# Algoritmi in podatkovne strukture 2

## Požrešni algoritmi

Luka Fürst

# Požrešni algoritmi

- Eden od možnih pristopov k reševanju optimizacijskih problemov
  - poišči zaporedje odločitev, ki optimizira (maksimizira ali minimizira) določeno količino
- Požrešni algoritmi so »kratkovidni«
  - na vsakem koraku se odločijo za tisto možnost, ki je po nekem kriteriju v tistem trenutku optimalna
- Lahko vse optimizacijske probleme rešujemo na požrešen način?

# Požrešni algoritmi

- Marsikaterega problema ne moremo rešiti s požrešno metodo
- Nekatere lahko, a je treba izbrati »pravi« požrešni pristop
  - včasih imamo več možnih požrešnih strategij
- Včasih je težko dokazati pravilnost požrešne metode
  - Možnost 1: dokažemo, da požrešni pristop v vsakem trenutku izvajanja »korak pred« vsemi drugimi pristopi (po vsaki iteraciji proizvede vsaj tako dobro (delno) rešitev kot vsi ostali)
  - Možnost 2: pričnemo s poljubno optimalno rešitvijo in jo brez izgube kakovosti preoblikujemo v požrešno rešitev

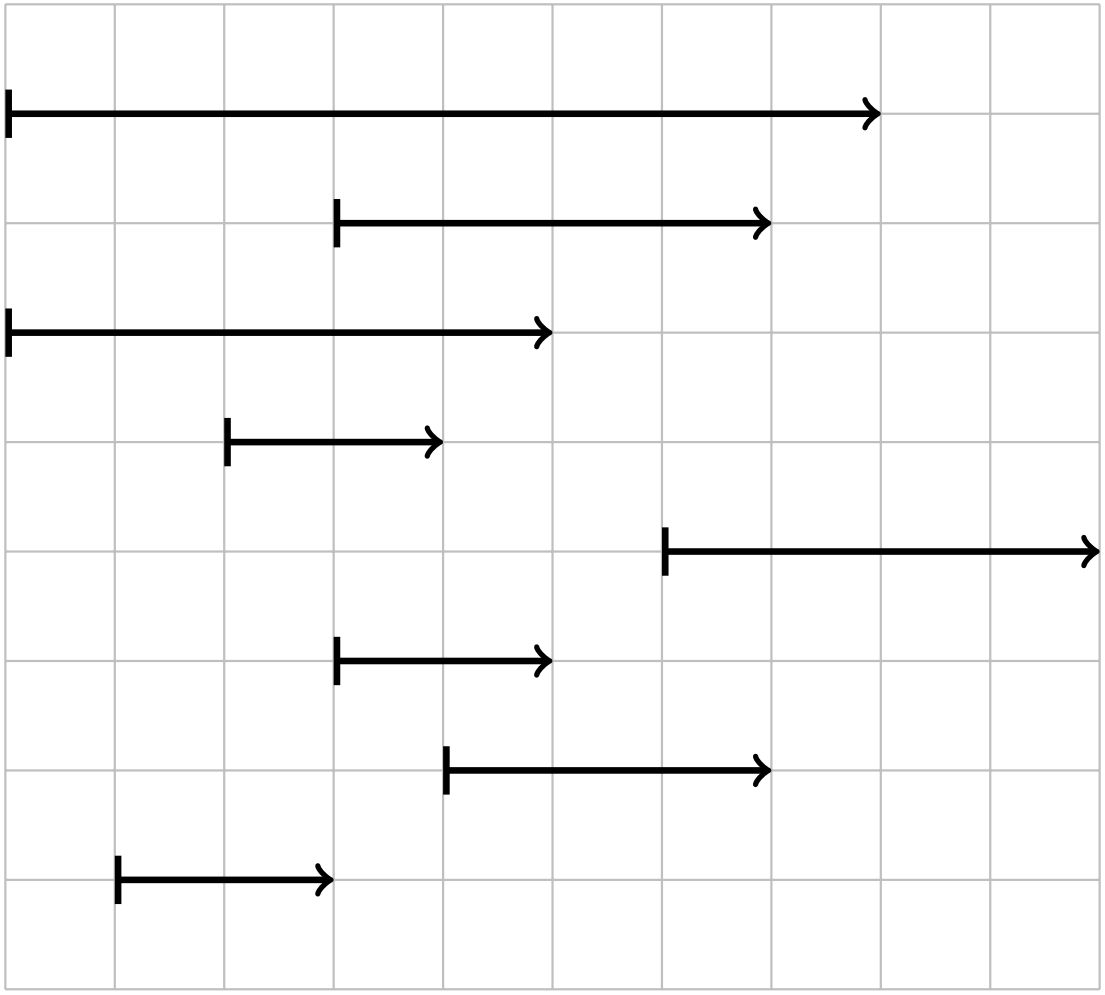
# Primeri požrešno rešljivih problemov

- Minimalno vpeto drevo
  - Primov in Kruskalov algoritem
- Najkrajše poti v grafu z nenegativnimi cenami povezav
  - Dijkstrov algoritem
- Izbira intervalov
- Huffmanovo kodiranje
- Zamenjevalna strategija predpomnilnika
- ...

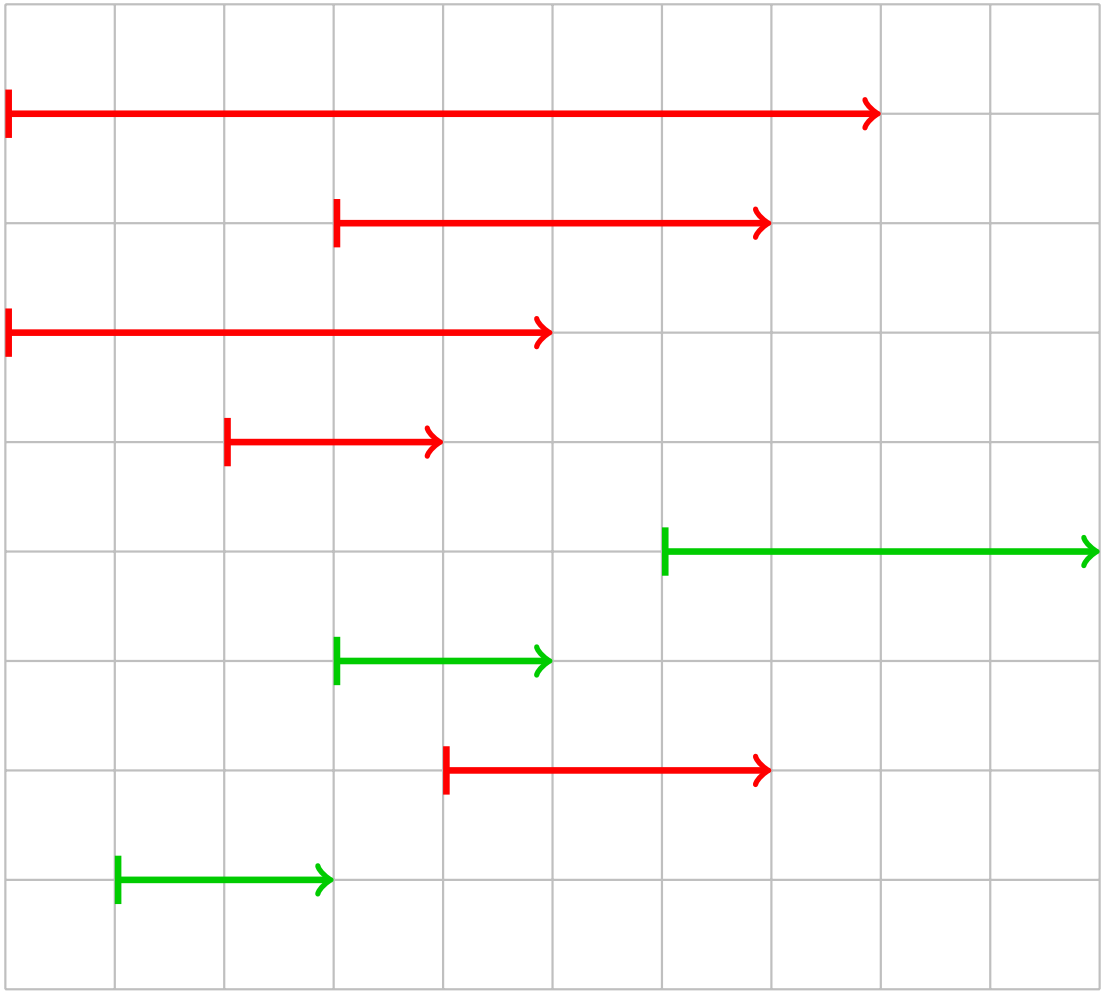
# Problem 1: Izbira intervalov

- Podanih je $n$ polodprtih intervalov $[s_i, f_i)$
  - $s_i$: začetni čas
  - $f_i$: končni čas
- Radi bi izbrali karseda veliko intervalov, ki se med seboj ne prekrivajo
- Praktični primer
  - predavanja v fiksnih terminih
  - ena sama predavalnica
  - na program želimo uvrstiti čimveč predavanj

