[K Balance](https://www.hackerrank.com/contests/may13/challenges/k-blance)

Find the sum of numbers between L and R having either <= K digits or having sum of first K digits equal to sum of last K digits.

**Difficulty Level**

Hard

**Problem Setter**

[Wanbo](https://www.hackerrank.com/wanbo)  
Editorial by [Piyush](https://www.hackerrank.com/piyush006)

**Required Knowledge**

Combinatorics, Dynamic Programming

**Approach**

Let’s reduce the problem to finding

f(n) = the sum of K-balance numbers <= N

The answer to the problem will then be

f(r) - f(l-1)

f(N){

// D is the number of digits of N.

D = Digits(N);

ans = 0;

//For numbers with lesser number of digits.

for(I = 1; I < |D|;I++) ans += f(I);

//For numbers with equal number of digits.

ans += Sum(); // what is sum?

}

We can reduce K if the first K digits overlap with last K digits, so that they don’t overlap. To find Sum() , we can break our sum into sum at individual digit position and then sum it as :

10^(d-1) \* S(d-1) + 10^(d-2)\*S(d-2) + .. 10^(0) \* S(0) // S(i) is the sum of digits at position i

For finding sum at individual digit position i, we can again break it into as :

0\*Count(0) + 1\*Count(1) + 2\*Count(2) + ... + 9\*Count(9) // Count(j) is the count of numbers having digit j at position i.

So our approach looks like :

Sum()

{

ans = 0;

for(I = 0;I <= |D|;I++){

SumAtI = 0;

for(J=1;J<=9;J++){

SumAtI += J\*Count(I , J);

}

ans += SumAtI \* 10^(I);

}

}

Finding Count(p , x)  
We have simply fixed the digit at some position ‘p’ to be ‘x’ , and we have to find the count of numbers which have digit at position ‘p’ as ‘x’. There have to be |D| - p digits before p , and p - 1 after p that will form a number . Also sum of first K digits should be equal to last K . Another condition is that since the number of digits is equal to |D| , we need to take care of the fact that the number does not exceed N itself. For doing this , we take a function: F(I , State , Sum) . This means that we are at the Ith digit , and State tells us that if all the digits taken before are equal to corresponding digits of N (if true would imply that Ith digit has to be <= Ith digit of N) , Sum stores the sum of digits taken in the first K digits.We will increase sum if Ith digit is in first K digits , decrease if it is in the last K digits , and make no change if it is between .This way if we get 0 sum in the end , that means our number satisfies all conditions.

F(I,state,sum){

//Base Case

if(N == -1) return (Sum == 0);

//If Ith is first digit , we can't take 0 .

//If state variable is true , that means we are in prefix of N , so Ith digit has to be less than Dig[I]

for(J = ((I == |D| - 1) ? 1 : 0) ;J <= (state ? Dig[I] : 9); J++){

//digit is in first K digits.

if(I >= |D| - K)

r += F(I - 1, state && (Dig[I] == J) , sum + J);

//digit is in last K digits.

else if(n < K){

if(sum >= i)

ans += F(I - 1, state && (Dig[I] == J), sum - J);

}

//digit is in between

else ans += F(I - 1, state && (Dig[I] == J), sum);

}

return ans;

}

Returning to Count(p, x) We only have the pth digit as x, but depending on starting |D|-p digits , we can have different values of (state,sum) at pth digit, so we will try all such values of (state,sum), and count the total numbers.

Count(p, x){

for(State = 0 ; State <= 1; State ++){

for(Sum = 0; Sum <= (|D| - p)\*9 ;Sum++){

ans += [ Numbers With (|D|-p)\_Digits And Reach (State,Sum) ] \* [ F(p,state,sum) and x as the pth digit] ;

}

}

}

Finding the count of numbers with |D|-P digits and reaching (state,sum) as required can be done either using another recursive function , or using function F itself . This completes our Sum() function . Find(x) function is easy to implement as well if you have understood the basic idea developed above( would be good exercise infact) . It can be also be eliminated, but at the cost of one more variable in the state of function F.

