

შეკუმშვის ალგორითმების ტიპები

შეკუმშვის ორი ძირითადი ტიპი:

დანაკარგებიანი შეკუმშვა

მაგალითად “JPEG”. შეკუმშვის და გახსნის შედეგად მიღებული ფაილი შეიძლება ცოტათი განსხვავდებოდეს ორიგინალისგან.

უდანაკარგო შეკუმშვა

მაგალითად “PNG”, “ZIP”, ა.შ. შეკუმშვის და გახსნის შემდეგ ზუსტად იგივე ფაილი მიიღება.

(ჩვენი ალგორითმი ამ ტიპის იქნება)

უდანაკარგო შეკუმშვის ძირითადი იდეა

ჩვეულებრივი კოდირებისას ყველა სიმბოლოს ერთნაირი სიგრძის კოდი შეესაბამება

თუ შეკუმშვა გვინდა უფრო ხშირად გამოყენებად სიმბოლოებს უფრო მოკლე კოდები უნდა შევუსაბამოთ

მაგალითად თუ განაწილებაა: {A: 20%, B: 5%, C: 10%, D: 40%, E: 25%}

(ეს ალბათობები შეგვიძლია მთელ ფაილში სიმბოლოების დათვლით გავიგოთ ხოლმე)

ჩვეულებრივი კოდი იქნება – {A – 000, B – 001, C – 010, D – 011, E – 100}

(საშუალო სიგრძე: 3 ბიტი)

შემკუმშავი კოდი შეიძლება იყოს – {A – 100, B – 1010, C – 1011, D – 0, E – 11}

(საშუალო სიგრძე: 2.1 ბიტი)

უპრეფიქსო კოდები

კოდების (კოდური სიტყვების) ერთმანეთზე მიდგმის შემდეგ მათი უკან გამოყოფა უნდა შეგვეძლოს:

$$\{A - 100, B - 1010, C - 1011, D - 0, E - 11\}$$

DECADE ---> 0111011100011

0111011100011 ---> 0 11 1011 100 0 11 ---> DECADE

ამისთვის ერთი კოდი მეორე კოდის დასაწყისი (პრეფიქსი) არ უნდა იყოს.

კოდების და ხეების შესაბამისობა

უპრეფიქსო კოდები შეესაბამება სრულ ორობით ხეებს, რომლების ფოთლებშიც სიმბოლოებია შენახული.