# Izbira intervalov



# Izbira intervalov



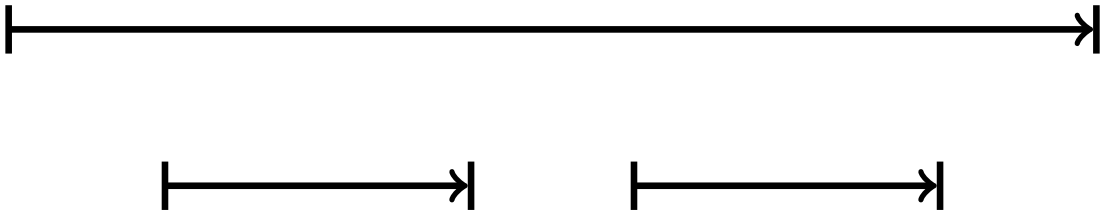
# Osnovna ideja

- Intervale obravnavamo v določenem vrstnem redu
- Na vsakem koraku upoštevamo le intervale, ki so združljivi z že izbranimi intervali
  - interval $i$ je združljiv z intervalom $j \iff$ $s_j \geq f_i \vee s_i \geq f_j$
- Kateri interval izbrati (med vsemi dopustnimi)?
  - obstaja več požrešnih strategij!

# Izberi interval, ki se prvi začne?

# Izberi interval, ki se prvi začne?

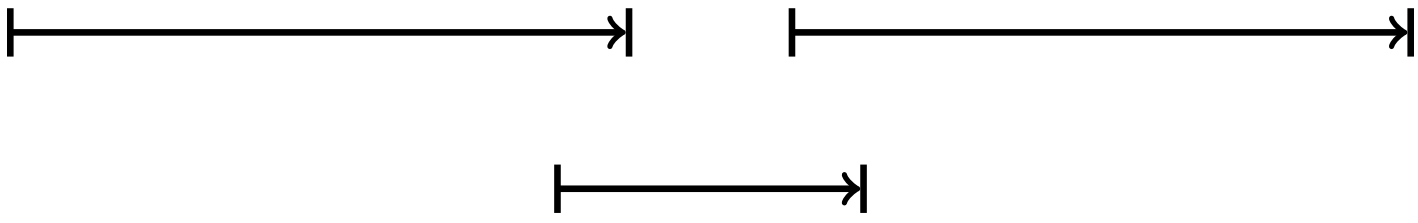
- Ne!



# Izberi najkrajši interval?

# Izberi najkrajši interval?

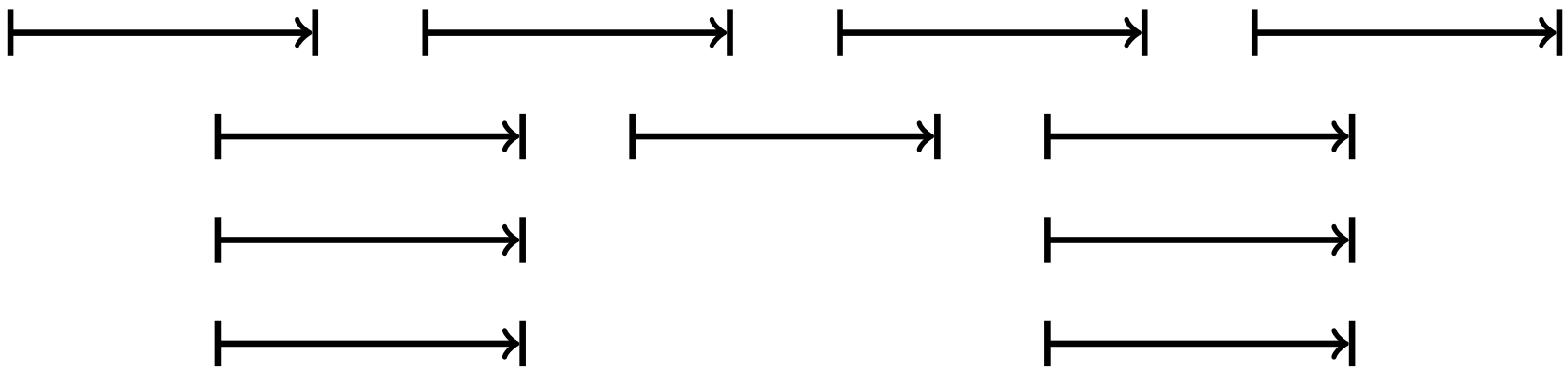
- Ne!



# Izberi interval z najmanj konflikti?

# Izberi interval z najmanj konflikti?

- Ne!



# Izberi interval, ki se prvi konča?

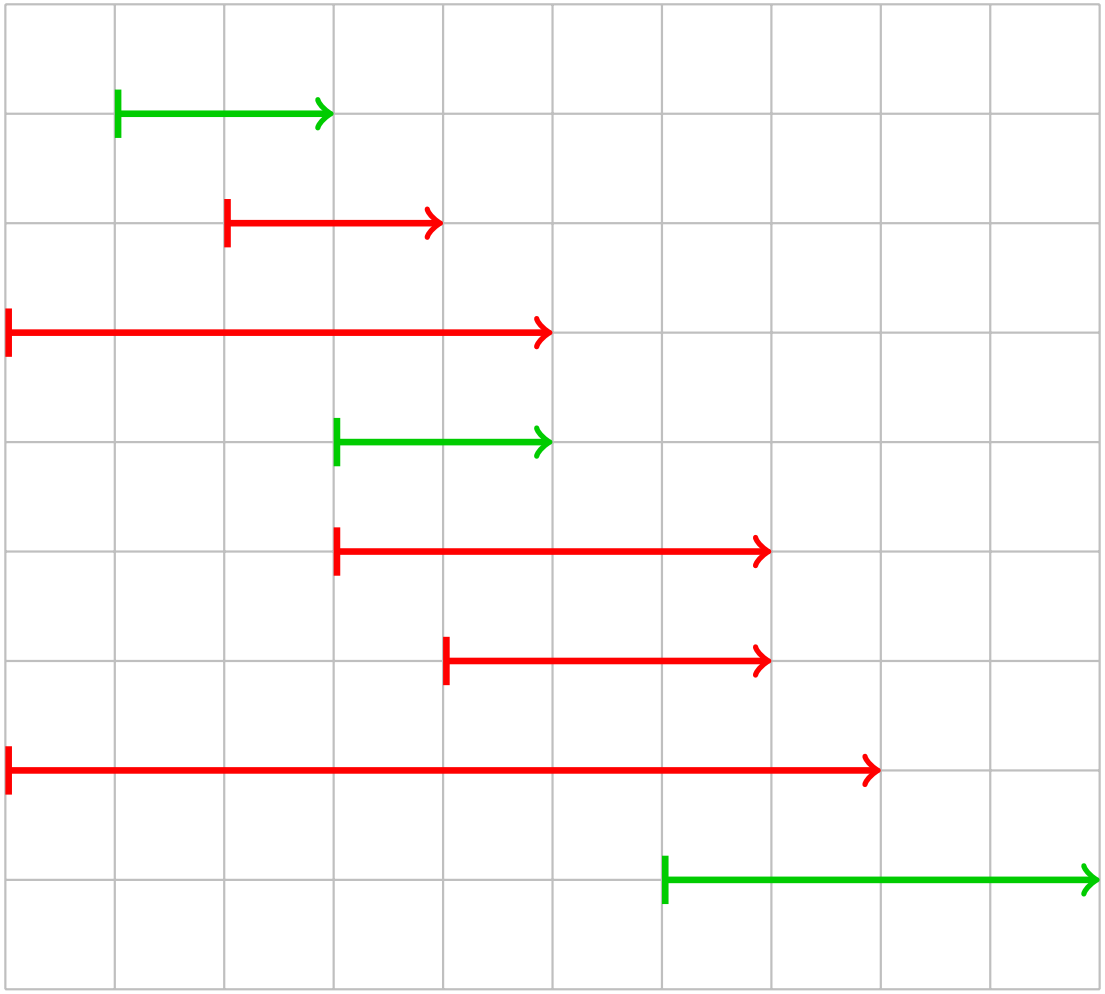
# Izberi interval, ki se prvi konča?

