ОЛІМПІАДА З ІНФОРМАТИКИ У МІСТІ КИЄВІ У 2012–2013 НАВЧАЛЬНОМУ РОЦІ

Продовження, початок у №8 за 2013 рік, №1, №2 за 2014 рік

Знов'як Юрій Володимирович,
інженер з програмного забезпечення Google New York Center,
Мисак Данило Петрович,
керівник гуртка СШ №52 м. Києва,
Рибак Олександр Владиславович,
аспірант Інституту математики НАН України,
Рудик Олександр Борисович,
кандидат фізико-математичних наук, доцент кафедри методики природничо-математичної освіти і технологій Інституту післядипломної педагогічної освіти Київського університету імені Бориса Грінченка.

2. Фарбування

```
{ Free Pascal }
{$n+} {Верхні межі кількостей:}
const ng_=22; {граней}
    ns_=120; {симетрій многогранника}
    nb_=61000000;
    full: set of byte=[0..255];
    simple=true;  {Без т. Редфілда-Пойа}
label NEXTi,NEXTh,RES; {Мітки для перебору}
            {Кількості:}
var ng,   {граней}
  ns,  {симетрій многогранника}
   m,  {кольорів}
   i,j,k,l: byte;  {Допоміжні лічильники}
       o: text;
                {Суміжні грані}
sg: array[0..ng_] of set of 1..ng_;
 b: array[0..nb_] of set of byte;
 {Група підстановок-симетрій многогранника}
s: array[1..ns_,1..ng_] of 1..ng_;
c,            {Номери кольорів граней
 або многочлен - ряд переліку конфігурацій}
h,                  {Номери гіпотез}
v: array[0..ng_] of byte;    {Перестановки}
fault: boolean;       {Хибність гіпотези}
ib,nb,imng,kmng,mng: longint;
nout: extended;
   {Перестановка елементів v при переборі}
procedure SWAP; BEGIN
      v[0]:=v[h[i]];
         v[h[i]]:=v[ng-i+1];
              v[ng-i+1]:=v[0] END;

  {Перехід  до наступного числа  у системі
  числення з основою m, нумерація розрядів
              - від наймолодшого}
procedure next;  var i: byte;       BEGIN
      inc(c[1]);  i:=1;
      while c[i]>=m do begin
           c[i]:=c[i]-m;
           inc(i);
           inc(c[i]) end      END;
    {Число, записане у системі числення
 з основою m номерами кольорів граней після
 застосування симетрії  j  до розфарбування
         з номерами кольорів граней c}
function number(k: byte): longint;
       var v: longint;  l: byte; BEGIN
            v:=c[s[k,ng]];
for l:=ng-1 downto 1 do v:=v*m+c[s[k,l]];
           number:=v  END;
        BEGIN
       {Зчитування і опрацювання даних}
for j:=1 to ng_ do sg[j]:=[];
assign(o,'farben.in');
reset (o);
readln(o,m);
ng:=0; while not seekeof (o) do begin
ng:=ng+1; while not seekeoln(o) do begin
      read(o,j);  include(sg[ng],j) end;
      readln(o) end;
close  (o);
assign (o,'farben.out');
rewrite(o);
 {Знаходження групи симетрій многогранника}
ns:=0;
 i:=0;
for j:=1 to ng do v[j]:=j;
NEXTi: inc(i); h[i]:=0;
NEXTh: inc(h[i]);
    if h[i]>ng-i+1 then begin
      dec(i);
      if i=0 then goto RES;
      SWAP;
      goto NEXTh end;
    if i>1 then begin
      j:=0;
      repeat inc(j);
       fault:= (j in sg[i]) and
       not (v[ng-j+1] in sg[v[h[i]]]) or
          not (j in sg[i]) and (v[ng-j+1] in sg[v[h[i]]]);
      until (j=i-1) or fault;
