Deljenje skrivnosti

1. Splošna predstavitev problema

Deljenje skrivnosti omogoča razbitje skrivnosti na n deležev, pri čemer zgolj en delež ne omogoča razkritja skrivnosti. Za razkritje skrivnosti je potrebno združiti vsaj k deležev. Na takšen način je možno implementirati sistem za varen prenos podatkov. Pri tem se začetno skrito sporočilo razbije v n datotek, ki se pošljejo po n različnih povezavah. Na cilj mora priti vsaj k datotek, ki se sestavijo v začetno sporočilo.

Shamirjevo metodo za deljenje skrivnosti je razvil Adi Shamir leta 1979. Metoda omogoča deljenje skrivnosti S na podlagi polinomske interpolacije:

1. Skrivnost S se najprej razbije na n deležev tako, da se najprej generira neničelne naključne celoštevilske vrednosti koeficientov polinoma od a_1 do a_{k-1}.
2. Koeficient a_0 naj bo enak skrivnosti S.
3. Za vsakega zaupnika x ($1 \leq x \leq n$) izračunamo delež skrivnosti kot par $<x, f(x)>$, kjer je $f(x)$ polinom: $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$. *To pomeni, da je* $S = f(0)$.

Vsak izmed x zaupnikov dobi svoj delež skrivnosti v obliki dveh števil, torej par $<x, f(x)>$, kjer je ($1 \leq x \leq n$). Na podlagi poljubnih k deležev izmed n zaupnikov je možno rekonstruirati originalno sporočilo na podlagi enačbe:

$$f(0) = \sum_{j=0}^{k-1} y_j \prod_{\substack{m=0 \\ m \neq j}}^{k-1} \frac{x_m}{x_m - x_j},$$

pri tem je k deležev predstavljenih v obliki $<x, y>$.

1.1 Primer

Želimo deliti skrivnost 1234 na n=6 delov, kjer za rekonstrukcijo zadostujejo k=3 deleži. Najprej tvorimo k-1 naključnih koeficientov polinoma, npr. a_1 = 166 in a_2 = 94.

Polinom zapišemo kot $f(x) = 1234 + 166x + 94x^2$. Na podlagi polinoma izračunamo deleže D:

- D_1 = <1, 1494>,
- D_2 = <2, 1942>,
- D_3 = <3, 2578>,
- D_4 = <4, 3402>,
- D_5 = <5, 4414>,
- D_6 = <6, 5614>

Če želimo iz deležev 2, 4, in 5 dobiti skrivnost, izvedemo sledeč izračun: 1942*4/(4-2)*5/(5-2)+3402*2/(2-4)*5/(5-4)+4414*2/(2-5)*4/(4-5) = 1234

2. Pomoč pri implementaciji

Implementacija osnovne verzije deljenja skrivnosti je možna s psevdokodo v izpisu 1.

```
function RAZBIJ(S, k, n)
 begin
  a_0 := S
  for i := 1 to k-1 do
  begin
    a_i = random(1, 2^32-1)
  end

  for x := 1 to n do
  begin
    f_x = \sum_{i=0}^{k-1} a_i x^i
    Izpiši <x, f_x> kot delež in dodaj v množico D
  end
  return D
end
```

Izpis 1: Razbitje skrivnosti na *n* deležnikov

Rekonstrukcija poteka na podlagi *k* deležev iz seznama D, ki jih zapišemo v obliki <*x*, *y*>.

```
function REKONSTRUIRAJ(D, k)
 begin
    Naj bo <x_i, y_i> i-ti delež iz seznama D velikosti k
    S = \sum_{j=1}^{k} y_j \prod_{i=1 \wedge i \neq j}^{k} \frac{x_i}{x_i - x_j}

    Izpiši skrivnost S;
end
```

