GREEDY ALGORITMI

UVOD U POHLEPU

- Za mnoge probleme optimizacije, algoritmi bazirani na dinamičkom programiranju su „overkill“
- Često su jednostavniji algoritmi, koji su efikasniji, bolje rešenje
- Pohlepni algoritmi uvek donose odluku koja u datom trenutku deluje najbolje
- Pravi se odluka na lokalnom nivou (na osnovu trenutnih podataka) u nadi da će odvesti do optimalnog rešenja na globalnom nivou
- Ne daju često optimalna rešenja, ali u mnogim, određenim situacijama daju
- Vrlo moćan alat koji se koristi za rešavanje mnogih problema (ili se rešenja tih problema baziraju na pohlepnim algoritmima)

UVOD U POHLEPU – JEDNOSTAVNI PRIMERI …

- **Penjanje na planinu**
  - Cilj nam je da dođemo na najviši vrh
  - U svakom momentu na raspolaganju su nam strmiji i manje strmi putevi
  - Kako da biramo staze? Pohlepni pristup?
  - **Ako uvek idemo najstrmijom stazom popećemo se na neki vrh, ne obavezno najviši**

… UVOD U POHLEPU – JEDNOSTAVNI PRIMERI

- **Vraćanje kusura**
  - Imamo na raspolaganju novčanice od 1, 2, 5, 10, 20, 50, 100, 200, 500 dinara i hoćemo da vratimo kusur sa najmanjim brojem novčanica
  - Kako vraćati kusur? Kako radi pohlepni algoritam u ovom slučaju?
  - **Uvek vraća najveću novčanicu** koja je manja od tekuće vrednosti kusura (npr. 188 = 100+50+20+10+5+2+1)
  - Koje je ograničenje?
  - **Ovakav pristup u opštem slučaju ne garantuje da ćemo vratiti kusur koji se sastoji od najmanjeg broja novčanica**
  - Recimo ako bi nam na raspolaganju bile novčanice od 5, 4, 2 i 1 dinara, a treba da vratimo kusur od 8 dinara, pohlepni algoritam bi vratio kombinaciju 5 + 2 + 1, a rešenje koje uključuje najmani broj novčanica je 4 + 4

RASPOREĐIVANJE AKTIVNOSTI

PROBLEM RASPOREĐIVANJA AKTIVNOSTI …

- Problem raspoređivanja aktivnosti odnosi se na situaciju kada je potrebno odlučiti kako rasporediti više različitih aktivnosti koje zahtevaju isti resurs
- Aktivnosti koriste ekskluzivno resurse (uvek samo jedna aktivnost može da ima pristup datom resursu u datom trenutku)
- Primer problema raspoređivanja je pravljenje rasporeda časova
- Pretpostaviti da postoji skup $\boldsymbol{S = \{a_1, a_2, \dots, a_n\}}$ koji sadrži $\boldsymbol{n}$ mogućih aktivnosti koje trebaju da koriste neki resurs (npr. neku učionicu)
- Svaka aktivnost $\boldsymbol{a_i}$ ima svoje početno vreme, $\boldsymbol{s_i}$, i svoje završno vreme, $\boldsymbol{f_i}$, pri čemu važi: $\boldsymbol{0 \leq s_i < f_i < \infty}$
- Ako se odabere aktivnost $\boldsymbol{a_i}$ ono traje tokom intervala $\boldsymbol{[s_i, f_i)}$
- Dve aktivnosti $\boldsymbol{a_i}$ i $\boldsymbol{a_j}$ su kompatibilne ako se intervali $\boldsymbol{[s_i, f_i)}$ i $\boldsymbol{[s_j, f_j)}$ ne poklapaju, tj. $\boldsymbol{s_i \geq f_j}$ ili $\boldsymbol{s_j \geq f_i}$

… PROBLEM RASPOREĐIVANJA AKTIVNOSTI

- U problemu raspoređivanja aktivnosti teži se da se pronađe najveći podskup međusobno kompatibilnih aktivnosti
- Pretpostavlja se da su aktivnosti sortirane u rastućem redosledu njihovih završnih vremena $\boldsymbol{f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{n-1} \leq f_n}$
- Pretpostaviti da postoji sledeći skup aktivnosti

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- Jedno od mogućih rešenja je $\{\boldsymbol{a_3, a_9, a_{11}}\}$
- Ali da li je to najveći podskup?
- Nije: $\{\boldsymbol{a_1, a_4, a_9, a_{11}}\}$, a ima i $\{\boldsymbol{a_2, a_4, a_9, a_{11}}\}$
- Da li se problem može rešiti dinamičkim programiranjem?

REŠENJE DINAMIČKIM PROGRAMIRANJEM …

- Problem raspoređivanja aktivnosti može se rešiti dinamičkim programiranja, jer ima optimalnu podstrukturu
- Naime, neka je S_{ij} skup koji sadrži sve aktivnosti koji počinju nakon što se završi aktivnost a_i i koje počinju pre nego što aktivnost a_j počne
- Želimo da pronađemo maksimalan podskup međusobno kompatibilnih aktivnosti iz S_{ij} i pretpostaviti da je to skup A_{ij} koji sadrži neku aktivnost a_k
- Uključivanjem a_k u A_{ij} razbijamo problem da 2 potproblema:
  - pronalaženje međusobno kompatibilnih aktivnosti iz S_{ik} (skup koji sadrži sve aktivnosti koji počinju nakon što se završi aktivnost a_i i koje počinju pre nego što aktivnost a_k počne)
  - pronalaženje međusobno kompatibilnih aktivnosti iz S_{kj} (skup koji sadrži sve aktivnosti koji počinju nakon što se završi aktivnost a_k i koje počinju pre nego što aktivnost a_j počne)

… REŠENJE DINAMIČKIM PROGRAMIRANJEM …

- Neka je
  - $\boldsymbol{A_{ik} = A_{ij} \cap S_{ik}}$ (aktivnosti iz $\boldsymbol{A_{ij}}$ koje počinju pre $\boldsymbol{a_k}$) i
  - $\boldsymbol{A_{kj} = A_{ij} \cap S_{kj}}$ (aktivnosti iz $\boldsymbol{A_{ij}}$ koje počinju posle $\boldsymbol{a_k}$)
- To znači da dobijamo da je:

  $$\boldsymbol{A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}}$$

- I da je maksimalan broj međusobno kompatibilnih aktivnosti koji se mogu rasporediti

  $$\boldsymbol{|A_{ij}| = |A_{ik}| + |A_{kj}| + 1}$$

- Vrlo lako se isecanje-pa-ubacivanje tehnikom može pokazati da optimalno rešenje $\boldsymbol{A_{ij}}$ mora uključivati i optimalna rešenja za $\boldsymbol{S_{ik}}$ i $\boldsymbol{S_{kj}}$, tj. da su $\boldsymbol{A_{ik}}$ i $\boldsymbol{A_{kj}}$ optimalna rešenja od za $\boldsymbol{S_{ik}}$ i $\boldsymbol{S_{kj}}$

… REŠENJE DINAMIČKIM PROGRAMIRANJEM …

- Vrlo lako se isecanje-pa-ubacivanje tehnikom može pokazati da optimalno rešenje $\boldsymbol{A_{ij}}$ mora uključivati i optimalna rešenja za $\boldsymbol{S_{ik}}$ i $\boldsymbol{S_{kj}}$, tj. da su $\boldsymbol{A_{ik}}$ i $\boldsymbol{A_{kj}}$ optimalna rešenja od za $\boldsymbol{S_{ik}}$ i $\boldsymbol{S_{kj}}$
- Da to nije tako, moglo bi da postoji na primer neko $\boldsymbol{A'_{ik} > A_{ik}}$ te bi tada $\boldsymbol{|A'_{ik}| + |A_{kj}| + 1 > |A_{ik}| + |A_{kj}| + 1 = |A_{ij}|}$, što bi značilo da $\boldsymbol{A_{ij}}$ nije optimalno rešenje od starta
- Simetrično se može dokazati i za aktivnosti u $\boldsymbol{S_{kj}}$
- Ovo sve ukazuje da se problem raspoređivanja aktivnost može rešiti dinamičkim programiranjem
- Ako se veličina optimalnog rešenja za skup $\boldsymbol{S_{ij}}$ označi kao $\boldsymbol{c[i,j]}$ onda se njeno izračunavanje može iskazati rekurentnom jednačinom:

$$\boldsymbol{c[i,j] = c[i,k] + c[k,j] + 1}$$

… REŠENJE DINAMIČKIM PROGRAMIRANJEM …

- Ako se veličina optimalnog rešenja za skup $\boldsymbol{S_{ij}}$ označi kao $\boldsymbol{c[i,j]}$ onda se njeno izračunavanje može iskazati rekurentnom jednačinom:

$$\boldsymbol{c[i,j] = c[i,k] + c[k,j] + 1}$$

- Naravno, kako se ne zna tačna pozicija aktivnosti $\boldsymbol{k}$ moraju se proučiti sve aktivnosti iz skupa $\boldsymbol{S_{ij}}$, tako da:

$$\boldsymbol{c[i,j]} = \begin{cases} \boldsymbol{0} & ako\ je\ S_{ij} = 0 \\ \max\limits_{a_k \in \boldsymbol{S_{ij}}} \{\boldsymbol{c[i,k] + c[k,j] + 1}\} & ako\ je\ S_{ij} \neq 0 \end{cases}$$

- Lako se može razviti algoritam baziran na dinamičkom programiranju primenom bottom-up ili top-down pristupom

