```
initialize_single_source(Graph G,Node s)
  for each vertex v in Vertices(G)
    G.d[v]:=infinity
    G.pi[v]:=nil
  G.d[s]:=0;
```

This sets up the graph so that each node has no predecessor (**pi[v] = nil**) and the estimates of the cost (distance) of each node from the source (**d[v]**) are infinite, except for the source node itself (**d[s] = 0**).

Note that we have also introduced a further way to store a graph (or part of a graph - as this structure can only store a spanning tree), the **predecessor sub-graph** - the list of predecessors of each node, **pi[j], 1 <= j <=** |**V**|. The edges in the predecessor sub-graph are **(pi[v],v)**.

The relaxation procedure checks whether the current best estimate of the shortest distance to **v** (**d[v]**) can be improved by going through **u** (*i.e.* by making **u** the predecessor of **v**):

```
relax(Node u,Node v,double w[][])
  if d[v] > d[u] + w[u,v] then
    d[v]:=d[u] + w[u,v]
    pi[v]:=u
```

Dijkstra Algorithm

Dijkstra's algorithm (invented by Edsger W. Dijkstra) solves the problem of finding the shortest path from a point in a graph (the *source*) to a destination. It turns out that one can find the shortest paths from a given source to *all* points in a graph in the same time, hence this problem is called the **Single-source shortest paths** problem.

There will also be no cycles as a cycle would define more than one path from the selected vertex to at least one other vertex. For a graph, **G=(V,E)** where **V** is a set of vertices and **E** is a set of edges.

Dijkstra's algorithm keeps two sets of vertices: **S (**the set of vertices whose shortest paths from the source have already been determined) *and* **V-S (**the remaining vertices). The other data structures needed are: **d** (array of best estimates of shortest path to each vertex) & **pi** (an array of predecessors for each vertex)

The basic mode of operation is:

```
1. Initialise d and pi,
2. Set S to empty,
```

```
3.  While there are still vertices in V-S,
4.  Sort the vertices in V-S according to the current best estimate of
      their distance from source,
5.  Add u, the closest vertex in V-S, to S,
6.  Relax all the vertices still in V-S connected to u
```

```
DIJKSTRA(Graph G,Node s)
  initialize_single_source(G,s)
  S:={ 0 }       /* Make S empty */
  Q:=Vertices(G) /* Put the vertices in a PQ */
  while not Empty(Q)
    u:=ExtractMin(Q);
    AddNode(S,u); /* Add u to S */
    for each vertex v which is Adjacent with u
      relax(u,v,w)
```

Bellman-Ford Algorithm

A more generalized single-source shortest paths algorithm which can find the shortest path in a graph with negative weighted edges. If there is no negative cycle in the graph, this algorithm will updates each d[v] with the shortest path from s to v, fill up the predecessor list "pi", and return TRUE. However, if there is a negative cycle in the given graph, this algorithm will return FALSE.

```
BELLMAN_FORD(Graph G,double w[][],Node s)
  initialize_single_source(G,s)
  for i=1 to |V[G]|-1
    for each edge (u,v) in E[G]
      relax(u,v,w)

  for each edge (u,v) in E[G]
    if d[v] > d[u] + w(u, v) then
      return FALSE
  return TRUE
```

TEST YOUR BELLMAN FORD KNOWLEDGE

Solve Valladolid Online Judge Problems related with Bellman Ford:

558 - Wormholes, simply check the negative cycle existence.

Single-source shortest paths in Directed Acyclic Graph (DAG)

There exist a more efficient algorithm for solving Single-source shortest path problem for a Directed Acyclic Graph (DAG). So if you know for sure that your graph is a DAG, you may want to consider this algorithm instead of using Djikstra.

```
DAG_SHORTEST_PATHS(Graph G,double w[][],Node s)
  topologically sort the vertices of G // O(V+E)
  initialize_single_source(G,s)

  for each vertex u taken in topologically sorted order
    for each vertex v which is Adjacent with u
      relax(u,v,w)
```

A sample application of this DAG_SHORTEST_PATHS algorithm (as given in CLR book) is to solve critical path problem, i.e. finding the longest path through a DAG, for example: calculating the fastest time to complete a complex task consisting of smaller tasks when you know the time needed to complete each small task and the precedence order of tasks.