      if fault then goto NEXTh end;

    SWAP;  if i<ng then goto NEXTi;
    inc(ns);
    for j:=1 to ng do s[ns,j]:=v[ng-j+1];
    SWAP;  goto NEXTh;

   {Перебір усіх способів розфарбування}
RES: if simple then  begin
 {без використання теореми Редфілда – Пойя}
if (ng*ln(m)>ln(256.*(nb_+1))) and
  (ng*ln(m)>ln(2147483647))  then halt;
mng:=m;  for j:=2 to ng do mng:=mng*m;
 nb:=mng div 256;
for ib:=0 to nb do b[ib]:=full;
for i :=1 to ng do c[i]:=0;
exclude(b[0],0);
nout:=1; imng:=0;
while imng < mng-1 do begin
 imng:=imng+1;
```

```
 next;
 if (imng mod 256) in b[imng div 256] then begin
 nout:=nout+1;
 for k:=1 to ns do begin
  kmng:=number(k);
  exclude(b[kmng div 256], kmng mod 256)
 end end end end
else begin
  {з використанням теореми Редфілда – Пойя}
for j:=1 to ng do c[j]:=0;
for k:=1 to ns do begin b[0]:=[1..ng]; i:=0;
for j:=1 to ng do if j in b[0] then begin l:=j;
repeat exclude(b[0],l); l:=s[k,l]; until not (l in b[0]);
inc(i) end;
inc(c[i]) end;
nout:=m*c[ng];
for i:=ng-1 downto 1 do nout:=(nout+c[i])*m;
                nout:=nout/ns end;
writeln(o,nout:0:0); close(o);
END.
```

3. Істотні інверсії

```
{ Free Pascal }
PROGRAM Inverses;
var n:longint; {Кількість елементів у
    послідовності з вводу.}
  x:array[1..50000] of longint;
   {Послідовність, для якої потрібно
    знайти кількість істотних інверсій.}
  x2:array[1..32769] of longint;
   {Запасний масив для сортування.}
  t:longint; {Величина, яку має перевищувати
    різниця x[j]-x[k], щоб пара (j,k) вважалася
    істотною інверсією за умови j<k.}
  d,i,j,k,p,q:longint; {Індекси для циклів.}
  inv:longint; {Кількість t-істотних інверсій.}
BEGIN
 {Відкриття файлів для вводу.}
 assign(input,'inverses.in'); reset(input);
 assign(output,'inverses.out'); rewrite(output);
 {Читання вхідних даних.}
 read(n,t);
 for i:=1 to n do read(x[i]);
 inv:=0; {Спочатку кількість знайдених
  інверсій дорівнює 0.}
 {Тепер відбуватиметься сортування
  послідовності методом злиття з
  підрахунком істотних інверсій.}
 d:=1; {Змінна d означає довжину відрізків, що
  зливаються. Спочатку ця довжина дорівнює 1.}
 while (d<n) do begin
  {Цикл виконується, поки довжина відрізків, які мають
   об'єднуватися, менша за кількість елементів.}
  i:=0;   while (i+d<n) do begin
   {Перший з двох об'єднуваних відрізків
    копіюється до запасного масиву.}
   for j:=1 to d do x2[j]:=x[i+j]; p:=i+d*2;
   if (p>n) then p:=n;
   q:=1; {У першому відрізку індексом q
    позначатиметься найлівіший елемент,
    який перевищує x[k] більше, ніж на t.
    До початку пошуку згаданого
    елемента значення q дорівнює 1.
    Далі воно лише зростатиме.}
   j:=1; k:=i+d+1;
   {Об'єднуємо два відсортованих відрізка,
    поки один з них не закінчиться.}
   while ((j<=d)and(k<=p)) do begin
     if (x2[j]<=x[k]) then begin
     x[k-d-1+j]:=x2[j]; Inc(j);
    end;
     if (j<=d) then
      if (x2[j]>x[k]) then begin
       {У першому відрізку шукатиметься номер
        найлівішого елемента, який більше, ніж на t.