REŠENJE POHLEPNIM ALGORITMOM …

- Šta ako ne istražujemo svaku mogućnost? Da li postoji način da izbegnemo sva ponavljanja?
- Može ako se napravi pohlepni izbor
- Kod problema raspoređivanja aktivnosti pohlepan izbor bi bio onaj koji nakon izbora neka aktivnosti ostavlja najviše resursa na raspolaganju ostalim aktivnostima što će omogućiti da se rasporedi najviše aktivnosti
- To znači da se bira aktivnost koja će prva da se završi
- Zato je bilo dobro sortirati aktivnosti u rastućem redosledu njihovih završnih vremena

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- Tako da je u ovom slučaju pohlepni izbor $\boldsymbol{a_1}$

… REŠENJE POHLEPNIM ALGORITMOM …

- Pohlepni izbor je na taj način eliminisao sve aktivnosti koje završavaju pre nego što a_k počne
- Zašto?
- Jer je završno vreme za a_1 najmanje, pa su sa njom kompatibilne samo aktivnosti koje počinju nakon što se ono završi i dalje se samo te aktivnosti i posmatraju
- Potrebno je sada posmatrati dalje samo one aktivnosti koje počinju nakon što se a_1 završi
- Na ovaj način je i dalje zadržana optimalna podstruktura, jedino je eliminisana leva strana strukture
- Neka je $S_k = \{a_i \in S : s_i \geq f_k\}$ skup svih aktivnosti koji počinju posle neke aktivnosti a_k
- Ako se napravi pohlepni izbor i njime odabere aktivnost a_1, onda je S_i jedini potproblem koji treba rešiti

… REŠENJE POHLEPNIM ALGORITMOM …

- Teorema:

  Ako je S_k neprazan potproblem (skup koji sadrži preostale kompatibilne aktivnosti u odnosu na aktivnost odabranu pohlepnim izborom) i ako je a_m aktivnost iz S_k sa najkraćim završnim vremenom, onda je a_m deo nekog podskupa međusobno kompatibilnih aktivnosti iz S_k maksimalne veličine

… REŠENJE POHLEPNIM ALGORITMOM …

- Dokaz:
  - Neka je A_k podskup međusobno kompatibilnih aktivnosti iz S_k maksimalne veličine i neka je a_j aktivnost iz A_k sa najkraćim završnim vremenom
  - Ako je $a_j = a_m$, onda je to dokazano
  - Ako je $a_j \neq a_m$, neka je onda $A'_k = A_k - \{a_m\} \cup \{a_j\}$, isto što i A_k samo što a_m menja a_j
  - Aktivnosti unutar A'_k su disjunktne, što proizilazi iz činjenice da su i aktivnosti unutar A_k disjunktne, a a_j je prva aktivnost koja će se završiti i $f_m \leq f_j$)
  - Kako je $|A'_k| = |A_k|$, zaključujemo da je i A'_k isto neki podskup međusobno kompatibilnih aktivnosti iz S_k maksimalne veličine i on uključuje u sebe a_m

… REŠENJE POHLEPNIM ALGORITMOM

- Kao što smo videli, ne treba nam dinamičko programiranje u ovom slučaju
- Ono što nam treba je da uzimamo onu aktivnost koja prva završava, ostavljamo samo aktivnosti koje su kompatibilne sa njom i među njima biramo sledeću aktivnost sve dok ih nismo sve potrošili
- Kako biramo aktivnosti sa najkraćim vremenom završavanja, sve odabrane aktivnosti imaju tendenciju da im vreme završavanja raste, tako da se svaka aktivnost razmatra samo jednom u striktno rastućoj vrednosti vremena završavanja
- Pohlepni algoritam u ovom primeru nema potreba da radi bottom-up pristupom, već top-down
- Pohlepni algoritmi generalno imaju top-down dizajn – napravi se izbor, pa se reševa potproblem (umesto rešavanja potproblema, pa pravljenje izbora)

