Fred’s Lotto Tickets

• Read the problem as many times as necessary
• Can you summarize the problem using your own words?
  – I need to check whether all integers in the range 1–49 appear on a given list of (possibly repeated) integers.
• Verify that your understanding of the problem is correct (this might save you time later)
Fred’s Lotto Tickets

• How will the data be represented?
  – Declare an array of >= 50 elements
  – Initialize each element in the array to 0
  – As you read each ticket number, set its corresponding element in the array to 1

• Now, array[k] == 1 if and only if you have the k-th ticket, and 0 otherwise.
Fred’s Lotto Tickets

• To check if you have all tickets in the 1-49 range:
  – Loop through the array (from 1 to 49)
  – If no element in the array is 0, then you have all the tickets and the answer is “YES”. Otherwise there is at least one missing ticket and the answer is “NO”
Fred’s Lotto Tickets

```cpp
#include <iostream>
using namespace std;
#define SZ 50

int main()
{
    int N;
    bool tickets[SZ];

    while (1)
    {
        for (int i = 0; i < SZ; i++)
            tickets[i] = false;

        cin >> N;
        if (N == 0)
            break;

        int t;
        for (int i = 0; i < N*6; i++)
        {
            cin >> t;
            tickets[t] = true;
        }

        bool all = true;
        for (int i = 1; i <= 49; i++)
        {
            if (!tickets[i])
            {
                all = false;
                break;
            }
        }

        if (all)
            cout << "Yes\n";
        else
            cout << "No\n";
    }

    return 0;
}
```
Lamps

• Read the problem as many times as necessary
• Can you summarize the problem in your own words?
  – An integer L will be given. Each of these integers represent a lamp which is initially turned off (0). L passes will be performed over all lamps. On the k-th pass, every k-th lamp will be toggled. Count how many lamps remain on after the last pass.
• Verify that your understanding of the problem is correct.
Lamps

• How will the data be represented?
  – Read the integer L (exit if L == 0)
  – Create a Boolean vector* of at least L+1 elements
  – Initialize each element to 0 (off)

```cpp
    cin >> L;
    if (L == 0)
        exit(0);

    vector<bool> lamps(L+1, false);
```

• The maximum possible L is not known in advance. It is better to avoid explicitly dealing with memory allocation if possible, at least in C/C++.
Lamps

• Use pseudocode when necessary:

  For i in range [1, L]:
    For j in range [i, L] in increments of i:
      toggle lamps[j]

• After this, you only need to count how many lamps[i] have a value of 1 (on).
Lamps

• Translate pseudocode to code:

```cpp
// perform the passes
for (int i = 1; i <= L; i++)
    for (int j = i; j <= L; j += i)
        lamps[j] = !lamps[j]; // toggle the lamp
```