        Якщо такого елемента немає, q стане рівним d+1.}
       while ((q<=d)and(x2[q]<=x[k]+t)) do
        Inc(q); inv:=inv+d-q+1;
        {Кількість знайдених інверсій збільшується
         на кількість елементів першого відрізка,
         що перевищують x[k] більше, ніж на t.}
       x[k-d-1+j]:=x[k]; Inc(k);
      end;
   end;
   if (j<=d) then
    for j:=j to d do x[p-d+j]:=x2[j];
   i:=i+d*2; {Перехід до наступних двох відрізків.}
  end;
  d:=d*2; {Подвоєння довжини відрізків.}
 end;
 writeln(inv); {Вивід відповіді.}
 {Закриття файлів вводу та виводу.}
 close(input); close(output);
END.
```

4. Фортеця

```
{ Free Pascal }
PROGRAM Fortress;
var n:word; {Кількість відмічених точок.}
  x,y:array[1..100] of integer;
   {Координати відмічених точок. На
    початку роботи програми ці точки
    сортуються лексикографічно, тобто за
    зростанням x-координати, а при рівних
    x-координатах - за зростанням y-координати.}
  sort:array[3..100,1..99] of byte;
   {Для кожного i від 3 до n числа
    sort[i,1], …, sort[i,i-1]
    складають послідовність, яка є
    перестановкою індексів 1, …, i-1.
    Вказані індекси впорядковуються
    наступним чином. Через точку
    (x[i],y[i]) проводиться промінь,
    направлений вертикально вниз. Потім
    промінь починає обертатися за
    годинниковою стрілкою, а індекси
    1, …, i-1 впорядковуються згідно з
    часом попадання точок
    (x[1],y[1]), …, (x[i-1],y[i-1]) на
    цей промінь. Порядок точок, які
    попадають на промінь одночасно, для
    нас не важливий.}
  f:array[3..100,1..100] of byte;
   {f[j,k] - це найбільша можлива
    кількість відмічених точок на опуклій
    ламаній, останньою точкою якої є
    (x[j],y[j]), а передостанньою з
    відмічених точок - (x[k],y[k]).
    Ламана має починатися в точці
    (x[i],y[i]), де i - індекс, який
    пробігає значення від 1 до n-2. Для
    кожного i всі f[j,k] обчислюються
    окремо. При підрахунку f[j,k] точка
    (x[j],y[j]) не враховується. Ми
    обчислюємо значення f[j,k] лише для
    j>k, а елементи f[j,j] - це
    максимуми з чисел f[j,1], …,
    f[j,j-1]. Спочатку розраховуються
    значення f[j,k] для ламаних, опуклих
    вгору, а потім — для опуклих вниз.
    При обчисленні масиву f розглядаються
    ламані, відмінні від відрізка.}
```

```
  seg:array[2..100] of byte;
   {seg[j] — це кількість відмічених
    точок на відрізку з кінцями
    (x[i],y[i]) та (x[j],y[j]), не
    враховуючи точки (x[j],y[j]). Тут i —
    індекс, який пробігає значення від 1
    до n-2. Для кожного i масив seg
    обчислюється окремо.}
   maxpoly:word; {Максимально можлива
    кількість веж на межі невиродженого
    опуклого многокутника. Вежі можна
    будувати лише у відмічених точках.}
   i,j,k,m,p,q:word; {Індекси для циклів.}
   vx1,vy1,vx2,vy2:longint;
   {Координати векторів з початками та
    кінцями у відмічених точках. За
    допомогою псевдоскалярного добутку
    векторів (vx1,vy1) та (vx2,vy2)
    з'ясовується, чи йдуть два промені за
    годинниковою стрілкою або чи лежать
    три точки на одній прямій.}
BEGIN
 {Відкриття файлів для вводу та виводу.}
 assign(input,'fortress.in'); reset(input);
 assign(output,'fortress.out'); rewrite(output);
 {Читання вхідних даних.}
 read(n);
 for i:=1 to n do read(x[i],y[i]);
 maxpoly:=0; {Для початкового варіанту
  відповіді встановлюється значення 0.
