NOI 2008—Solutions to Contest Tasks

Martin Henz

12/4, 2008

The Scientific Committee of NOI 2008

- Ben Leong
- Chang Ee-Chien
- Colin Tan
- Frank Stephan
- Low Kok Lim
- Martin Henz
- Sung Wing Kin

1. PRIME
2. LVM
3. GECKO
4. HOUSING
5. RANK
6. 4SUM

1. PRIME
   - Problem
   - Algorithm
   - Program in C++
2. LVM
3. GECKO
4. HOUSING
5. RANK
6. 4SUM

Problem

A prime number is a natural number which has exactly two distinct natural number divisors: 1 and itself. The first prime number is 2. Can you write a program that computes the nth prime number, given a number $n \leq 10000$?

Input and Output

- Input file: `PRIME.IN`

  `30`

- Output file: `PRIME.OUT`

  `113`

Sieving Integers

Idea

Generate list of candidate integers, and systematically eliminate non-prime numbers. Remaining numbers are prime.

Sieving Integers

Idea

Generate list of candidate integers, and systematically eliminate non-prime numbers. Remaining numbers are prime.

Suitability

This approach is reasonable, because we need to generate the first n primes in order to find the nth prime number.

Sieving Integers

Idea

Generate list of candidate integers, and systematically eliminate non-prime numbers. Remaining numbers are prime.

Suitability

This approach is reasonable, because we need to generate the first n primes in order to find the nth prime number.

Examples of Sieves

Sieving Integers

Idea

Generate list of candidate integers, and systematically eliminate non-prime numbers. Remaining numbers are prime.

Suitability

This approach is reasonable, because we need to generate the first n primes in order to find the nth prime number.

Examples of Sieves

- Sieve of Eratosthenes (following slides)

Sieving Integers

Idea

Generate list of candidate integers, and systematically eliminate non-prime numbers. Remaining numbers are prime.

Suitability

This approach is reasonable, because we need to generate the first n primes in order to find the nth prime number.

Examples of Sieves

- Sieve of Eratosthenes (following slides)
- Sieve of Atkin (involving modulo 60 computation)

Sieving Integers

Idea

Generate list of candidate integers, and systematically eliminate non-prime numbers. Remaining numbers are prime.

Suitability

This approach is reasonable, because we need to generate the first n primes in order to find the nth prime number.

Examples of Sieves

- Sieve of Eratosthenes (following slides)
- Sieve of Atkin (involving modulo 60 computation)
- Wheel sieves (based on wheel with first few prime numbers)

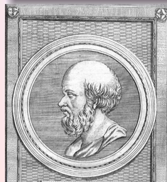
Erastosthenes

Biographical data

276 BCE–194 BCE; born in Cyrene (today in Libya); chief librarian of the Great Library of Alexandria

Famous for

Estimating the size of Earth with surprising accuracy



Program in C++

```
#include<iostream.h>
int main() {
  int n,i,j,MAX = 100; cin >> n;
  int prime[MAX];
  for (i=0;i<MAX;i++) prime[i] = 1;
  for (i=2; i<MAX; i++)
    if (prime[i])
      if (n--==0) break;
      else for (j = 2*i; j<MAX; j+=i)
             prime[j] = 0;
  cout << i << endl;
}
```

Complexity

- Iterating through `prime` array: Depends on size of array `MAX`.

Complexity

- Iterating through `prime` array: Depends on size of array `MAX`.
- In addition, execute `prime[j] = 0;` for each prime factor for each j.

Complexity

- Iterating through `prime` array: Depends on size of array `MAX`.
- In addition, execute `prime[j] = 0;` for each prime factor for each j.

The number of distinct prime factors of j is called $\omega(j)$

Complexity

- Iterating through `prime` array: Depends on size of array `MAX`.
- In addition, execute `prime[j] = 0;` for each prime factor for each j.

The number of distinct prime factors of j is called $\omega(j)$
$\omega(j) \sim \ln \ln j$

Complexity

- Iterating through `prime` array: Depends on size of array `MAX`.
- In addition, execute `prime[j] = 0;` for each prime factor for each j.