REKURZIVNI POHLEPNI ALGORITAM …


```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]      // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

- $\boldsymbol{s}$ je niz početnih vremena, $\boldsymbol{f}$ je niz završnih vremena
- $\boldsymbol{k}$ definiše potproblem $\boldsymbol{S_k}$ koji se rešava
- $\boldsymbol{n}$ definiše veličinu originalnog problema
- pretpostavka je da je $\boldsymbol{n}$ aktivnosti sortirano monotono u rastućem redosledu spram niza završnih vremena
- da bi algoritam počeo da radi sve se proširuje fiktivnom aktivnošću $\boldsymbol{a_0}$ čije je vreme završetka $\boldsymbol{f_0 = 0}$
- algoritam započinje pozivom RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$

… REKURZIVNI POHLEPNI ALGORITAM …

- Kako algoritam radi:
  - U bilo kom rekurzivnom pozivu u ***while*** petlji se traži prva kompatibilna a_m aktivnost u S_k potproblemu (koja je ujedno ima i najkraće vreme završavanja)

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

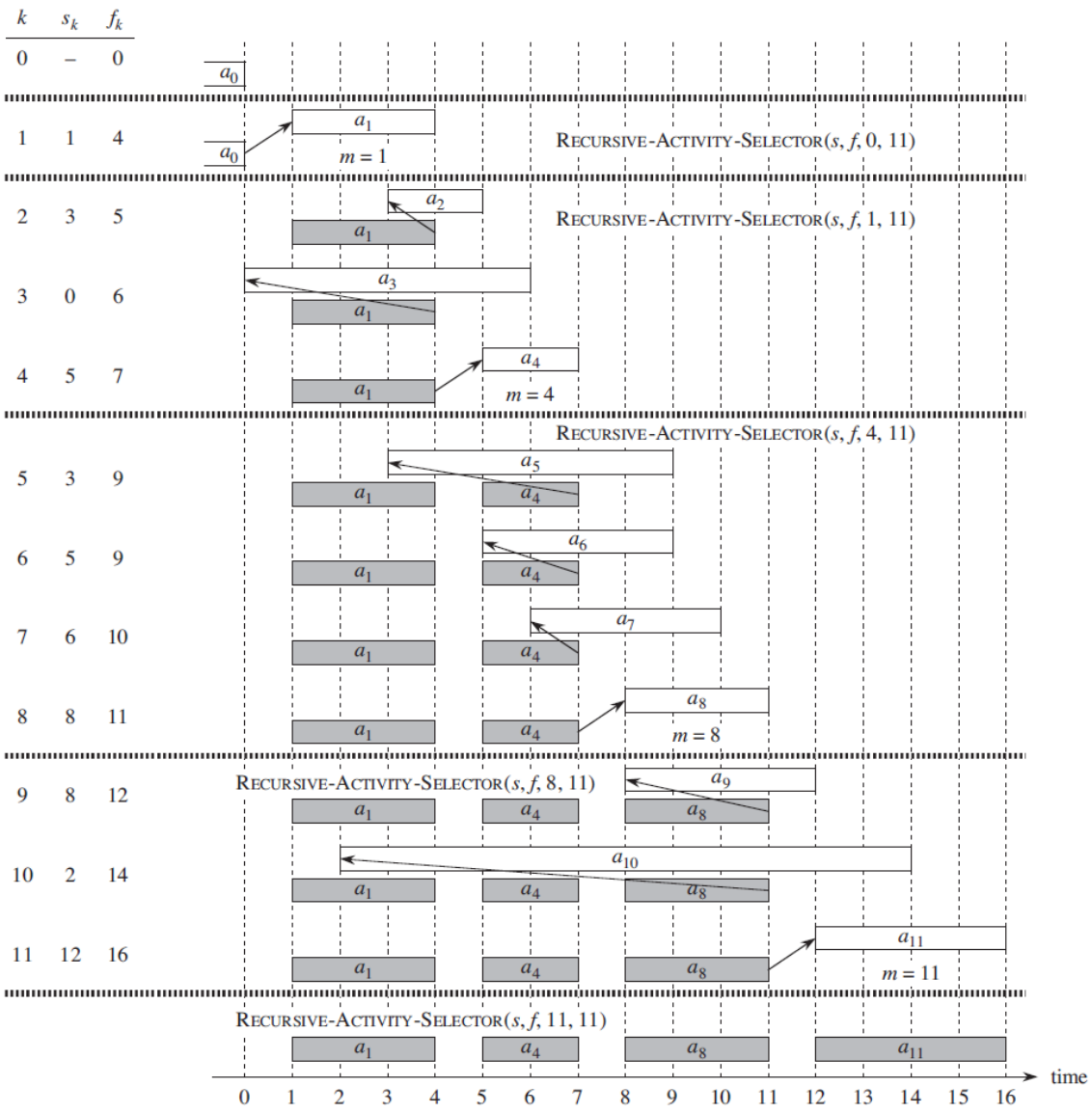
  - Petlja će proveriti aktivnosti $a_{k+1}, a_{k+2}, \ldots, a_m$ (m može biti i $m = n$)
  - Petlja se završi ili kada je pronađena aktivnost ili kada se stiglo do kraja
  - Ako je pronađena aktivnost, onda se ona vraća a_m i unija sa rekurzivnim pozivom procedure koja će rešiti potproblem S_m
  - U slučaju da aktivnost nije pronađena, vraća se prazan skup
- Vreme izvršavanja algoritma je $\Theta(n)$. Zašto?
- Svaka aktivnost se kroz sve rekurzije proveri samo jednom i u samo jednoj rekurziji

… REKURZIVNI POHLEPNI ALGORITAM

- Kako algoritam radi na primeru:

i	1	2	3	4	5	6
s_i	1	3	0	5	3	5
f_i	4	5	6	7	9	9

i	7	8	9	10	11
s_i	6	8	8	2	12
f_i	10	11	12	14	16



ITERATIVNI POHLEPNI ALGORITAM …

- Kada se rekurzivni poziv nalazi skoro na kraju (ili na samom kraju) procedure, lako je transformisati rekurzivno rešenje u iterativno

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

- Isto kao i u rekurzivnom algoritmu, s je niz početnih vremena, f je niz završnih vremena (i sortirano je u rastućem redosledu po f)
- Izabrane aktivnosti se smeštaju u novi niz, A

… ITERATIVNI POHLEPNI ALGORITAM …

- Kako algoritam radi:
  - Promenljiva $\boldsymbol{k}$ čuva u sebi indeks zadnje aktivnosti dodate u $\boldsymbol{A}$ što odgovara aktivnosti $\boldsymbol{a_k}$ u rekurzivnoj verziji
  - Zbog načina sortiranja, $\boldsymbol{f_k}$ je uvek najmanje završno vreme
  - Linije 2-3 selektuju aktivnost $\boldsymbol{a_1}$, inicijalizuju $\boldsymbol{A}$, i inicijalizuju promenljivu $\boldsymbol{k}$

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

  - ***for*** petlja u linijama 4-7 pronalaze koja će aktivnost u $\boldsymbol{S_k}$ prva da završi, ali i da li je ona kompatibilna sa svim prethodnim odabranim aktivnostima
  - Ako je aktivnost kompatibilna, onda se ona, $\boldsymbol{a_m}$, u linijama 6-7 dodaje u $\boldsymbol{A}$
- Kao i kod rekurzivnog pristupa, vreme izvršavanja je $\Theta(\boldsymbol{n})$. Zašto?
- Samo jedna petlja

… ITERATIVNI POHLEPNI ALGORITAM

- Za kraj da razmislimo
- Da li je praktično vreme izvršavanja algoritma zaista $\Theta(n)$?
- Šta je sa sortiranjem?
- Da li će se ono magično desiti?
- Smemo li da se oslonimo na to da će aktivnosti biti sortirane?
- Ako ne smemo koliko je onda kompleksnost algoritma, ako uzmemo najbrži algoritam za sortiranje?
- $O(n^2 \log n)$?
- Da li se pohlepni algoritam dešava unutar algoritma za sortiranje ili posle njega?
- Tj. da li se operacije množe ili sabiraju?
- Da nije ipak $O(n \log n)$?

ELEMENTI POHLEPNOG PRISTUPA

KORACI U POHLEPNOM PRISTUPU …

- U primeru rešavanja problema raspoređivanja aktivnosti koristili smo nekoliko koraka:
  1. Određivanje optimalne podstrukure problema
  2. Razvoj rekurzivnog rešenja
  3. Pokazati da se pohlepnim izborom smanjuje broj potproblema
  4. Dokazati da je uvek pouzdano napraviti pohlepni izbor
  5. Razviti rekurzivni algoritam koji implementira pohlepnu strategiju
  6. Pretvaranje rekurzivnog rešenja u iterativno
- U primeru, problemu se pristupilo sa tačke dinamičkog programiranja, pa se analizom shvatilo da se može ići sa pohlepnim pristupom
- Moglo se pristupiti problemu direktno sa aspekta pohlepne strategije

… KORACI U POHLEPNOM PRISTUPU

- U opštijem slučaju pohlepna strategija se realizuje kroz sledeće korake:
  1. Modelovanje (postavka) problema tako da se prvo napravi odluka koja rezultuje jednim potproblemom koji treba rešiti
  2. Dokazati da uvek postoji optimalno rešenje originalnog problema koje pravi pohlepni izbor, tako da je pouzdano napraviti pohlepni izbor
  3. Demonstrirati optimalnu podstrukturu tako što se pokaže da se nakon pohlepnog izbora dobija potproblem koji ima osobinu da ako kombinujemo optimalno rešenje potproblema sa pohlepnim izborom dolazimo do optimalnog rešenja potproblema
- Ako problem pokazuje osobinu pohlepnog izbora (*greedy-choice property*) i osobinu optimalne podstrukture, postoji velika verovatnoća da je pohlepna strategija prava za rešavanje datog problema

OSOBINA POHLEPNOG IZBORA ...

- Glavna osobina problema koji se mogu rešiti pohlepnim algoritmima jeste da se do globalnog (opšteg) optimalnog rešenja može doći pravljenjem lokalno-optimalnog (pohlepnog) izbora
- To znači da se u nekom potproblemu bira rešenje (pravi izbor) koji deluje najbolji za dati problem, pri čemu se ne vodi računa o drugim potproblemima
- Razlika u odnosu na dinamičko programiranje ogleda se u tome što kod dinamičkog programiranja moraju da se reše svi potproblemi na istom nivou, pa da se na osnovu njihovih rešenja donese odluka o izboru rešenja, dok se kod pohlepnog algoritma, pravi izbor na osnovu lokalnih informacija i rešava potproblem koji ostaje nakon tog izbora