- Na ta način se bo »prostor« najhitreje sprostil
- Sliši se smiselno, toda . . .
- . . . ali znamo to dokazati?

# Algoritem

**function** IZBIRA-INTERVALOV(intervali)
    Uredi intervale po naraščajočih časih zaključka
    ▷ *Dobimo zaporedje* $\langle [s_1, f_1), \ldots, [s_n, f_n) \rangle$
    $\mathcal{A} \leftarrow \emptyset$
    $t \leftarrow 0$
    **for** $i \leftarrow 1 : n$ **do**
        **if** $s_i \geq t$ **then**
            $\mathcal{A} \leftarrow \mathcal{A} \cup [s_i, f_i)$
            $t \leftarrow f_i$
    **return** $\mathcal{A}$

# Primer



# Dokaz pravilnosti in optimalnosti

- Intervali v množici $\mathcal{A}$, ki jo zgradi algoritem, so po konstrukciji medsebojno združljivi
- Je množica $\mathcal{A}$ tudi maksimalna?
- Recimo, da je optimalna neka druga množica (npr. $\mathcal{O}$)
- Dokažimo, da velja $|\mathcal{A}| = |\mathcal{O}|$

# Optimalnost množice $\mathcal{A}$

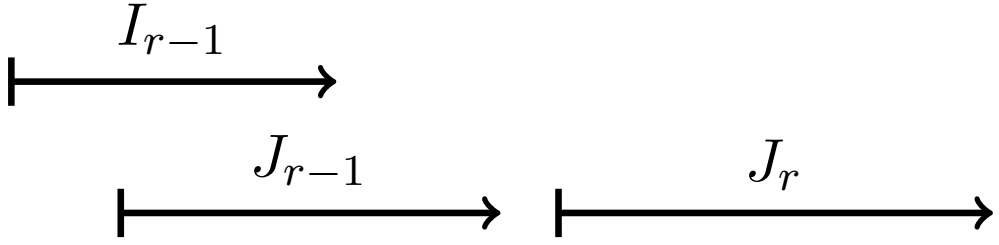
- Naj bo $\mathcal{A} = \{I_1, \ldots, I_k\}$ in $\mathcal{O} = \{J_1, \ldots, J_m\}$
- Recimo, da so intervali $I_1$, $\ldots$, $I_k$ in $J_1$, $\ldots$, $J_m$ urejeni po naraščajočih časih zaključka
- Dokažimo, da je $k = m$

# Optimalnost množice $\mathcal{A}$

- Naj bo $f(I)$ čas zaključka intervala $I = [s_i, f_i)$
  - $f(I) = f_i$
- Trditev: za vsak $r \leq k$ velja $f(I_r) \leq f(J_r)$
- Dokaz z indukcijo
- Za $r = 1$ trditev drži
  - interval $I_1$ ima po konstrukciji najmanjši čas zaključka
- Predpostavimo, da trditev velja za $r - 1$
- Pokažimo, da trditev velja tudi za $r$

# Optimalnost množice $\mathcal{A}$

- Po induktivni predpostavki: $f(I_{r-1}) \leq f(J_{r-1})$
- $I_r$ je združljiv z intervali $I_1, \ldots, I_{r-1}$
- $J_r$ je združljiv z intervali $J_1, \ldots, J_{r-1}$
- Ker je $f(I_{r-1}) \leq f(J_{r-1})$, je $J_r$ združljiv tudi z intervali $I_1, \ldots, I_{r-1}$
- Naš algoritem ima torej (v najslabšem primeru) vedno možnost, da za $I_r$ izbere kar interval $J_r$




- Zato je $f(I_r) \leq f(J_r)$

# Optimalnost množice $\mathcal{A}$

- Algoritem proizvede $\mathcal{A} = \{I_1, \ldots, I_k\}$
- Optimalna množica je $\mathcal{O} = \{J_1, \ldots, J_m\}$
- Ker je $f(I_r) \leq f(J_r)$ za vsak $r \leq k$, velja $f(I_k) \leq f(J_k)$
- Od tod sledi $k \geq m$
- Torej je množica $\mathcal{A}$ tudi optimalna

# Časovna zahtevnost

- $O(n \log n)$ za urejanje intervalov po času zaključka
- $O(n)$ za izbiranje urejenih intervalov
- $O(n \log n)$ za celoten algoritem

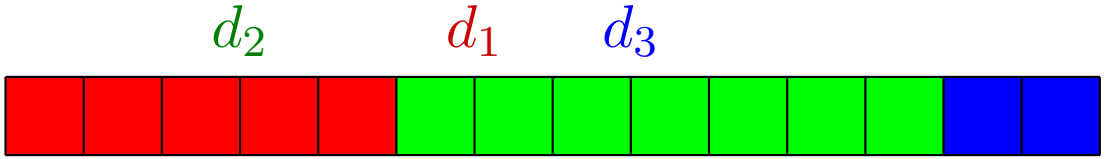
# Problem 2: Minimizacija maksimalne zamude

- Podanih je $n$ opravil
- Za vsako opravilo je podano trajanje $t_i$ in rok $d_i$
- Za vsako opravilo je treba določiti njegov začetek $s_i$
- Konec opravila $i$: $f_i = s_i + t_i$
- Zamuda opravila $i$: $\ell_i = f_i - d_i$
- Radi bi minimizirali maksimalno zamudo $\ell = \max_{i=1}^{n} \ell_i$

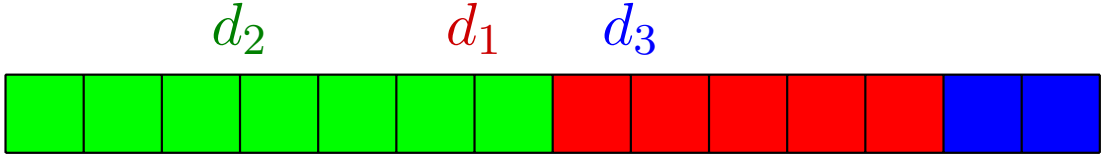
# Primer

| | Trajanje ($t_i$) | Rok ($d_i$) |
|---|---|---|
| Opravilo 1 | 5 | 6 |
| Opravilo 2 | 7 | 3 |
| Opravilo 3 | 2 | 8 |

- Razpored 1, 2, 3 ima maksimalno zamudo 9:




- Razpored 2, 1, 3 (optimalen) ima maksimalno zamudo 6:



# Rešitev

- Tudi tokrat pride v poštev več (napačnih) požrešnih strategij, pravilna pa je tale . . .
- Razvrsti opravila po naraščajočih rokih
- Dolžine opravil lahko povsem ignoriramo!

