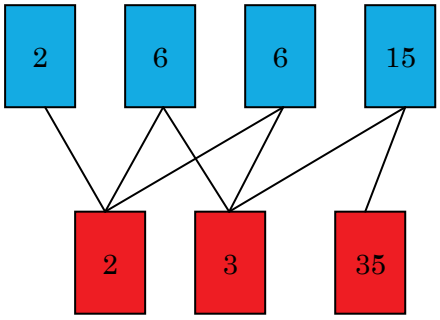
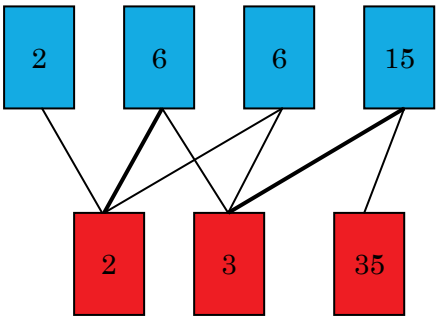
[Problem E] カードゲーム

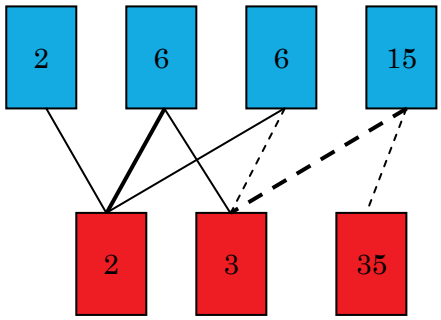
- 青いカードと赤いカードの最大公約数が２以上であればそれらのカードをペアにすることができる。
- ペアに出来るカード同士を線でつなぐと２部グラフとなる。



- この問題は２部グラフの最大マッチング問題となる。
- あるマッチング M（ペアが組める枝の集合）が最大マッチングとなるための必要十分条件は、増大路が存在しないことである。
- 増大路とは、ペアが組めていない節点からはじまり、M 以外の枝、M の枝を交互に通ってペアが組めていない節点にいたる経路である。

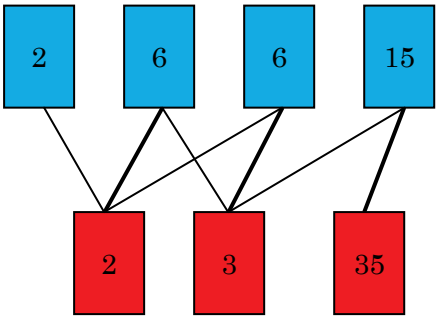


マッチング M（太線）



増大路の例（破線）

- 増分路の枝に対し、元の M の枝を M から取り除き、M 以外の枝を M に追加することにより新たなマッチング M’が得られ、M’の枝数は M の枝数より 1 増加する。