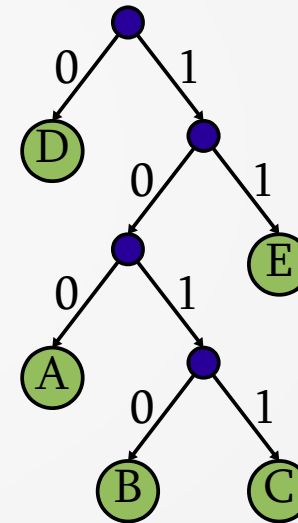
A – 100

B – 1010

C – 1011

D – 0

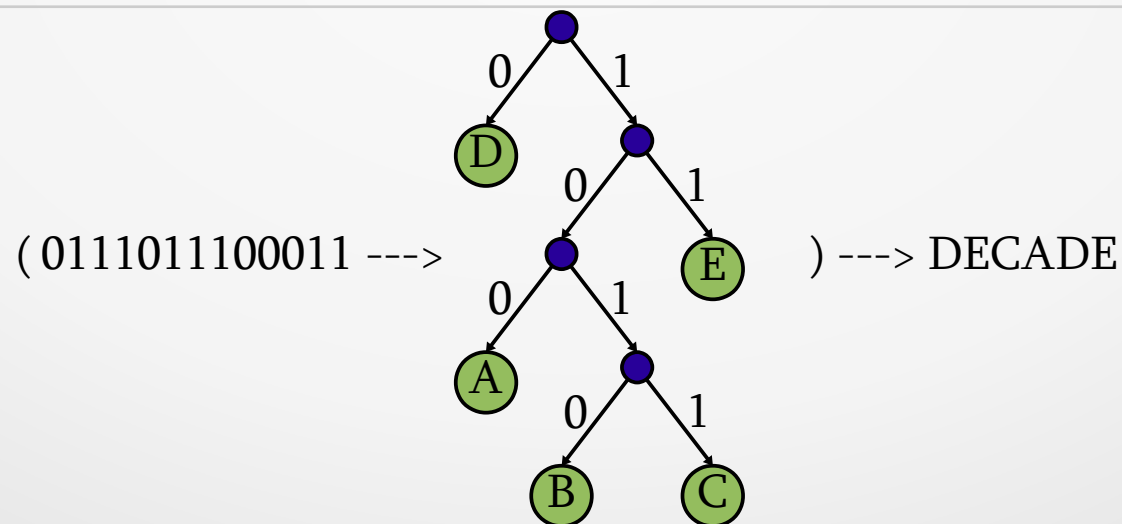
E – 11



კოდირება და დეკოდირება

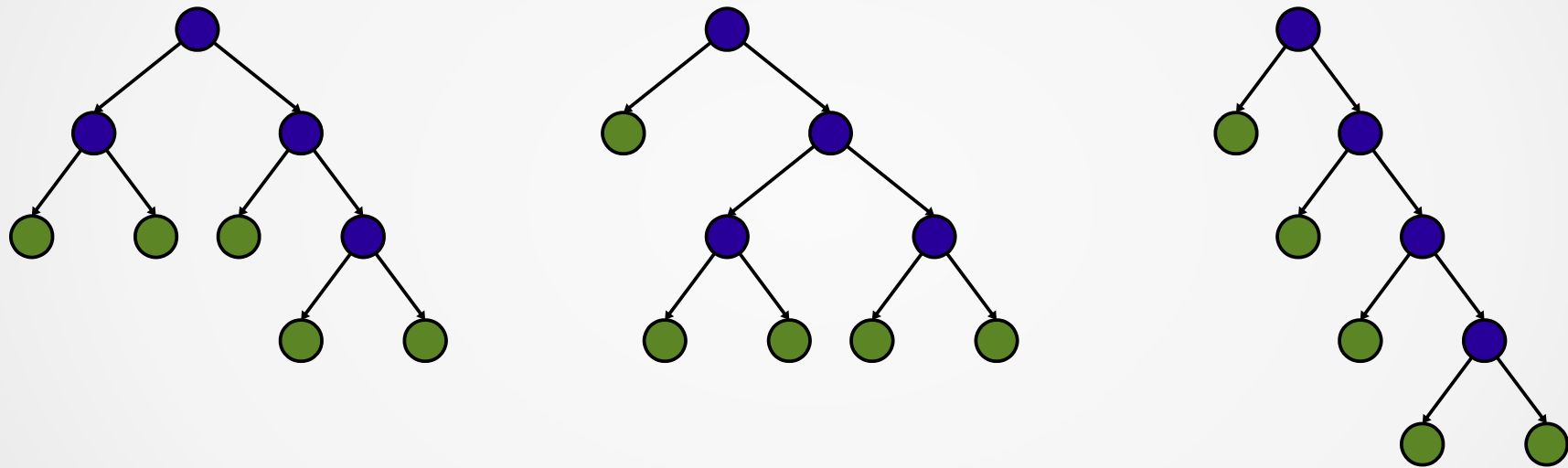
მოცემული სტრინგის კოდირებისთვის უბრალოდ სიმბოლოების და კოდების შესაბამისობის ცხრილი (მაგალითად Map) გვჭირდება.

კოდის დეკოდირებისას შეგვიძლია ორობით ხეს გავუყვეთ და როდესაც ფოთოლში მივალთ ამ ფოთოლში არსებული სიმბოლო გამოვიტანოთ



რომელი უპრეფიქსო კოდი ჯობია?