- Pohlepni algoritmi obično koriste top-down pristup za razliku od dinamičkog programiranja kod kojeg se prirodno primenjuje bottom-up pristup

… OSOBINA POHLEPNOG IZBORA

- Naravno, na kraju je potrebno dokazati da je pohlepni izbor onaj pravi, tj. da u svakom koraku pohlepni izbor dovodi do globalno optimalnog rešenja
- To se radi tako što se prvo istraži globalno optimalno rešenje nekog problema, nakon čega se modifikuje to rešenje tako što se neki od potproblema zameni pohlepnim izborom, čime se dobija sličan potproblem, ali obično manjih dimenzija, tj. jednostavniji
- Pohlepni izbor je efikasniji, jer najčešće eliminiše potrebu da se ispituje širi skup mogućih izbora
- Pretprocesiranjem ili izborom adekvatne strukture podataka se može ubrzati pohlepni izbor
- Kako bi se ubrzao izbor u primeru problema raspoređivanja aktivnosti?
- Sortiranjem ulaznih nizova u rastućem redosledu završnih vremena aktivnosti

OPTIMALNA PODSTRUKTURA

- Problem ima osobinu optimalne podstrukture, ako je optimalno rešenje problema sadržano unutar optimalnih rešenja njegovih potproblema
- Osobina koju pohlepni algoritmi dele sa dinamičkim programiranjem
- Kod pohlepnih algoritama pronalaženje optimalne podstrukture je malo jednostavnije
- Pretpostavka je da smo pohlepnim izborom došli do potproblema iz originalnog problema
- Sve što treba dokazati je da optimalno rešenje potproblema u kombinaciji sa pohlepnim izborom (koji je već napravljen) dovodi do optimalnog rešenja originalnog problema
- Ovo implicira upotrebu indukcije nad potproblemom kako bi se dokazalo da pohlepni izbor u svakom koraku dovodi do optimalnog rešenja

POHLEPA vs. DINAMIKA …

- Kako pohlepni algoritmi dele osobinu optimalne podstrukture sa dinamičkim programiranjem, često se može desiti da se pogreši i izabere pogrešan algoritam
  - Da se bez potrebe koristi dinamičko programiranje u slučaju kada je dovoljno napraviti pohlepni izbor
  - Da se upotrebi pohlepni algoritam u situaciji kada je zapravo potrebno dinamičko programiranje
- Ovo se lepo može demonstrirati na problemu ruksaka (*knapsack problem*):
  - Lopov uđe u prodavnicu sa ruksakom u kome može stati roba težine W
  - U radnji postoji roba, pri čemu svaki artikal ima svoju težinu w_j i vrednost v_j
  - Problem se ogleda u želji lopova da uzme robu u najvećoj mogućnoj vrednosti, ali da može da je ponese u svom ruksaku

… POHLEPNA vs. DINAMIKA …

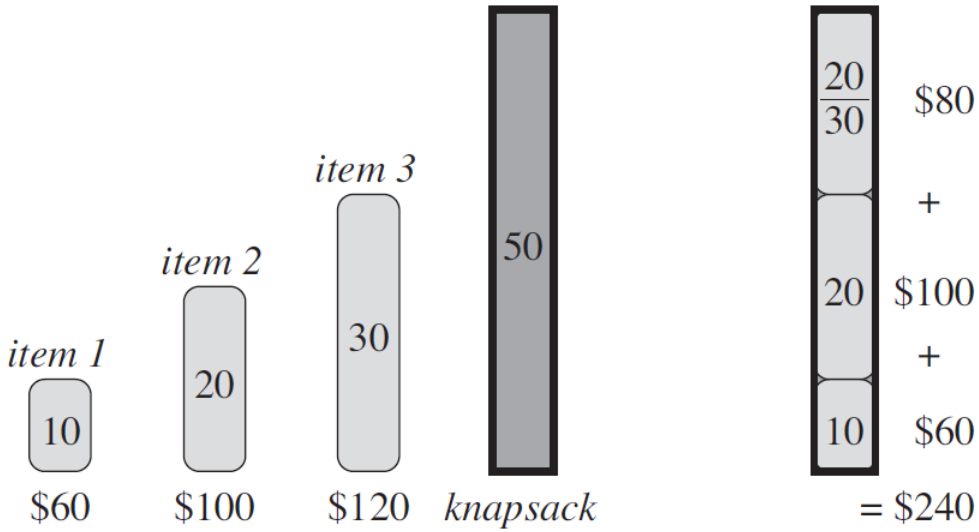
- Ovaj problem može imati dve varijante:
  - Uzmi ili ostavi varijantu (0-1 *knapsack problem*) kada lopov može da uzme samo ceo artikal i to po jedan od svakog artikla
  - Tradicionalnu varijantu kada može da uzme bilo koju količinu bilo kog artikla
- Razlika između ove dve varijante može se zamisliti kao da u prvoj situaciji imamo zlatne poluge, dok u drugoj zlatni prah
- Obe varijantne imaju optimalnu podstrukturu
- Za uzmi ili ostavi varijantu, to je sledeća struktura:
  - Pretpostaviti da teret sa najvećom vrednošću ukupno ima težinu do W (to je i maksimalna nosivost ranca)
  - Ako se iz ovog tereta izbaci artikal j težine w_j, onda je preostao tovar sa najvećom vrednošću za težinu $W - w_j$ koja je sastavljena iz nekih od $n - 1$ originalnih artikala bez j

… POHLEPNA vs. DINAMIKA …

- Za tradicionalnu varijantu, to je sledeća struktura:
  - Pretpostaviti da teret sa najvećom vrednošću ukupno ima težinu do W
  - Ako se iz ovog tereta izbaci neka težina w od artikla j (ne mora ceo artikal da se izbaci, samo deo njegove težine), onda je preostao tovar sa najvećom vrednošću za težinu $W - w$ koja je sastavljena iz nekih od $n - 1$ originalnih artikala plus $w_j - w$ težine artikla j
- Tradicionalna varijanta može se rešiti primenom pohlepnog izbora, ali se uzmi ili ostavi varijanta ne može rešiti pohlepnim izborom
- Zašto?
- Da bi se napravio pohlepni izbor, potrebno je odrediti odnos cene i težine, v_j/w_j i birati uvek onaj artikal koji daje najbolji odnos

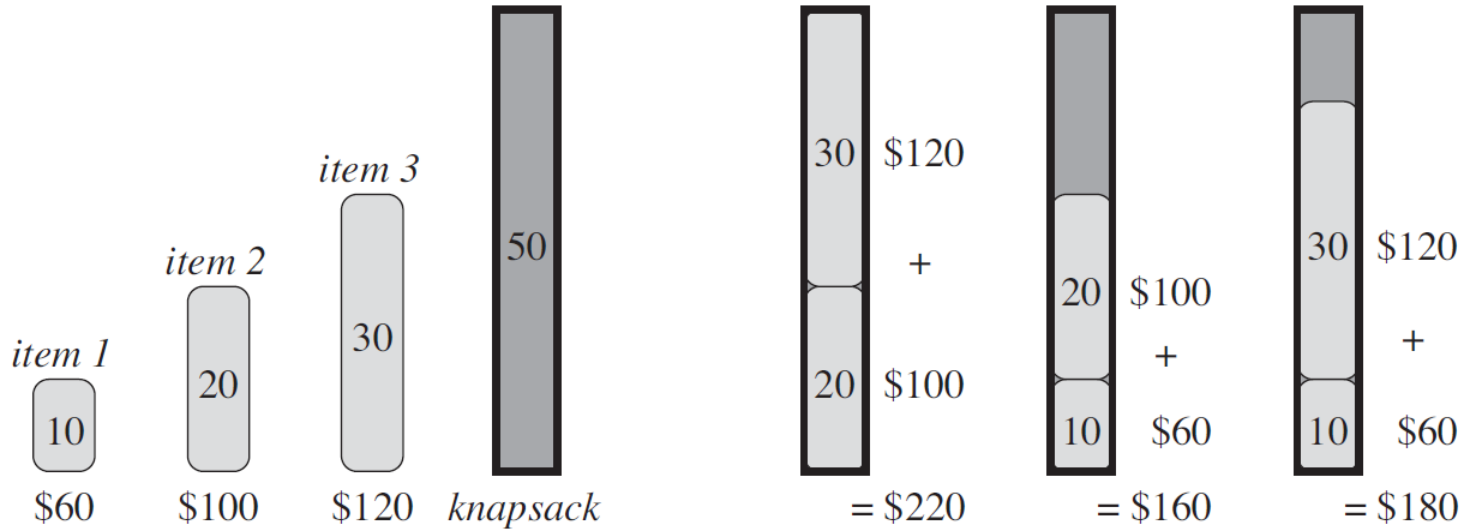
… POHLEPNA vs. DINAMIKA …

- Da bi se napravi pohlepni izbor, potrebno je odrediti odnos cene i težine, v_j/w_j i birati uvek onaj artikal koji daje najbolji odnos
- Kod tradicionalnog je to dovoljno, jer će ruksak uvek biti popunjen do kraja, odabere se artikal sa najboljim odnosom cene/težine i stavi u ruksak, pa sledeći takav, i tako sve dok ima mesta
- Pogledati primer u kojem ima 3 artikla



… POHLEPNA vs. DINAMIKA

- Da bi se napravio pohlepni izbor, potrebno je odrediti odnos cene i težine, v_j/w_j i birati uvek onaj artikal koji daje najbolji odnos
- Kod uzmi ili ostavi to nije dovoljno, jer se može desiti da posle izbora artikla sa najboljim odnosom cene/težine, ne ostane dovoljno mesta u ruksaku da se napravi optimalni izbor
- Pogledati primer u kojem ima 3 artikla



TRADICIONALNI KNAPSACK PROBLEM …

RAZLOMLJENI-RUKSAG (W, w, n)

1: $S = 0$
2: $i = 1$
3: **for** $j = 1$ **to** n
4: $\quad p[j] = 0$
5: **while** $S \leq W$
6: $\quad S = S + w[i]$
7: $\quad p[i] = 1$
8: $\quad i = i + 1$
9: **if** $S > W$
10: $\quad i = i - 1$
11: $\quad$ // ako poslednji artikal ne može ceo da stane, uzimamo samo deo
12: $\quad p[i] = 1 - (S - W)/w[i]$
13: **return** p

… TRADICIONALNI KNAPSACK PROBLEM

- W - nosivost ranca,
- w - vektor težina artikala,
- n - broj artikala
- Pretpostavilja se da su artikli sortirani nerastuće prema vrednosti po jedinici mere
- izlaz: p - procenat svakog predmeta koji je stavljen u ranac