  Якщо існує многокутник, що задовольняє
  умову задачі, maxpoly набуде значення,
  рівне кількості відмічених точок на
  межі цього многокутника. Потім зміна
  значення maxpoly відбуватиметься
  кожного разу, коли з'являтиметься
  варіант відповіді з більшою кількістю
  відмічених точок на межі. Якщо
  потрібного многокутника не існує,
  maxpoly залишиться рівним 0, як і
  вимагає постановка задачі.}
 if (n>=3) then begin
  {Для n<3 відповідь залишається рівною 0,
   бо потрібного многокутника точно немає.
   Для випадку n>=3 відбуватиметься перевірка,
   чи існує многокутник, що задовольняє умову
   задачі. Якщо він існує, буде визначено, яка
   найбільша кількість відмічених точок може
   бути на межі такого многокутника.}
  for i:=1 to n-1 do begin
   {Відмічені точки сортуються в
    лексикографічному порядку.}
   m:=i;
   for j:=i+1 to n do
    if ((x[j]<x[m])or((x[j]=x[m])and(y[j]<y[m])))
    then m:=j; vx1:=x[i]; x[i]:=x[m];
    x[m]:=vx1; vy1:=y[i]; y[i]:=y[m]; y[m]:=vy1;
  end;
  for i:=3 to n do begin
   {Для кожного i від 3 до n будуються послідовності
    sort[i,1], ..., sort[i,i-1]. Зміст масиву sort описаний
    у розділі var.}
   for j:=1 to i-1 do sort[i,j]:=j;
   for j:=1 to i-2 do begin
    m:=j;
    for k:=j+1 to i-1 do begin
     vx1:=x[sort[i,m]]-x[i]; vy1:=y[sort[i,m]]-y[i];
     vx2:=x[sort[i,k]]-x[i]; vy2:=y[sort[i,k]]-y[i];
     if (vx1*vy2-vy1*vx2>0) then m:=k;
    end;
    p:=sort[i,j]; sort[i,j]:=sort[i,m]; sort[i,m]:=p;
   end;
  end;
  {Для кожного i від 1 до n-2 шукається
   максимальний розв'язок, в якому
   найлівішою вершиною є (x[i],y[i]).}
  for i:=1 to n-2 do begin
   {Для даного i шукається найбільша
    можлива кількість відмічених точок,
    які лежать на опуклій угору ламаній
    з початком в (x[i],y[i]).}
   if (i>1) then
    {Якщо цикл виконується не вперше,
     з масиву sort видаляються
     елементи, рівні i-1. Точки з
     номерами менше i при виконанні
     циклу вже не розглядатимуться.}
    for j:=i+2 to n do begin
     k:=1;
     while (sort[j,k]<>i-1) do Inc(k);
     for k:=k to j-i do sort[j,k]:=sort[j,k+1];
    end;
   {Побудова масиву seg для даного i.}
   for j:=i+1 to n do begin
    seg[j]:=1; vx1:=x[j]-x[i]; vy1:=y[j]-y[i];
    for k:=i+1 to j-1 do begin
     vx2:=x[k]-x[i]; vy2:=y[k]-y[i];
     if (vx1*vy2=vy1*vx2) then Inc(seg[j]);
    end;
   end;
   {Основний крок динамічного програмування,
    на якому для певного j обчислюються значення
    f[j,i+1], ..., f[j,j-1].}
   for j:=i+2 to n do begin
    {Для всіх m, які в масиві sort[j] стоять перед i,
     то встановлюється f[j,m]=0, бо відповідної опуклої
     ламаної немає або вона є відрізком.}
    k:=1;
    while (sort[j,k]<>i) do begin
     f[j,sort[j,k]]:=0; Inc(k);
    end;
    vx1:=x[i]-x[j]; vy1:=y[i]-y[j];
    vx2:=vx1; vy2:=vy1;