**Problem Setter’s Code: C++**

[Setter’s code](https://gist.github.com/dheerajrav/5757729)

**Problem Tester’s Code: C++**

[Tester’s code](https://www.hackerrank.com/rest/contests/may13/challenges/k-blance/hackers/piyush006/download_solution)

[Cut-Tree](https://www.hackerrank.com/contests/may13/challenges/cutTree)

Given a tree T with n nodes, how many subtrees (T’) of T have at most K edges connected to (T - T’)?

**Difficulty level**

Medium

**Problem Setter**

[Wanbo](https://www.hackerrank.com/wanbo)

Editorial by [Piyush](https://www.hackerrank.com/piyush006)

**Required Knowledge**

Recursion , Memoization

**Approach**

Let us consider the brute force approach . In this we would simply need to enumerate all subsets of vertices and check if they form a valid tree and also satisfy the other conditions in the problem statement. But , for N = 50 , we will have to check 2^(50 C 2) vertices which is highly inefficient. Lets consider enumerating subsets which are valid trees in a recursive fashion.

Let F(u,K) denote the function which finds the number of subtrees **rooted** at node number u , and with **exactly** K edges connected with the remaining graph. Let S denote the set of children of u. Now we have an option of choosing some children, and ignoring some of them in the subtree we want to form. Suppose we take only L( <= |S|) of them in the subtree. Then , it means that we have (|S| - L) edges to the remaining graph from u itself .Now we have to iterate over all combinations of L children and ask each of them to have exactly Ri edges to the remaining graph , where Sum (Ri) should be equal to K-L . Simply going through all such ways would again become inefficient. So we need to define another recursive function, which would have to take only L children among |S| , and use up K - (|S|-L) edges to the remaining graph . Lets call this G(I, D , U) which would mean that we are at Ith children of u and are left to ignore D children, and still to have U edges with the remaining graph .

Our approach now is something like this :

F(u,K){

ans = 0 ;

for(i = 0; i <= |S| ; i++){

//Ignore some i children( taking only |S| - i )

ans += G( 0 , i , K - i ) ; //Assume children are 0 indexed

}

return ans;

}

For finding G(I,D,U) : At the Ith child , we have two options :

Either we can take Ith child or ignore it .

1. If we ignore it , answer is simply G(I + 1, D - 1, U)
2. If we take it ,then we have an option of asking this child to have exactly J edges with remaining graph (1 <=J <= U).

So this will function will be :

G(I, D, U){

//Base Case

if(I == |S|) return (D == 0) && (U == 0) ;

ans = 0;

//Ignore it

ans += G(I + 1, D - 1, U)

//Take it

for(J = 0; J <= U; J++){

//Ask this child to have exactly J edges with remaining graph .

ans += F(S[I] , J) \* G(I + 1, D , U - J)

}

return ans;

}

This is all about finding number of subtrees **rooted** at u and with K edges with the remaining graph . So lets return to the original problem . We will have to consider subtrees rooted at any vertex .So lets root the given tree at vertex 1 . Then , our answer will be Sum of F(i ,K) , where 1 <= i <= N , but for F(i, K) we will consider children of i ,leaving its parent in the tree rooted at vertex 1. Also, simply using these recursive functions will lead to an exponential time solution again, unless we[memoize](http://en.wikipedia.org/wiki/Memoization) it. So we make arrays F[][] , and G[][][] which will store results for each state so that we don’t compute it more than one time.

**Problem Setter’s Code: C++**

#include <map>

#include <set>

#include <queue>

#include <ctime>

#include <cmath>

#include <vector>

#include <bitset>

#include <cstring>

#include <cassert>

#include <numeric>

#include <sstream>

#include <iterator>

#include <iostream>

#include <algorithm>

using namespace std;

typedef long long LL;

#define MM(a, b) memset(a,b,sizeof(a))

#define P(x) cout << #x << " = " << x << endl;

#define PP(a,n) for(int o=0;o<n;o++)cout<<#a<<"["<<o<<"] = "<<a[o]<<endl;

vector<int> g[55];

vector<int> tr[55];

LL dp[55][55][55][2];

bool used[55];

int n, maxCut;

void dfs(int u) {

used[u] = 1;

for(int i = 0; i < g[u].size(); i++) {

int v = g[u][i];

if(!used[v]) {

tr[u].push\_back(v);

dfs(v);