```
RAZLOMLJENI-RUKSAG (W,w,n)
1:  S = 0
2:  i = 1
3:  for j = 1 to n
4:      p[j] = 0
5:  while S ≤ W
6:      S = S + w[i]
7:      p[i] = 1
8:      i = i + 1
9:  if S > W
10:     i = i − 1
11:     // ako poslednji artikal ne može ceo da stane, uzimamo samo deo
12:     p[i] = 1 − (S − W)/w[i]
13: return p
```

HAFMANOVO KODIRANJE

HAFMANOVO KODIRANJE …

- Vrlo efikasan način kompresije podataka
  (20% do 90% zavisnosti od prirode podataka)
- Kod se formira na osnovu statistike
  - koliko puta se neki znak pojavljuje unutar nekog ulaza podataka, ili
  - unutar čitave populacije ulaza
- Kada je potrebno kodirati neki skup znakova, on se može predstaviti binarnim kodom
- Moguće je koristiti binarni kod
  - fiksne dužine uniformno za svaki ulaz podataka bez obzira na statistiku (uvek celobrojni umnožak bajta),
  - fiksne dužine uniformno prilagođeno ulazu podataka
  - promenljive dužine

… HAFMANOVO KODIRANJE …

- Kada se koristi binarni kod fiksne dužine mora se odabrati broj bita dovoljno veliki da se svaki znak može predstaviti tim kodom
- Nekada se zbog jednostavnosti čitavog sistema čitanja/pisanja uzima predefinisana veličina (celobrojni umnožak bajta) nezavisno od toga da li se u nekom ulazu koriste svi znakovi iz alfabeta ili samo neki
- Zašto ASCII kod nije bio dovoljan? Čemu Unicod?
- Alternativa je da se koristi binarni kod fiksne dužine, ali se koristi najmanji broj bita koji je dovoljan da se predstave svi znakovi koji čine ulaznu reč
- Na primer, ako se ulazna reč sastoji od 6 znakova (***a***, ***b***, ***c***, ***d***, ***e***, ***f***), onda su potrebna 3 bita kako bi se oni predstavili/kodirali (***a=000***, ***b=001***, ***c=010***, ***d=011***, ***f=100***, ***e=101***)
- Zašto?
- Jer se sa 3 bita može predstaviti 8 vrednosti

… HAFMANOVO KODIRANJE …

- Za kodiranje znakova fiksne dužine nije potrebno znati frekvenciju pojavljivanja znakova unutar ulaza podataka
- Svakako smo na ovaj način smanjili veličinu poruke spram fiksnog kodiranja uniformne dužine (nepotrebno koristiti 8 bita za 6 znakova)
- Da li se ovo može još unaprediti?
- Da, ako poznajemo frekvenciju pojavljivanja znakova u ulaznoj poruci
- Kako to može pomoći?
- Zamislimo da u primeru sa 6 znakova znamo frekvenciju pojavljivanja svakog znaka ($\boldsymbol{p_a = 0.45}$, $\boldsymbol{p_b = 0.13}$, $\boldsymbol{p_c = 0.12}$, $\boldsymbol{p_d = 0.16}$, $\boldsymbol{p_e = 0.09}$, $\boldsymbol{p_f = 0.05}$)
- Sada znakove možemo kodirati promenljivom dužinom tako znak sa najvećom frekvencijom pojavljivanja ima najmanji kod (***a=0***, ***b=101***, ***c=100***, ***d=111***, ***f=1101***, ***e=1100***)

… HAFMANOVO KODIRANJE …

- Pravi doprinos i uticaj kodiranja promenljive dužine može se videti kada je potrebno kodirati ulaznu poruku ***100 000*** znakova
- Pretpostaviti da za to koristimo onih 6 znakova i pretpostavimo da važi opisana statistika

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- Kada se koristi kodiranje fiksne dužine, izlazna poruka će imati veličinu od ***300 000*** bita
- Kada se koristi kodiranje promenljive dužine, dobijamo da će izlazna poruka imati veličinu od ***224 000*** bita
- Kako?

$$(p_a \cdot a_{bit} + p_b \cdot b_{bit} + p_c \cdot c_{bit} + p_d \cdot d_{bit} + p_e \cdot e_{bit} + p_f \cdot f_{bit}) \cdot 100000 =$$
$$(0.45 \cdot 1 + 0.13 \cdot 3 + 0.12 \cdot 3 + 0.16 \cdot 3 + 0.09 \cdot 4 + 0.05 \cdot 4) \cdot 100000 =$$
$$(0.45 + 0.39 + 0.36 + 0.48 + 0.36 + 0.20) \cdot 100000 = 2.24 \cdot 100000 =$$
$$224000$$

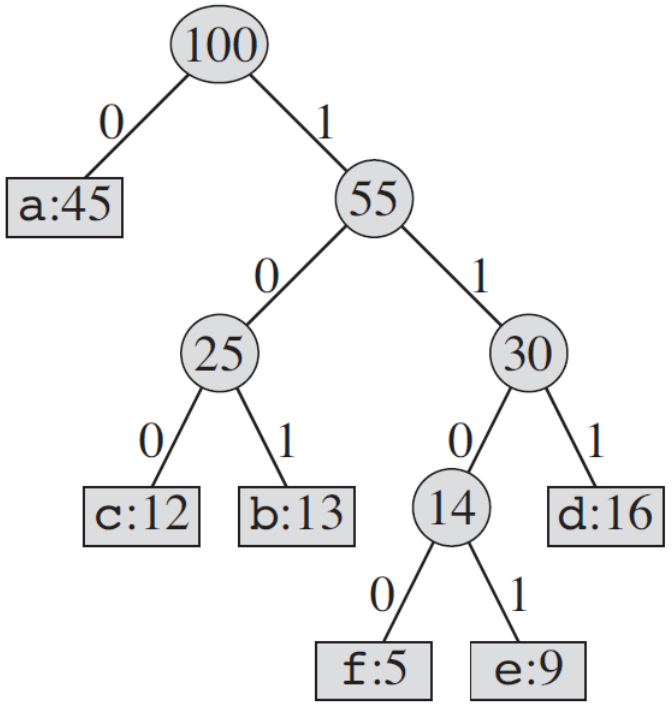
… HAFMANOVO KODIRANJE …

- Kodiranje promenljive dužine opisano u primeru naziva se prefiksno kodiranje
- To znači da je svaka kodna reč jedinstvena i da ni jedna kodna reč ne započinje drugom kodnom rečju (***a=0***, ***b=101***, ***c=100***, ***d=111***, ***f=1101***, ***e=1100***)
- Može se dokazati da prefiksno kodiranje uvek rezultuje optimalnom kompresijom podataka za bilo koji tip karakter kodiranja
- Izlazna reč se u prefiksnom kodiranju formira na jednostavan način
  - prostim nadovezivanjem kodova za različite karaktere koji slede jedan iza drugog ($\mathbf{0 \cdot 101 \cdot 100 = 0101100}$)
- Zašto je ovo moguće?
- Kako je svaki kod jedinstven i ni jedna kodna reč ne deli svoj početak ni sa jednom drugom kodnom rečju, ne postoji razlog za posebnim separatorima

… HAFMANOVO KODIRANJE …

- Ovo značajno pojednostavljuje dekodiranje – identifikuje se početna kodna reč, pronađe se njoj odgovarajući znak i čitav postupak se ponavlja
  ($\mathbf{00101100 = 0 \cdot 0 \cdot 101 \cdot 100}$, što je ***aabe***)
- U Hafmanovom kodiranju čitav postupak kodiranja/dekodiranja bazira se na Hamanovom stablu