Izpis 2: Rekonstrukcija skrivnosti iz k deležev v množici D

Težava postopka v izpisu 2 je v tem, da lahko pride do deljenja z ostankom, za kar bi bilo potrebno računati z decimalnimi števili. V ta namen je smiselno izpis 2 razširiti na takšen način, da izračunamo vse imenovalce, jih zmnožimo in potem šele na koncu celoten ulomek množimo z zmnožkom imenovalcev. Pred tem pa moramo vse ulomke postaviti na skupen imenovalec. Rešitev je prikazana v izpisu 3 in omogoča delo zgolj s celimi števili.

```
function REKONSTRUIRAJ_ROBUST(D, k)
 begin
   Naj bo <x_i, y_i> i-ti delež iz množice D velikosti k
   downGlobal := \prod_{j=1}^{k} y_j \prod_{i=1 \wedge i \neq j}^{k} (x_i - x_j)

   S : = 0
   for j := 1 to k do
   begin
```

```
    up := ∏_{i=1∧i≠j}^{k} x_i
    downLocal := ∏_{i=1∧i≠j}^{k} (x_i − x_j)
    S := S + y_j * up * (downGlobal / downLocal)
  end
  S := S/downGlobal;
  Izpiši S
end
```

Izpis 3: Rekonstrukcija skrivnosti iz k deležev v množici D, kjer deljenjem poteka s celimi števili

3. Zahteve naloge

3.1 Implementacija (4 točke)

Implementirajte aplikacijo za deljenje skrivnosti, ki omogoča razbitje poljubnih datotek na *n* deležev. Za vsak delež ustvarite ločeno datoteko. Uporabnik bo nato lahko *k* deležev/datotek ročno prenesel po različnih nosilcih do prejemnika datotek, ki bo lahko iz deležev rekonstruiral datoteko. Uporabnik naj poda: *n*, *k,* vhodno datoteko in predpono imena datotek z deleži.

Nato razvijte aplikacijo, ki bo na podlagi algoritma iz izpisa 3 omogočala rekonstrukcijo originalne datoteke iz *k* datotek/deležev. Uporabnik naj v tem primeru poda imena datotek z deleži, in ime izhodne datoteke.

Glede na to, da lahko pričakujemo zahtevo po kodiranju večjih datotek, omogočite dva načina razbitja datotek. V obeh aplikacijah naj način razbitja določi uporabnik:

- predstavitev celotne datoteke z BigInt in predstavitev enega deleža s števili *x* in y (**1 točka**),
- branje vhodne datoteke po zlogih in zapis deleža v obliki zaporedja števil: x $y^{(1)}$ $y^{(2)}$ $y^{(3)}$.... Torej zaporedni številki deleža *x* sledijo vrednosti deleža za vsak zlog datoteke posebej (**1 točka**).

Vse obravnavane algoritme implementirajte s knjižnico za predstavitev števil z večjim številom bitov (BigInt), npr.:

- https://www.boost.org/doc/libs/1_71_0/libs/multiprecision/doc/html/boost_multiprecision/tut/ints/cpp_int.html
- https://gmplib.org/,
- https://www.boost.org/doc/libs/1_71_0/libs/multiprecision/doc/html/boost_multiprecision/intro.html.

3.2 Analiza (3 točke)

V naslednjem koraku naloge sledi analiza. Najprej nas zanima, ali je implementacija primerna za večje datoteke. Narišite sledeče grafikone časovne zahtevnosti (**2 točki**):

- čas razbitja glede na velikost datoteke v zlogih z naključno datoteko in uporabo BigInt,

- čas razbitja glede naključne datoteke glede na velikost datoteke v zlogih in obdelavo datoteke po zlogih,
- čas rekonstrukcije glede na velikost datoteke v zlogih in uporabo BigInt,
- čas rekonstrukcije glede na velikost datoteke v zlogih in obdelavo datotek po zlogih.

Pri tem pri vsaki velikosti datotek generirajte več naključnih datotek in izmerite povprečen čas. Testiranje lahko izvedete pri poljubnih vrednostih n in k. Pri tem test izvedite pri vsaj treh različnih kombinacijah vrednosti n in k, da preverite njun vpliv na časovno zahtevnost.

Na koncu preučite slabosti algoritma iz izpisa 2 (**1 točka**). Zraven algoritma iz izpisa 3 implementirajte še algoritem iz izpisa 2. Pri deljenju uporabite 32-bitna decimalna števila. Pri manjših vrednostih a, n in k, bi program moral delovati brez večjih napak. V primeru večjih števil pa lahko pride do napačne rekonstrukcije. Preverite, pri kako velikih vrednostih pride do napak in ugotovitve zapišite v poročilu.