To zadanie można podzielić na etapy. Najpierw obliczamy wartość liczby, której reprezentacja binarna jest zapamiętana w ciągu, a następnie sprawdzamy, czy liczba ta jest liczbą półpierwszą.

Krok pierwszy można wykonać na kilka sposobów. Jednym z nich jest wykorzystanie schematu Hornera, co realizuje poniższa funkcja:

```
long long int bin2wart(string bin)
{
long long int w=0;
for(int i=0; i<bin.length(); i++)
   w=w*2+(bin[i]-'0');
return w;
}
```

Zauważmy, że ciągi zostały wczytane jako napisy, zatem `bin[i]` jest zmienną typu znak, pamiętającą *i*-tą cyfrę rozwinięcia binarnego. Aby użyć tej zmiennej w obliczeniach wartości liczby, musimy zamienić jej wartość (znakową) na wartość liczbową. Możemy to zrobić, odejmując od niej kod ASCII cyfry 0: `bin[i]-'0'` lub `bin[i]-48`.

Sprawdzanie, czy liczba jest półpierwsza, można także zrealizować na kilka sposobów. Pierwszy z nich korzysta z algorytmu obliczania czynników pierwszych liczby. Możemy napisać funkcję, która w pętli sprawdza kolejnych kandydatów na czynniki pierwsze: jeśli badana liczba dzieli się bez reszty przez kandydata, to zwiększamy o 1 liczbę znalezionych czynników i modyfikujemy badaną liczbę (dzielimy ją przez znaleziony czynnik). Oczywiście, gdy liczba znalezionych czynników przekroczy 2, możemy przerwać obliczenia — badana liczba nie jest liczbą półpierwszą. Jeśli obliczenia nie zostały przerwane, mamy dwie możliwości: albo badana liczba jest liczbą pierwszą (gdy liczba czynników jest równa 1), albo jest liczbą półpierwszą. Musimy sprawdzić, z którym przypadkiem mamy do czynienia. Poniżej przedstawiamy fragment kodu:

```
bool czy_polpierwsza(long long int x)
{
int ile=0;
int i=2;
while (x>1){
      if (x%i==0){
         ile++;
         x=x/i;
         if (ile>2) return false;
      }
      else i++;
}
if (ile==1) return false;
return true;
}
```

Innym sposobem jest skorzystanie wprost z definicji liczby półpierwszej: jest ona iloczynem dwóch liczb pierwszych. Dla badanej liczby sprawdzamy kolejnych kandydatów na czynniki iloczynu, zaczynając od dwójki. Jeżeli badana liczba dzieli się przez kandydata bez reszty oraz liczbami pierwszymi są zarówno kandydat, jak i wynik dzielenia badanej liczby przez kandydata, to taka liczba jest półpierwsza. Przykładem realizacji opisanej metoda jest poniższa funkcja:

```
bool czy_polpierwsza2(long long int x)
{
for (long long int j=2; j<=x; j++)
   if (x%j == 0 && czy_pierwsza(j) && czy_pierwsza(x/j))
       return true;
return false;
}
```

Zauważmy jednak, że obydwie metody mogą być bardzo czasochłonne. Przykładowo, jeśli badana liczba x jest liczbą pierwszą, to obydwie metody będą sprawdzać wszystkich kandydatów z przedziału $[2,x]$. Ponieważ x może być rzędu 2^{18}, liczba wykonywanych operacji może być nieakceptowalnie wielka.

Można to poprawić, modyfikując funkcję `czy_polpierwsza2()` tak, by nie sprawdzać kandydatów o wartości większej niż pierwiastek z x. Ponadto można usprawnić tę funkcję, likwidując niepotrzebne wywołania funkcji `czy_pierwsza()`. Zauważmy bowiem, że:

- jeśli liczba x jest liczbą pierwszą, to nie może być liczbą półpierwszą;
- jeśli liczba x jest parzysta, to wystarczy sprawdzić, czy liczba $x/2$ jest liczbą pierwszą;
- jeśli liczba x dzieli się bez reszty przez *dzielnik* nieparzysty, to wystarczy sprawdzić, czy liczba *x/dzielnik* jest liczbą pierwszą. Jest tak dlatego, że pierwszy napotkany przez algorytm dzielnik nieparzysty jest liczbą pierwszą.

Prowadzi to do następującej funkcji:

```
bool czy_polpierwsza2_v2(long long int x)
{    if (czy_pierwsza(x)) return false;
     if (x%2==0) return (czy_pierwsza(x/2));
     for (long long int j=3; j*j<=x; j+=2)
          if (x%j == 0) return (czy_pierwsza(x/j));
}
```

Funkcja `czy_pierwsza()`, która jest wywoływana powyżej, jest realizacją algorytmu sprawdzania, czy liczba jest pierwsza. Niech badaną liczbą będzie x. Jeżeli x jest równe 1, to ta liczba nie jest pierwsza. Jeżeli x dzieli się przez 2 bez reszty i jednocześnie nie jest równe 2, to badana liczba również nie jest pierwsza. Jeżeli x podzieli się przez jakąkolwiek spośród liczb: 3, 5, 7, ..., $\sqrt{n}$, to także nie będzie liczbą pierwszą. Algorytm ten realizuje poniższa funkcja:

```
bool czy_pierwsza(long long int x)
{
     if (x==1) return false;
     if (x%2 == 0 && x!=2) return false;
     for(long long int i=3; i*i<=x; i+=2)
        if (x%i == 0) return false;
     return true;
}
```

Dodatkowo należy wyznaczyć minimalną i maksymalną liczbę półpierwszą. Najpierw znajdziemy pierwszą liczbę półpierwszą w danych, przyjmując ją jako początkową wartość minimalnej i maksymalnej liczby półpierwszej:

```
long long int min, max, liczba;
int j=0;
bool wystepuje=false;
while(j<1000 && (!wystepuje)){
   liczba = bin2wart(ciagi[j]);
   if (czy_polpierwsza(liczba)){
      wystepuje=true;
      min=max=liczba;
   } else j++;
}
```

W wyniku wykonania powyższego fragmentu programu zmienna `j` wskazuje na pierwszą liczbę półpierwszą (jeśli `j<1000`) lub wskazuje na brak liczb półpierwszych (gdy `j=1000`). Ponadto `wystepuje=true` dokładnie wtedy, gdy w ciągu znaleźliśmy liczbę półpierwszą.

Następnie przeglądamy kolejne liczby (począwszy od `j`-tej), stosując klasyczny algorytm znajdowania minimum (lub maksimum) w nieuporządkowanym ciągu liczb. Pamiętamy jednak o tym, że minimum i maksimum wybieramy tylko spośród liczb półpierwszych. Dlatego

też wartość kolejnej liczby porównywana jest z `min` i `max` tylko wtedy, gdy dana liczba jest półpierwsza. Poniżej prezentujemy fragment kodu realizujący powyżej opisany algorytm:

```
ile=0;
for(int i=j; i<1000; i++)
{
   liczba = bin2wart(ciagi[i]);
   if (czy_polpierwsza(liczba))
   {
      ile++;
      if (liczba>max) max = liczba;
      if (liczba<min) min = liczba;
   }
}
```

Po wykonaniu powyższego fragmentu programu zmienna `wystepuje` wskazuje, czy w pliku wystąpiła choć jedna liczba półpierwsza. Gdy `wystepuje=true`, zmienne `min` i `max` zawierają odpowiednio wartość najmniejszej liczby półpierwszej i wartość największej liczby półpierwszej. Ponadto w zmiennej `ile` wyznaczona zostanie ilość liczb półpierwszych.