    {Якщо деякі точки (x[m],y[m]) лежать на відрізку
     з кінцями (x[i],y[i]) та (x[j],y[j]), то для таких m
     встановлюється f[j,m]=0, бо відповідна опукла
     ламана є відрізком.}
    while ((k<=j-i) and (vx1*vy2-vy1*vx2=0)) do
    begin
     f[j,sort[j,k]]:=0; Inc(k);
     if (k<=j-i) then begin
      vx2:=x[sort[j,k]]-x[j]; y2:=y[sort[j,k]]-y[j];
     end;
    end;
    for k:=k to j-i do begin
     {Для тих m, для яких (x[m],y[m])
      лежить вище відрізка з кінцями
      (x[i],y[i]) та (x[j],y[j]),
      значення f[j,m] визначається на
      основі значень f[p,q] для p<j.}
     m:=sort[j,k]; {Індекс sort[j,k]
      для зручності позначається через m.}
     f[j,m]:=seg[m]+1; {Змінна f[j,m]
      отримує значення, яке
      відповідає ламаній з вершинами
      (x[i],y[i]), (x[m],y[m]) та
      (x[j],y[j]). У цьому виразі не
      враховані проміжні точки
      відрізка з кінцями (x[m],y[m])
      та (x[j],y[j]). Але якщо такі
      точки дають максимальну
      відповідь, вони будуть враховані
```

```
    в інших елементах масиву f.}
   if (m>i+1) then begin
     vx1:=x[j]-x[m]; vy1:=y[j]-y[m];
     p:=1; q:=1;
     while (q shl 1<m-i) do q:=q shl 1;
     {За допомогою двійкового пошуку знаходимо
      таке s, що точка (x[s],y[s]) утворює з
      точками (x[m],y[m]) та (x[j],y[j]) найбільший
      кут, який не більше 180 градусів.
      Вершиною кута має бути (x[m],y[m]), а сам кут
      відраховується за годинниковою стрілкою від
      променя, що починається в (x[m],y[m]) та
      проходить через (x[j],y[j]). Індекс s має
      задовольняти умову i<=s<m. У якості s
      виступатиме sort[m,p].}
     while (q<>0) do begin
      if (p+q<=m-i) then begin
        vx2:=x[sort[m,p+q]]-x[m];
        vy2:=y[sort[m,p+q]]-y[m];
        if (vx1*vy2-vy1*vx2<=0) then p:=p+q;
      end;
      q:=q shr 1;
     end;
     if (f[m,sort[m,p]]+1>f[j,m])
       then f[j,m]:=f[m,sort[m,p]]+1;
     {Якщо двійковий пошук дає точку, яка дозволяє
     побудувати ламану з більшою кількістю точок,
     значення f[j,m] поновлюється.}
   end;
   {Тепер елементу f[j,m] замість розв'язку для
    точок з номерами j та m присвоїться значення
    максимального серед розв'язків для точок з
    номерами j та s, де i<=s<j. Максимум обирається
    лише з тих f[j,s], для яких s у послідовності
    sort[j,1], …, sort[j,j-1] знаходиться не далі за m.
    Саме це переприсвоєння дає можливість
    застосовувати двійковий пошук.}
   if (f[j,sort[j,k-1]]>f[j,m])
     then f[j,m]:=f[j,sort[j,k-1]];
 end;
   f[j,j]:=f[j,sort[j,j-i]];
   {У елементі f[j,j] зберігатиметься
   максимальна кількість відмічених
   точок, які можуть знаходитися на
   опуклій вгору ламаній з кінцями
   (x[i],y[i]) та (x[j],y[j]).}
end;
{Аналогічний максимум шукається для ламаних,
 відмінних від відрізка та опуклих вниз.