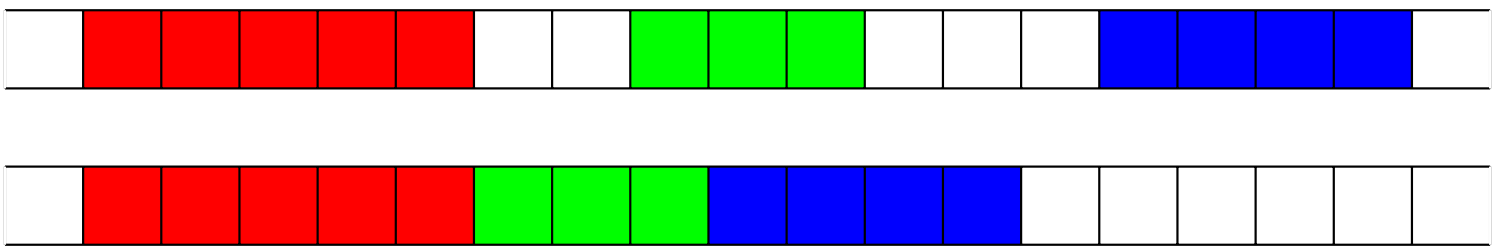
**function** RAZVRSTI-OPRAVILA(opravila)
    Uredi opravila po naraščajočih rokih $(d_i)$
    ▷ *Dobimo zaporedje* $\langle (t_1, d_1), \ldots, (t_n, d_n) \rangle$
    $s_1 \leftarrow 0$ ▷ $s_i$*: izračunani začetni čas* $i$*-tega opravila*
    **for** $i \leftarrow 2 : n$ **do**
        $s_i \leftarrow s_{i-1} + t_{i-1}$
    **return** $\langle s_1, s_2, \ldots s_n \rangle$

# Lastnosti razporeda, ki ga proizvede algoritem

- Lastnost 1: razpored je strnjen (nima »lukenj«)
- Lastnost 2: razpored nima inverzij
  - inverzija: situacija, ko je opravilo s kasnejšim rokom v razporedu pred opravilom z zgodnejšim rokom
- Ni nujno, da ima vsak optimalen razpored ti dve lastnosti
- Pokazali pa bomo, da lahko vsak optimalen razpored pretvorimo v enakovreden razpored s tema lastnostma

# Trditev 1: obstaja strnjen optimalni razpored

- To je jasno kot beli dan!
- »Lukenj« se vedno lahko znebimo in s tem kvečjemu pridobimo (zamude ne moremo povečati)



# Trditev 2: obstaja optimalni razpored brez inverzij

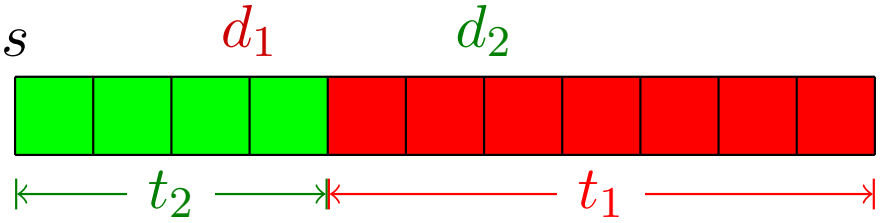
- Inverzijo imamo, ko opravilo $i$ z rokom $d_i < d_j$ v razporedu nastopa kasneje kot opravilo $j$ ($s_i > s_j$)
- Če imamo v razporedu vsaj eno inverzijo, lahko najdemo tudi zaporedni opravili z medsebojno inverzijo
  - Predstavljajmo si »skoraj« naraščajoče zaporedje $a_1$, $a_2$, ..., $a_n$
  - V takem zaporedju imamo inverzijo, če obstajata števili $a_i > a_j$ za $i < j$
  - Ker je zaporedje na indeksu $j$ na nižji točki kot na indeksu $i$, gotovo obstaja vsaj en indeks med $i$ in $j$, pri katerem se zgodi padec

# Trditev 2: obstaja optimalni razpored brez inverzij

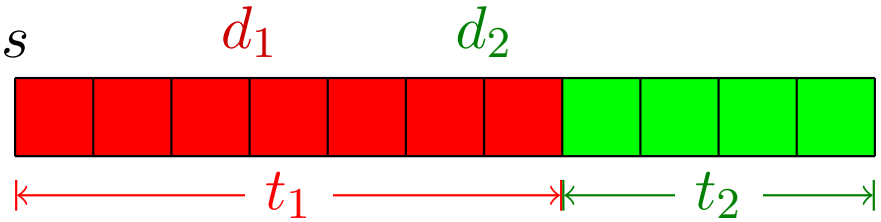
- Razpored z inverzijami lahko torej pretvorimo v razpored brez inverzij zgolj z medsebojnimi zamenjavami zaporednih opravil
  - Točno to počne urejanje z mehurčki — *bubblesort*!
- Ključno vprašanje: ali lahko s takšno pretvorbo povečamo maksimalno zamudo?

# Trditev 2: obstaja optimalni razpored brez inverzij

- Odgovor je ne
- Odstranitev inverzij kvečjemu zmanjša maksimalno zamudo




- $\ell_1 = s + t_2 + t_1 - d_1$, $\ell_2 = s + t_2 - d_2$




- $\ell'_1 = s + t_1 - d_1$, $\ell'_2 = s + t_1 + t_2 - d_2$
- $\ell'_1 < \ell_1$ (ker je $t_2 > 0$)
- $\ell'_2 < \ell_1$ (ker je $d_2 > d_1$)
- $\max(\ell'_1, \ell'_2) < \max(\ell_1, \ell_2)$

# Dokaz optimalnosti

- Naj bo $\mathcal{O}$ nek optimalen razpored (z minimalno maksimalno zamudo)
- Iz razporeda $\mathcal{O}$ odstranimo »luknje« in dobimo razpored $\mathcal{O}'$, ki ne more biti slabši kot $\mathcal{O}$ (zato je takisto optimalen)
- Iz razporeda $\mathcal{O}'$ odstranimo inverzije in dobimo razpored $\mathcal{O}''$, ki ne more biti slabši kot $\mathcal{O}'$ (zato je tudi ta optimalen)
- Razpored $\mathcal{O}''$ pa je točno tisto, kar proizvede naš algoritem

# Problem 3: Huffmanovo kodiranje

- Recimo, da imamo abecedo z $n$ znaki
- Vsak znak želimo zapisati z določenim številom bitov
- Enakomerno kodiranje
  - vsakemu znaku dodelimo isto število ($\lceil \lg n \rceil$) bitov
  - A $\mapsto 00000$, B $\mapsto 00001$, ..., Ž $\mapsto 11000$
- Neenakomerno kodiranje
  - različnim znakom lahko dodelimo različno število bitov
  - smiselno je pogostejše znake zapisati z manj, redkejše pa z več biti

# Povprečna bitna dolžina kodiranja

- Kodiranje
  - funkcija $\gamma\colon \Sigma \to \{0,1\}^*$
  - $\Sigma$ je abeceda z najmanj dvema znakoma
- Naj bo $f_c$ pogostost znaka $c \in \Sigma$
  - pogostost = delež pojavitev v besedilu
- Povprečna bitna dolžina kodiranja (*average bit length*)

$$\mathrm{ABL}(\gamma) = \sum_{c \in \Sigma} f_c |\gamma(c)|$$

# Povprečna bitna dolžina kodiranja

| $c$ | $f_c$ | $\gamma(c)$ |
|---|---|---|
| A | 0,25 | 01 |
| B | 0,17 | 111 |
| C | 0,08 | 1101 |
| D | 0,20 | 00 |
| E | 0,27 | 10 |
| F | 0,03 | 1100 |

$$\begin{aligned}\mathrm{ABL}(\gamma) &= 2 \cdot 0{,}25 + 3 \cdot 0{,}17 + 4 \cdot 0{,}08 \\ &\quad + 2 \cdot 0{,}20 + 2 \cdot 0{,}27 + 4 \cdot 0{,}03 \\ &= 2{,}39\end{aligned}$$

