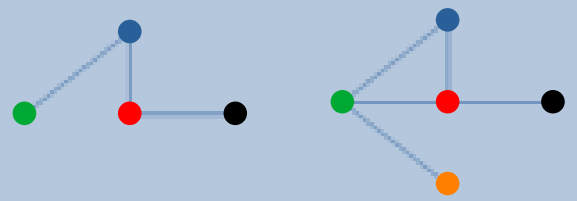
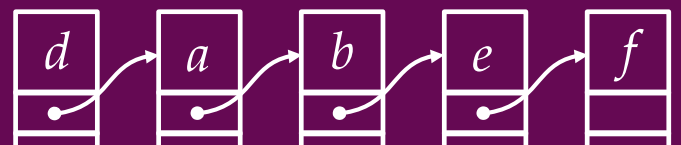
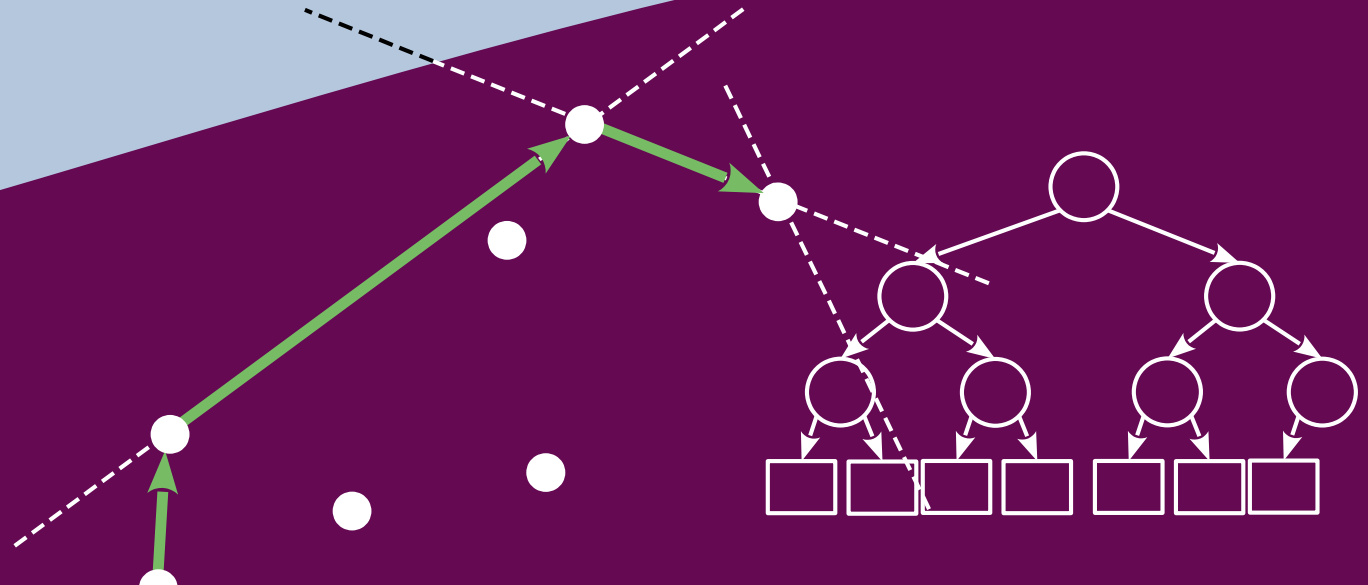
Translated from Croatian to English - www.onlinedoctranslator.com