Floyd Warshall

Given a directed graph, the Floyd-Warshall All Pairs Shortest Paths algorithm computes the shortest paths between each pair of nodes in O(n^3). In this page, we list down the Floyd Warshall and its variant plus the source codes.

Given:
w : edge weights
d : distance matrix
p : predecessor matrix

w[i][j] = length of direct edge between i and j
d[i][j] = length of shortest path between i and j
p[i][j] = on a shortest path from i to j, p[i][j] is the last node before j.

Initialization

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++) {
    d[i][j] = w[i][j];
    p[i][j] = i;
  }
for (i=0; i<n; i++)  d[i][i] = 0;
```

The Algorithm

```
for (k=0;k<n;k++) /* k -> is the intermediate point */
  for (i=0;i<n;i++) /* start from i */
    for (j=0;j<n;j++) /* reaching j */
      /* if i-->k + k-->j is smaller than the original i-->j */
      if (d[i][k] + d[k][j] < d[i][j]) {

        /* then reduce i-->j distance to the smaller one i->k->j */
        graph[i][j] = graph[i][k]+graph[k][j];
        /* and update the predecessor matrix */
        p[i][j] = p[k][j];
      }
```

In the k-th iteration of the outer loop, we try to improve the currently known shortest paths by considering k as an intermediate node. Therefore, after the k-th iteration we know those shortest paths that only contain intermediate nodes from the set {0, 1, 2,...,k}. After all n iterations we know the real shortest paths.

Constructing a Shortest Path

```
print_path (int i, int j) {
  if (i!=j)    print_path(i,p[i][j]);
  print(j);
}
```

TEST YOUR FLOYD WARSHALL KNOWLEDGE

Solve Valladolid Online Judge Problems related with Floyd Warshall:

104 - Arbitrage - modify the Floyd Warshall parameter correctly
423 - MPI Maelstrom
436 - Arbitrage (II) - modify the Floyd Warshall parameter correctly
567 - Risk - even though you can solve this using brute force

Transitive Hull

Given a directed graph, the Floyd-Warshall algorithm can compute the Transitive Hull in O(n^3). Transitive means, if i can reach k and k can reach j then i can reach j. Transitive Hull means, for all vertices, compute its reachability.

w : adjacency matrix
d : transitive hull

w[i][j] = edge between i and j (0=no edge, 1=edge)
d[i][j] = 1 if and only if j is reachable from i

Initialization

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    d[i][j] = w[i][j];

for (i=0; i<n; i++)
  d[i][i] = 1;
```

The Algorithm

```
for (k=0; k<n; k++)
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      /* d[i][j] is true if d[i][j] already true
         or if we can use k as intermediate vertex to reach j from i,
         otherwise, d[i][j] is false */
      d[i][j] = d[i][j] || (d[i][k] && d[k][j]);
```

TEST YOUR TRANSITIVE HULL FLOYD WARSHALL KNOWLEDGE

Solve Valladolid Online Judge Problems related with Transitive Hull:
334 - Identifying Concurrent Events - internal part of this problem needs transitive hull, even though this problem is more complex than that.

MiniMax Distance

Given a directed graph with edge lengths, the Floyd-Warshall algorithm can compute the minimax distance between each pair of nodes in O(n^3). For example of a minimax problem, refer to the Valladolid OJ problem below.

w : edge weights
d : minimax distance matrix
p : predecessor matrix

w[i][j] = length of direct edge between i and j
d[i][j] = length of minimax path between i and j

Initialization

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    d[i][j] = w[i][j];

for (i=0; i<n; i++)
  d[i][i] = 0;
```

The Algorithm

```
for (k=0; k<n; k++)
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      d[i][j] = min(d[i][j], max(d[i][k], d[k][j]));
```

TEST YOUR MINIMAX FLOYD WARSHALL KNOWLEDGE

Solve Valladolid Online Judge Problems related with MiniMax:

534 - Frogger - select the minimum of longest jumps
10048 - Audiophobia - select the minimum of maximum decibel along the path

MaxiMin Distance

You can also compute the maximin distance with the Floyd-Warshall algorithm. Maximin is the reverse of minimax. Again, look at Valladolid OJ problem given below to understand maximin.

w : edge weights
d : maximin distance matrix
p : predecessor matrix

w[i][j] = length of direct edge between i and j
d[i][j] = length of maximin path between i and j

Initialization

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    d[i][j] = w[i][j];

for (i=0; i<n; i++)
  d[i][i] = 0;
```