The number of distinct prime factors of j is called $\omega(j)$
$\omega(j) \sim \ln \ln j$
Therefore, overall runtime grows on average with $MAX \cdot \ln \ln MAX$.

1. PRIME
2. LVM
   - Problem
   - Algorithm
   - Program in Java
3. GECKO
4. HOUSING
5. RANK
6. 4SUM

Problem

You are designing a virtual machine for a new programming language called Lombok. The Lombok Virtual Machine (LVM) runs an assembler-like machine code. It operates on a stack and a single register.

PUSH -11

Before:

2
-3

After:

-11
2
-3

STORE

Before:

3
9
4

After:

9
4

and the register contains the integer 3.

LOAD

Before: Register contains 6

8
7

After: Register still contains 6

6
8
7

PLUS

Before:

2
3
4

After:

5
4

TIMES

Before:

2
3
4

After:

6
4

`IFZERO n`

Removes the topmost integer from the stack, and checks if it is equal to 0.
If yes, jump to the nth instruction.
If not, continues as usual with next instruction.

Example: IFZERO 3

Stack:

0
20

IFZERO 3 removes 0 from the stack and jumps to instruction with address 3.

DONE

Prints out the integer on top of the stack, and terminates the program.

Example: Compute $5! = 120$

```
0: PUSH 5
1: STORE
2: LOAD
3: LOAD
4: PUSH -1
5: PLUS
6: STORE
7: LOAD
8: IFZERO 13
9: LOAD
10: TIMES
11: PUSH 0
12: IFZERO 3
13: DONE
```

Algorithm

1. Read program into array of instructions
2. Initialize register, stack and program counter
3. Execute instructions until `DONE` is encountered
4. Print top of stack

Program in Java

```java
import java.util.Stack;
public class LVM {
    // opcodes declared as constants
    private static final int PUSH = 0;
    private static final int STORE = 1;
    private static final int LOAD = 2;
    private static final int PLUS = 3;
    private static final int TIMES = 4;
    private static final int IFZERO = 5;
    private static final int DONE = 6;
```

Program in Java

```
static abstract class Instruction {
    public int opcode;
};
// instruction classes
static class Store extends Instruction  {
  public Store(){
    opcode = STORE;
  };
}
...
```

Program in Java

```
static class IfZero extends Instruction {
    public int arg;
    public IfZero(int i) {
        opcode = IFZERO;
        arg = i;
    }
};
```

Program in Java

```
static class Push extends Instruction {
        public int arg;
        public Push(int i) {
            opcode = PUSH;
            arg = i;
        }
    };
```

Program in Java

```
public static void main(String[] args){
   // (1) read program into ``code'' (not shown)
   // (2) initialize stack and register
   Stack<Integer> s = new Stack<Integer>();
   Integer reg = 0;
```

Program in Java

```
// (3) execute instructions until DONE
loop:
for (int pc=0;;) {
  Instruction i = code[pc];
  switch (i.opcode) {
  case PUSH: {
    s.push(((Push) i).arg);
    pc++;
    break;
 }
```

Program in Java

```
case STORE: {
  reg = s.pop();
  pc++;
  break;
}
case LOAD: {
  s.push(reg);
  pc++;
  break;
}
```

Program in Java

```
      case IFZERO: {
        if (s.pop() == 0) {
          pc = ((IfZero) i).arg;
        } else pc++;
        break;
      }
      case DONE: {
        break loop;
    } } }
    // (4) print top of the stack
    System.out.println(s.pop());
} }
```

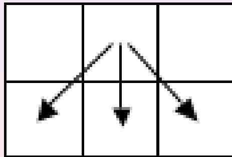
Complexity

Depends on the given LVM program!

1. PRIME
2. LVM
3. GECKO
    - Problem
    - Algorithm
4. HOUSING
5. RANK
6. 4SUM

Problem

During the rainy season, one of the walls in the house is infested with mosquitoes. The wall is covered by $h \times w$ square tiles, where there are h rows of tiles from top to bottom, and w columns of tiles from left to right. Each tile has 1 to 1000 mosquitoes resting on it.