• Count how many lamps remain on:

```cpp
int cnt = 0;
for (int i = 1; i <= L; i++)
    cnt += (int) lamps[i]; // 0 -> off; 1 -> on.
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int L;
    cin >> L;
    if (L == 0)
        break;
    vector<bool> lamps(L+1, false);
    int cnt = 0;
    for (int i = 1; i <= L; i++)
        for (int j = i; j <= L; j+=i)
            lamps[j] = !lamps[j];
    for (int i = 1; i <= L; i++)
        cnt += (int) lamps[i];
    cout << cnt << 'n';
    return 0;
}
What if $L$ is very large, say $2^{40}$?

A Boolean array of $2^{40}$ elements takes around 1 TB of space. The memory limit on most contests is usually in the 64 MB – 128 MB range.

Also, the previous approach will perform at least $2^{40}$ iterations, which will take too long to complete (there are also time constraints).

There’s no partial credit
Lamps

- 1 2 3 4 5 6 7 8 9
- 0 0 0 0 0 0 0 0 0 (initially)
- 1 1 1 1 1 1 1 1 1 (k = 1, all numbers)
- 1 0 1 0 1 0 1 0 1 (k = 2, multiples of 2)
- 1 0 0 0 1 1 1 0 0 (k = 3, multiples of 3)
- 1 0 0 1 1 1 1 1 0 (k = 4, multiples of 4)
- 1 0 0 1 0 1 1 1 0 (k = 5, multiples of 5)
- 1 0 0 1 0 0 1 1 0 (k = 6, multiples of 6)
- 1 0 0 1 0 0 0 1 0 (k = 7, multiples of 7)
- 1 0 0 1 0 0 0 0 0 (k = 8, multiples of 8)
- 1 0 0 1 0 0 0 0 1 (k = 9, multiples of 9)

- Only 1, 4 and 9 remain on after the last pass.
Lamps

• What do 1, 4 and 9 have in common?
• They are all perfect squares (their square root is an integer)
• This suggests a hypothesis:
  – If \( L \) is the number of lamps, then the number of perfect squares in the range \([1, L]\) is the number of lamps that remain on after the last pass
• Is this hypothesis true?
Lamps

• A lamp that is toggled an even number of times will end up off.

• On the \( k \)-th pass only multiples of \( k \) are toggled. In other words, on the \( k \)-th pass, a lamp is toggled if and only if its index has \( k \) as a factor.

• Therefore, lamp \( s \) is toggled exactly \( f \) times if \( s \) has \( f \) distinct factors.
Lamps

• Example:
  – Lamp 12 is toggled on passes 1, 2, 3, 4, 6, 12.
  – Lamp 12 is therefore toggled an even amount of times, so it remains off after the last pass.

• Thus the problem is reduced to finding how many integers in the range \([1, L]\) have an **odd** number of factors.
Lamps

• Suppose we are dealing with lamp $s$.

• If $u$ is a factor of $s$, then $u * v = s$, for some integer $v$. So $v$ is also a factor of $s$.

• A factor $u$ implies another factor $v$. In other words factors tend to come in pairs.

• How can there be an unpaired factor?
  – What if $u = v$?
  – Then $u * u = s$ and $s$ has an odd number of factors.
  – Multiplication is a binary operation so there must always be two operands. Thus, this is the only possible case.
Lamps

• This shows that the hypothesis is true:
  – If $L$ is the number of lamps, then the number of perfect squares in the range $[1, L]$ is the number of lamps that remain on after the last pass.

• For example, if $L = 100$, then there are 10 lamps on at the end of the process: 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.
Given L, how do we compute the number of perfect squares less than or equal to L?

Try small cases, as before:

<table>
<thead>
<tr>
<th>L</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSQ(L)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SQRT(L)</td>
<td>1</td>
<td>1.41</td>
<td>1.73</td>
<td>2</td>
<td>2.24</td>
<td>2.45</td>
<td>2.65</td>
<td>2.83</td>
<td>3</td>
</tr>
<tr>
<td>FLOOR(SQRT(L))</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Therefore floor(sqrt(L)) seems to give the right answer.
Lamps

• In a contest, the second solution is only better than the first solution only if:
  – L is so large that the first solution becomes incorrect (on the account of spatial/temporal constraints)
  – You already knew the second solution (easier to code)
  – You came up with the second solution first (…)

• Usually the simplest solution is the best solution
The Heart of the Country

- Read the problem as many times as necessary
- Can you summarize the problem using your own words?
  - Two integers $N$ (cities) and $K$ (min troops) will be given.
  - Each city has a certain amount of troops and is connected to some other cities.
  - A city $C$ is *defendable* if $K$ is less than or equal to the number of troops in $C$ plus the number of troops in cities directly connected to $C$.
  - If a city is not defendable it is destroyed and its troops are lost.
  - What is the largest subset of mutually defendable cities?
The Heart of the Country - Input

• Two integers $N$ (cities) and $K$ (min troops)
• $N$ lines (one for each city):
  – First integer $T_i$ is the number of troops at that city
  – Second integer $H_i$ is the number of highways going out of that city
  – The next $H_i$ integers indicate to which city each of these highways go to
• How should this information be represented?
The Heart of the Country

Sample Input

4   900
100 2   1   2
200 2   0   3
500 2   0   3
1000 2   1   2

• 4 cities
• Minimum number of troops required to protect a city = 900
The Heart of the Country
Internal Representation

• How to represent this information (efficiently) internally?

• Keep a two dimensional array (matrix) $country$ such that:
  – $country[i][j] = 1$ if and only if there is a highway between city $i$ and city $j$

• Also, use an array to keep track of troops in each city
The Heart of the Country

Example

country =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- country[0][2] = 1, so there is a highway between 0 and 2
- country[3][0] = 0, so there is no highway between 0 and 3
The Heart of the Country
Declaring the data structures

• The maximum number of cities is 1000
• To keep things static declare your data structures using the maximum possible size if possible

#define MAXCITIES 1005
int country[MAXCITIES][MAXCITIES];
int troops[MAXCITIES];
while (1) {
    int N;
    int K;
    cin >> N >> K;

    if (N == 0 && K == 0)
        exit(0);

    initialize(country, 0);
    initialize(troops, 0);

    int city_troops;
    int total_nbs;
    int neighbor;

    for (int i = 0; i < N; i++) {
        cin >> city_troops >> total_nbs;
        troops[i] = city_troops;

        for (int j = 0; j < total_nbs; j++) {
            cin >> neighbor;
            graph[i][neighbor] = true;
        }
    }

    .. .. //find the heart
    .. .. //output the answer
} // end while
The Heart of the Country

• A city is not defendable if the number of troops in the city plus the number of troops in the neighboring cities is less than K
• If a city is not defendable it is destroyed and its troops are lost.
• Find the largest mutually defendable subset of cities
  – Key observation: the total number of troops protecting a city can never increase (but it can decrease)
The Heart of the Country

– First, eliminate all cities that can be directly destroyed. This must eventually be done by every algorithm since none of these cities can belong to the final answer

– This will generate more directly destroyable cities. These must also be eliminated since they cannot be in the final answer either

– Continue the above process until no more cities become destroyable. The remaining cities are the heart of the country
• Algorithm (pseudocode):

do
    done = true
    for each city c that has not been destroyed:
        p ← count troops protecting c
        if c < K:
            done = false
            destroy c and its troops
    while not done
The Heart of the Country

do {
    done = true;
    for (int i = 0; i < N; i++) //for each city
    {
        //if the city has been destroyed, skip it
        if (troops[i] == DESTROYED)
            continue;

        //count how many troops are protecting city i
        int cnt = troops[i];
        for (int j = 0; j < N; j++)
            if (country[i][j] && troops[j] != DESTROYED)
                cnt += troops[j];

        if (cnt < K)
        {
            troops[i] = DESTROYED;
            done = false;
        }
    }
} while (!done);
The Heart of the Country

• What is the time complexity of this solution?
  – $O(n^3)$

• Think about how would you reduce this
The Heart of the Country

Output

• You need to output the number of cities in the heart of the country as well as the total number of troops that remain