The Algorithm

```
for (k=0; k<n; k++)
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      d[i][j] = max(d[i][j], min(d[i][k], d[k][j]));
```

TEST YOUR MAXIMIN FLOYD WARSHALL KNOWLEDGE

Solve Valladolid Online Judge Problems related with MaxiMin:

544 - Heavy Cargo - select the maximum of minimal weight allowed along the path.
10099 - The Tourist Guide - select the maximum of minimum passenger along the path, then divide total passenger with this value to determine how many trips needed.

Safest Path

Given a directed graph where the edges are labeled with survival probabilities, you can compute the safest path between two nodes (i.e. the path that maximizes the product of probabilities along the path) with Floyd Warshall.

w : edge weights
p : probability matrix

w[i][j] = survival probability of edge between i and j
p[i][j] = survival probability of safest path between i and j

Initialization

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    p[i][j] = w[i][j];

for (i=0; i<n; i++)
  p[i][i] = 1;
```

The Algorithm

```
for (k=0; k<n; k++)
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      p[i][j] = max(p[i][j], p[i][k] * p[k][j]);
```

Graph Transpose

Input: directed graph G = (V,E)
Output: graph $G^T = (V,E^T)$, where E^T = {(v,u) in VxV : (u,v) in E}.
i.e. G^T is G with all its edges reversed.

Describe efficient algorithms for computing G^T from G, for both the adjacency-list and adjacency-matrix representations of G. Analyze the running times of your algorithms.

Using Adjacency List representation, array B is the new array of Adjacency List G^T

```
for (i=1;i<=p;i++)
  B[i] = nil;

for (i=1;i<=p;i++)
  repeat {
    append i to the end of linked list B[A[i]];
    get next A[i];
  } until A[i] = nil;
```

Eulerian Cycle & Eulerian Path

Euler Cycle
Input: Connected, directed graph G = (V,E)
Output: A cycle that traverses every edge of G exactly once, although it may visit a vertex more than once.
Theorem: A directed graph possesses an Eulerian cycle iff
1) It is connected
2) For all {v} in {V} indegree(v) = outdegree(v)

Euler Path
Input: Connected, directed graph G = (V,E)
Output: A path from v1 to v2, that traverses every edge of G exactly once, although it may visit a vertex more than once.

Theorem: A directed graph possesses an Eulerian path iff
1) It is connected
2) For all {v} in {V} indegree(v) = outdegree(v) with the possible exception of two vertices v1,v2 in which case,
a) indegree(v1) = outdegree(v2) + 1
b) indegree(v2) = outdegree(v1) - 1

Topological Sort

Input: A directed acyclic graph (DAG) G = (V,E)
Output: A linear ordering of all vertices in V such that if G contains an edge (u,v), then u appears before v in the ordering.

If drawn on a diagram, Topological Sort can be seen as a vertices along horizontal line, where all directed edges go from left to right. A directed graph G is acylic if and only if a DFS of G yields no back edge.

Topological-Sort(G)

1. call DFS(G) to compute finishing times f[v] for each vertex v
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices

Topological-Sort runs in O(V+E) due to DFS.

Strongly Connected Components

Input: A directed graph G = (V,E)
Output: All strongly connected components of G, where in strongly connected component, all pair of vertices u and v in that component, we have u ~~> v and v ~~> u, i.e. u and v are reachable from each other.

Strongly-Connected-Components(G)
1. call DFS(G) to compute finishing times f[u] for each vertex u
2. compute GT, inversing all edges in G using adjacency list
3. call DFS(GT), but in the main loop of DFS, consider the vertices in order of decreasing f[u] as computed in step 1
4. output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected component.

Strongly-Connected-Components runs in O(V+E)

CHAPTER 13 COMPUTATIONAL GEOMETRY

Computational Geometry is an important subject. Mastering this subject can actually help you in programming contests since every contest usually include 1-2 geometrical problems.[2]

Geometrical objects and its properties

Earth coordinate system

People use *latitudes* (horizontal lines) and *longitudes* (vertical lines) in Earth coordinate system.

Longitude spans from 0 degrees (Greenwich) to +180* East and -180* West. Latitude spans from 0 degrees (Equator) to +90* (North pole) and -90* (South pole).

The most interesting question is what is the spherical / geographical distance between two cities p and q on earth with radius r, denoted by (p_lat,p_long) to (q_lat,q_long). All coordinates are in radians. (i.e. convert [-180..180] range of longitude and [-90..90] range of latitudes to [-pi..pi] respectively.

After deriving the mathematical equations. The answer is as follow:

```
spherical_distance(p_lat,p_long,q_lat,q_long) =
acos( sin(p_lat) * sin(q_lat) + cos(p_lat) * cos(q_lat) * cos(p_long -
q_long) ) * r

since cos(a-b) = cos(a)*cos(b) + sin(a)*sin(b), we can simplify the above
formula to:

spherical_distance(p_lat,p_long,q_lat,q_long) =
acos( sin(p_lat) * sin(q_lat) +
        cos(p_lat) * cos(q_lat) * cos(p_long) * cos(q_long) +
        cos(p_lat) * cos(q_lat) * sin(p_long) * sin(q_long)
      ) * r
```

TEST YOUR EARTH COORDINATE SYSTEM KNOWLEDGE

Solve UVa problems related with Earth Coordinate System:

535 - Globetrotter
10075 - Airlines - combined with all-pairs shortest path

Convex Hull

Basically, Convex Hull is the most basic and most popular computational geometry problem. Many algorithms are available to solve this efficiently, with the best lower bound O(n log n). This lower bound is already proven.

Convex Hull problem (2-D version):

Input: A set of points in Euclidian plane
Output: Find the minimum set of points that enclosed all other points.

Convex Hull algorithms:

a. Jarvis March / Gift Wrapping
b. Graham Scan
c. Quick Hull
d. Divide and Conquer

CHAPTER 14 VALLADOLID OJ PROBLEM CATEGORY[2]

Math	
(General)	113,202 ,256,275,276,294326,332,347, 350,356,374,377,382,386, 412,465,471,474,485,498550,557,568,594,725,727,846,10006,10014,10019,10042,10060,10071,10093,10104,10106,10107,10110,10125,10127,10162,10190,10193,10195,10469
Prime Numbers	406,516,543,583,686,10140,10200,10490
Geometry	190, 191, 378,438,476,477,478,10112,10221,10242,10245,10301,10432,10451
Big Numbers	324,424,495,623,713,748,10013,10035,10106,10220, 10334
Base Numbers	343,355,389,446,575,10183,10551
Combinations / Permutations	369,530
Theories / Formulas	106,264,486,580
Factorial	160, 324, 10323, 10338
Fibonacci	495,10183,10334,10450
Sequences	138,10408
Modulo	10176,10551
Dynamic Programming	
General	108, 116,136,348,495,507,585,640,836,10003,10036,10074,10130,10404
Longest Inc/Decreasing Subsequence	111,231,497,10051,10131
Longest Common Subsequence	531, 10066, 10100, 10192,10405
Counting Change	147, 357, 674
Edit Distance	164,526
Graph	
Floyd Warshall All-Pairs Shortest Path	1043, ,436,534,544,567,10048,10099,10171,112,117,122,193, 336,352,383, 429,469, 532, 536, 590, 614, 615, 657, 677, 679, 762,785,10000,10004,10009,10010,10116,10543
Network Flow	820, 10092, 10249
Max Bipartite Matching	670,753,10080
Flood Fill	352,572
Articulation Point	315,796
MST	10034,10147,10397
Union Find	459,793,10507
Chess	167,278,439,750
Mixed Problems	
Anagram	153,156,195,454,630
Sorting	120,10152,10194,10258
Encryption	458,554,740,10008,10062
Greedy Algorithm	10020,10249,10340
Card Game	162,462,555
BNF Parser	464,533
Simulation	130,133,144, 151, 305, 327,339,362,379402,440,556,637,758,10033,10500
Output-related	312,320, 330, 337, 381,391,392400,403,445,488,706,10082

Ad Hoc	101,102,103, 105,118, 119,121,128, 142,145,146,154, 155,187,195220,227,232, 271,272,291,297,299,300,311,325,333,335,340,344,349,353,380,384,394,401,408,409,413,414,417,434,441,442,444,447,455,457,460,468,482,483,484,489,492,494,496,499537,541,542,551,562,573,574,576,579,586,587,591602,612,616,617,620,621,642,654,656,661,668,671,673729,755,837,10015,10017,10018,10019,10025,10038, 10041,10045,10050,10055,10070,10079,10098,10102, 10126, 10161,10182,10189, 10281,10293, 10487
Array Manipulation	466,10324,10360,10443
Binary Search	10282,10295,10474
Backtracking	216,291422,524,529, 539, 571, 572, 574,10067,10276,10285,10301,10344,10400,10422,10452
3n+1 Problem	100,371,694

This problems are available at (http://acm.uva.es/p). New problems are added after each online contest at Valladolid Online Judge as well as after each ACM Regional Programming Contest, problems are added to live ACM archive (http://cii-judge.baylor.edu/).

APPENDIX A
ACM PROGRAMMING PROBLEMS

This part of this book contains some interesting problems from ACM/ICPC. Problems are collected from Valladolid Online Judge. You can see the reference section for finding more problem sources.

Find the ways !

The Problem

An American, a Frenchman and an Englishwoman had been to Dhaka, the capital of Bangladesh. They went sight-seeing in a taxi. The three tourists were talking about the sites in the city. The American was very proud of tall buildings in New York. He boasted to his friends, "Do you know that the Empire State Building was built in three months?"

"Really?" replied the Frenchman. "The Eiffel Tower in Paris was built in only one month! (However, The truth is, the construction of the Tower began in January 1887. Forty Engineers and designers under Eiffel's direction worked for two years. The tower was completed in March 1889.)

"How interesting!" said the Englishwoman. "Buckingham Palace in London was built in only two weeks!!"

At that moment the taxi passed a big slum (However, in Bangladesh we call it "Bostii"). "What was that? When it was built ?" The Englishwomen asked the driver who was a Bangladeshi.

"I don't know!" , answered the driver. "It wasn't there yesterday!"

However in Bangladesh, illegal establishment of slums is a big time problem. Government is trying to destroy these slums and remove the peoples living there to a far place, formally in a planned village outside the city. But they can't find any ways, how to destroy all these slums!

Now, can you imagine yourself as a slum destroyer? In how many ways you can destroy ***k*** slums out of ***n*** slums ! Suppose there are 10 slums and you are given the permission of destroying 5 slums, surly you can do it in 252 ways, which is only a 3 digit number, Your task is to find out the digits in ways you can destroy the slums !

The Input

The input file will contain one or more test cases. Each test case consists of one line containing two integers n (n>=1) and k (1<=<k=<n).

Sample Input

