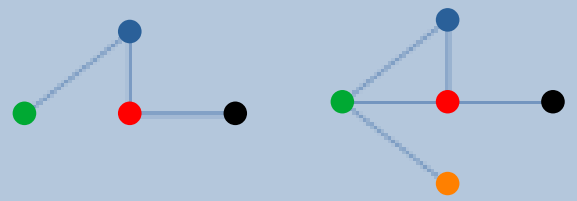
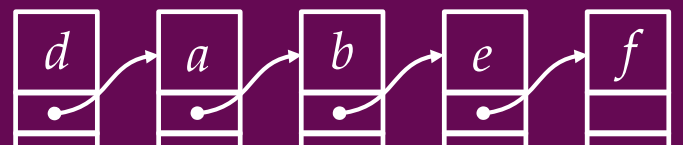
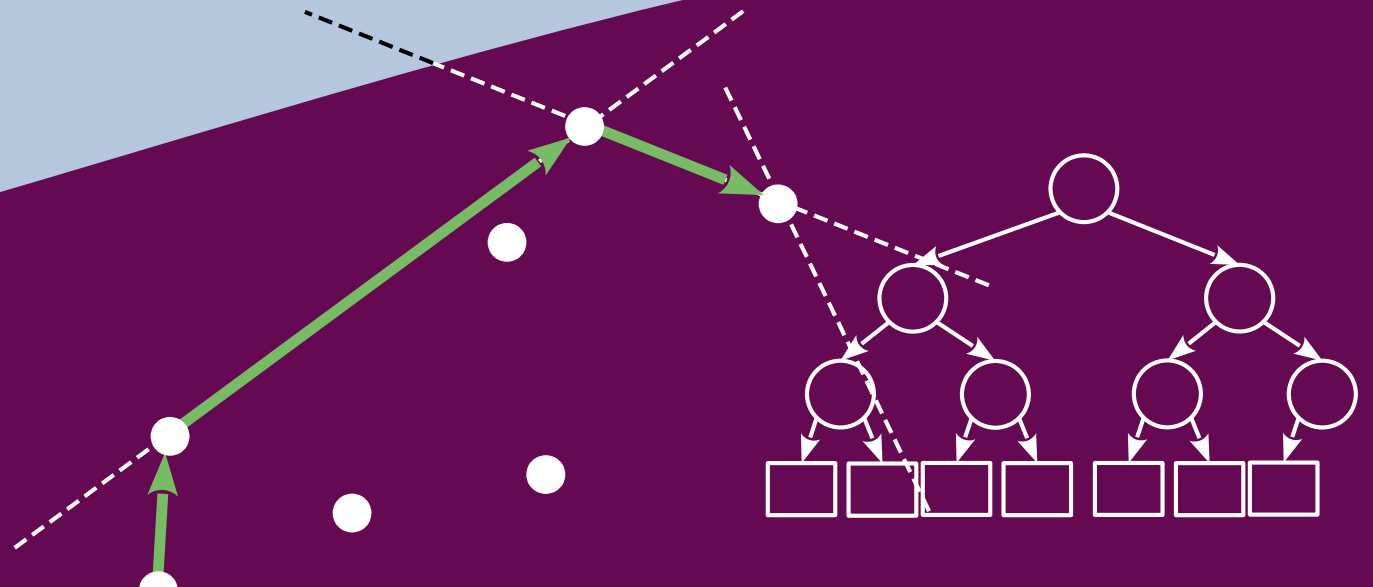
Napredni algoritmi i strukture podataka

Tjedan 7: Kompresijski algoritmi







Sadržaj

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

Kompresijski algoritmi

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

Teorijska pozadina

- Dva pristupa

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

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

SIT

- Jednostavna i algoritmi manje složenosti

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

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 – 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!