Advanced algorithms and structures data

Week 7: Compression algorithms

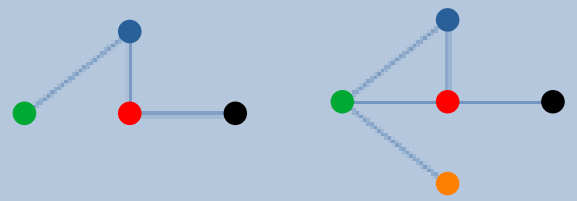


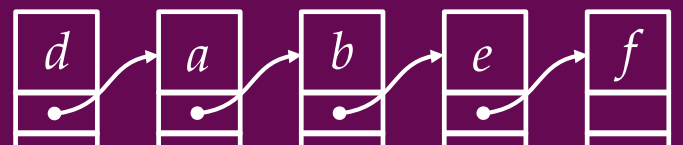


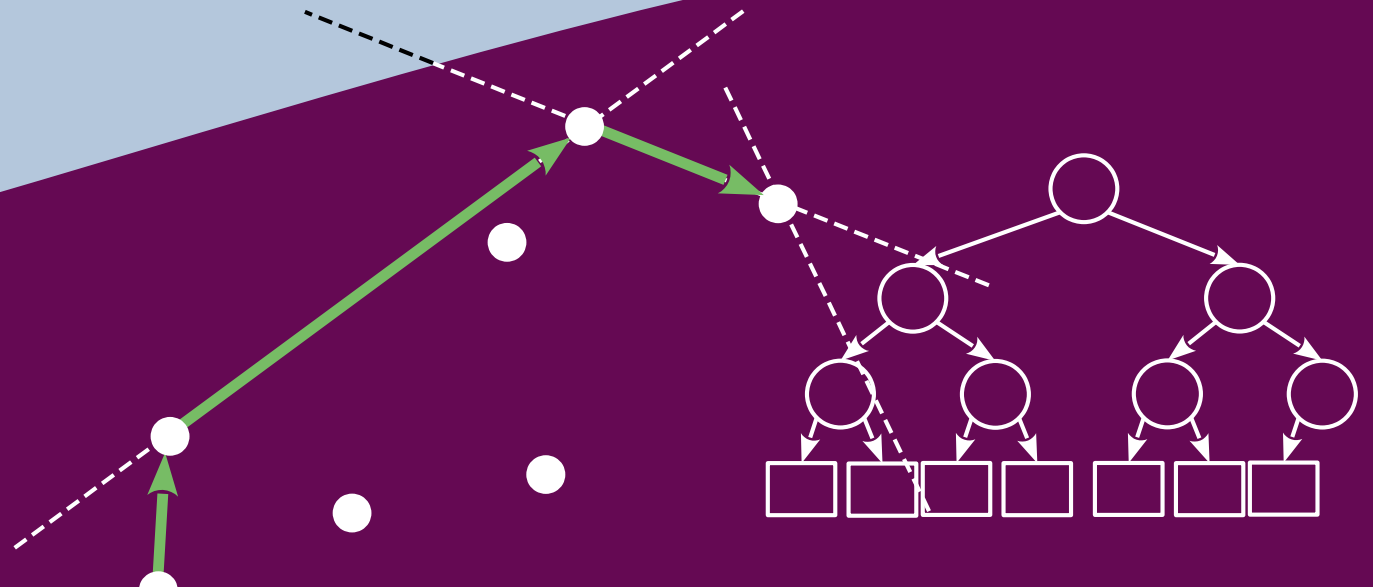


Napredni algoritmi i strukture podataka

Tjedan 7: Kompresijski algoritmi







Content

- Compression algorithms
  - The basics
  - Huffman coding
  - Lempel-Ziv 77

Literature:
- Krleža, Brčić: "Advanced Algorithms and Data Structures", Lecture book.
- Drozdek: "Elements of data compression ”, Brooks/Cole Thomson Learning, 2002. (chapters1, 3, 5.1)

Sadržaj

- Kompresijski algoritmi
  - Osnove
  - Huffmanovo kodiranje
  - Lempel-Ziv 77

Literatura:

- Krleža, Brčić: „Advanced Algorithms and Data Structures”, Lecture book.
- Drozdek: ”Elements of data compression”, Brooks/Cole Thomson Learning, 2002. (chapters 1, 3, 5.1)

Compression algorithms

- Compression - transforms the input into a more compact output
- Satisfactory input reconstruction possible ("approximate" inverse) - decompression
- **No losses**–perfect reconstruction
- **With losses**–pragmatically satisfactory reconstruction

Kompresijski algoritmi

- Kompresija - transformira ulaz u kompaktniji izlaz
- Zadovoljavajuća rekonstrukcija ulaza moguća („približni” inverz) – dekompresija
- **Bez gubitaka** – savršena rekonstrukcija
- **Sa gubitcima** – pragmatično zadovoljavajuća rekonstrukcija

Theoretical background

- Two approaches

- **Statistical Information Theory (SIT)**
  - Shannon entropy and ensembles

- Algorithmic Information Theory (AIT)
  - Kolmogorov complexity, Turing machines

Teorijska pozadina

- Dva pristupa

- **Statistička teorija informacija (SIT)**
  - Shannonova entropija i ansambli

- Algoritamska teorija informacija (AIT)
  - Kolmogorovljeva složenost, Turingovi strojevi

SIT

- Simple and less complex algorithms

- Randomness rating
  - Redundancies are compressed, randomness remains in the output
  - Recursively generated data identified as random - incompressible

SIT

- Jednostavna i algoritmi manje složenosti

- Ocjena slučajnosti
  - Redundacije se komprimiraju, slučajnost ostaje u izlazu
  - Rekurzivno generirani podatci identificirani kao slučajni – nekompresibilni

SIT

- *n* of independent events $S=\{x_1,\ldots,x_n\}$
- probability of occurrence $P=\{p_1,\ldots p_n\}$

Entropy

$$( ) = - \log_2$$

$$=1$$

SIT

- *n* nezavisnih događaja $S=\{x_1,...,x_n\}$
- vjerojatnosti pojave $P=\{p_1,...p_n\}$

Entropija

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

AIT

- U – reference universal Turing machine

Kolmogorov complexity of a string of characters x is the length of the shortest program p that generates x:

$$:=(m)\text{in}\{\quad :\quad =(\ \})\quad (\ )$$

Incalculable!

AIT

- U – referentni univerzalni Turingov stroj

Kolmogorovljeva složenost nekog niza znakova x jest duljina najkraćeg programa p koji generira x:

$$K_U(x) \coloneqq \min_p \{l(p) : U(p) = x\}$$

Neizračunljiva!