```
20 5
100 10
200 15
```

Sample Output

```
5
14
23
```

Problem Setter : Ahmed Shamsul Arefin

I Love Big Numbers !

The Problem

A Japanese young girl went to a Science Fair at Tokyo. There she met with a Robot named Mico-12, which had AI (You must know about AI-Artificial Intelligence). The Japanese girl thought, she can do some fun with that Robot. She asked her, "Do you have any idea about maths ?"."Yes! I love mathematics", The Robot replied.

"Okey ! Then I am giving you a number, you have to find out the Factorial of that number. Then find the sum of the digits of your result!. Suppose the number is 5.You first calculate 5!=120, then find sum of the digits 1+2+0=3.Can you do it?"

"Yes. I can do!"Robot replied."Suppose the number is 100, what will be the result ?".At this point the Robot started thinking and calculating. After a few minutes the Robot head burned out and it cried out loudly "Time Limit Exceeds".The girl laughed at the Robot and said "The sum is definitely 648". "How can you tell that ?" Robot asked the girl. "Because I am an ACM World Finalist and I can solve the Big Number problems easily." Saying this, the girl closed her laptop computer and went away. Now, your task is to help the Robot with the similar problem.

The Input

The input file will contain one or more test cases. Each test case consists of one line containing an integers n (n<=1000).

The Output

For each test case, print one line containing the required number. This number will always fit into an integer, i.e. it will be less than 2^31-1.

Sample Input