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**
  - Binarni kod $C=\{c_1,\ldots,c_n\}$ duljina $\{l_1,\ldots,l_n\}$ jest prefix-free ako i samo ako

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

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+ decimala)

FYI: trenutni (Kolovoz 2021.) rekord $62.8*10^{12}$ znamenki

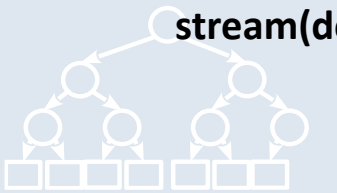
- **SIT nekompresibilno**
  - Prolazi sve testove slučajnosti

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

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

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

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()

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

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 - 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

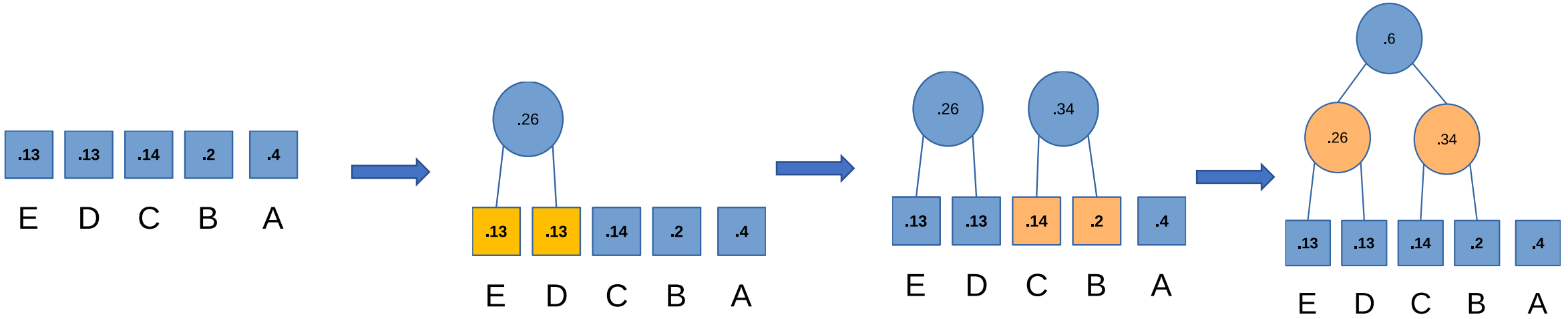
b) Enkodirajte ulazni niz ABCABCABCE i izračunajte faktor kompresije

c) Dekodirajte dolazni tok 01110111

Huffman – primjer a)

