j;
  if( abs(i * log((double) x) - j * log((double) y)) < ERROR )
    return numberA;  //argument 1 or 2
  else if( i * log((double) x) > j * log((double) y) )
    return numberA;  //argument 1
  else
    return numberB;  //argument 2
}
```
public class MassiveNumbers {
    private static final double ERROR = 1E-7;

    public String getLargest(String numberA, String numberB) {
        String[] partsA = numberA.split(“\^”);
        int x = Integer.parseInt(partsA[0]);
        int i = Integer.parseInt(partsA[1]);
        String[] partsB = numberB.split(“\^”);
        int y = Integer.parseInt(partsB[0]);
        int j = Integer.parseInt(partsB[1]);
        if( Math.abs(i * Math.log(x) - j * Math.log(y)) < ERROR )
            return numberA;  //argument 1 or 2
        else if( i * Math.log(x) > j * Math.log(y) )
            return numberA;  //argument 1
        else
            return numberB;  //argument 2
    }
}
A note about floating-point arithmetic

- When comparing two floating point values for equality, never use "==" as the comparison operator; this can fail due to round-off error.
- Always check if the two numbers are equal within some error bound, e.g. $1 \times 10^{-7}$.
BusinessTasks:
Problem Statement

• A busy businessman has a set of tasks. He selects one using the following algorithm (of which you must execute steps 3, 4, & 5):
  1) He places the tasks in a circular list
  2) He picks a positive random integer seed n
  3) Starting with the first task, he counts from 1 to n, and removes the n^{th} task
  4) Starting from the task after the one that was just removed, he removes the n^{th} task again
  5) He repeats step 4 until the list has only one element, which is the task he selects
Business Tasks:
An example

- \( n = 3 \)

\[
\begin{array}{cccc}
  a & b & c & d & e \\
\end{array}
\]
Business Tasks: An example

- $n = 3$

\[1\]

\[
\begin{array}{cccc}
  a & b & c & d & e \\
\end{array}
\]
Business Tasks:
An example

- n = 3
Business Tasks:

An example

- n = 3

```
a b c d e
```

3
Business Tasks: An example

- $n = 3$

```
 a b e d e
```
Business Tasks:

An example

- n = 3
Business Tasks: An example

• $n = 3$

```
  a  b  d  e
```

1
Business Tasks: An example

- n = 3

```
 a b d e
```

2
Business Tasks: An example

- $n = 3$

\[ \begin{array}{c}
3 \\
\downarrow \\
\begin{array}{cccc}
a & b & d & e \\
\end{array}
\end{array} \]
BusinessTasks: An example

- $n = 3$

```
  a  b  d  e
```

3
Business Tasks: An example

- $n = 3$

| b | d | e |
BusinessTasks: An example

• $n = 3$

```
1
```

```
\[
\begin{array}{ccc}
  b & d & e \\
\end{array}
\]
```
BusinessTasks:
An example

• n = 3

\[ 2 \]

\[ b \quad d \quad e \]
Business Tasks:
An example

- \( n = 3 \)

\[ \begin{array}{ccc}
    b & d & e \\
\end{array} \]
BusinessTasks: An example

- $n = 3$
Business Tasks:
An example

- $n = 3$

1

\[
\begin{array}{c}
\text{b} \\
\downarrow \\
\text{d}
\end{array}
\]
Business Tasks: An example

- $n = 3$

```
  b  d
  
  2  
```
Business Tasks:
An example

- $n = 3$

3

\[ \begin{array}{c|c}
  \text{x} & d \\
\end{array} \]
Business Tasks:
An example

- $n = 3$
Business Tasks:
An example

- n = 3

\[ d \rightarrow \text{return value} \]
BusinessTasks

- Notice that we are really just doing modular arithmetic on each iteration
Business Tasks

Which is the \( n^{th} \) element (\( n = 3 \)), counting from the current position?

\[ \text{current} \]

\[
\begin{array}{cccccc}
a & b & c & d & e \\
\end{array}
\]

\((L = 5)\)
BusinessTasks

\[ \text{\(n^{th}\) element from current} = (\text{current} + (n - 1)) \mod L \]

(L = 5)

(\text{current position} + (3 - 1)) \mod 5
• Taking this into account, we can remove each element one at a time by just computing \(((\text{current} + (n - 1)) \mod L)\) each time instead of iterating \((n - 1)\) times.
```cpp
#include <vector>
#include <string>
using namespace std;

class BusinessTasks {
    public: string getTask(vector<string>, const int n);
};

string BusinessTasks::getTask(vector<string> list, const int n) {
    for(int currentTask = (n - 1) % list.size();
        list.size() > 1;
        currentTask = (currentTask + n - 1) % list.size())
    {
        list.erase(list.begin() + currentTask);
    }