Coding

- **Unique decodability**(code)
  - If and only if there is only one way to split the code into separate code words
- **Prefix-free property**(code)
  - If neither codeword is a prefix of another
- **Kraft's inequality**
  - There is a prefix-free binary code $C=\{c_1,\ldots, c_n\}$ with lengths $\{l_1,\ldots,l_n\}$ if and only if

$$2^{-} \leq 1$$

$$=1$$

Kodiranje

- **Jedinstvena dekodabilnost** (koda)
  - Ako i samo ako postoji samo jedan način razdvajanja koda u odvojene kodne riječi
- **Prefix-free svojstvo** (koda)
  - Ako nijedna kodna riječ nije prefiks druge
- **Kraftova nejednakost**
  - Postoji prefix-free binarni kod $C=\{c_1,\ldots,c_n\}$ sa duljinama $\{l_1,\ldots,l_n\}$ ako i samo ako

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1$$

Coding

- **Shannon's fundamental theorem for discrete noiseless coding**
  - For S having entropy H(S), prefix-free encoding is possible **sequence of *k* signs** from S with average code word length L_k by an element from S that satisfies:

$$H(S) \leq \frac{L_k}{k} < H(S) + \frac{1}{k}$$

- By increasing the size of the grouping ("supersymbol"), the average length of the code word per input symbol approaches the entropy of the source
  - "fractional" source coding
  - Code complexity (and code table size) increases

Kodiranje

- **Shannonov osnovni teorem za diskretno bešumno kodiranje**
  - Za S koji ima entropiju H(S), moguće je prefix-free kodiranje **sekvenci od *k* znakova** iz S sa prosječnom duljinom kodne riječi L_k po elementu iz S koja zadovoljava:

$$H(S) \leq \frac{L_k}{k} < \mathrm{H(S)} + \frac{1}{k}$$

- Povećanjem veličine grupiranja („supersimbola”) se prosječna duljina kodne riječi po ulaznom simbolu približava entropiji izvora
  - „frakcionalno” kodiranje izvora
  - Povećava se složenost koda (i veličina kodne tablice)

SIT vs AIT

3.14159265358979323846264338327950288419716939937510…
(1,000,000+ decimals)

FYI: the current (July 2022) record is 10^{14} digits

- **SIT incompressible**
  - Passes all randomness tests

SIT vs AIT

3.14159265358979323846264338327950288419716939937510...
(1,000,000+ decimala)

FYI: trenutni (Srpanj 2022.) rekord 10^{14} znamenki

- **SIT nekompresibilno**
  - Prolazi sve testove slučajnosti

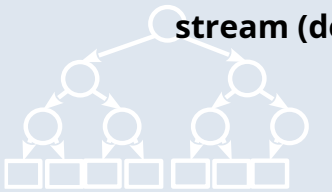
SIT vs AIT

3.14159265358979323846264338327950288419716939937510... (1,000,000+ decimals)

```
def stream(length):
    a = 2 * sqrt(2) / 9801
    b=1
    s=0
    n=0
    while b>10**-length:
        b=a*(fact(4*n)/pow(fact(n),4))*((26390*n+1103)/pow(396,4*n)) s+=b

        n+=1
    return 1/s
```

stream (decimals)



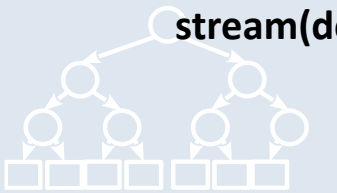
FYI:Ramanujan's formula

SIT vs AIT

3.14159265358979323846264338327950288419716939937510... (1,000,000+ decimala)

```
def stream(length):
    a = 2 * sqrt(2) / 9801
    b=1
    s=0
    n=0
    while b>10**-length:
        b=a*(fact(4*n)/pow(fact(n),4))*((26390*n+1103)/pow(396,4*n))
        s+=b
        n+=1
    return 1/s

stream(decimals)
```



FYI: Ramanujanova formula

Lossless compression

- Removing statistical redundancy:
  - Reducing the number of unique symbols
  - Encoding of more common symbols with fewer bits
  - Minimize average length = σ

- Techniques:
  - Frequency-based coding
  - Trees
  - Recursive divisions
  - Sampling
  - Quantization (rounding)
  - Dictionaries (character groupings)
  - Functional transformations

Kompresija bez gubitaka

- Uklanjanje statističke redundacije:
  - Reduciranje broja jedinstvenih simbola
  - Enkodiranje češćih simbola sa manje bitova
  - Minimizirati prosječnu duljinu $L_{avg} = \sum_i p_i l_i$

- Tehnike:
  - Kodiranje bazirano na frekvenciji
  - Stabla
  - Rekurzivne podjele
  - Uzorkovanje
  - Kvantizacija (zaokruživanje)
  - Riječnici (grupiranja znakova)
  - Funkcijske transformacije

Huffman coding (+trees)

- Conversion from the original alphabet $S=\{x_1,...,x_n\}$ with probabilities $P=\{p_1,..., p_n\}$ into code words $C=\{c_1,..., c_n\}$ with corresponding lengths $L=\{l_1,...,l_n\}$.

- First coding for **optimal** variable length code
  - A greedy algorithm

- "Rounding" of probabilities in length (quantization)
  - Integer number of bits for code words

Huffmanovo kodiranje (+stabla)

- Pretvorba iz izvornog alfabeta $S=\{x_1,...,x_n\}$ sa vjerojatnostima $P=\{p_1,...,p_n\}$ u kodne riječi $C=\{c_1,...,c_n\}$ sa odgovarajućim duljinama $L=\{l_1,...,l_n\}$.