Given the values of h and w, and the number of mosquitoes resting on each tile, write a program to compute the maximum possible number of mosquitoes the gecko can eat in one single trip from the top to the bottom of the wall.

Example

	0	1	2	3	4
0	3	1	7	4	2
1	2	1	3	1	1
2	1	2	2	1	8
3	2	2	1	5	3
4	2	1	4	4	4
5	5	7	2	5	1

Example

	0	1	2	3	4
0	3	1	7	4	2
1	2	1	3	1	1
2	1	2	2	1	8
3	2	2	1	5	3
4	2	1	4	4	4
5	5	7	2	5	1

Algorithm

Idea

In what ways can the gecko start?

	0	1	2	3	4
0	3	1	7	4	2
1	2	1	3	1	1
2	1	2	2	1	8
3	2	2	1	5	3
4	2	1	4	4	4
5	5	7	2	5	1

Algorithm

How best to reach the second row of tiles?

	0	1	2	3	4
0	3	1	7	4	2
1	2	1	3	1	1
2	1	2	2	1	8
3	2	2	1	5	3
4	2	1	4	4	4
5	5	7	2	5	1

Algorithm

How best to reach the second row of tiles?

	0	1	2	3	4
0	3	1	7	4	2
1	5	8	10	8	5
2	1	2	2	1	8
3	2	2	1	5	3
4	2	1	4	4	4
5	5	7	2	5	1

Algorithm

Observation

The best way to reach the next row only depends on the best way to reach the previous row.

"Greedy" algorithm

Compute best results from the top to bottom, each time using the previous row.

Ending

When reaching the bottom, the highest value tile contains the result.

Complexity

We need to perform a constant amount of work for each tile. Thus, the overall amount of work grows with the number of tiles (number of rows times number of columns).

1. PRIME
2. LVM
3. GECKO
4. HOUSING
    - Problem
    - Algorithm
5. RANK
6. 4SUM

Problem

For the Youth Olympic Games in Singapore, the administration is considering to house each team in several units with at least 5 people per unit. A team can have from 5 to 100 members, depending on the sport they do.

Problem

For the Youth Olympic Games in Singapore, the administration is considering to house each team in several units with at least 5 people per unit. A team can have from 5 to 100 members, depending on the sport they do.

For example, if there are 16 team members, there are 6 ways to distribute the team members into units: (1) one unit with 16 team members; (2) two units with 5 and 11 team members, respectively; (3) two units with 6 and 10 team members, respectively; (4) two units with 7 and 9 team members, respectively; (5) two units with 8 team members each; (6) two units with 5 team members each plus a third unit with 6 team members.

Problem

How many choices are there to distribute n team members into units?

Input and Output

Input:

20

Output:

13

Avoid Double-Counting

Idea

In order to compute the number of ways to split 15 people into groups of at least 5, find the number of ways to

1. split 10 people into groups of 5 (add a group of 5)
2. split 9 people into groups of 6 (add a group of 6)
3. split 8 people into groups of 7 (add a group of 7)
4. split 7 people into groups of 8 (add a group of 8)
5. split 6 people into groups of 9 (add a group of 9
6. split 5 people into groups of 10 (add a group of 10)

Naive Approach

```
int options(int n, int k) {
  if (n < k   ) return 0;
  if (n < 2*k) return 1;
  int result = 1;
  for (int i=k; i<n; i++)
    result += options(n-i,i);
  return result;
}
```

Complexity

Analysis

This leads to an exponential number of calls of *options*.

Complexity

Analysis

This leads to an exponential number of calls of *options*.

Judgement

The amount of time taken will quickly exceed the allotted time of 10 seconds.

Complexity

Analysis

This leads to an exponential number of calls of *options*.

Judgement

The amount of time taken will quickly exceed the allotted time of 10 seconds.

Idea

Remember previously computed intermediate results.

Complexity

Analysis

This leads to an exponential number of calls of *options*.

Judgement

The amount of time taken will quickly exceed the allotted time of 10 seconds.

Idea

Remember previously computed intermediate results.

Terminology

This idea is called *memoization*. The corresponding programming technique is called *dynamic programming*.