სიმბოლოების ერთნაირ რაოდენობაზე შეიძლება რამდენიმე განსხვავებული ხე აიგოს.



სხვადასხვა სიტუაციაში (სიმბოლოების ალბათობების მიხედვით) ამ ხეებიდან სხვადასხვა ხის გამოყენება ჯობია.

ჰაფმანის ხის აგება

ჰაფმანის ალგორითმი მოცემული ალბათობების (ან სიხშირეების) მიხედვით აგებს ოპტიმალურ ხეს (და შესაბამისად კოდს).

ჰაფმანის ალგორითმი

დააღაგეთ სიმბოლოები ალბათობების კლებადობის მიხედვით

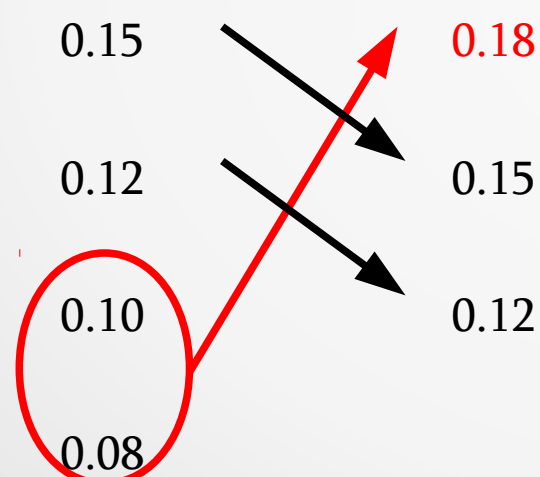
თუ სულ ორი სიმბოლოა – ერთს მიანიჭეთ - “0”, მეორეს - “1” (base case)

თუ არა:

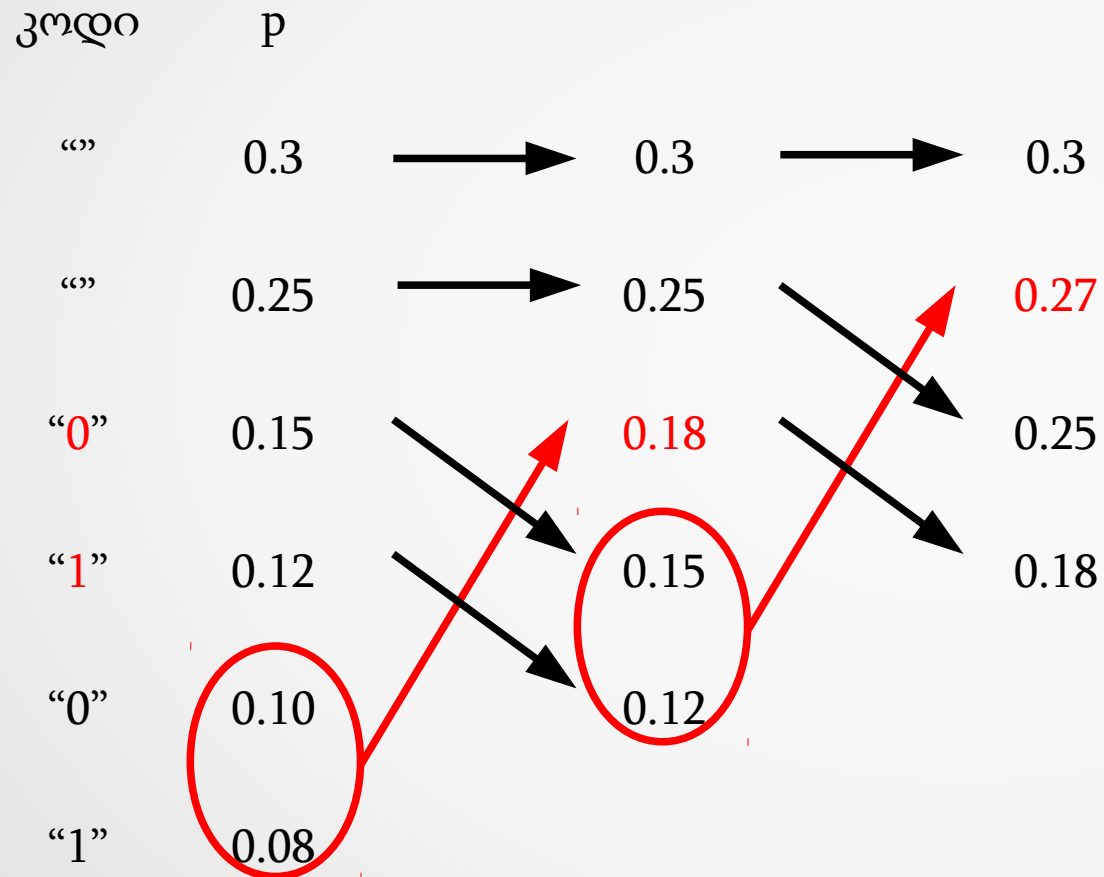
1. გააერთიანეთ ბოლო ორი სიმბოლო ერთ სიმბოლოდ (ალბათობები შეკრიბეთ)
2. რეკურსიულად ააგეთ ჰაფმანის კოდი დარჩენილი ერთით ნაკლები სიმბოლოსთვის
3. გახლიჩეთ გაერთიანებული სიმბოლო (კოდით x , სადაც x ნებისმიერი სიგრძის სტრინგია) ისევ ორ ნაწილად და მიანიჭეთ ამ ნაწილებს კოდები: $x + “0”$ და $x + “1”$