# Predponsko kodiranje

- Kodiranje, pri katerem nobena koda ni predpona druge
  - $c \neq d \implies \neg\exists w \in \Sigma^* : \gamma(c) = w\gamma(d)$
  - vsako enakomerno kodiranje je predponsko kodiranje
  - predponsko kodiranje imamo tudi v primeru s prejšnje prosojnice
- Predponsko kodiranje omogoča enostavno dekodiranje
  - po vrsti beremo bite
  - ko prebrani biti tvorijo kodo nekega znaka, vemo, da je to edini možni znak

# Problem

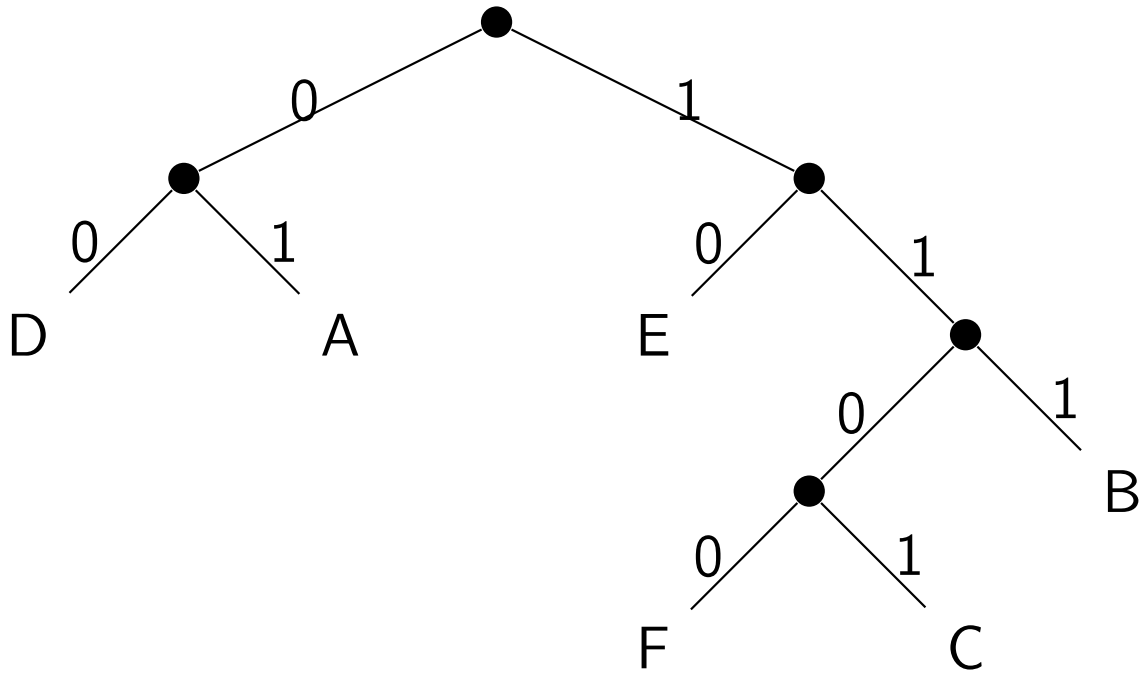
- Vhod
  - abeceda $\Sigma$
  - pogostost $f_c$ za vsak $c \in \Sigma$
- Izhod
  - predponsko kodiranje $\gamma \colon \Sigma \to \{0,1\}^*$ z najmanjšo povprečno bitno dolžino

# Predstavitev predponskega kodiranja z drevesom

- Predponsko kodiranje lahko predstavimo z dvojiškim drevesom
- Vsako levo vejo označimo z 0, desno pa z 1
- Zaporedje oznak na poti od korena navzdol predstavlja kodo
- Listi predstavljajo posamezne znake
  - če kodiranje ne bi bilo predponsko, bi lahko znaki pripadali tudi notranjim vozliščem

# Predstavitev predponskega kodiranja z drevesom

| $c$ | $\gamma(c)$ |
|---|---|
| A | 01 |
| B | 111 |
| C | 1101 |
| D | 00 |
| E | 10 |
| F | 1100 |

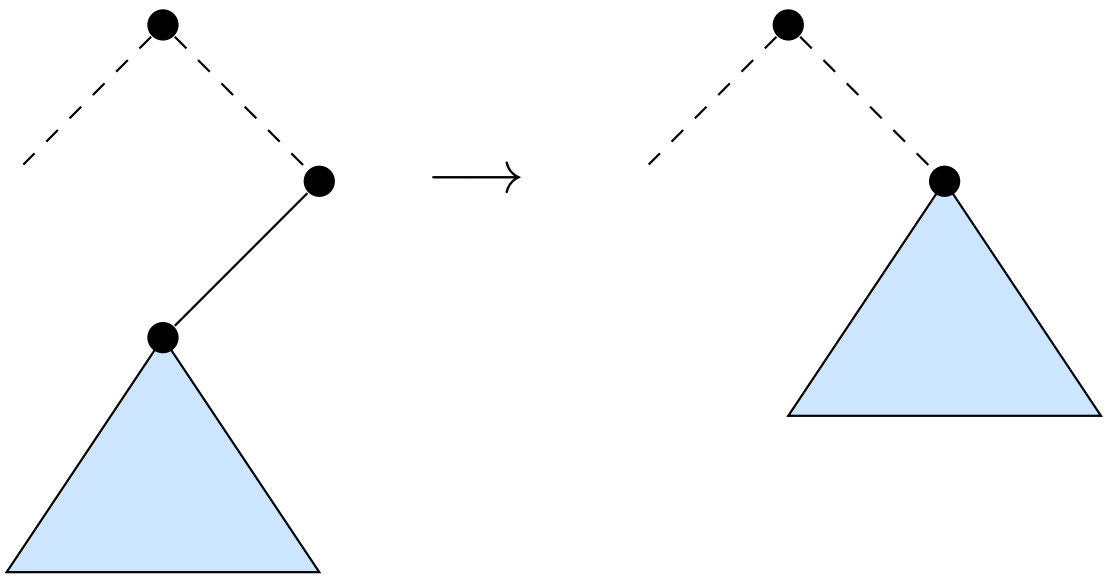


# Drugačna formulacija problema

- Globina lista = dolžina kode pripadajočega znaka
  - $d(c) = |\gamma(c)|$
- Povprečna bitna dolžina drevesa
  - = povprečna bitna dolžina kodiranja
  - $\mathrm{ABL}(T) = \sum_{c \in \Sigma} f_c d(c)$
- Poišči drevo z najmanjšo povprečno bitno dolžino!
- Problemski prostor je ogromen (vsa možna drevesa), k sreči pa obstaja učinkovit požrešni pristop

# Trditev 1

- Lahko se omejimo na polna (*full*) dvojiška drevesa
  - polno drevo = drevo, v katerem ima vsako vozlišče razen listov natanko dva otroka
- Drevo, ki vsebuje vozlišče z enim samim otrokom, lahko pretvorimo v drevo z manjšim $\mathrm{ABL}$