- Prvo kodiranje za **optimalan** kod varijabilne duljine
  - Pohlepni algoritam

- „Zaokruživanje” vjerojatnosti u duljini (kvantizacija)
  - Cjelobrojni broj bitova za kodne riječi

Building a Huffman tree

- Input: S,P sorted by P
- BuildHuffTree(S,P):
  1. Q_1=Queue(),Q_2=Queue()// Q_1contains leaves, Q_2subtrees
  2. For each ∈ :
     1. Q_1.enqueu((=nodes( ),))
  3. while |Q_1|+|Q_2|>1:
     1. Get , , ,(,the two(elements)with the lowest probabilities of all the elements on $Q_1$and Q_2
     2. R=Node(), R.left=N_{And}, R.right=N_b// left-right invariant
     3. Q_2.enqueu(, +)
  4. return root=Q_2.dequeue()

Izgradnja Huffmanovog stabla

- Ulaz: S,P sortirani prema P
- IzgradiHuffStablo(S,P):
  1. Q_1=Queue(),Q_2=Queue() // Q_1 sadrži listove, Q_2 podstabla
  2. For each $x_i \in S$:
     1. Q_1.enqueu((N_i =Node(x_i),p_i))
  3. while |Q_1|+|Q_2|>1:
     1. Dohvati $(N_a, p_a), (N_b, p_b)$, dva elementa sa najmanjim vjerojatnostima od svih elemenata na Q_1 i Q_2
     2. R=Node(), R.left=N_a, R.right=N_b // lijevo-desno invarijantno
     3. Q_2.enqueu($R, p_a + p_b$)
  4. return root=Q_2.dequeue()

Huffman encoding

- To encode each symbol x_{and}, traversal from the tree to the corresponding leaf matches the code word c_{and}
  - Each transition to the left subtree adds 0 to the end of the codeword, transition to the right adds 1 to the end of the codeword
- Caching code words in the code table for efficiency
  - Key-value store

Huffmanovo enkodiranje

- Za kodiranje svakog simbola x_i, obilazak od stabla do pripadajućeg lista slaže kodnu riječ c_i
  - Svaki prelazak u lijevo podstablo dodaje 0 na kraj kodne riječi, prelazak u desno dodaje 1 na kraj kodne riječi
- Caching kodnih riječi u kodnu tablicu radi efikasnosti
  - Key-value store

Huffman decoding

- The decoder must have a tree or code table
  - Building a tree from the code table
- Input: encoded bit-stream Y, T Huffman tree
- HuffDecoding (Y, T):

1. C=T.root
2. **While** is non-empty:
   1. **If**C is a leaf node:
      1. Output C.symbol
      2. C=T.root
   2. B=Y.fetch()
   3. **If**B=0: C=C.left**otherwise**C=C.right

Huffmanovo dekodiranje

- Dekoder mora imati stablo ili kodnu tablicu
  - Izgradnja stabla iz kodne tablice
- Ulaz: enkodirani bit-stream Y, T Huffmanovo stablo
- HuffDekodiranje (Y, T):
  1. C=T.root
  2. **While** Y is non-empty:
     1. **If** C is leaf node:
        1. Output C.symbol
        2. C=T.root
     2. B=Y.fetch()
     3. **If** B=0: C=C.left **else** C=C.right

Huffman - example

Prev. original alphabetS={A,B,C,D,E},probabilitiesP={0.4, 0.2, 0.14, 0.13, 0.13}

a) Build a Huffman code table

b) Encode the input string ABCABCABCE and calculate the compression factor

c) Decode the incoming stream 01110111

Huffman - primjer

Pretp. izvorni alfabet `S={A,B,C,D,E}`, vjerojatnosti `P={0.4, 0.2, 0.14, 0.13, 0.13}`

a) Izgradite Huffmanovu kodnu tablicu

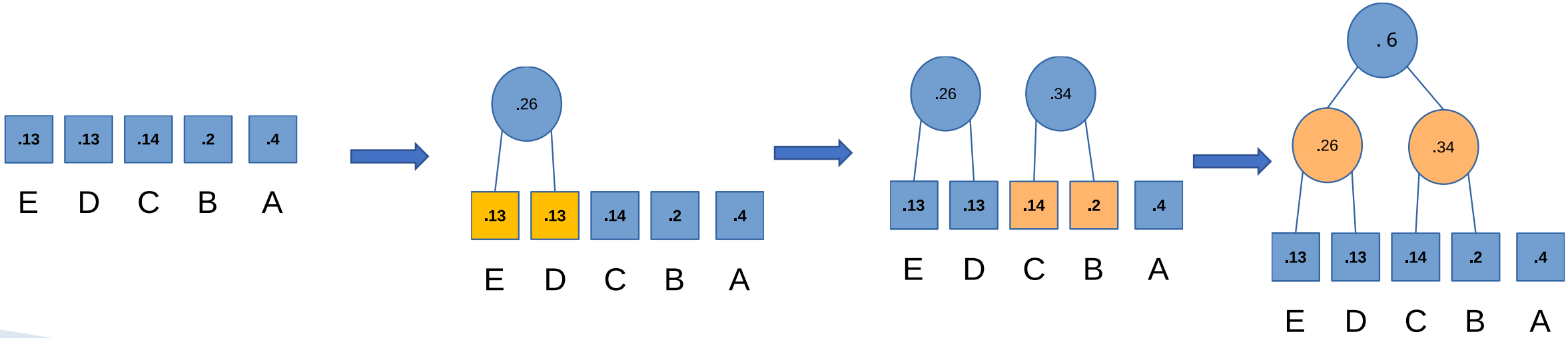
b) Enkodirajte ulazni niz ABCABCABCE i izračunajte faktor kompresije

c) Dekodirajte dolazni tok 01110111

Huffman – example a)