Розв'язок для ламаної з кінцями (x[i],y[i]) та (x[j],y[j])
 знаходитиметься в елементі f[j,sort[j,1]].}
for j:=i+2 to n do begin
 k:=j-i;
 while (sort[j,k]<>i) do begin
   f[j,sort[j,k]]:=0; Dec(k);
 end;
 vx1:=x[i]-x[j]; vy1:=y[i]-y[j];
 vx2:=vx1; vy2:=vy1;
 while ((k>0)and(vx1*vy2-vy1*vx2=0)) do
 begin
   f[j,sort[j,k]]:=0; Dec(k);
   if (k>0) then begin
     vx2:=x[sort[j,k]]-x[j];
     vy2:=y[sort[j,k]]-y[j];
   end;
 end;
 for k:=k downto 1 do begin
   m:=sort[j,k]; f[j,m]:=seg[m]+1;
   if (m>i+1) then begin
     vx1:=x[j]-x[m]; vy1:=y[j]-y[m];
     p:=m-i; q:=1;
     while (q shl 1<m-i) do q:=q shl 1;
     while (q<>0) do begin
      if (p>q) then begin
        vx2:=x[sort[m,p-q]]-x[m];
        vy2:=y[sort[m,p-q]]-y[m];
        if (vx1*vy2-vy1*vx2>=0)
          then p:=p-q;
      end;
      q:=q shr 1;
     end;
     if (f[m,sort[m,p]]+1>f[j,m])
       then f[j,m]:=f[m,sort[m,p]]+1;
   end;
   if (f[j,sort[j,k+1]]>f[j,m])
     then f[j,m]:=f[j,sort[j,k+1]];
 end;
 {Перевірка, чи буде розв'язок для ламаних
  з кінцями в точках (x[i],y[i]) та (x[j],y[j])
  більшим за знайдений раніше.
  Розглядаються три варіанти:
  відрізок замість нижньої ламаної,
  відрізок замість верхньої ламаної
  та коли обидві ламані не є відрізками.}
 k:=sort[j,1];
 if ((f[j,j]+seg[j]>maxpoly)and(f[j,j]<>0)) then
   maxpoly:=f[j,j]+seg[j];
 if ((seg[j]+f[j,k]>maxpoly)and(f[j,k]<>0)) then
   maxpoly:=seg[j]+f[j,k];
 if ((f[j,j]+f[j,k]>maxpoly)and(f[j,j]<>0)and(f[j,k]<>0))
   then maxpoly:=f[j,j]+f[j,k];
 end;
end;
writeln(maxpoly); {Вивід відповіді.}
{Закриття файлів вводу та виводу.}
close(input); close(output);
END.
```

5. Вкладені множини

```
/* GCC */
#include <cstdio>
// Константи з умови задачі
const int MOD = 41;
const int MAX_N = 5;
const int MAX_M = 20;

// Звичніший синонім для 64-х бітових чисел
typedef long long int64;
// Вхідні данні
int N, M; int h[MAX_N];
// f(n, Q) з розбору задачі
int64 f[MAX_N][1 + (1 << MAX_M) / MOD];

// Так як в C/C++ нумерація масивів
// починається з 0, то f(n, Q) з розбору
// задачі відповідатиме f[n-1][Q] в
// програмі, а h_n з умови задачі
// відповідатиме h[n-1] в програмі.

int main() {
 // Зчитуємо вхідні дані
 freopen("nested.in", "r", stdin);
 scanf("%d %d", &N, &M);
 for (int i = 0; i < N; i++)
  scanf("%d", &h[i]);

 // За означенням f: f(1, Q) = 1
 // (для всіх можливих Q).
 //
 // Тут mask — двійкове представлення
 // усіх можливих множин S_1.
```

```
 for (int mask = h[0]; mask < (1 << M); mask += MOD)
  f[0][mask / MOD] = 1;

 // Використовуючи рекурентну формулу
 // з розбору задачі обчислимо f(j, Q) маючи f(j-1, Q').
 for (int j = 1; j < N; j++) {
  // Тут mask_prev — двійкове представлення Q'.
  for (int mask_prev = h[j-1]; mask_prev < (1 << M);
    mask_prev += MOD) {

   // f_prev = f(j-1, Q').
   int64 f_prev = f[j-1][mask_prev / MOD];

   // Тут mask — двійкове
   // представлення Q, підмножини Q'.