- Ovo je binarno stablo koje je lako interpretirati i koje se lako tumači
- Stablo je vrlo značajno za jednostavno dekodiranje, jer čitanje ulazne poruke odgovara kretanju kroz Hafmanovo stablo od korena ka listovima
- Iščitava se bit po bit i u zavisnosti od vrednosti bita bira se jedna od mogućih putanja u stablu
- Kada se stigne do lista stabla, stiglo se i do znaka koji treba upisati u izlaznu poruku, postupak se ponavlja sve dok se ne iščitaju svi kodovi

… HAFMANOVO KODIRANJE …

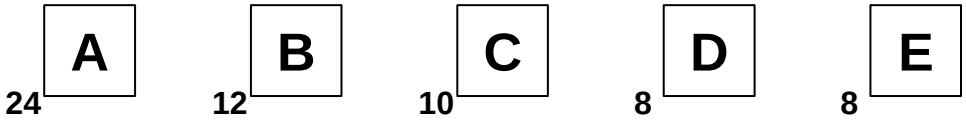


- Kako to izgleda na primeru: ($\mathbf{00101100} = \mathbf{0 \cdot 0 \cdot 101 \cdot 100}$, što je ***aabc***)
- Hafmanovo stablo je kompletno binarno stablo koje za alfabet karaktera $\boldsymbol{C}$ (gde svaki karakter ima frekvenciju pojavljivanja koja je pozitivan broj) ima tačno $|\boldsymbol{C}|$ listova i $|\boldsymbol{C}| - \mathbf{1}$ čvorova
- Ako uzmemo stablo $\boldsymbol{T}$ koje odgovara prefiksnom kodu, lako je odrediti broj bita potrebnih da se kodira čitav fajl
- Ako je $\boldsymbol{c.freq}$ frekvencija pojavljivanja karaktera $\boldsymbol{c}$ iz $\boldsymbol{C}$ u ulaznoj reči, $\boldsymbol{d_T(c)}$ dubina lista u stablu koji odgovara $\boldsymbol{c}$ ($\boldsymbol{d_T(c)}$ je ujedno i broj bita kojima se kodira $\boldsymbol{c}$), broj bita $\boldsymbol{B(T)}$ potrebnih da se kodira ulazna reč je:

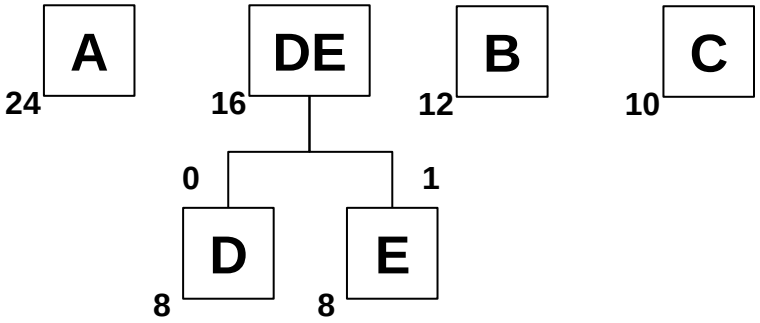
$$\boldsymbol{B(T)} = \sum_{\boldsymbol{c \in C}} \boldsymbol{c.freq \cdot d_T(c)}$$

… HAFMANOVO KODIRANJE ..

- Kako se formira Hafmanovo stablo?
- Znakovi iz izvora podataka se organizuju u sortirani niz listova na osnovu njihovih težina, npr.

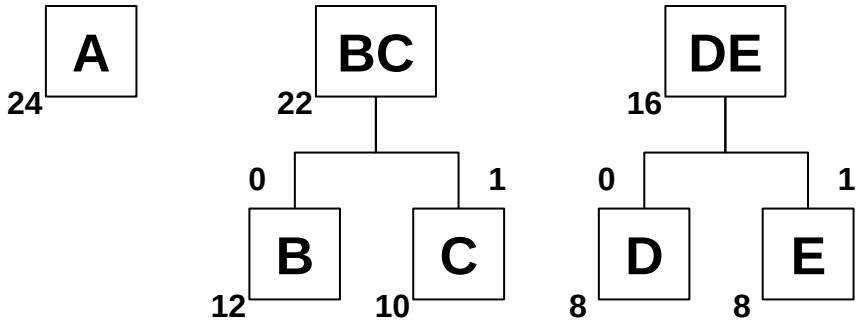


- U prvom koraku pronalaze se čvorovi sa najmanjim težinama, D i E, čije težine su 8 i 8
- Njihovim spajanjem formira se novi čvor čija težina je jednaka sumi težina podređenih čvorova
- Putanja do čvora D dobija kod 0, a do čvora E dobija kod 1
- Čvorovi D i E više nisu slobodni za povezivanje



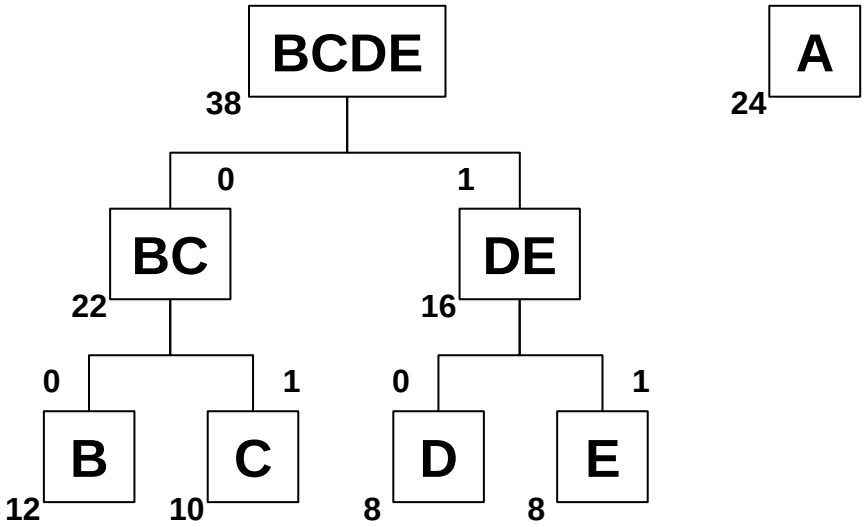
… HAFMANOVO KODIRANJE …

- Algoritam se nastavlja dalje i ponovo se traže dva čvora koji imaju najmanju težinu (to su ujedno i susedni čvorovi)
- U primeru su to B i C čvorovi sa težinama 12 i 10
- Njihovim spajanjem nastaje novi čvor sa težinom jednakom sumi njihovih težina
- Putanja do čvora B dobija kod 0, a putanja do čvora C dobija kod 1
- Čvorovi B i C više nisu u slobodni za povezivanje



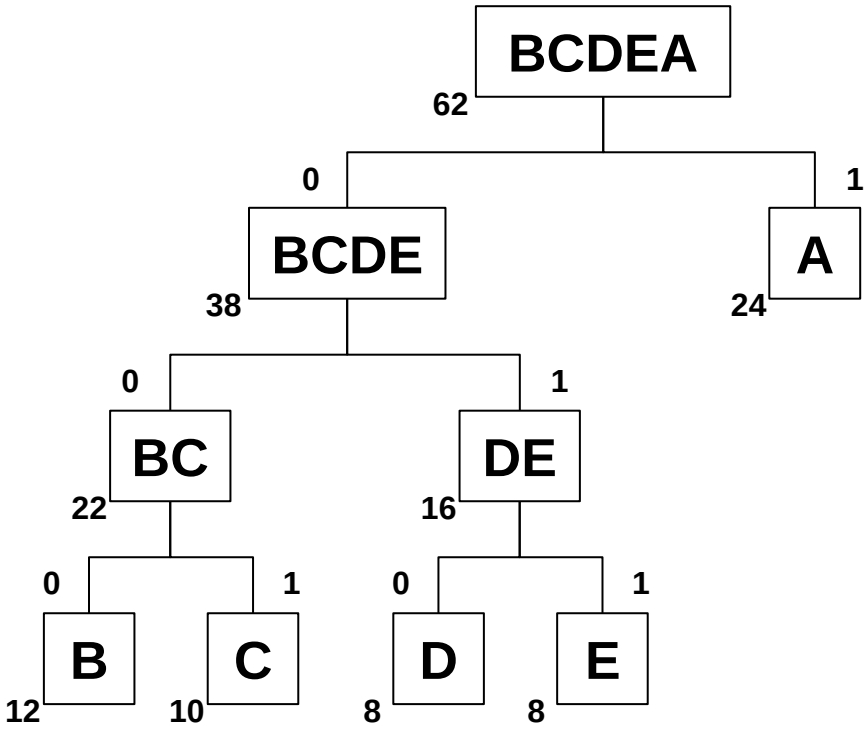
… HAFMANOVO KODIRANJE …

- U sledećem koraku, povezuju se BC kombinovani čvor i DE kombinovani čvor, jer su njihove težine, 22 i 16, manje od težine čvora A
- Putanja do BC kombinovanog čvora dobija kod 0, a putanja do DE kombinovanog čvor dobija kod 1
- BC i DE kombinovani čvorovi više nisu u slobodni za povezivanje



… HAFMANOVO KODIRANJE …

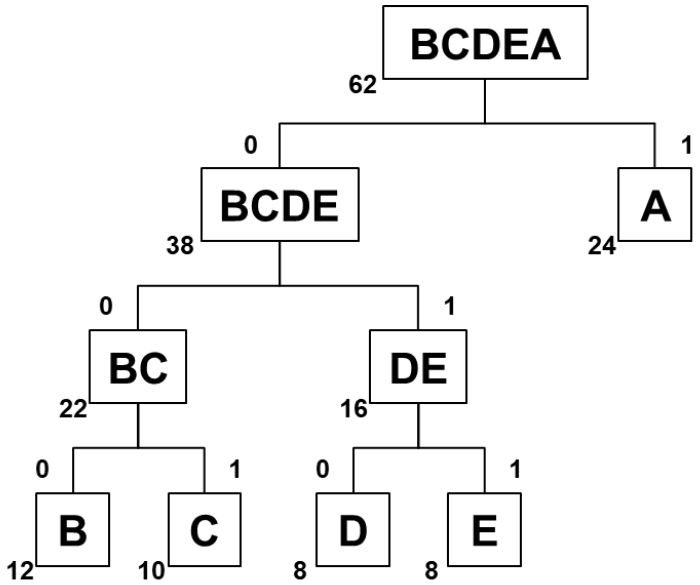
- Na kraju čvor A se povezuje sa BCDE kombinovanim čvorom i formira se Hafmanovo binarno stablo
- Putanja do čvora A dobija kod 0, a putanja do čvora BCDE kombinovanog BCDE kombinovanog A dobija kod 1



… HAFMANOVO KODIRANJE

- Na kraju se dobija sledeći Hafmanov kod:

Znaci	Frekvencija pojavljivanja	Kod
A	24	1
B	12	000
C	10	001
D	8	010
E	8	011



HAFMANOVO ALGORITAM …

- Za potrebe kodiranja, Hafman je implementirao svoj pohlepni algoritam

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

… HAFMANOVO ALGORITAM …

- Ovaj algoritam se oslanja na Minimalne redove prioriteta
- Kreće se od alfabeta $\boldsymbol{C}$ koji sadrži $\boldsymbol{n}$ karaktera i svaki karakter $\boldsymbol{c} \in \boldsymbol{C}$ je objekat koji sadrži kao atribut njegovu frekvenciju pojavljivanja $\boldsymbol{c.freq}$

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

- Algoritam gradi stablo $\boldsymbol{T}$ koje odgovara optimalnom kodu bottom-up pristupom
- Počinje sa $|\boldsymbol{C}|$ listova i zatim izvrši $|\boldsymbol{C}| - \boldsymbol{1}$ spajanja i stvara $|\boldsymbol{C}| - \boldsymbol{1}$ čvorova
- Sortiranje elemenata u Minimalnom redu prioriteta $\boldsymbol{Q}$ ide preko $\boldsymbol{freq}$ atributa

… HAFMANOVO ALGORITAM …

- Takođe, algoritam koristi minimalni red prioriteta Q kako bi identifikovao dva objekta koji imaju najmanju vrednost $freq$ atributa
- Ova dva objekta se spajaju ka nadređenom čvoru

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

- Taj nadređeni čvor ima frekvenciju koja je jednaku sumi frekvencija njemu podređenih čvorova
- Ovo bukvalno prati način pravljenja Hafmanovog stabla iz primera
- Linija 2 inicijalizuje minimalni red prioriteta

… HAFMANOVO ALGORITAM …

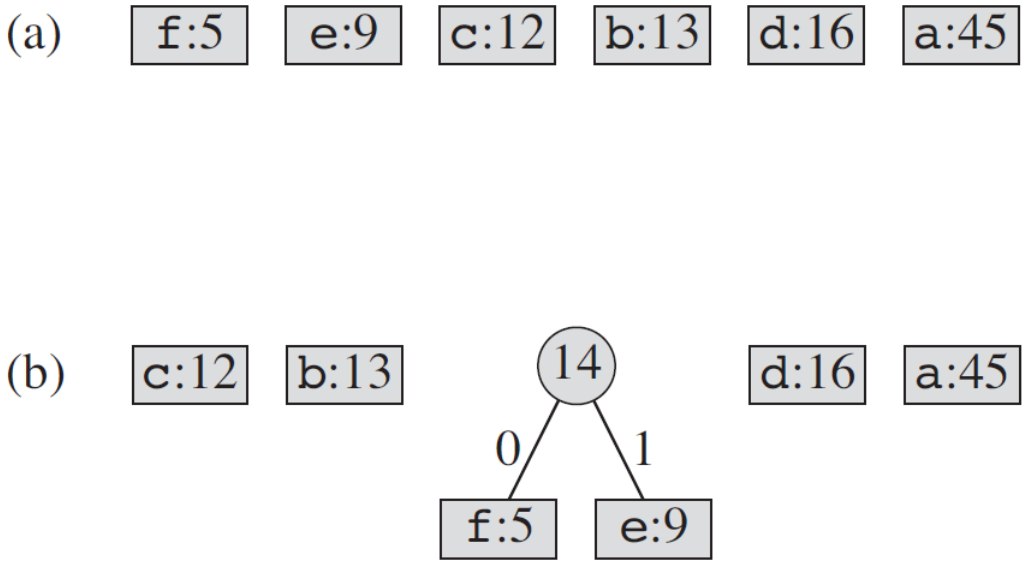