ჰაფმანის ხის აგება (მაგალითი)

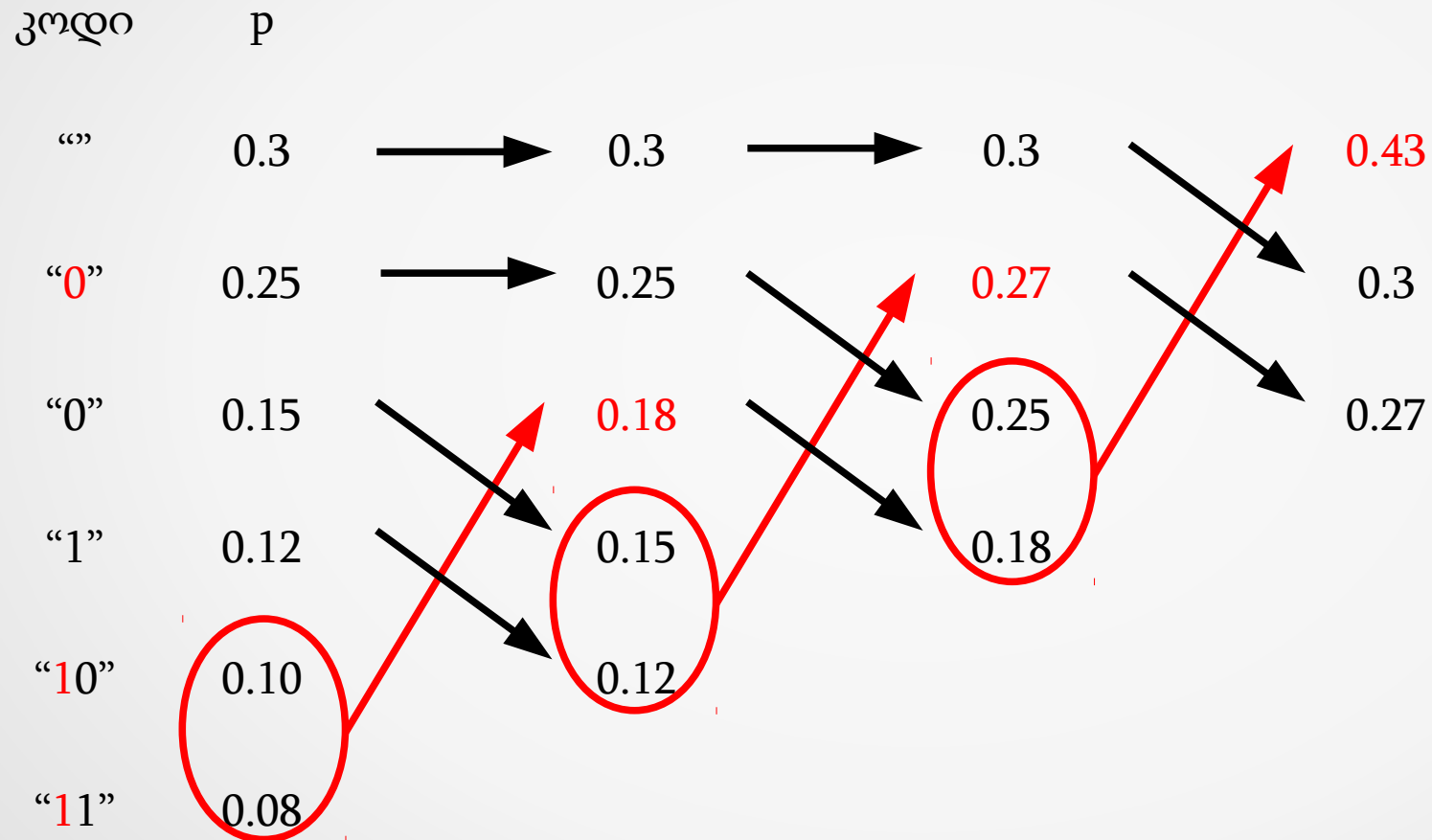
კოდი	p		
“”	0.3	→	0.3
“”	0.25	→	0.25
“”	0.15		0.18
“”	0.12		0.15
“0”	0.10		0.12
“1”	0.08		