```python
num_heart = 0
troops_heart = 0
for each city c:
    if c has not been destroyed:
        num_heart += 1
        troops_heart += troops[c]
output num_heart + " " + troops_heart + "\n"
```
int num_heart = 0;
int troops_heart = 0;
for (int i = 0; i < N; i++)
    if (troops[i] != DESTROYED) {
        num_heart++;
        troops_heart += troops[i];
    }

cout << num_heart << "  " << troops_heart << "\n";
The Heart of the Country

• This problem can be modeled by using Graph Theory
• Graphs are frequently found in programming competitions
• Many problems can be modeled by using graphs
• Leonhard Euler published what is considered to be the first paper on Graph Theory in 1736
Graphs

• Formally, a graph is the following:
  – A set of vertices $V$
  – A set of edges $E$ consisting of pair of vertices

• Think of vertices as the cities. In this case, $V$ was the set of all cities

• Think of edges as the highways between cities. Then, $E$ is the set of all highways
Graphs

- **Representation:**
  - Vertices (nodes) are usually depicted as circles or dots
  - Edges are usually depicted as lines between them

In this example graph:

- \( V = \{1, 2, 3, 4, 5, 6\} \)
- \( E = \{(1,3), (1,6), (2,5), (3,4), (3,6)\} \)

Two vertices \( u \) and \( v \) are said to be adjacent if there is an edge \((u,v)\) in the graph.
Graphs

- Sometimes numerical weights are associated with the edges. These graphs are known as edge-weighted graphs (or weighted graphs).
- Similarly, nodes can also be weighted and are known as node-weighted graphs.
- In THOTC problem the graph was node-weighted.
Graphs

• Graphs can also be **undirected**
• In undirected graphs edges go only one way

Undirected graphs are a special case of directed graphs

Why?

If you want an undirected edge between node u and node v, just add two edges to the graph: (u, v) and (v, u).
Identify the graph - I

• Given a set of computers and a set of wires running between pairs of computers, what is the minimum number of machines whose crash causes two given machines to be unable to communicate? (The two given machines will not crash.) (USACO Championship 1996)

• The vertices of the graph are the computers. The edges are the wires between the computers.
Identify the graph - II

• Two squares on an 8x8 chessboard. Determine the shortest sequence of knight moves from one square to the other

• Each location on the chessboard represents a vertex. There is an edge between two positions if it is a legal knight move
Graphs – Data Structures

• There are four main ways to represent graphs:
  – Edges List
  – Adjacency List
  – Adjacency Matrix (this is what we used in THOTC)
  – Implicit Representation (chess problem)

• More about this next time
Assigned Problems

• 1) http://online-judge.uva.es/p/v100/10037.html
• 2) http://online-judge.uva.es/p/v101/10188.html
• Go to: http://icpcres.ecs.baylor.edu/onlinejudge/
• Doesn’t work with IE
• Register (check “results email”, this is important)
• Submit the problems using the language of your choice and the problem ID given in the problem statement
• Please record how long it took you to solve the problem
• When your problem is accepted, forward the email to rosentha@cs.fsu.edu with the problem ID as the subject followed by your name, for example:
  Subject: 100342 – Andre Rodriguez
TopCoder Competitions

• Go to:
  – Register
  – Click on the O(n) sign near the upper left corner
  – Go to a practice room and take a look at the problems as well as at other people’s code (especially red-rated coders)
  – Also check out the problem archive, editorials, tutorials, etc
SPOJ

• Go to: www.spoj.pl
• Register
• You can choose from a wider variety of languages (Haskell, Python, Ada, Whitespace, Scheme, Perl, Bash and many more) along with the usual languages