Prev. original alphabetS={A,B,C,D,E},probabilitiesP={0.4, 0.2, 0.14, 0.13, 0.13}

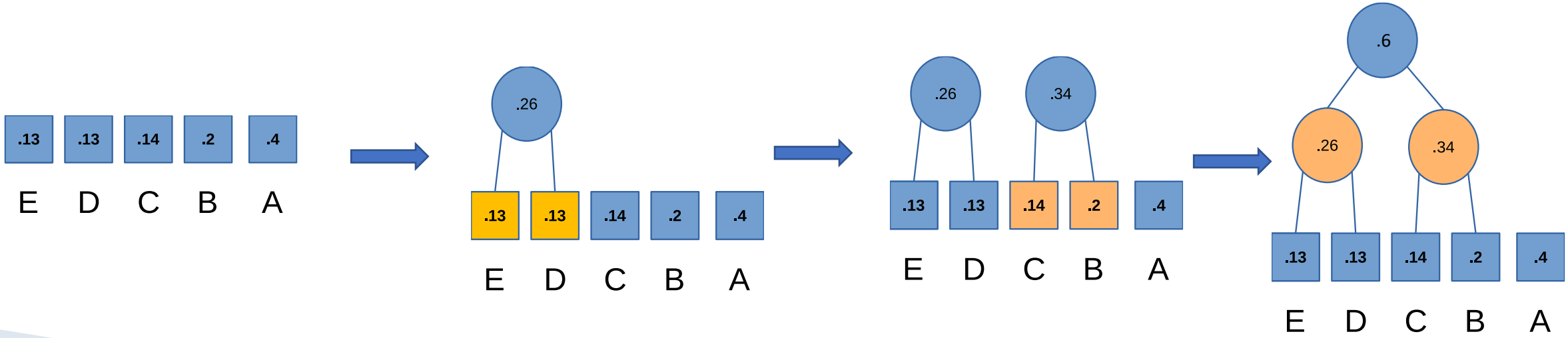
a) Build a Huffman code table



Huffman – primjer a)

Pretp. izvorni alfabet `S={A,B,C,D,E}`, vjerojatnosti `P={0.4, 0.2, 0.14, 0.13, 0.13}`