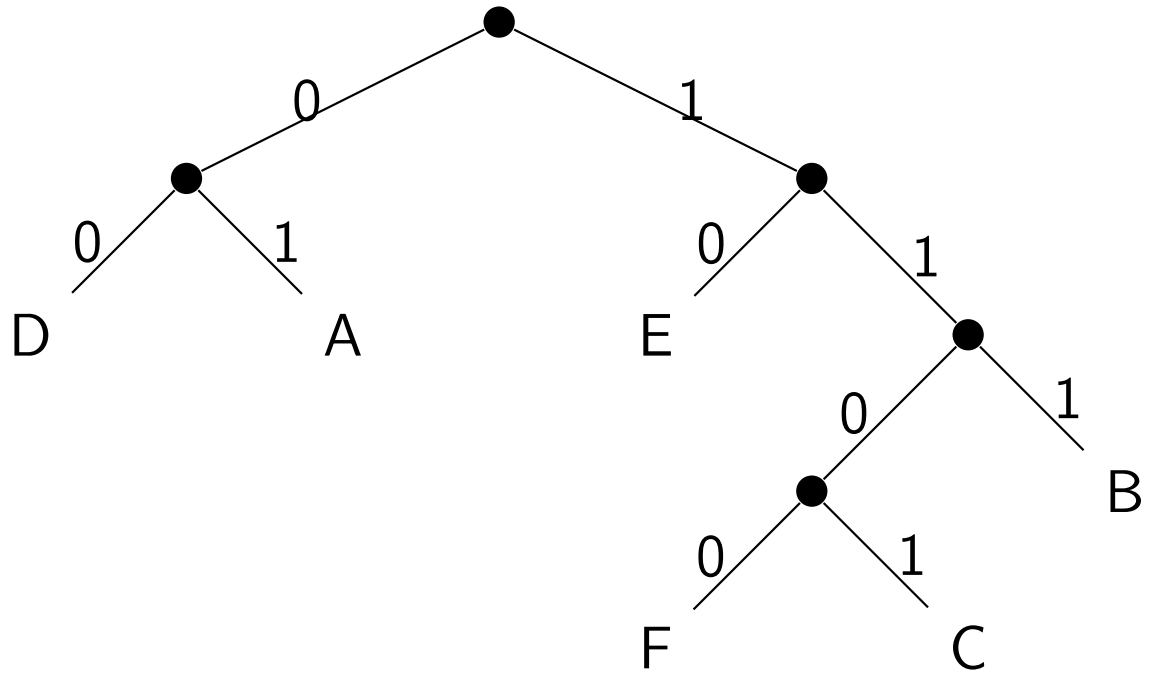
# Trditev 2

- Naj bosta $u$ in $v$ lista optimalnega drevesa
- Če je $d(u) < d(v)$, potem je $f_u \geq f_v$
- Če bi v domnevno optimalnem drevesu $T$ za lista $u$ in $v$ z $d(u) < d(v)$ veljalo $f_u < f_v$, potem bi lahko lista $u$ in $v$ med seboj zamenjali in dobili drevo $T'$ z manjšim $\mathrm{ABL}$

$$\begin{aligned}
\mathrm{ABL}(T') &= \mathrm{ABL}(T) + f_u d(v) + f_v d(u) - f_u d(u) - f_v d(v) \\
&= \mathrm{ABL}(T) + f_u(d(v) - d(u)) + f_v(d(u) - d(v)) \\
&= \mathrm{ABL}(T) + f_u(d(v) - d(u)) - f_v(d(v) - d(u)) \\
&= \mathrm{ABL}(T) + \underbrace{(f_u - f_v)}_{<0}\underbrace{(d(v) - d(u))}_{>0} \\
&< \mathrm{ABL}(T)
\end{aligned}$$

# Trditev 3

- Liste na isti globini lahko med seboj poljubno zamenjujemo




- Če med seboj zamenjujemo D, A in E ali F in C, se spreminja samo sestava kod, ne pa tudi njihova dolžina

# Trditev 4

- List $u$ na maksimalni globini ima brata $v$, ki je prav tako list na maksimalni globini
- $u$ ima brata ima zato, ker je dvojiško drevo polno
- Če $v$ ne bi ždel na maksimalni globini, potem bi to veljalo tudi za $u$

# Trditev 5

- Obstaja optimalno kodiranje, pri katerem znaka z dvema najmanjšima pogostostma pripadata bratoma, ki sta lista na maksimalni globini
- Vozlišči, ki pripadata znakoma z najmanjšima pogostostma, sta gotovo lista na največji globini
  - če vsaj za eno vozlišče to ne velja, ga zamenjamo z višje ležečim vozliščem, ki pripada znaku z manjšo pogostostjo, in dobimo drevo z manjšim $\mathrm{ABL}$
- Če omenjeni vozlišči nista brata, lahko z medsebojnimi zamenjavami vozlišč na maksimalni globini dosežemo, da postaneta brata

# Algoritem

**function** HUFFMANOVO-KODIRANJE($\Sigma$, $f$)
    Za vsak znak $a \in \Sigma$ ustvari nepovezano vozlišče $a$
    **while** $|\Sigma| > 1$ **do**
        $a$, $b \leftarrow$ znaka v $\Sigma$ z najmanjšima pogostostma
        $c \leftarrow ab$ ▷ *nov znak abecede, ki nadomešča $a$ in $b$*
        $f_c \leftarrow f_a + f_b$
        $\Sigma \leftarrow (\Sigma \setminus \{a, b\}) \cup \{c\}$
        Ustvari vozlišče $c$ in mu dodeli otroka $a$ in $b$
    Nazadnje dodano vozlišče razglasi za koren drevesa
    Nastalo drevo pretvori v kodiranje $\gamma$
    **return** $\gamma$

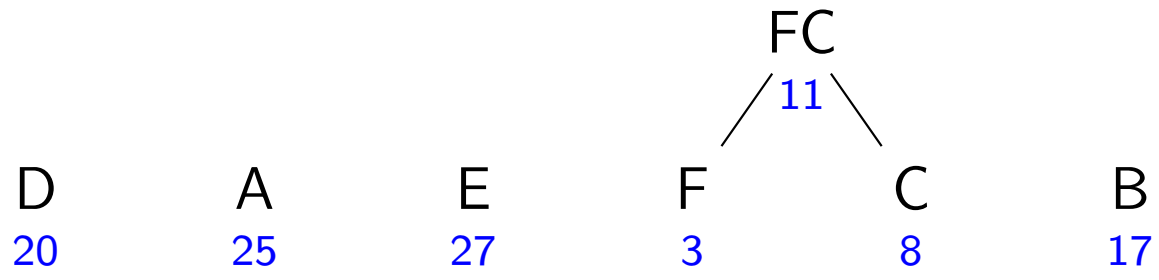
# Primer (pogostosti so v stotinah)

| $c$ | $f_c$ |
|---|---|
| A | 25 |
| B | 17 |
| C | 8 |
| D | 20 |
| E | 27 |
| F | 3 |

| D | A | E | F | C | B |
|---|---|---|---|---|---|
| 20 | 25 | 27 | 3 | 8 | 17 |

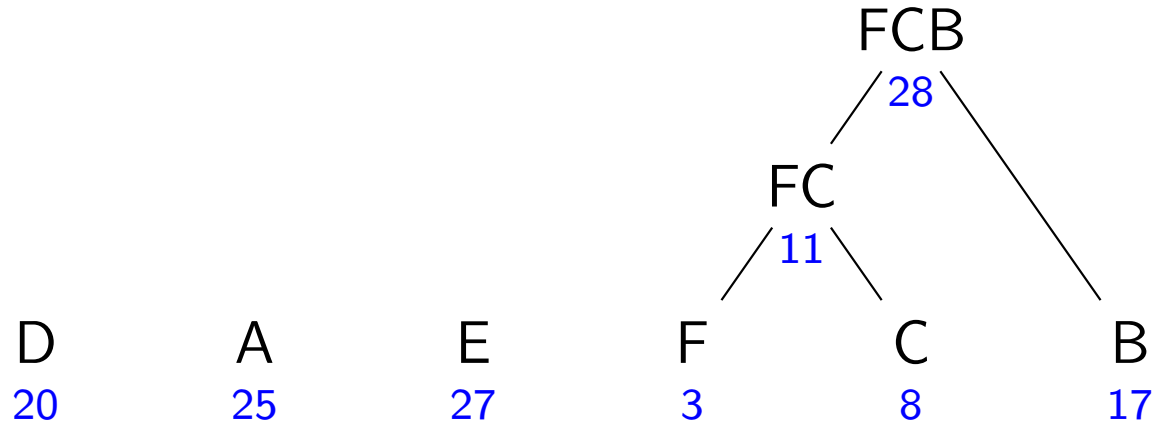
# Primer (pogostosti so v stotinah)

| $c$ | $f_c$ |
|---|---|
| A | 25 |
| B | 17 |
| FC | 11 |
| D | 20 |
| E | 27 |