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

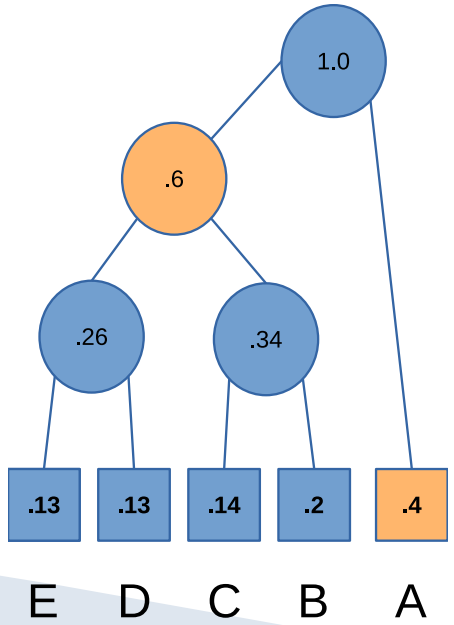
a) Izgradite Huffmanovu kodnu tablicu

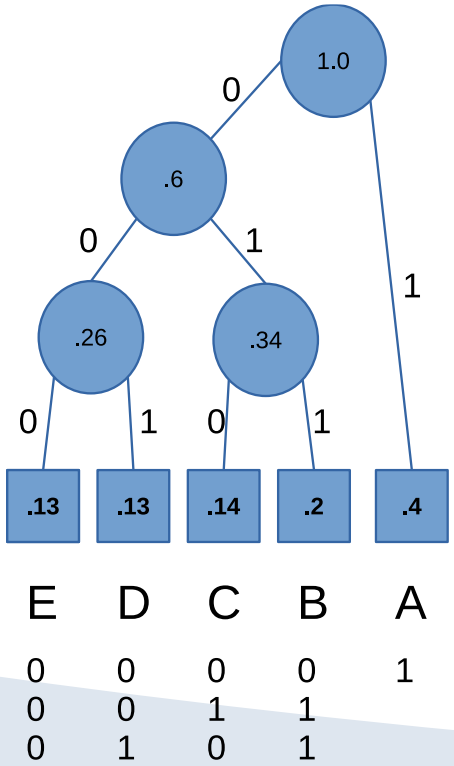


Huffman – primjer a)

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 – 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

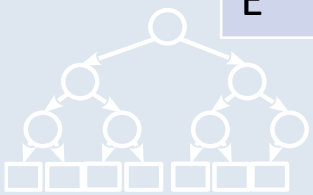
Enkodirani niz:
1 011 010 1 011 010 1 011 010 000

24bita za naš enkodirani niz

- Veličina tablice: 5*8+13=53bita

Enkodirani podatci= niz + tablica = 77bitova

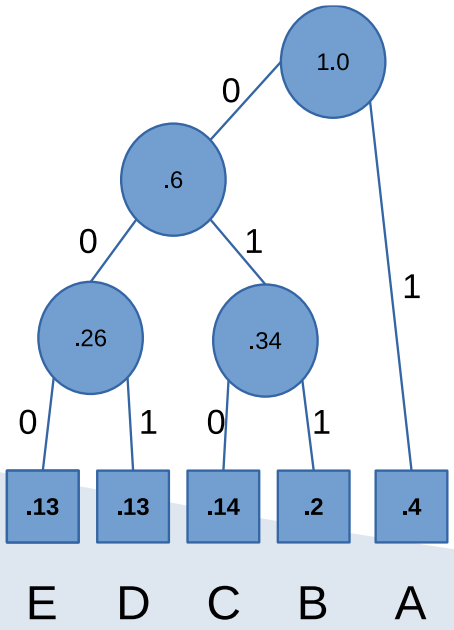
Faktor kompresije
80/77=**1.04**



Huffman – 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 - problemi

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

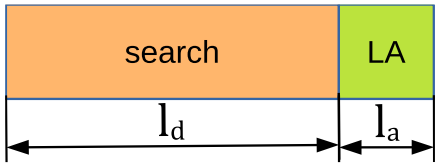
- Permutacijska invarijantnost - poboljšanja
  - Transformacije ulaza
    - Blokovsko kodiranje – grupe znakova postaju simboli
  - Približavanje Kolmogorovljevoj složenosti

Metode zasnovane na riječniku

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

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 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],* Output*(L[0])*
  3. while *L* is non-empty:
     1. Nađi najdulji prefiks *p* od *L* koji počinje unutar *D* *//spec. slučajevi*
     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. Pomak kompozicije ***D,L,I*** za *j+1* znakova udesno. Konzumira znakove iz ***I***

LZ77 dekoder

- Ulaz: E – enkodirani niz, L – lookahead spremnik, D – dictionary spremnik, B – spremnik (spoj D,L)
- LZ77Decode(E,L,D):
  1. Inicijaliziraj *D* kopijama konzumiranog *E[0]*
  2. while *E* is non-empty:
     1. Konzumiraj prvi triplet (i,j,k) iz E
     2. *start=l_d-i, end=start+j, riječ = B[start:end]+k* *//python "slice" notacija*
     3. Output *(riječ)*
     4. Pomak B-a udesno za *len(riječ)+1*
     5. *B = B+riječ* *//konkatenacija*

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**

Iznimni slučajevi:
- Nema poklapanja
- Poklapanje cijelog LA

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

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