```
5
60
100
```

Sample Output

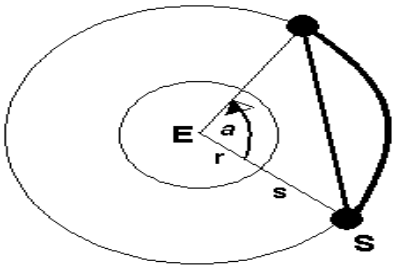
```
3
288
648
```

Problem Setter : Ahmed Shamsul Arefin

Satellites

The Problem

The radius of earth is 6440 Kilometer. There are many Satellites and Asteroids moving around the earth. If two Satellites create an angle with the center of earth, can you find out the ***distance*** between them? By ***distance*** we mean both the **arc** and **chord** distances. Both satellites are on the same orbit. (However, please consider that they are revolving on a circular path rather than an elliptical path.)



E = Earth S = Satellite

The Input

The input file will contain one or more test cases.

Each test case consists of one line containing two-integer ***s*** and ***a*** and a string "**min**" or "**deg**". Here ***s*** is the distance of the satellite from the surface of the earth and ***a*** is the angle that the satellites make with the center of earth. It may be in minutes (′) or in degrees (^{0}). Remember that the same line will never contain minute and degree at a time.

The Output

For each test case, print one line containing the required distances i.e. both ***arc distance*** and ***chord distance*** respectively between two satellites in Kilometer. The distance will be a floating-point value with six digits after decimal point.

Sample Input

```
500 30 deg
700 60 min
200 45 deg
```

Sample Output

```
3633.775503 3592.408346
124.616509 124.614927
5215.043805 5082.035982
```

Problem Setter : Ahmed Shamsul Arefin

Decode the Mad man

The Problem

Once in BUET, an old professor had gone completely mad. He started talking with some peculiar words. Nobody could realize his speech and lectures. Finally the BUET authority fall in great trouble. There was no way left to keep that man working in university. Suddenly a student (definitely he was a registered author at UVA ACM Chapter and hold a good rank on 24 hour-Online Judge) created a program that was able to decode that

professor's speech. After his invention, everyone got comfort again and that old teacher started his everyday works as before.

So, if you ever visit BUET and see a teacher talking with a microphone, which is connected to a IBM computer equipped with a voice recognition software and students are taking their lecture from the computer screen, don't get thundered! Because now your job is to write the same program which can decode that mad teacher's speech!

The Input

The input file will contain only one test case i.e. the encoded message.

The test case consists of one or more words.

The Output

For the given test case, print a line containing the decoded words. However, it is not so hard task to replace each letter or punctuation symbol by the two immediately to its left alphabet on your standard keyboard.

Sample Input

```
k[r dyt I[o
```

Sample Output

```
how are you
```

Problem Setter: Ahmed Shamsul Arefin

How many nodes ?

The Problem

One of the most popular topic of Data Structures is Rooted Binary Tree. If you are given some nodes you can definitely able to make the maximum number of trees with them. But if you are given the maximum number of trees built upon a few nodes, Can you find out how many nodes built those trees?

The Input

The input file will contain one or more test cases. Each test case consists of an integer n (n<=4,294,967,295). Here n is the maximum number of trees.

The Output

For each test case, print one line containing the actual number of nodes.

Sample Input

```
5
14
42
```

Sample Output

```
3
4
5
```

Problem Setter: Ahmed Shamsul Arefin

Power of Cryptography

Background

Current work in cryptography involves (among other things) large prime numbers and computing powers of numbers modulo functions of these primes. Work in this area has resulted in the practical use of results from number theory and other branches of mathematics once considered to be of only theoretical interest.

This problem involves the efficient computation of integer roots of numbers.

The Problem

Given an integer $n \geq 1$ and an integer $p \geq 1$ you are to write a program that determines $\sqrt[n]{p}$, the positive n^{th} root of p. In this problem, given such integers n and p, p will always be of the form k^n for an integer k (this integer is what your program must find).