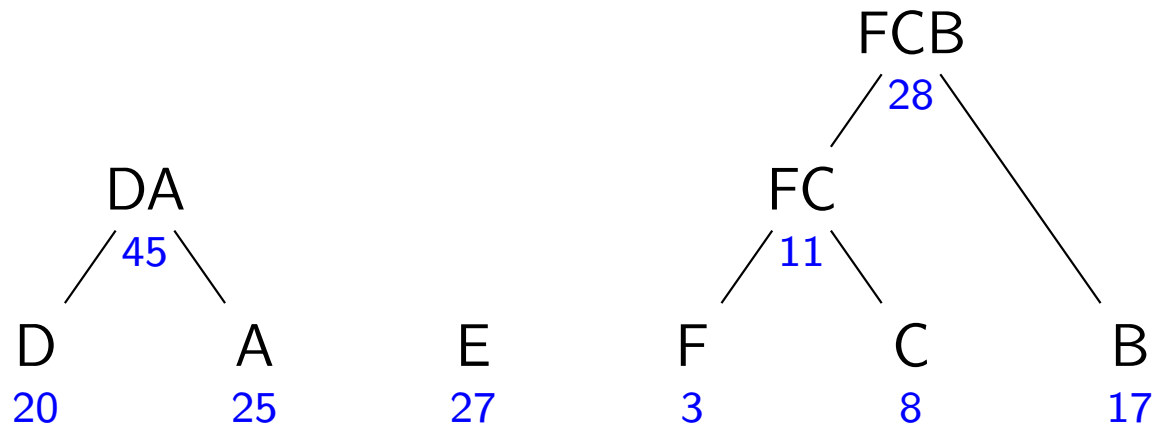
# Primer (pogostosti so v stotinah)

| $c$ | $f_c$ |
|---|---|
| A | 25 |
| FCB | 28 |
| D | 20 |
| E | 27 |



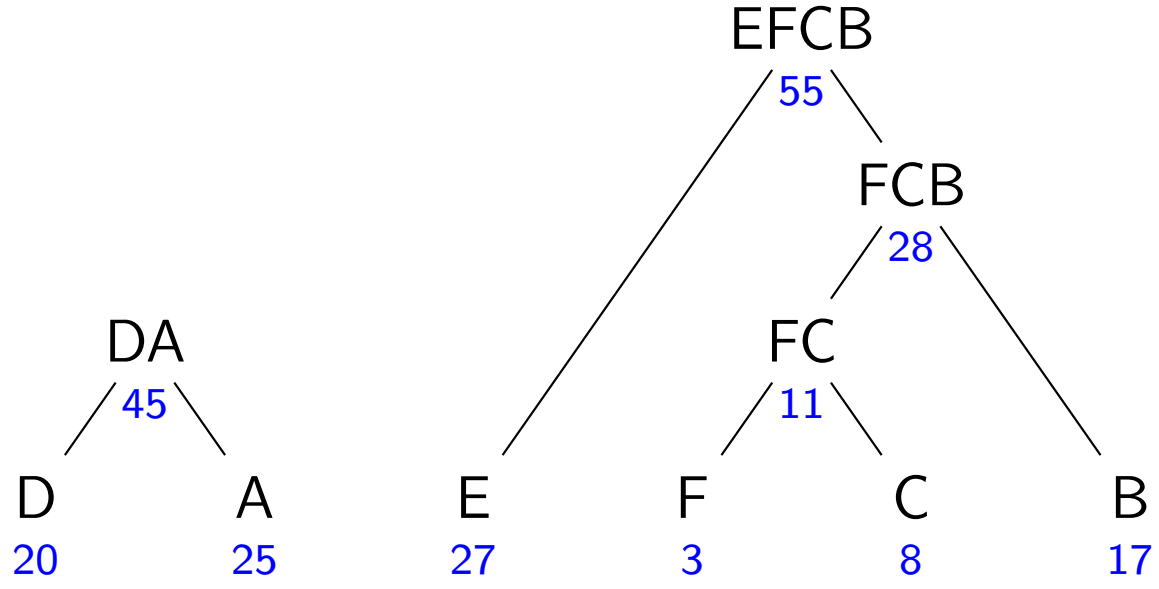
# Primer (pogostosti so v stotinah)

| $c$ | $f_c$ |
|---|---|
| DA | 45 |
| FCB | 28 |
| E | 27 |



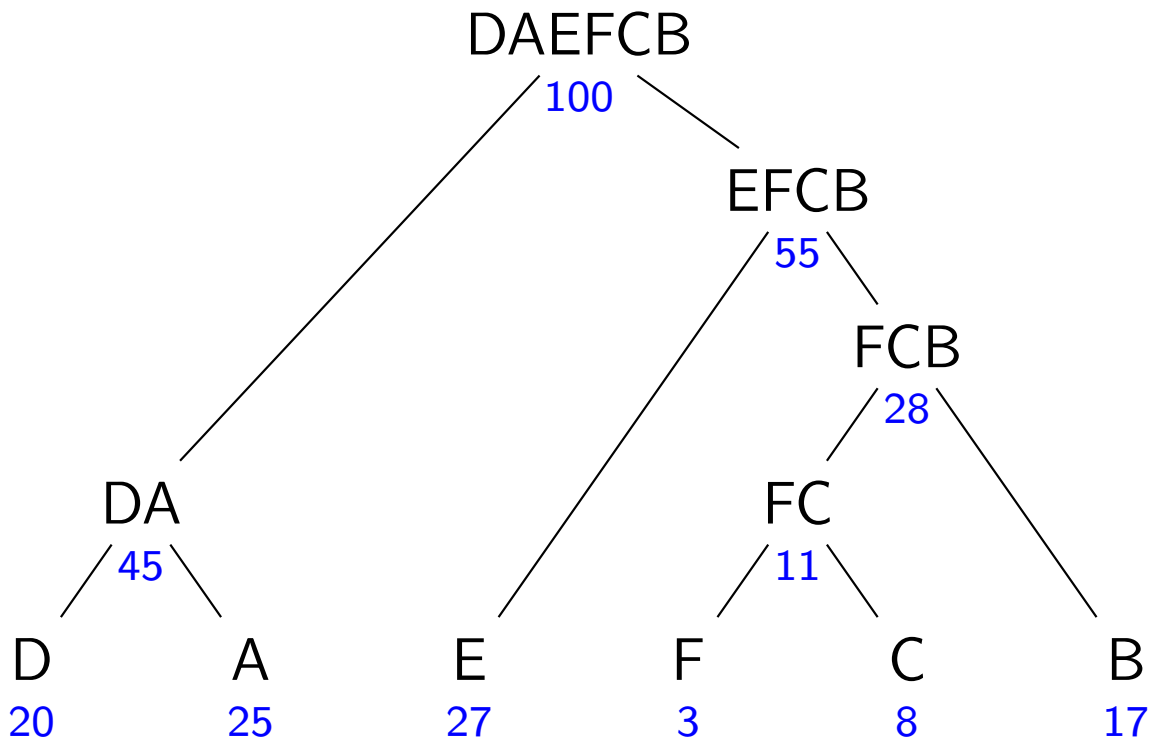
# Primer (pogostosti so v stotinah)