- ***for*** petlja od 3-8 linije
  - kreira novi čvor, tako što stalno izvlači 2 čvora sa najmanjom frekvencijom i menja ih sa čvorom koji je suma njihovih frekvencija
  - novi čvor u sebi čuva veze ka čvorovima koji su njemu podređeni
  - nakon $n-1$ spajanja ***for*** petlje red sadrži samo jedan čvor, koren Hafmanovog stabla

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

- Na kraju se vrati jedini preostali čvor u redu
- x i y su ostavljeni u liniji 7 zbog dokaza korektnosti algoritma

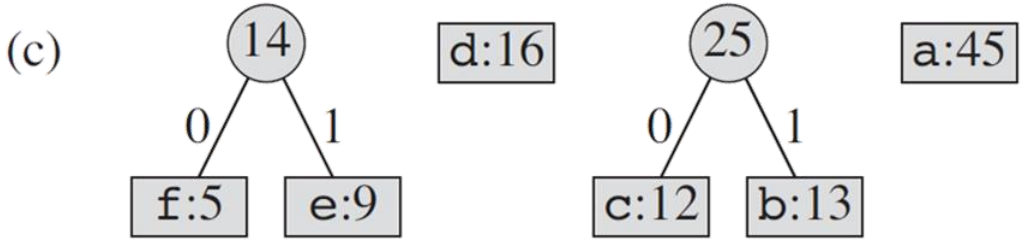
… HAFMANOVO ALGORITAM …

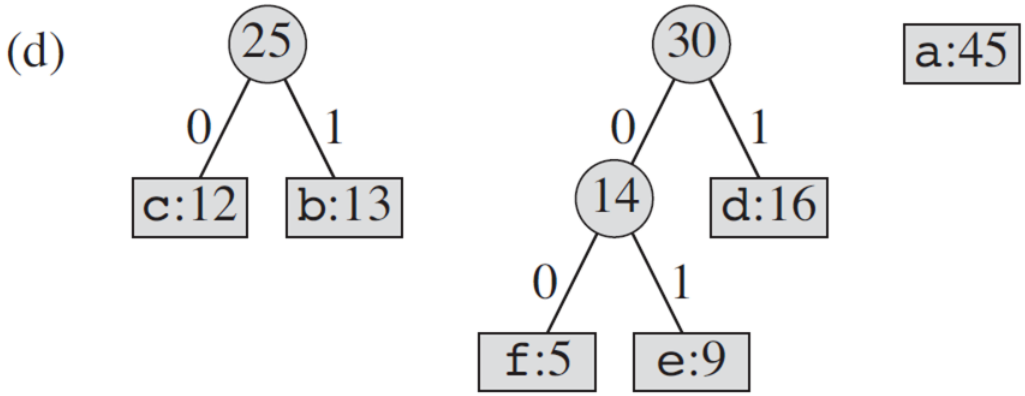
- Kako Hafmanov algoritam rad na primeru



… HAFMANOVO ALGORITAM …

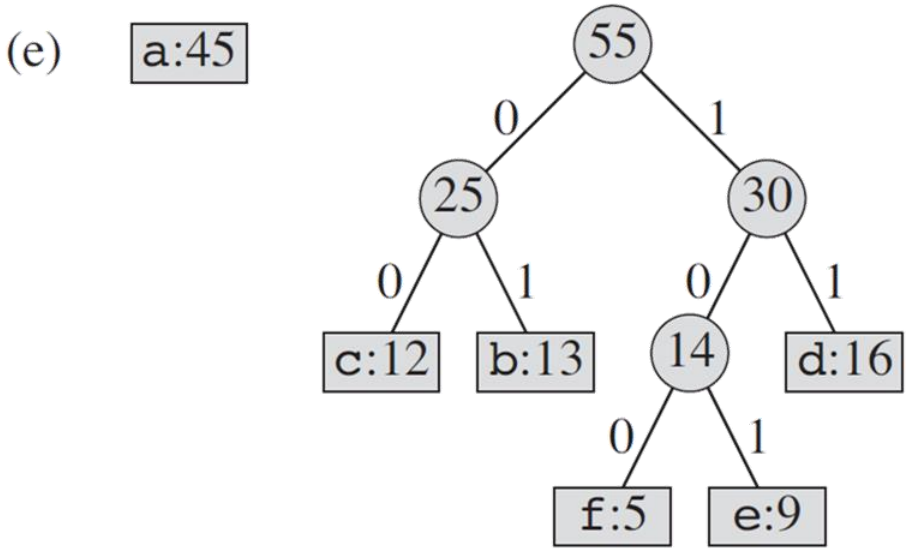
- Kako Hafmanov algoritam rad na primeru (nastavak)





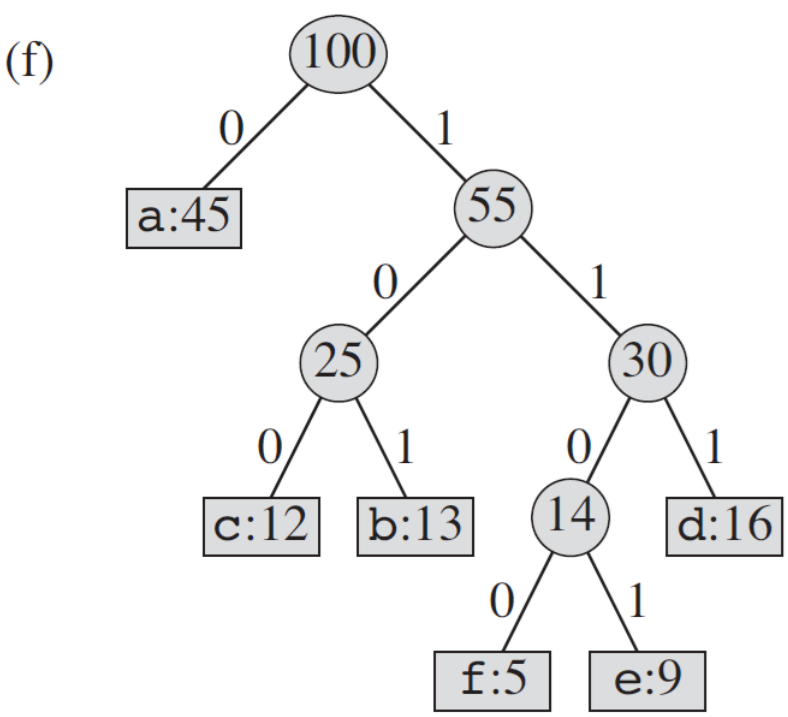
… HAFMANOVO ALGORITAM …

- Kako Hafmanov algoritam rad na primeru (nastavak)



… HAFMANOVO ALGORITAM …

- Kako Hafmanov algoritam rad na primeru (nastavak)



… HAFMANOVO ALGORITAM

- Vreme izvršavanja ovog algoritma se računa na osnovu vremena izvršavanja Minimalnog reda prioriteta
- Koliko je to vreme?
- $\boldsymbol{O}(\log \boldsymbol{n})$
- Kako se ***for*** petlja izvršava $\boldsymbol{n-1}$ puta ukupno vreme izvršavanja je $\boldsymbol{O}(\boldsymbol{n}\log \boldsymbol{n})$

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

KOREKTNOST HAFMANOVOG ALGORITAM …

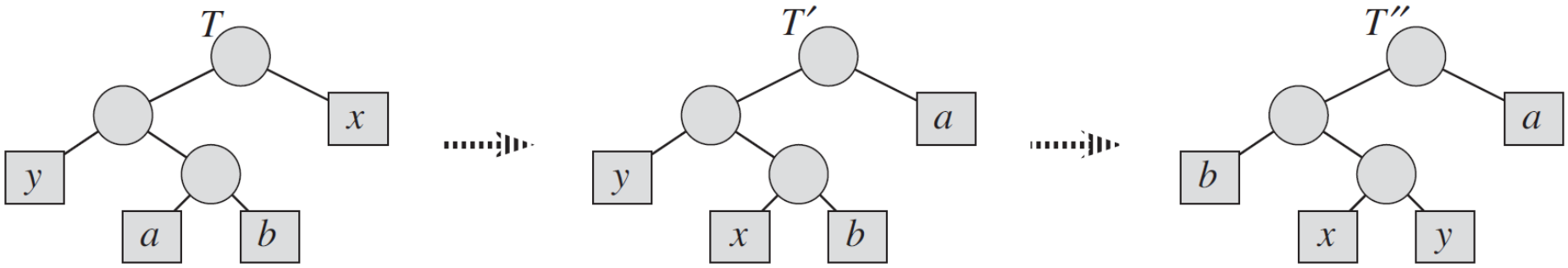
- Da bi se dokazala korektnost Hafmanovog algoritma, potrebno je dokazati da određivanje optimalnog prefiksnog koda ima osobinu pohlepnog izbora i optimalnu podstrukture
- Sledeća lema je dokaz pohlepnog izbora:
  - Neka je $\boldsymbol{C}$ alfabet gde svaki karakter $\boldsymbol{c} \in \boldsymbol{C}$ ima frekvenciju pojavljivanja $\boldsymbol{c.freq}$
  - Neka su $\boldsymbol{x}$ i $\boldsymbol{y}$ karakteri iz $\boldsymbol{c} \in \boldsymbol{C}$ sa najmanjom frekvencijom pojavljivanja
  - Onda postoji optimalni prefiksni kod za $\boldsymbol{C}$ u kojoj su kodne reči za $\boldsymbol{x}$ i $\boldsymbol{y}$ iste dužine i razlikuju se samo u poslednjem bitu
- Dokaz ide po principu se uzme stablo $\boldsymbol{T}$ koje reprezentuje arbitrarni optimalni prefiksni kod i modifikovati ga tako da karakteri $\boldsymbol{x}$ i $\boldsymbol{y}$ budu u susednim listovima (maksimalne dužine) stabla koji dele nadređeni čvor, što direktno implicira da im kodne reči imaju istu dužinu i da se razlikuju za samo jedan bit

… KOREKTNOST HAFMANOVOG ALGORITAM …

- Dokaz za pohlepan izbor:
  - Neka su $\boldsymbol{a}$ i $\boldsymbol{b}$ dva proizvoljna karaktera iz $\boldsymbol{C}$ koji se nalaze u susednim listovima stabla koji dele nadređeni čvor i koji su maksimalne dužine stabla $\boldsymbol{T}$
  - Bez gubitka opštosti, može se pretpostaviti da je $\boldsymbol{a.freq \leq b.freq}$ i $\boldsymbol{x.freq \leq y.freq}$
  - Kako su $\boldsymbol{x}$ i $\boldsymbol{y}$ karakteri iz $\boldsymbol{C}$ sa najmanjom frekvencijom pojavljivanja, $\boldsymbol{a}$ i $\boldsymbol{b}$ dva proizvoljna karaktera iz $\boldsymbol{C}$, tada važi $\boldsymbol{x.freq \leq a.freq}$ i $\boldsymbol{y.freq \leq b.freq}$
  - Moguće je $\boldsymbol{x.freq = a.freq}$ i/ili $\boldsymbol{y.freq = b.freq}$, ali ako je $\boldsymbol{x.freq = b.freq}$, onda iz toga sledi da $\boldsymbol{a.freq = b.freq = x.freq = y.freq}$ i dokaz je trivijalan, tako da se pretpostavlja da je $\boldsymbol{x.freq \neq b.freq}$, što znači $\boldsymbol{x \neq b}$

… KOREKTNOST HAFMANOVOG ALGORITAM …

- Dokaz za pohlepan izbor (nastavak):
  - U sledećim koracima menja se mesto čvorova a i x iz stabla T tako da nastane stablo T', a zatim i mesto čvorova b i y iz stabla T' tako da nastane stablo T''



  - a i b se nalaze u susednim listovima stabla koji dele nadređeni čvor i koji su maksimalne dužine stabla T
  - a x i y su karakteri sa najmanjom težinom
  - ako je $x \neq b$, onda je moguće zameniti a i x, a posle i b i y
  - zamenom se ne menja ukupna cena, tako da je stablo T'' optimalno

… KOREKTNOST HAFMANOVOG ALGORITAM …

- Dokaz za pohlepan izbor (nastavak):
  - Na osnovu jednačine $\boldsymbol{B(T) = \sum_{c \in C} c.freq \cdot d_T(c)}$ može se uporediti cena stabla $\boldsymbol{T}$ i $\boldsymbol{T'}$

$$\boldsymbol{B(T) - B(T') = \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c)}$$
$$\boldsymbol{= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_{T'}(x) - a.freq \cdot d_{T'}(a)}$$
$$\boldsymbol{= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_T(a) - a.freq \cdot d_T(x)}$$
$$\boldsymbol{= (a.freq - x.freq) \cdot \big(d_T(a) - d_T(x)\big)}$$
$$\boldsymbol{\geq 0}$$