}

}

}

LL f(int u, int nson, int cut, int choose) {

if(cut < 0) return 0;

if(nson == 0) return (cut == 0);

LL &ret = dp[u][nson][cut][choose];

if(ret != -1) return ret;

ret = 0;

if(choose == 0) {

for(int i = 0; i < nson; i++) {

int v = g[u][i];

ret += f(v, g[v].size(), cut, 0);

ret += f(v, g[v].size(), cut - 1, 1);

}

if(cut == 0) ret -= nson - 1;

} else {

int rson = g[u][nson - 1];

ret += f(u, nson - 1, cut - 1, 1);

for(int i = 0; i <= cut; i++) {

ret += f(u, nson - 1, cut - i, 1) \* f(rson, g[rson].size(), i, 1);

}

}

return ret;

}

int main() {

cin >> n >> maxCut;

for(int i = 0; i < n - 1; i++) {

int u, v;

cin >> u >> v;

--u;

--v;

g[u].push\_back(v);

g[v].push\_back(u);

}

dfs(0);

for(int i = 0; i < 55; i++) g[i] = tr[i];

MM(dp, -1);

LL ret = 0;

for(int cut = 0; cut <= maxCut; cut++) {

LL r1 = f(0, g[0].size(), cut, 0);

LL r2 = f(0, g[0].size(), cut, 1);

ret += r1 + r2;

}

cout << ret << endl;

return 0;

}

**Problem Tester’s Code: C++**

#include<cstdio>

#include<cstdlib>

#include<iostream>

#include<algorithm>

#include<vector>

#include<cstring>

#include<map>

#include<set>

#include<stack>

#include<queue>

#include<string>

#include<iterator>

#include<string>

#include<sstream>

#include<cassert>

#include<ctime>

#include<cmath>

#define MP make\_pair

#define PB push\_back

#define X first

#define Y second

#define oo 2000000000

#define MOD 1000000007

#define LL long long int

#define PII pair<int,int>

#define DEBUG 0

using namespace std;

LL f[60][60],g[60][60][60];

vector<int> adj[60];

int Prev[60];

LL func(int,int,int);

LL junc(int,int,int,int,int);

///Spend exactly K units with the root starting at u

LL func(int u,int p,int k){

LL &ret = f[u][k];

if(ret != -1) return ret;

ret = 0;

for(int i = 0 ; i <= min(k , (int)(adj[u].size() - (p != -1))) ; i++){

///Root spends i units, rest spending by children

ret += junc(0 , u, k - i, i, p);

}

return ret ;

}

///Drop exactly r children and distribute exactly k units among the children

LL junc(int idx ,int p , int k, int r,int actp){

if(idx == adj[p].size()){

return ((k == 0) && (r==0));

}

if(adj[p][idx] == actp){

return junc(idx + 1, p, k , r, actp);

}

LL &ret = g[adj[p][idx]][k][r];

if(ret != -1){

return ret;

}

ret = 0;

///Drop this node

if(r) ret += junc(idx + 1,p , k, r - 1, actp);

///Take it

for(int i = 0;i <= k; i++){

///Allow this child to use exactly i units

ret += func(adj[p][idx], p , i) \* junc(idx + 1, p , k - i, r, actp);

}

return ret;

}

void dfs(int u,int p){

Prev[u]=p;

for(int i=0;i<adj[u].size();i++){

if(p != adj[u][i]) dfs(adj[u][i],u);

}

}

int main(){

int n,k,u,v;

scanf("%d%d",&n,&k);

for(int i=1;i<n;i++){

scanf("%d%d",&u,&v);

--u,--v;

adj[u].PB(v);

adj[v].PB(u);

}

LL ans = 0;

memset(f,-1,sizeof(f));

memset(g,-1,sizeof(g));

dfs(0,-1);

for(int i=0;i<n;i++){

for(int j=0;j<=k;j++){

if(Prev[i] != -1 && j == k) continue;

LL t=func(i,Prev[i],j);

ans += t;

}

}

printf("%lld\n",ans+1);

return 0;

}

[N Puzzle](https://www.hackerrank.com/contests/may13/challenges/n-puzzle)

N Puzzle is a sliding blocks game that takes place on a k \* k grid with k \* k - 1 tiles each numbered from 1 to N. Your task is to reposition the tiles in their proper order

**Difficulty Level**

Medium ( Implementation heavy )

**Problem Setter**

[Dheeraj M R](https://www.hackerrank.com/dheeraj)

**Required Knowledge**

A\* algorithm

**Approach**

N-Puzzle is a classic 1 player game that teaches the basics of heuristics in arriving at solutions in Artificial Intelligence.