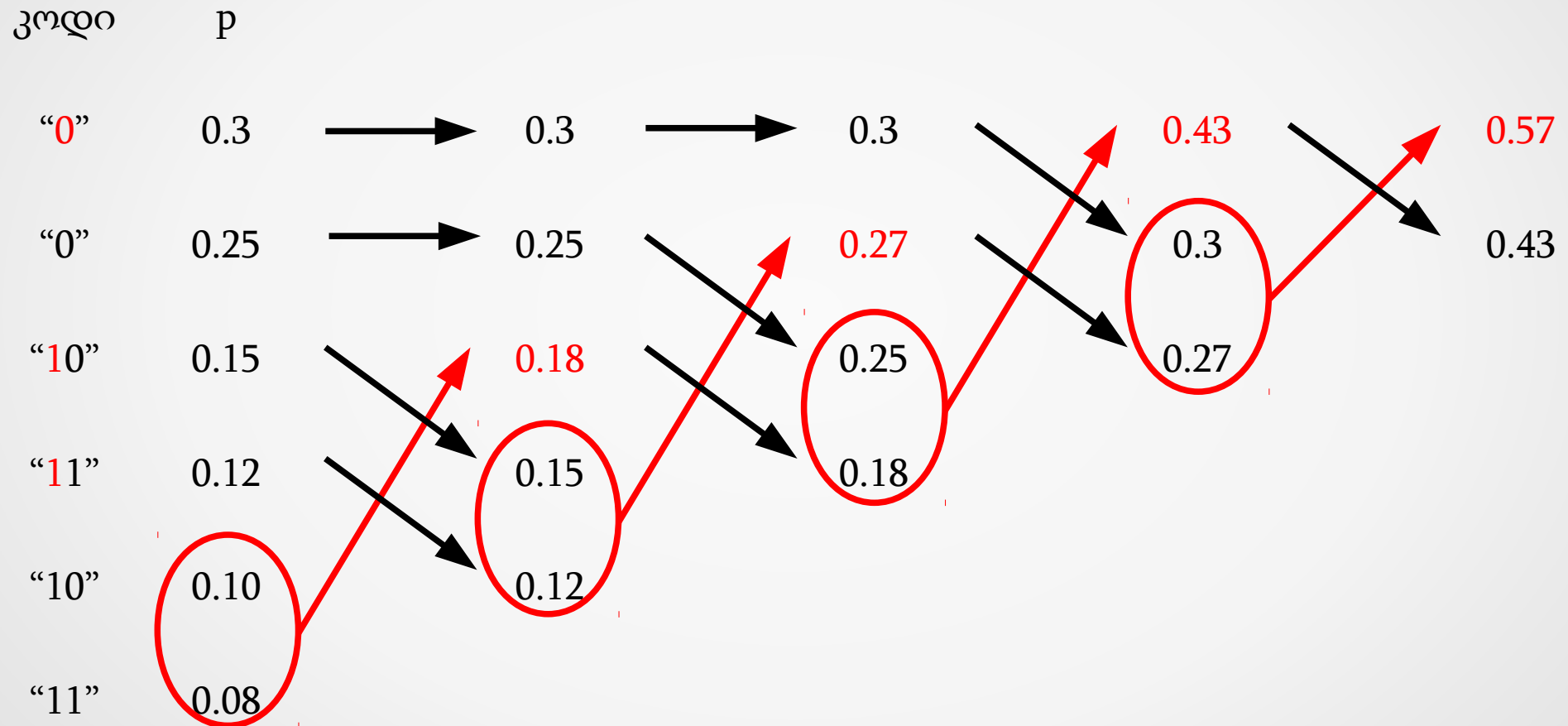
ჰაფმანის ხის აგება (მაგალითი)