  - I $\boldsymbol{(a.freq - x.freq)}$ i $\boldsymbol{\big(d_T(a) - d_T(x)\big)}$ su nenegativne vrednosti
  - Zašto?
  - $\boldsymbol{x.freq}$ je najmanja vrednost, a $\boldsymbol{d_T(a)}$ list maksimalne dužine u $\boldsymbol{T}$

… KOREKTNOST HAFMANOVOG ALGORITAM …

- Dokaz za pohlepan izbor (nastavak):
  - Na sličan način kao što je pokazano da je $\boldsymbol{B}(\boldsymbol{T}') \leq \boldsymbol{B}(\boldsymbol{T})$ se može pokazati i da je $\boldsymbol{B}(\boldsymbol{T}'') \leq \boldsymbol{B}(\boldsymbol{T}')$, tj. da je $\boldsymbol{B}(\boldsymbol{T}'') \leq \boldsymbol{B}(\boldsymbol{T})$
  - E sad kako je $\boldsymbol{T}$ optimalno, onda je i $\boldsymbol{B}(\boldsymbol{T}) \leq \boldsymbol{B}(\boldsymbol{T}'')$ što implicira da je $\boldsymbol{B}(\boldsymbol{T}'') = \boldsymbol{B}(\boldsymbol{T})$
  - Dakle, $\boldsymbol{T}''$ je takođe optimalno stablo u kojoj se $\boldsymbol{x}$ i $\boldsymbol{y}$ listovi koji dele nadređeni čvor i koji su maksimalne dužine, što dokazuje početnu lemu
  - Ova lema implicira da se izgradnja optimalnog stabla spajanjem može realizovati pohlepni izborom kojim se za kandidate spajanja biraju dva čvora najmanje frekvencije
  - Zašto je ovo pohlepni izbor?
  - Ako se zbir frekvencija čvorova koji se spajaju posmatra kao cena spanja, onda uvek biramo najmanju cenu za spajanje – pohlepno!

… KOREKTNOST HAFMANOVOG ALGORITAM …

- Sledeća lema zapravo pokazuje da problem izgradnje optimalnog prefiksnog koda zapravo ima osobine optimalne podstrukture:
  - Neka je $\boldsymbol{C}$ alfabet gde svaki karakter $\boldsymbol{c} \in \boldsymbol{C}$ ima frekvenciju pojavljivanja $\boldsymbol{c.freq}$
  - Neka su $\boldsymbol{x}$ i $\boldsymbol{y}$ karakteri iz $\boldsymbol{c} \in \boldsymbol{C}$ sa najmanjom frekvencijom pojavljivanja
  - Neka je $\boldsymbol{C'}$ alfabet iz kojeg su izbačeni karakteri $\boldsymbol{x}$ i $\boldsymbol{y}$ i neka su oni zamenjeni novim karakterom $\boldsymbol{z}$ tako da je $\boldsymbol{C' = C - \{x, y\} \cup \{z\}}$
  - Frekvencije pojavljivanja svih karaktera u $\boldsymbol{C'}$ su iste kao i u $\boldsymbol{C}$, jedino je $\boldsymbol{z.freq = x.freq + y.freq}$
  - Neka je $\boldsymbol{T'}$ neko stablo koje predstavlja optimalni prefiksni kod za alfabet $\boldsymbol{C'}$
  - Onda stablo $\boldsymbol{T}$, koje nastaje iz stabla $\boldsymbol{T'}$ zamenom lista $\boldsymbol{z}$ sa čvorom koji za listove ima $\boldsymbol{x}$ i $\boldsymbol{y}$, predstavlja optimalni prefiksni kod za alfabet $\boldsymbol{C}$

… KOREKTNOST HAFMANOVOG ALGORITAM …

- Dokaz za osobinu optimalne podstrukture:
  - Prvo se cena gradnje $\boldsymbol{B(T)}$ stabla $\boldsymbol{T}$ iskaže kroz cenu gradnje $\boldsymbol{B(T')}$ stabla $\boldsymbol{T'}$ preko jednačine $\boldsymbol{B(T) = \sum_{c \in C} c.freq \cdot d_T(c)}$
  - Za svaki karakter $\boldsymbol{c \in C - \{x, y\}}$ važi da je $\boldsymbol{d_T(c) = d_{T'}(c)}$, a samim tim je i $\boldsymbol{c.freq \cdot d_T(c) = c.freq \cdot d_{T'}(c)}$
  - Kako je $\boldsymbol{d_T(x) = d_T(y) = d_{T'}(z) + 1}$ imamo da je:

$$\begin{aligned} & \boldsymbol{x.freq \cdot d_T(x) + y.freq \cdot d_T(x)} \\ & \boldsymbol{= (x.freq + y.freq) \cdot (d_{T'}(z) + 1)} \\ & \boldsymbol{= (x.freq + y.freq) \cdot d_{T'}(z) + (x.freq + y.freq)} \\ & \boldsymbol{= z.freq \cdot d_{T'}(z) + (x.freq + y.freq)} \end{aligned}$$

  - Odavde može da se zaključi da je:

$$\boldsymbol{B(T) = B(T') + (x.freq + y.freq)}$$

… KOREKTNOST HAFMANOVOG ALGORITAM …

- Dokaz za osobinu optimalne podstrukture (nastavak):
  - Odavde može da se zaključi da je:

    $$\boldsymbol{B(T) = B(T') + (x.freq + y.freq)}$$

  - Ili da je:

    $$\boldsymbol{B(T') = B(T) - (x.freq + y.freq)}$$

  - Dalje se lema dokazuje kontradikcijom
  - Pretpostaviti da $\boldsymbol{T}$ stablo ne reprezentuje optimalni prefiksni kod
  - Onda mora postojati neko stablo $\boldsymbol{T''}$ takvo da je $\boldsymbol{B(T') \leq B(T)}$
  - Bez gubitka opštosti, može se pretpostaviti da su $\boldsymbol{x}$ i $\boldsymbol{y}$ čvorovi u $\boldsymbol{T''}$ koji dele nadređeni čvor

… KOREKTNOST HAFMANOVOG ALGORITAM

- Dokaz za osobinu optimalne podstrukture (nastavak):
  - Pretpostaviti da postoji i stablo T''' koje je identično stablu T'' samo što je kod njega nadređeni čvor čvorovima x i y zamenjen čvorem z čija frekvencija je $z.freq = x.freq + y.freq$, onda je:

$$B(T''') = B(T'') - (x.freq + y.freq)$$
$$= B(T'') - x.freq - y.freq$$
$$< B(T) - x.freq - y.freq$$
$$= B(T')$$

  - Što je kontradikcija sa pretpostavkom da je T' optimalni prefiksni kod za C'
  - Dakle, T je optimalni prefiksni kod za C
- Na osnovu ove dve leme se zaključuje da Hafmanov algoritam generiše optimalan prefiksni kod