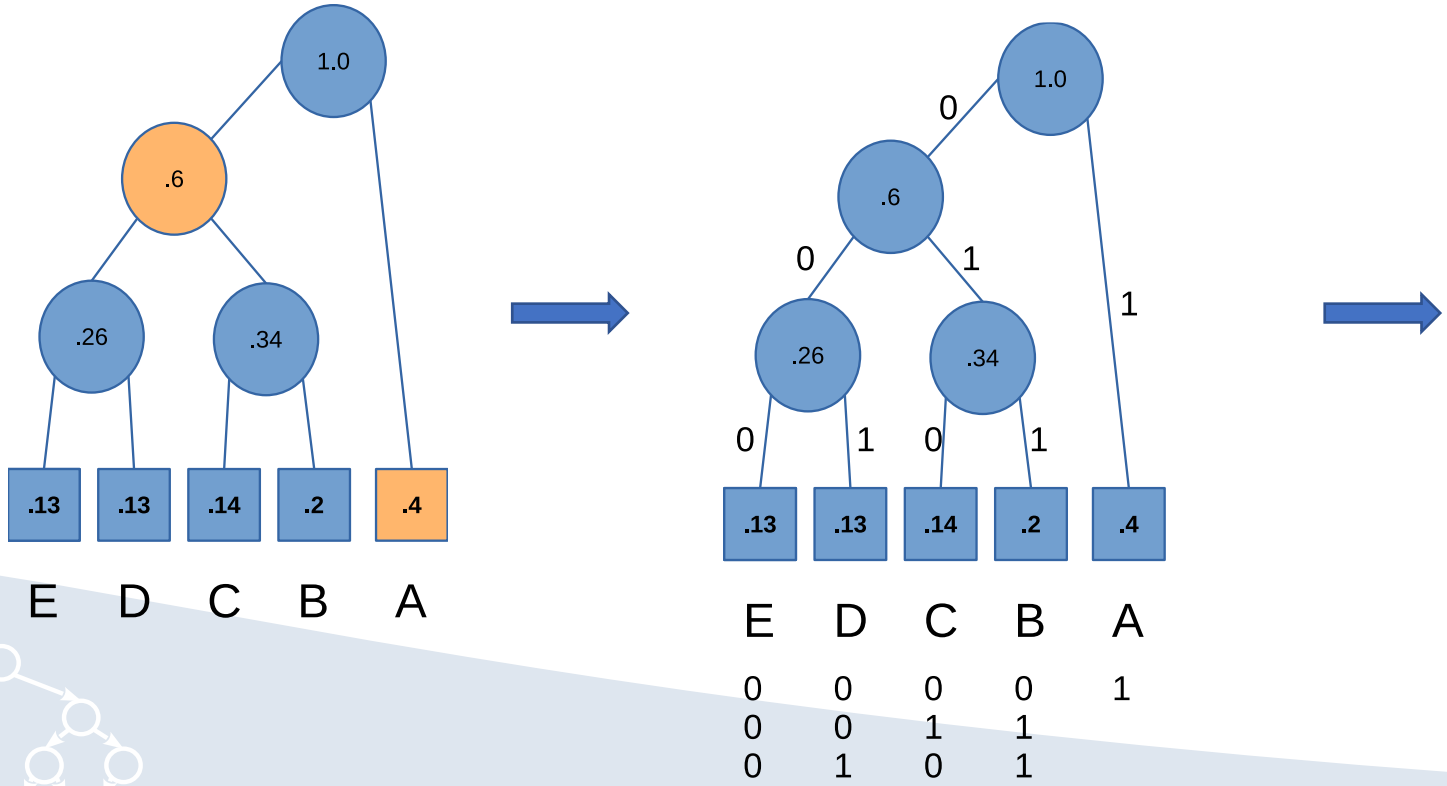
a) Izgradite Huffmanovu kodnu tablicu



Huffman – example a)

Prev. original alphabetS={A,B,C,D,E},probabilitiesP={0.4, 0.2, 0.14, 0.13, 0.13}

a) Build a Huffman code table

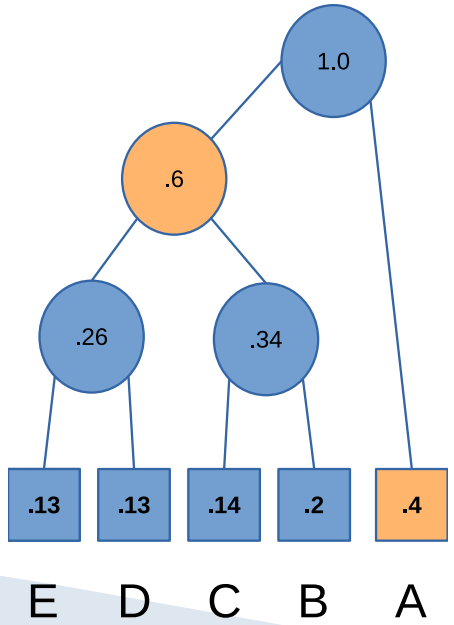


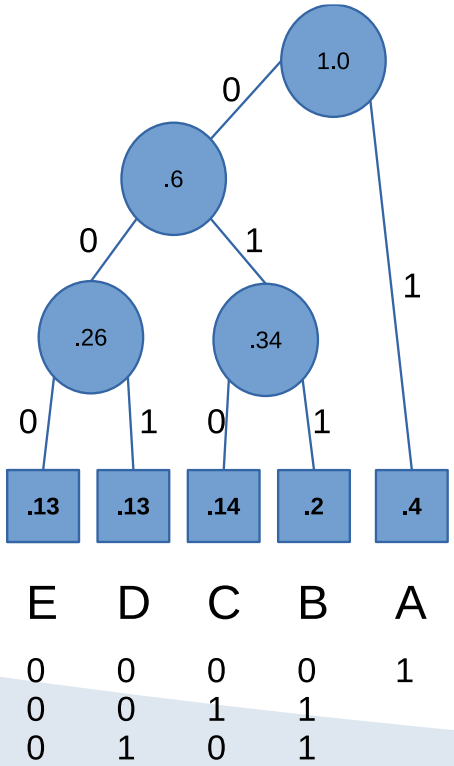
Symbol	Code word
AND	1
B	011
C	010
D	001
E	000

Huffman – primjer a)

Pretp. izvorni alfabet S={A,B,C,D,E}, vjerojatnosti P={0.4, 0.2, 0.14, 0.13, 0.13}

a) Izgradite Huffmanovu kodnu tablicu





Simbol	Kodna riječ
A	1
B	011
C	010
D	001
E	000

Huffman - example b)

Prev. original alphabetS={A,B,C,D,E},probabilitiesP={0.4, 0.2, 0.14, 0.13, 0.13}

b) Encode the input string ABCABCABCE

Symbol	Code word
AND	1
B	011
C	010
D	001
E	000

80 bits for an ASCII input string

Encoded string:
1 011 010 1 011 010 1 011 010 000

24bits for our encoded string

- Table size: 5*8+13=53 bits

Encoded data = string + table = 77 bits

Compression factor
80/77=**1.04**

Huffman – primjer b)

Pretp. izvorni alfabet `S={A,B,C,D,E}`, vjerojatnosti `P={0.4, 0.2, 0.14, 0.13, 0.13}`

b) Enkodirajte ulazni niz ABCABCABCE

Simbol	Kodna riječ
A	1
B	011
C	010
D	001
E	000

80 bitova za ASCII ulazni niz

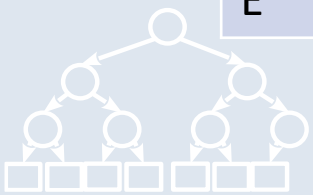
Enkodirani niz:
1 011 010 1 011 010 1 011 010 000