Let us consider for the sake of simplicity, N = 8

The final configuration of an 8 - puzzle will look like

0 1 2

3 4 5

6 7 8

[A\*](https://www.hackerrank.com/challenges/pacman-astar) algorithm talks about a cost function and a heuristic.

cost = heuristic\_cost + #of steps taken to reach the current state.

the tiles are numbered. For a given unsolved 8 - puzzle configuration say

0 3 8

4 1 7

2 6 5

We use 2 heuristics for every state and at every iteration

1 such heuristic is

h1 => The number of misplaced tiles.

In the above example, the number of misplaced tiles are 8

2nd heuristic we use is

h2 => The manahattan distance between a tile’s original position to its current position.

In the above example, the manhattan distance for all tiles are as follows

ab => where a is the manhattan distance of tile b.

h2 = 23 + 28 + 14 + 11 + 27 + 42 + 16 + 15

h2 = 14

Given 2 heuristics, it is always advised to use the maximum as the maximum never underestimates the cost to reach the required end state ( initial configuration of 8 Puzzle )

If it took 10 moves to reach the current state as shown above, then the cost function of a given state of N - puzzle

is given as

cost = max( h1, h2 ) + 10

cost = max( 14, 8 ) + 10 = 24

We can then implement a priority queue and record each of possible moves from a given initial state of the board and then pop the state with the lowest cost function. This procedure is guaranteed to give us a solution.

**Problem Setter’s code: C++**

[here](https://gist.github.com/dheerajrav/5748717)

[Equal](https://www.hackerrank.com/contests/may13/challenges/equal)

Christy has erred by distributing chocolates unevenly among her coworkers and now has to correct her mistake by equalizing chocolates of coworkers using minimum number of operations.

**Difficulty level**

Easy

**Problem Setter**

[Amit Pandey](https://www.hackerrank.com/amitiitkgp)

**Required Knowledge**

Greedy Algorithm and Coin change problem.

**Approach**

Christy has to equalize the number of chocolates for all the coworkers. The only action she can make at every operation is to increase the count of every others’ chocolate by 1,2 or 5 except one of them. This is equivalent to saying, christy can take away the chocolates of one coworker by 1, 2 or 5 while keeping others’ chocolate untouched.

Let’s consider decreasing a coworker’s chocolate as an operation. To minimize the number of operations, we should try to make the number of chocolates of every coworker equal to the minimum one in the group(min). We have to decrease the number of chocolates the ith person A[i] by (A[i] - min). Let this value be x. For this you may consider [Coin change algorithms](http://en.wikipedia.org/wiki/Change-making_problem).

we now follow a greedy algorithm so number of operations required is minimum. This can be done in k operations.

k = x/5 +(x%5)/2 + (x%5)%2

Let f(min) be sum of operations performed over all coworkers to reduce each of their chocolates to min.

However, sometimes f(min) might not always give the correct answer. It can also be a case when

f(min) > f(min-1)

but it is safe to assume that

f(min) < f(min-5)

as f(min-5) takes N operations more than f(min) where N is the number of coworkers.

Therefore, if

A = {min,min-1,min-2,min-3,min-4}

then,

f(A) <= f(min) < f(min-5)

Compute f(y) ∀ y ∈ A and print the minimum as the answer.