| $c$ | $f_c$ |
|---|---|
| DA | 45 |
| EFCB | 55 |

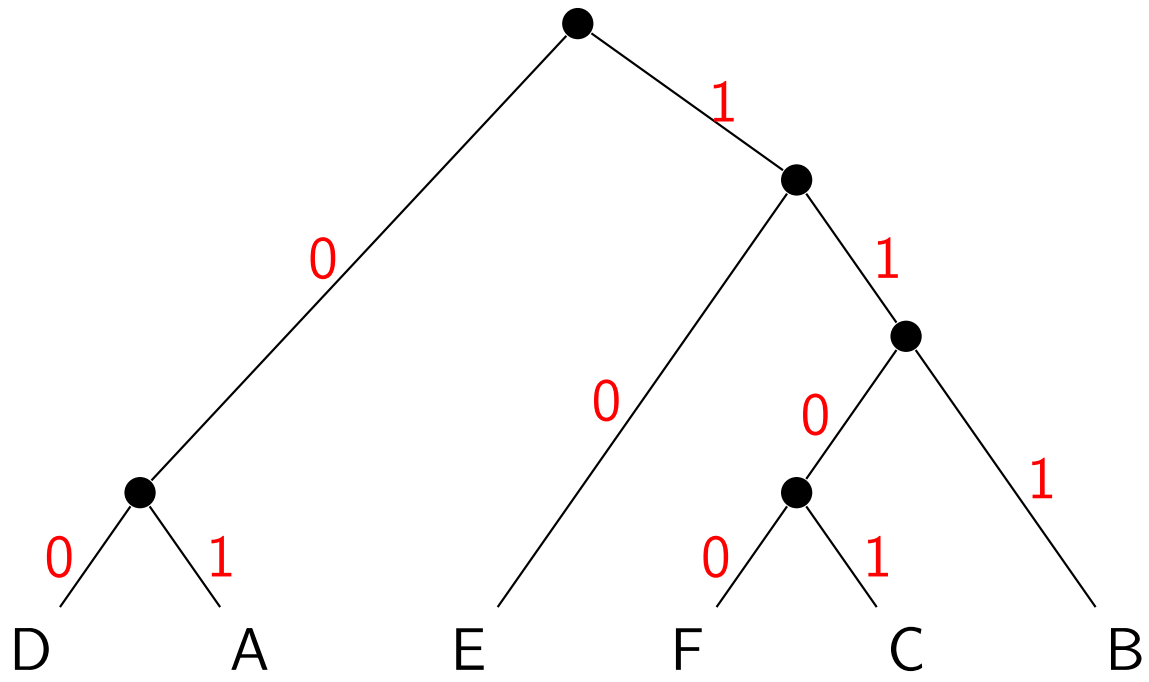


# Primer (pogostosti so v stotinah)

| $c$ | $f_c$ |
|---|---|
| DAEFCB | 100 |



# Primer



# Dokaz optimalnosti

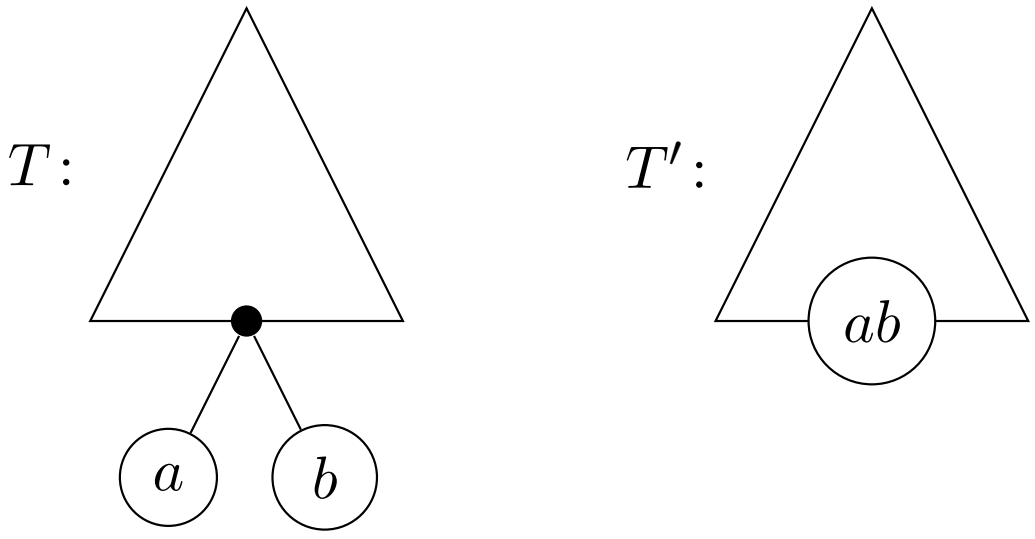
- Dokažimo, da Huffmanov algoritem zgradi drevo z minimalnim $\mathrm{ABL}$ za poljuben $n \geq 2$ (poljuben nabor $n$ znakov in njihovih pogostosti)
- Dokaz z indukcijo
- Baza
  - primer za $n = 2$ je trivialen
- Induktivna predpostavka
  - predpostavimo, da so drevesa za abecede velikosti $n - 1$ optimalna

# Dokaz pravilnosti: induktivni korak

- Kako pri abecedi velikosti $n$ zgradimo drevo $T$?
- Nekoliko drugačen pogled na algoritem
  - znaka $a$ in $b$ z najmanjšima pogostostma ($f_a$ in $f_b$) nadomestimo z »združenim« znakom $ab$ s pogostostjo $f_a + f_b$
  - algoritem rekurzivno poženemo na dobljeni abecedi velikosti $n - 1$ in dobimo drevo $T'$
  - drevo $T$ dobimo iz drevesa $T'$ tako, da na vozlišče, ki pripada znaku $ab$, pripnemo vozlišči za znaka $a$ in $b$
- V kakšnem odnosu sta $\mathrm{ABL}(T)$ in $\mathrm{ABL}(T')$?

# Dokaz pravilnosti: induktivni korak

- Vozlišči $a$ in $b$ sta brata na največji globini ($d$)
- Vozlišče $ab$ leži na globini $d - 1$




$$\begin{aligned}\mathrm{ABL}(T') &= \mathrm{ABL}(T) - f_a d - f_b d + (f_a + f_b)(d-1) \\ &= \mathrm{ABL}(T) - f_a d - f_b d + f_a d + f_b d - f_a - f_b \\ &= \mathrm{ABL}(T) - f_a - f_b\end{aligned}$$

# Dokaz pravilnosti: induktivni korak

- $\mathrm{ABL}(T') = \mathrm{ABL}(T) - f_a - f_b$
- $T'$ je drevo za abecedo velikosti $n-1$ in je po induktivni predpostavki optimalno (ima najmanjši $\mathrm{ABL}$)
- Recimo, da $T$ ne bi bilo optimalno
- Potem bi obstajalo drevo $U$ z $\mathrm{ABL}(U) < \mathrm{ABL}(T)$, ki ima (po trditvi 5) na največji globini brata $a$ in $b$ z najnižjima pogostostma
- Za drevo $U'$, ki nastane iz drevesa $U$ po enem koraku Huffmanovega algoritma, velja $\mathrm{ABL}(U') = \mathrm{ABL}(U) - f_a - f_b$
- Potem je $\mathrm{ABL}(U') < \mathrm{ABL}(T')$, kar je v nasprotju s predpostavko o optimalnosti drevesa $T'$

# Časovna zahtevnost

- $O(n)$ iteracij
- $O(\log n)$ za vsako iteracijo (z uporabo prioritetne vrste)
- Skupaj $O(n \log n)$

# Je Huffmanovo kodiranje res optimalno?

- Da, če kot mero optimalnosti vzamemo $\mathrm{ABL}$
- Drevo oz. kodiranje, ki ga zgradi Huffmanov algoritem, ima za podano besedilo dokazano najmanjšo povprečno število bitov na znak

# Je Huffmanovo kodiranje res optimalno?

- Ne, če se pogovarjamo o kodiranju nasploh
- Protiprimer 1
  - besedilo z $m$ A-ji in $n$ B-ji, kjer je $m \gg n$
  - Huffmanovo kodiranje nam dá $\gamma(\mathrm{A}) = 0$ in $\gamma(\mathrm{B}) = 1$
  - obstajajo boljši načini (npr. lahko preprosto zapišemo indekse B-jev)
- Protiprimer 2
  - besedilo, pri katerem se distribucija skozi čas spreminja
  - npr. v prvi polovici $n$ A-jev in $n$ B-jev; v drugi polovici $n$ C-jev in $n$ D-jev
  - Huffmanonovo kodiranje nam dá kode 00, 01, 10 in 11
  - bolje bi bilo A in C zakodirati z 0 ter B in D z 1, prelom pa označiti z nekim posebnim zaporedjem