24bita za naš enkodirani niz

- Veličina tablice: 5*8+13=53bita

Enkodirani podatci= niz + tablica = 77bitova

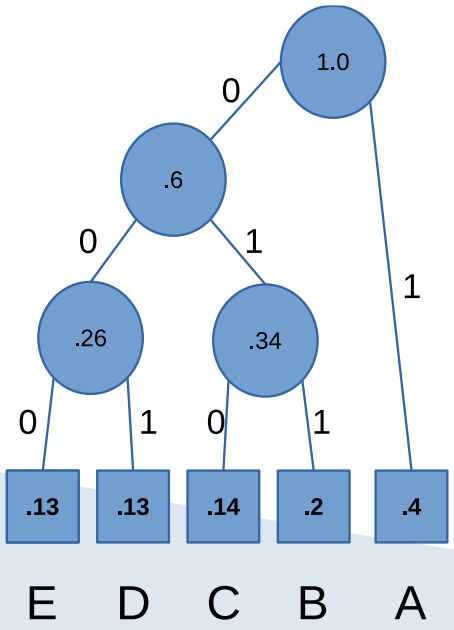
Faktor kompresije
80/77=**1.04**



Huffman - example c)

Prev. original alphabetS={A,B,C,D,E},probabilitiesP={0.4, 0.2, 0.14, 0.13, 0.13}

c) Decode the incoming stream 01110111

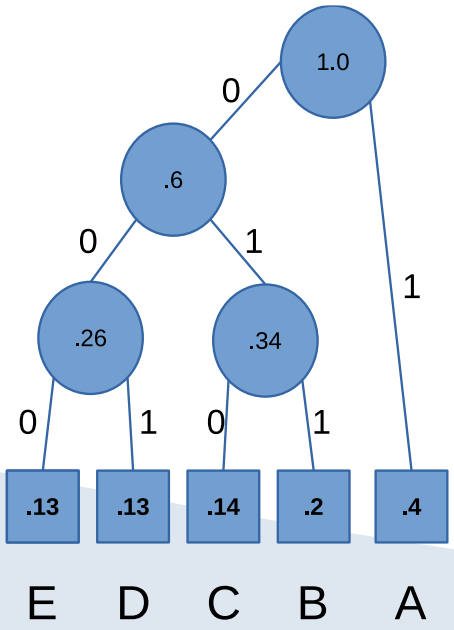


Decoded:**BABA**

Huffman – primjer c)

Pretp. izvorni alfabet `S={A,B,C,D,E}`, vjerojatnosti `P={0.4, 0.2, 0.14, 0.13, 0.13}`

c) Dekodirajte dolazni tok 01110111



Dekodirano: **BABA**

Huffman - problems

- Quantization
  - Best code for individual symbols, but not the set S
  - Improvement: Arithmetic coding

- Permutational invariance - improvements
  - Input transformations
    - Block encoding – groups of characters become symbols
  - Approaching Kolmogorov complexity

Huffman - problemi

- Kvantizacija
  - Najbolji kod za individualne simbole, ali ne i skup S
  - Poboljšanje: Aritmetičko kodiranje

- Permutacijska invarijantnost - poboljšanja
  - Transformacije ulaza
    - Blokovsko kodiranje – grupe znakova postaju simboli
  - Približavanje Kolmogorovljevoj složenosti

Dictionary-based methods

- Automatic stacking of groupings for block encoding

- Dictionary!
  - Static
  - Dynamic

- It transforms the input stream into a smaller and more compressible one

Metode zasnovane na riječniku

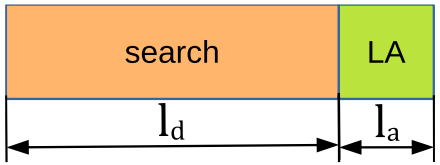
- Automatsko slaganje grupiranja za blokovsko enkodiranje
- Riječnik!
  - Statički
  - Dinamički
- Transformira ulazni tok u manji i prikladniji za komprimiranje

Lempel-Ziv77

- Dynamic dictionary

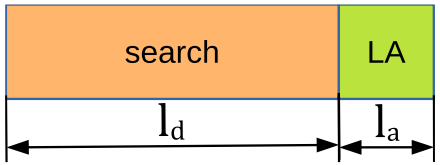
- Sliding window
  - Finding the largest overlap (greedy)

- Tank (buffer) of two parts
  - Dictionary container (for searching)
    - thousands of characters (usually powers of 2)
  - Lookahead
    - tens of characters



Lempel-Ziv77

- Dinamički riječnik
- Kližući prozor
  - Traženje najvećeg preklapanja (pohlepno)
- Spremnik (buffer) od dva dijela
  - Riječnički spremnik (za pretraživanje)
    - tisuće znakova (obično potencije broja 2)
  - Unaprijedni spremnik (lookahead)
    - desetci znakova