    return list.at(0);
}
```
import java.util.Vector;

public class BusinessTasks {
    public String getTask(final String[] list, final int n) {
        final Vector<String> tasks = new Vector<String>();
        for(String temp : list) {
            tasks.add(temp);
        }

        for(int currentTask = (n - 1) % tasks.size();
            tasks.size() > 1;
            currentTask = (currentTask + n - 1) % tasks.size()){
            tasks.remove(currentTask);
        }

        return tasks.get(0);
    }
}
ComputerExpert

• Given an array of strings representing facts, and a set of rules of the form (fact/.../fact,...,fact \rightarrow fact), return a sorted array of all the outcome facts which can be deduced from the initial observations using the set of rules provided.
ComputerExpert

• Rule format:
  fact/.../fact,...,fact -> fact

• “/” denotes or
• “,” denotes and
• In this problem only, “/” (or) has precedence over “,” (and)
ComputerExpert: Example

- Observations:
  
  \{"SKY_IS_CLOUDY", "HAVE_MONEY"\}

- Rules:

  \{"HAVE_MONEY,BAD_WEATHER -> WATCH_MOVIE",
  "SKY_IS_CLOUDY -> BAD_WEATHER",
  "GOOD_WEATHER -> PLAY_SOCCER"\}
ComputerExpert: Example

- Observations:
  
  \{"SKY_IS_CLOUDY", "HAVE_MONEY"\}

- Rules:

  \{"HAVE_MONEY,BAD_WEATHER -> WATCH_MOVIE",
  "SKY_IS_CLOUDY -> BAD_WEATHER",
  "GOOD_WEATHER -> PLAY_SOCCER"\}
ComputerExpert: Example

• Observations:
  {"SKY_IS_CLOUDY", "HAVE_MONEY"}

• Rules:
  {"HAVE_MONEY,BAD_WEATHER -> WATCH_MOVIE",
   "SKY_IS_CLOUDY -> BAD_WEATHER",
   "GOOD_WEATHER -> PLAY_SOCCER"}
ComputerExpert: Example

• Observations:
  
  \{"SKY\_IS\_CLOUDY", "HAVE\_MONEY"\}

• Rules:

  \{"HAVE\_MONEY,BAD\_WEATHER\ -> \ WATCH\_MOVIE", 
   "SKY\_IS\_CLOUDY\ -> \ BAD\_WEATHER", 
   "GOOD\_WEATHER\ -> \ PLAY\_SOCCER"\}
ComputerExpert: Example

- **Observations:**
  
  `{"SKY_IS_CLOUDY", "HAVE_MONEY"}`

- **Rules:**
  
  `{"HAVE_MONEY,BAD_WEATHER -> WATCH_MOVIE", "SKY_IS_CLOUDY -> BAD_WEATHER", "GOOD_WEATHER -> PLAY_SOCCER"}`
ComputerExpert: Example

- Observations:
  
  \[
  \{ \text{"SKY IS CLOUDY"}, \text{"HAVE MONEY"} \} 
  \]

- Rules:
  
  \[
  \{ \text{"HAVE MONEY, BAD WEATHER -> WATCH MOVIE"}, \\
  \text{"SKY IS CLOUDY -> BAD WEATHER"}, \\
  \text{"GOOD WEATHER -> PLAY SOCCER"} \} 
  \]
ComputerExpert: Example

• Observations:
  {"SKY_IS_CLOUDY", "HAVE_MONEY"}

• Rules:
  {"HAVE_MONEY,BAD_WEATHER -> WATCH_MOVIE",
   "SKY_IS_CLOUDY -> BAD_WEATHER",
   "GOOD_WEATHER -> PLAY_SOCCER"}
ComputerExpert: Example

• Observations:
  {"SKY_IS_CLOUDY", "HAVE_MONEY"}

• Rules:
  {"HAVE_MONEY,BAD_WEATHER -> WATCH_MOVIE",
  "SKY_IS_CLOUDY -> BAD_WEATHER",
  "GOOD_WEATHER -> PLAY_SOCCER"}

• Returns:
  {"BAD_WEATHER", "WATCH_MOVIE"}
ComputerExpert: An algorithm

- Maintain a set of knowledge (known facts)
ComputerExpert: An algorithm

- Maintain a set of knowledge (known facts)
- Add the set of input facts (observations) to the set of knowledge
ComputerExpert: An algorithm

- Maintain a set of knowledge (known facts)
- Add the set of input facts (observations) to the set of knowledge
- Remove all occurrences of known facts from the left-hand side of the rules.
ComputerExpert: An algorithm

- Maintain a set of knowledge (known facts)
- Add the set of input facts (observations) to the set of knowledge
- Remove all occurrences of known facts from the left-hand side of the rules.
- If any remaining rules have all conditions satisfied, add them to the current set of knowledge and continue recursively until no progress can be made
ComputerExpert: An algorithm

- Maintain a set of knowledge (known facts)
- Add the set of input facts (observations) to the set of knowledge
- Remove all occurrences of known facts from the left-hand side of the rules.
- If any remaining rules have all conditions satisfied, add them to the current set of knowledge and continue recursively until no progress can be made
- The set of knowledge minus the set of input facts is the set of outcome facts
ComputerExpert: An algorithm

- Maintain a set of knowledge (known facts)
- Add the set of input facts (observations) to the set of knowledge
- Remove all occurrences of known facts from the left-hand side of the rules.
- If any remaining rules have all conditions satisfied, add them to the current set of knowledge and continue recursively until no progress can be made
- The set of knowledge minus the set of input facts is the set of outcome facts
- Sort this result and return it as the answer
Computer Expert: Data Representation

- Maintain knowledge as a set
  (std::set<std::string> in C++)
  (HashSet<String> in Java)
ComputerExpert: Data Representation

• Maintain rules as a vector of string vectors, with the last element of the vector representing an outcome fact

(vector<vector<string>> in C++)
(Vector<Vector<String>> in Java)
ComputerExpert

- Java solution online:
  
Rubik’s Cube

• How many people have a Rubik’s Cube?