Dynamic Programming (Chang Ee Chien)

```
int options(int n, int k) {

  if (n < k  ) return 0;
  if (n < 2*k) return 1;
  int result = 1;
  for (int i=k; i<n; i++)
    result += options(n-i,i);

  return result;
}
```

Dynamic Programming (Chang Ee Chien)

```
int options(int n, int k) {
  if (T[n][k] >= 0)  return T[n][k];
  if (n < k   ) return 0;
  if (n < 2*k) return 1;
  int result = 1;
  for (int i=k; i<n; i++)
    result += options(n-i,i);
  T[n][k]=result;
  return result;
}
```

Complexity

Each table entry for n and k requires $n - k - 1$ additions.

Complexity

Each table entry for n and k requires $n - k - 1$ additions.
Overall runtime grows with the cube of the number of players.

1. PRIME
2. LVM
3. GECKO
4. HOUSING
5. RANK
   - Problem
   - Algorithm
6. 4SUM

Problem

In a new sports discipline called Triball proposed for the Youth Olympic Games in Singapore, a set of k players $\{1, 2, \ldots, k\}$ are involved (where $k < 1000$). In every match, 3 players are selected and the result of the match is either 1 winner or 1 loser. Given the result of n matches (where $n < 10000$), our task is to give a ranking list of the players.

Example

Consider 6 players $\{1, 2, 3, 4, 5, 6\}$. Suppose the results of 5 matches are:

1. $4 > 1, 3$,
2. $3 > 1, 2$,
3. $2, 5 > 6$,
4. $5 > 2, 3$,
5. $1 > 2, 6$.

Then, a valid ranking of the players is $4 > 5 > 3 > 1 > 2 > 6$.

Another Example

Consider 4 players $\{1, 2, 3, 4\}$. Suppose the results of 2 matches are:

1. $1 > 2, 4$ and
2. $2, 4 > 1$.

Then, there is no valid ranking of the players.

Input and Output File

Input:

```
6 5
4>1,3
3>1,2
2,5>6
5>2,3
1>2,6
```

Output:

```
4 5 3 1 2 6
```

Input and Output File

Input:

```
4 2
1>2,4
2,4>1
```

Output:

```
0
```

Idea

Topological sort

Find team that is never defeated and remove it.

Idea

Topological sort

Find team that is never defeated and remove it.
Repeat until no more teams.

Idea

Topological sort

Find team that is never defeated and remove it.
Repeat until no more teams.

Implementation

Pascal program by Frank Stephan

Which team is never defeated?

```
for v := 1 to k-1 do begin
    for u := 0 to k do begin
        for w := v+1 to k do begin
            if loses[teamAtPosition[v],teamAtPosition[w]]
            then begin temp := teamAtPosition[v];
                   teamAtPosition[v]:=teamAtPosition[w];
                   teamAtPosition[w] := temp
    end end end end;
```

Finding Inconsistencies

```
for v := 1 to k-1 do begin
  for w := v+1 to k do begin
      if loses[teamAtPosition[v],teamAtPosition[w]]
      then inconsistent:=true
  end end
```

Complexity

Three nested loops, each iterating over the number of teams k.

Complexity

Three nested loops, each iterating over the number of teams k. Therefore, complexity grows with k^3.

Problem

In this task, you are provided with four sets of integers. Your task consists of selecting one integer from each set, such that their sum is 0. You can assume that exactly one such selection exists.

Input File

```
3 2 4 2
5
17
-8
-13
19
6
-9
10
0
-14
7
```

Input File

```
3 2 4 2
5   |
17  |
-8  |
-13   |
19    |
6   |
-9  |
10  |
0   |
-14   |
7     |
```

Output File

```
17 -13 10 -14
```

Algorithm

Idea

In order for the overall sum to be 0, the sum of the first two values must be minus the sum of the last two values.

Procedure

List the sums of all possible first two values, and the negatives of the sums of all possible second two values.

Algorithm

Idea

In order for the overall sum to be 0, the sum of the first two values must be minus the sum of the last two values.

Procedure

List the sums of all possible first two values, and the negatives of the sums of all possible second two values. Then find the place where the two lists have same value.