LZ77 encoder

- Input: I – input array, L – lookahead container, D – dictionary container
- LZ77Encode(I,L,D):
  1. L=first l_{And}characters within I
  2. Initialize *D* copies *L[0]*
  3. while *L* is non-empty:
     1. Find the longest prefix *p* from *L* which begins within *D*
     2. *i* = position *p* relatively from the end *D*, *j*=length of *p*, *k*=first next character after *p* in L
     3. Output *(i,j,k)*
     4. Left shift *j+1* signs through the composition ***D, L, I***. Consumes characters from ***AND***

LZ77 enkoder

- Ulaz: I – ulazni niz, L – lookahead spremnik, D – dictionary spremnik
- LZ77Encode(I,L,D):
  1. L=prvih l_a znakova unutar I
  2. Inicijaliziraj *D* kopijama *L[0]*
  3. while *L* is non-empty:
     1. Nađi najdulji prefiks *p* od *L* koji počinje unutar *D*
     2. *i*=pozicija *p* relativno od kraja *D*, *j* = duljina od *p*, *k*=prvi sljedeći znak iza *p* u L
     3. Output *(i,j,k)*
     4. Lijevi pomak *j+1* znakova kroz kompoziciju ***D,L,I***. Konzumira znakove iz ***I***

LZ77 decoder

- Input: E – encoded string, L – lookahead container, D – dictionary container, B – container (connection D,L)
- LZ77Decode(E,L,D):
  1. L=first *l*And characters within E
  2. Initialize *D* copies *L[0]*
  3. while *E* is non-empty:
     1. Consume the first triplet (i,j,k) from E
     2. *Word = B[i:i+j]+k*
     3. Output *(word)*
     4. Left shift Ba for *len(word)+1*, put the word at the end of B

LZ77 dekoder

- Ulaz: E – enkodirani niz, L – lookahead spremnik, D – dictionary spremnik, B – spremnik (spoj D,L)
- LZ77Decode(E,L,D):
  1. L=prvih l_a znakova unutar E
  2. Inicijaliziraj *D* kopijama *L[0]*
  3. while *E* is non-empty:
     1. Konzumiraj prvi triplet (i,j,k) iz E
     2. *Riječ = B[i:i+j]+k*
     3. Output *(riječ)*
     4. Lijevi pomak B-a za *len(riječ)+1*, stavi riječ na kraj B

LZ77 – example encoding

Encode the input string ABCABCABCE using LZ77 and calculate the compression factor. Let the tank sizes be $l_{And}=4$, $l_d=4$.

container (4 dict;4LA)	entrance	Exit
	ABCABCABCE	AND
AAAAND ;ANDBCA	BCABCE	(0,1,B)
AAAB; CABC	ABC	(0,0,C)
ANDABC; ABC	BCE	(2,3,A)
ANDBC AND;BCE		(2,2,E)
ABC;		

- Size
  encoded output:
  8+4*12=56 bits

- Compression factor:
  80/56=**1.43**

LZ77 – primjer enkodiranje

Enkodirajte ulazni niz ABCABCABCE koristeći LZ77 i izračunajte faktor kompresije. Veličine spremnika neka su l_a=4, l_d=4.

spremnik (4 dict;4LA)	ulaz	izlaz
	ABCABCABCE	A
AAAA;ABCA	BCABCE	(0,1,B)
AAAB;CABC	ABCE	(0,0,C)
AABC;ABCA	BCE	(2,3,A)
ABCA;BCE		(2,2,E)
ABCE;		

- Veličina enkodiranog izlaza: 8+4*12=56bitova
- Faktor kompresije: 80/56=**1.43**

LZ77 – example - decoding

Decode the string A(0,1,B)(0,0,C)(2,3,A)(2,2,E). The tank sizes are l_{And} =4, l_d=4.

entrance	container (4 dict; 4LA)	Exit	Move tank
AND	AAAA;		AAAA;
(0,1,B)	AAAA; AB	AB	AAAB;
(0,0,C)	AAAB; C	C	AABC;
(2,3,A)	AABC; ABCA	ABC	ABCA;
(2,2,E)	ABCA; BCE	BCE	ABC;
	ABC;		

LZ77 – primjer - dekodiranje

Dekodirajte niz A(0,1,B)(0,0,C)(2,3,A)(2,2,E). Veličine spremnika su l_a=4, l_d=4.

ulaz	spremnik (4 dict;4LA)	izlaz	Pomaknuti spremnik
A	AAAA;		AAAA;
(0,1,B)	AAAA;AB	AB	AAAB;
(0,0,C)	AAAB;C	C	AABC;
(2,3,A)	AABC;ABCA	ABCA	ABCA;
(2,2,E)	ABCA;BCE	BCE	ABCE;
	ABCE;		

LZ77 + Huffman

- **evil**program library
  - DEFLATE algorithm (LZ77 variant + Huffman + tricks)
  - Linux, MacOS, iOS
  - PS3, PS4, Xbox, Wii, iPhone
  - Optimized versions: Intel, CloudFlare
  - Good analysis

LZ77 + Huffman

- **zlib** programska knjižnica
  - DEFLATE algoritam (LZ77 varijanta + Huffman + trikovi)
  - Linux, MacOS, iOS
  - PS3, PS4, Xbox, Wii, iPhone
  - Optimirane verzije: Intel, CloudFlare
  - Dobra analiza