   // Використовується трюк для перебору
   // всіх підномжин заданої множини,
   // що описаний в розборі.
   for (int mask = mask_prev;
      ;
      mask = (mask - 1) & mask_prev) {
    if (mask % MOD == h[j])
     f[j][mask / MOD] += f_prev;
    if (mask == 0)
     break;
   }
  }
 }

 // Щоб отримати правильну відповідь
 // просумуємо f(N-1, Q) для всіх
 // можливих Q. Реалізація формули
 // [1] з розбору задачі.
 //
 // Тут total — відповідь на задачу
 // mask — двійкове представлення усіх
 // можливих множин S_n.
 int64 total = 0;
 for (int mask = h[N-1]; mask < (1 << M); mask += MOD)
  total += f[N-1][mask / MOD];

 // Виводимо результат
 freopen("nested.out", "w", stdout);
 printf("%lld\n", total);

 return 0;
}
```

6. Портали

```
/* GCC */
#include <stdio.h>
#include <vector>
#define maxn 1000
using namespace std;

int n, m, p[maxn], i, a, b, cur, sum, minimum, count;
vector<int> g[maxn]; // Граф:
// g[i] міститиме динамічний масив номерів
// вершин, суміжних із вершиною i
// (індексація з нуля)
bool visited[maxn]; // visited[i] матиме
// значення true тоді й тільки тоді, коли
// ми вже відвідали вершину i під час
// пошуку в глибину

// Рекурсивна функція для пошуку в глибину
// vert — номер вершини
// Повертає мінімальне з чисел на піддереві
// пошуку, починаючи з вершини vert
int dfs(int vert)
{
  visited[vert] = true;  // Позначаємо
// вершину як уже відвідану, щоб уникнути
// повторного її проходження
  int result = p[vert];  // Початкове
// значення результату, який поверне
// функція, — число у вершині, з якої
// починається пошук
  for (int i = 0; i < g[vert].size(); i++)
// Для всіх вершин, суміжних із даною
    if (! visited[g[vert][i]]) // Якщо
// суміжну вершину ще не було відвідано
      result=min(result, dfs(g[vert][i]));
// Відвідуємо суміжну вершину і оновлюємо
// найменше значення на піддереві
  return result;
}

int main()
{
  freopen("portals.in", "r", stdin);
  freopen("portals.out", "w", stdout);

  scanf("%d %d", &n, &m);
  for (i = 0; i < n; i++)
  {
    scanf("%d", &p[i]);
    visited[i] = false; // Позначаємо
// кожну вершину як іще не відвідану
  }
  for (i = 0; i < m; i++)
  {
    scanf("%d %d", &a, &b);
    g[a - 1].push_back(b - 1);
// Додаємо ребра до графа, враховуючи,
// що індексація у задачі з одиниці,
// а у програмі — з нуля
    g[b-1].push_back(a-1);
  }
  sum = count = 0; // Ініціалізуємо
// суму найменших чисел на всіх компонентах
// зв'язності і кількість компонент
// зв'язності графа
  for (i = 0; i < n; i++)
// Для кожної вершини
    if (! visited[i]) // Якщо вершина
// не належить уже розглянутим компонентам
// зв'язності
    {
      cur = dfs(i); // Здійснюємо
// пошук у глибину на компоненті
// зв'язності, до якої належить вершина i,
// та заносимо у змінну cur найменше число
// з цієї компоненти
      if (count == 0 || cur < minimum)
// Якщо це перша компонента зв'язності,
// яку ми розглядаємо, або якщо мінімальне
// число на ній менше за знайдені раніше
        minimum = cur;
// Оновлюємо значення змінної minimum,
// яка буде містити найменше з усіх чисел
      sum += cur; // Оновлюємо суму
// найменших чисел кожної компоненти
// зв'язності
      count++; // Оновлюємо
// кількість компонент зв'язності
    }

  printf("%d\n", sum + minimum * (count-2));
// Виводимо відповідь, підраховану за формулою
  return 0;
}
```