**Time Complexity**

O(N)

where N : Number of coworkers

**Problems Setter’s Code: C++**

#include<iostream>

#include<cstdio>

#define INT\_MAX 2000000000

using namespace std;

long long int functn (long long int temp) // similar to greedy Coin-change

{

long long int x=0;

if(temp>=5)

{

x = temp/5;

temp = temp%5;

}

if(temp>=2)

{

x += temp/2;

temp = temp%2;

}

x += temp;

return x;

}

int array\_smallest(long long int array[],int array\_length)

{

long long int smallest = INT\_MAX;

long long int i;

for (i = 0; i < array\_length; i++)

{

if (array[i] < smallest) {

smallest = array[i];

}

}

return smallest;

}

long long int mod(long long int x)

{

if(x>0)

return x;

else

return (-1)\*x;

}

int main()

{

long long int T,N,i,j,min,sum,temp;

cin>>T;

while(T--)

{

min = 1000000;

cin>>N;

int A[N];

for(i=0 ; i< N ; i++)

{

cin>>A[i];

if(A[i]<min)

min = A[i];

}

long long int sum[6];

for(j=0 ; j<=5 ; j++)

{

sum[j]=0;

for(i=0 ; i< N ; i++)

{

temp = mod(A[i] - (min-j));

sum[j] = sum[j] + functn(temp);

}

}

cout<<array\_smallest(sum,6)<<endl;

}

return 0;

}

**Problem tester’s code: C++**

#include<cassert>

#include<iostream>

#include<algorithm>

using namespace std;

int T, N;

int d[100010];

int main() {

cin >> T;

assert(T <= 100);

while(T--) {

cin >> N;

assert(N <= 10000);

for(int i = 0; i < N; i++)

{cin >> d[i]; assert(d[i] < 1000);}

int M = \*min\_element(d, d + N);

int r = 1e9;

for(int t = M - 4; t <= M; t++) {

int s = 0;

for(int i = 0; i < N; i++) {

int D = d[i] - t;

s += D / 5, D %= 5;

s += D / 2, D %= 2;

s += D;

}

r = min(r, s);

}

cout << r << endl;

}

return 0;

}

class Customer{

public:

int number\_of\_companies;

int ar[101];

Company(int x, int \*a){

number\_of\_companies = x;

for( int i = 0;i < x;i++ ){

ar[i] = a[i];

}

}

}

[Restaurant](https://www.hackerrank.com/contests/may13/challenges/restaurant)

Cut a bread of size l \* b into smaller identical pieces such that each piece is a square of the same dimension and having maximum possible side length with no left over piece of bread.

**Difficulty level**

Very Easy

**Problem Setter**

[Amit Pandey](https://www.hackerrank.com/amitiitkgp)

**Required Knowledge**

Properties of GCD(Greatest Common Divisor)

**Approach**

In the given problem, you have to cut a rectangular bread having size l \* b into squares of equal dimension such that no piece of original bread is left over. So we have to make cuts only vertically or horizontally. Hence we can conclude that if the length of a side of the square is a, both l and b both has to be divisible by a.

Now, there’s another constraint. a has to be as large as possible.

The problem reduces to finding an integer whose value is maximum as well as divisible by both l & b which is equivalent to finding the greatest common divisor (gcd) of l & b.

For more information refer [Greatest Common Divisor](http://en.wikipedia.org/wiki/Greatest_common_divisor)

So total number of square of maximum size will be

(l/gcd(l,b))\*(b/gcd(l,b))

**Time complexity**

T\*(log(l)+log(b))

Where

T:number of test cases,   
l : length of rectangle,  
b: width of rectangle.

**Problem setter’s code: java**

import java.util.Scanner;

class Solution {

public static int gcd ( int a, int b)

{

if(a==0)

return b;

else if(b==0)

return a;

else

return gcd(b,a%b);

}

public static void main(String args[])

{

Scanner in = new Scanner(System.in);

int num = in.nextInt();

int a,b;

while(num-- > 0)

{

a = in.nextInt();

b = in.nextInt();

System.out.println((a\*b)/(gcd(a,b)\*gcd(a,b)));

}

}

}

**Tester’s code: python**

import sys

def gcd(a,b):

if a==0 :

return b

if b==0 :

return a

else :

return gcd(b,a%b)

t = int(raw\_input())

i = 0

for i in range(0,t):

a,b = map(int,sys.stdin.readline().split())

print (a\*b)/(gcd(a,b)\*gcd(a,b))