Example

First value: 5,17,-8
Second value: -13,19

Example

First value: 5,17,-8
Second value: -13,19
Possible sums: -8

Example

First value: 5,17,-8
Second value: -13,19
Possible sums: -8, 24

Example

First value: 5,17,-8
Second value: -13,19
Possible sums: -8, 4, 24

Example

First value: 5,17,-8
Second value: -13,19
Possible sums: -8, 4, 24, 36

Example

First value: 5,17,-8
Second value: -13,19
Possible sums: -21, -8, 4, 24, 36

Example

First value: 5,17,-8
Second value: -13,19
Possible sums: -21, -8, 4, 11, 24, 36

Example

Third value: 6, -9, 10, 0
Fourth value: -14, 7

Example

Third value: 6, -9, 10, 0
Fourth value: -14, 7
Possible negative sums: 8

Example

Third value: 6, -9, 10, 0
Fourth value: -14, 7
Possible negative sums: -13, 8

Example

Third value: 6, -9, 10, 0
Fourth value: -14, 7
Possible negative sums: -17, -13, -7, 2, 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: -21, -8, 4, 11, 24, 36
Possible negative sums: -17, -13, -7, 2, 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: -8, 4, 11, 24, 36
Possible negative sums: -17, -13, -7, 2, 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: -8, 4, 11, 24, 36
Possible negative sums: -13, -7, 2, 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: -8, 4, 11, 24, 36
Possible negative sums: -7, 2, 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: 4, 11, 24, 36
Possible negative sums: -7, 2, 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: 4, 11, 24, 36
Possible negative sums: 2, 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: 4, 11, 24, 36
Possible negative sums: 4, 8, 14, 23

Example

Find match between sorted lists:
Possible positive sums: 4, 11, 24, 36
Possible negative sums: 4, 8, 14, 23
Bingo!

Complexity

We need to calculate all possible sums of the first two numbers and all possible sums of the second two numbers.

Complexity

We need to calculate all possible sums of the first two numbers and all possible sums of the second two numbers.

The amount of work to be done grows with the square of the list size.

Complexity

We need to calculate all possible sums of the first two numbers and all possible sums of the second two numbers.

The amount of work to be done grows with the square of the list size.

Then we need to sort the two lists, incurring an amount of work that grows with the square of the list size times the logarithm of the square of the list size.

Complexity

We need to calculate all possible sums of the first two numbers and all possible sums of the second two numbers.

The amount of work to be done grows with the square of the list size.

Then we need to sort the two lists, incurring an amount of work that grows with the square of the list size times the logarithm of the square of the list size.

This turns out to be equivalent to saying: The amount of work grows with the square of the list size times the logarithm of the list size.

Program in Java (Ben Leong)

```
static class Value implements Comparable<Value> {
  int value;
  int x;
  int y;

  public Value(int value, int x, int y) {
    this.value = value;
    this.x = x;
    this.y = y;
  }

  public int compareTo(Value o) {
    return value - o.value;
  }
}
```

Program in Java (Ben Leong)

```
// Create a+b and c+d arrays
PriorityQueue<Value> q1 = new PriorityQueue<Value>();
PriorityQueue<Value> q2 = new PriorityQueue<Value>();
for (int i = 0; i < a.length; i++) {
  for (int j = 0; j < b.length; j++) {
    q1.add(new Value(a[i] + b[j], i, j));
  }
}
for (int i = 0; i < c.length; i++) {
  for (int j = 0; j < d.length; j++) {
    q2.add(new Value(-(c[i] + d[j]), i, j));
  }
}
```

Program in Java (Ben Leong)

```
FileOutputStream fs = new FileOutputStream(output_file);
PrintWriter pw = new PrintWriter(fs);
// Traverse priority queues to find a match.
while (!q1.isEmpty() && !q2.isEmpty()) {
  int v1 = q1.peek().value;
  int v2 = q2.peek().value;
  if (v1 == v2) {
    Value V1 = q1.peek();
    Value V2 = q2.peek();
    pw.print(a[V1.x]+" "+b[V1.y]+" "+
             c[V2.x]+" "+d[V2.y]);
    q1.poll();
  } else if (v1 < v2) q1.poll();
  else q2.poll();
}
```