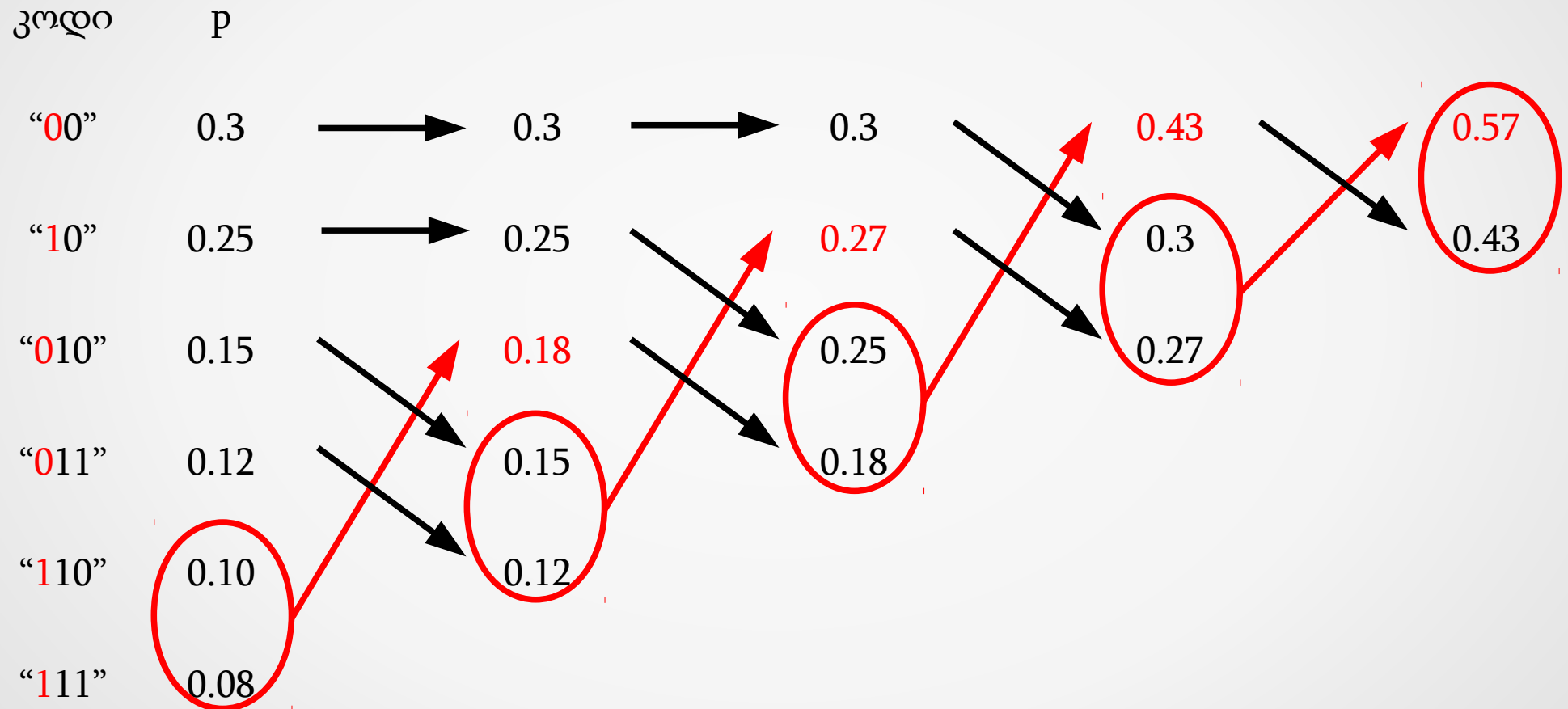
ჰაფმანის ხის აგება (მაგალითი)