マッチング M’（太線）

- 増大路を見つけるのは、深さ優先探索で行える。
- 増大路が存在しなければ、その時点のマッチングが最大マッチングである。

プログラム例

```
/* ACM-ICPC2009 国内予選 Problem E */
// http://www.waseda.jp/assoc-icpc2009/preliminary/contest/all_ja.html#section_E
// filename = pe.c
// コンパイル： cc -O2 pe.c
// 実行方法：   ./a.out < E0 > E0.result 等
// 確認方法：   diff E0.ans E0.result 等
#include <stdio.h>
#include <stdlib.h>

int m,n;        // 1 <= m <= 500, 1 <= n <= 500
int b[501],r[501];
int bp[501], rp[501];
struct edge_t {
  int blue;
  int red;
  int paired;
} edge[501*501];
int ne;         // number of edges
struct edge_list_t {
  int en;       // edge number
  struct edge_list_t *next;
} *bel[501], *rel[501], *inc_path;
int np;         // number of pairs

void greedy()
{
  int i,j;
  struct edge_list_t *elp;

  for(i=1; i<=m; i++){
    if(bp[i] == 0){
      elp = bel[i];
      while(elp){
        if(rp[edge[elp->en].red] == 0){
          bp[i] = 1;
          edge[elp->en].paired = 1;
          rp[edge[elp->en].red] = 1;
          np++;
          break;
        }
        elp = elp->next;
      }
    }
  }
}

int dfs(int node, int mode)
{
  struct edge_list_t *elp, *elnew;
  int eno;
  if(mode == 0){
    elp = bel[node];
    while(elp){
```

```
      eno = elp->en;
      if(edge[eno].paired == 0){
        if(rp[edge[eno].red] != 2){
          if(rp[edge[eno].red] == 0){
            elnew = (struct edge_list_t *)malloc(sizeof(struct edge_list_t));
            elnew->en = eno;
            elnew->next = inc_path;
            inc_path = elnew;
            return 1;
          }
          rp[edge[eno].red] = 2;
          if(dfs(edge[eno].red, 1)){
            elnew = (struct edge_list_t *)malloc(sizeof(struct edge_list_t));
            elnew->en = eno;
            elnew->next = inc_path;
            inc_path = elnew;
            //      rp[edge[eno].red] = 1;
            return 1;
          }
          //      rp[edge[eno].red] = 1;
        }
      }
      elp = elp->next;
    }
    return 0;
  }
  else {
    elp = rel[node];
    while(elp){
      eno = elp->en;
      if(edge[eno].paired == 1){
        if(dfs(edge[eno].blue, 0)){
          elnew = (struct edge_list_t *)malloc(sizeof(struct edge_list_t));
          elnew->en = eno;
          elnew->next = inc_path;
          inc_path = elnew;
          return 1;
        }
      }
      elp = elp->next;
    }
    return 0;
  }
}

int solve()
{
  int i, j, flag;
  struct edge_list_t *elp, *elptmp;
  greedy();
  if(np==0 || np==m || np==n)
    return np;

  flag = 1;
```

```
  while(flag){
    if(np == m || np == n) return np;
    flag = 0;
    for(i=1; i<=m; i++){
      if(bp[i]) continue;
      for(j=1; j<=n; j++)
        if(rp[j] == 2) rp[j]=1;
      inc_path = NULL;
      flag = dfs(i,0);
      if(flag) break;
    }
    if(flag){
      elp = inc_path;
      while(elp){
        if(edge[elp->en].paired)
          edge[elp->en].paired = 0;
        else
          edge[elp->en].paired = 1;
        bp[edge[elp->en].blue] = 1;
        rp[edge[elp->en].red] = 1;
        elptmp = elp;
        elp = elp->next;
        free(elptmp);
      }
      np++;
    }
  }
  return 0;
}

int gcd(int a, int b)
{
  int tmp,r;
  if(a<b) {
    tmp = a;
    a = b;
    b = tmp;
  }
  // now, a >= b
  r = a%b;
  while(r){
    a = b;
    b = r;
    r = a%b;
  }
  return b;
}

int main()
{
  int i,j;
  int bn, rn;
  struct edge_list_t *ecp, *ecptmp;
  while(1){
```

```
scanf("%d %d", &m, &n);
if(m == 0 && n == 0) break;
for(i=1; i<=m; i++) scanf("%d", &b[i]);
for(i=1; i<=n; i++) scanf("%d", &r[i]);
for(i=0; i<500*500; i++)
  edge[i].blue = edge[i].red = edge[i].paired = 0;
ne = 0;
for(i=1; i<=m; i++){
  bp[i] = 0;
  bel[i] = NULL;
}
for(j=1; j<=n; j++){
  rp[j] = 0;
  rel[j] = NULL;
}
for(i=1; i<=m; i++)
  for(j=1; j<=n; j++)
    if(gcd(b[i],r[j]) > 1){
      edge[ne].blue = i;
      edge[ne++].red = j;
    }
for(i=0; i<ne; i++){
  bn = edge[i].blue;
  rn = edge[i].red;
  ecp = (struct edge_list_t *)malloc(sizeof(struct edge_list_t));
  ecp->en = i;
  ecp->next = bel[bn];
  bel[bn] = ecp;
  ecp = (struct edge_list_t *)malloc(sizeof(struct edge_list_t));
  ecp->en = i;
  ecp->next = rel[rn];
  rel[rn] = ecp;
}
np = 0;

solve();
printf("%d¥n", np);

for(i=1; i<=m; i++){
  ecp = bel[i];
  while(ecp){
    ecptmp = ecp;
    ecp = ecp->next;
    free(ecptmp);
  }
}
for(i=1; i<=n; i++){
  ecp = rel[i];
  while(ecp){
    ecptmp = ecp;
    ecp = ecp->next;
    free(ecptmp);
  }
}
```

```
    }
}
```