#include<bits/stdc++.h>

using namespace std;

// tabular form (for best approach look below this code)

int tabular(int e, int f){

vector<vector<int> > dp(e+1,vector<int> (f+1));

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

dp[i][0] = 0;

dp[i][1] = 1;

}

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

dp[0][j] = 0;

dp[1][j] = j;

}

for(int i=2;i<=e;i++){

for(int j=2;j<=f;j++){

int mn = INT\_MAX;

for(int mj=j-1,pj=0;mj>=0 && pj<=j; mj--,pj++){

int v1 = dp[i][mj]; //egg survives

int v2 = dp[i-1][pj]; // egg break

int val = max(v1,v2);

mn = min(mn,val);

}

dp[i][j] = 1 + mn;

}

}

return dp[e][f];

}

// best approach(binarySearch + recursion + dp)

int solve\_optimised(int e, int f, vector<vector<int> > &dp){

if(f==0 || f==1 || e==1 || e==0){

return dp[e][f] = f;

}

if(dp[e][f]!=-1) return dp[e][f];

int mn = INT\_MAX;

int l = 1;

int r = f;

while(l<=r){

int mid = (l+r)/2;

int right = solve\_optimised(e,f-mid,dp);

int left = solve\_optimised(e-1,mid-1,dp);

int temp\_ans = 1+ max(left,right);

if(left<right){

l = mid+1;

}

else{

r = mid-1;

}

mn = min(mn,temp\_ans);

}

return dp[e][f] = mn;

}

// recursion with dp

int solve\_optimised(int e, int f, vector<vector<int> > dp){

if(f==0 || f==1 || e==1 || e==0){

return dp[e][f] = f;

}

if(dp[e][f]!=-1) return dp[e][f];

int mn = INT\_MAX;

for(int k=1;k<=f;k++){

int temp\_ans = 1 + max((dp[e][f-k]!=-1)?dp[e][f-k] : solve\_optimised(e,f-k,dp),

(dp[e-1][k-1]!=-1)?dp[e-1][k-1] : solve\_optimised(e-1,k-1,dp));

mn = min(mn, temp\_ans);

}

return dp[e][f] = mn;

}

int solve(int e, int f, vector<vector<int> > dp){

if(f==0 || f==1 || e==1 || e==0){

return dp[e][f] = f;

}

if(dp[e][f]!=-1) return dp[e][f];

int mn = INT\_MAX;

for(int k=1;k<=f;k++){

int temp\_ans = 1 + max(solve(e,f-k,dp), solve(e-1,k-1,dp));

mn = min(mn, temp\_ans);

}

return dp[e][f] = mn;

}

int main(){

int e,f;

cin>>e>>f;

vector<vector<int> > dp(e+1, vector<int> (f+1,-1));

cout<<solve\_optimised(e,f,dp);

return 0;

}