ჰაფმანის ხის აგება (მაგალითი)



ჰაფმანის ხის აგება (მაგალითი)



ფაილის დამთავრების სიმბოლო

ფაილი ბაიტებში ინახება. ამიტომ, თუ ჩვენი კოდირებული ინფორმაცია არასრულ ბაიტს იკავებს ეს ბაიტი შეივსება 0-ებით.

$\{A - 100, B - 1010, C - 1011, D - 0, E - 11\}$

DECADE ---> 0111011100011 ---> 0111011100011000

0111011100011000 ---> DECADEDDD

ამის თავიდან ასაცილებლად უნდა შემოვიღოთ ერთი დამატებითი სიმბოლო ე.წ. “EOF” და გავაკეთოთ $D + E + C + A + D + E + EOF$ მიმდევრობის კოდირება.

დეკოდირებისას EOF-ის მერე ყველაფერს იგნორირებას გავუკეთებთ

ცხადია EOF სიმბოლო კოდირების ხის აგების დროსაც უნდა გავითვალისწინოთ

სიმბოლოების და კოდების შესაბამისობის შენახვა

შეკუმშული ფაილის მიმღებმა არ იცის ჩვენს ფაილში რა ალბათობით გვხვდებოდა სიმბოლოები. ამიტომ რამენაირად უნდა მივაწოდოთ ეს ინფორმაცია.

მეთოდი 1: გადავცეთ სიმბოლოების და კოდური სიტყვების შესაბამისობა ფაილიდან ერთად (ჩავდეთ ეს ინფორმაცია შეკუმშურ ფაილში)

(ეს მეთოდი დიდ ადგილს იკავებს და არაეფექტურია)

მეთოდი 2: გადავცეთ სიმბოლოების შეხვედრის სიხშირეები ფაილიდან ერთად. ამ ინფორმაციით მიმღებს თავისითაც შეეძლება იგივე ხის აგება რაც შემკუმშავს ჰქონდა.

დავალებაში წინასწარ გაქვთ გამზადებული მეთოდი რომელიც კოდირება/აღდგენისთვის საჭირო ინფორმაციას შეინახავს ფაილში.