Príklady z predmetu IV100: Distribuované výpočty

1 Na prednáške sme ukazovali, že neexistuje deterministický algoritmus na problém dohody medzi dvoma procesormi v modeli so stratou správ (v skratke: máme dva procesy so známymi identifikátormi pracujúce v synchrónnom režime, každý z nich má vstupnú hodnotu 0/1; cieľom je napísať protokol, v ktorom po fixnom počte krokov sa oba procesy vždy rozhodnú na tú istú hodnotu, pričom ak majú na začiatku oba 0, musia sa dohodnúť na 0 a ak majú na začiatku oba 1 a žiadna správa sa nestratí, musia sa dohodnúť na 1). Modifikujme model tak, že správy sa síce môžu strácať ľubovoľne, ale v žiadnom kole sa nestratia všetky poslané správy. Dá sa problém dohody pre dva procesory deterministicky riešiť v takto modifikovanom modeli?

---

2 Graf G s n vrcholmi nazveme *takmer úplný*, ak vznikne z úplného grafu K_n odobratím nanajvýš 42 hrán. Popíšte, ako by pracoval asymptoticky optimálny algoritmus na voľbu šéfa v takmer úplných grafoch (v základnom modeli, t.j. asynchrónne, bez zmyslu pre orientáciu, s identifikátormi, ...) a zdôvodnite jeho optimalitu.

---

3 Navrhnite asynchrónny algoritmus pre voľbu šéfa na d-rozmernej hyperkocke, ktorý pracuje s počtom správ $O(n)$, kde $n = 2^d$. Prepokladajte, že hyperkocka má zmysel pre orientáciu.

---

4 Uvažujme model byzantínskych chýb z prednášky: procesy majú identifikátory, komunikujú v synchrónnom režime, môžu posielať správy každý každému a poznajú identifikátory susedov (t.j. proces vie, že správa, ktorá mu prišla je od procesu s identifikátorom x). Na začiatku sú zobudené všetky procesy. V systéme môže byť f chybných procesov, ktorých správanie nie je nijak obmedzené (môžu posielať úplne hociaké správy). Proces s minimálnym identifikátorom (všetci vedia, ktorý to je, lebo poznajú všetky identifikátory) je "generál", ktorý má vstupnú hodnotu $x \in \{0, 1\}$. Chceme riešiť nasledovný problém: V konečnom čase každý dobrý proces v skončí s hodnotou $x_v \in \{0, 1\}$, pričom všetky dobré procesy skončia s rovnakou hodnotou a ak je generál dobrý, je $x_v = x$ pre každý dobrý proces v, t.j. všetci "poslúchnu" generála. (Ak je generál chybný, dobrí sa môžu dohodnúť hociako, ale musia povedať všetci to isté). Zdôvodnite, aké veľké môže byť f (v závislosti od n), aby úloha bola riešiteľná? Navrhnite algoritmus pre prípustné hodnoty f.

---

5 Uvažujme $n > 2$ procesov spojených obojsmernými linkami do kruhu. Procesy štartujú naraz, pričom na začiatku má každý proces i vstupnú hodnotu $x_i \in \{0, 1\}$. Procesy nemajú identifikátory. Cieľom je zistiť (tak, že každý proces v konečnom čase zastane s odpoveďou `0` alebo `1`), či je v kruhu aspoň 42 procesov so vstupnou hodnotou 1.

Uvažujme dva prípady:

- procesy poznajú n a pracujú asynchrónne
- procesy nepoznajú n a pracujú synchrónne

Analyzujte riešiteľnosť úlohy v oboch prípadoch (t.j. ak je úloha riešiteľná, popíšte algoritmus; ak nie je riešiteľná, dokážte).

---

6 Nájdite čo najlepší horný odhad počtu krokov potrebných na routovanie permutácie pomocou greedy "farthest-to-go" algoritmu na stromoch.

Sieť má topológiu ľubovoľného stromu, t.j. medzi každými dvoma vrcholmi existuje práve jedna cesta. Na začiatku má každý uzol jeden paket a každý uzol je cieľom jedného paketu. Algoritmus je rovnaký ako bol na prednáške: začína sa naraz a postupuje sa v synchrónnych krokoch, pričom pakety sa presúvajú po najkratších (jediných) cestách a spomedzi paketov čakajúcich na jednej linke sa v každom kroku vyberie ten, ktorého cieľ je najvzdialenejší.

Horný odhad $O(f(n))$ znamená, že pre ľubovoľný strom s n vrcholmi a ľubovoľné rozloženie cieľov paketov algoritmus skončí po najviac $O(f(n))$ krokoch.

---

7 Dá sa riešiť problém voľby šéfa v silno súvislých orientovaný grafoch? Orientované grafy sú také, že správa môže ísť po linke iba jedným smerom. Orientovaný graf je silno súvislý, ak existuje orientovaná cesta medzi každými dvoma vrcholmi.

Návody a rady k riešeniam

---

1 Dá sa. Predpokladajme, že máme dva procesy spojené linkou. Zoberme si nasledovný jednoduchý dvojkolový algoritmus, ktorý budú symetricky vykonávať obidva procesy: v prvom kole pošli svoju vstupnú hodnotu. V druhom kole, ak si v prvom kole nedostal správu, nerob nič, inak pošli svoju hodnotu. Platí, že po dvoch kolách každý proces pozná vstupnú hodnotu druhého procesu: v prvom kole poslali správu oba procesy, a teda aspoň jedna sa doručila. Ak sa doručili obe, oba procesy poznajú obe vstupné hodnoty. Ak sa doručila iba jedna správa z procesu A do procesu B, proces B pozná po prvom kole obe vstupné hodnoty. Zároveň v prvom kole dostal správu iba B, takže v druhom kole pošle správu iba B. Keďže je táto správa jediná, nestratí sa, a po druhom kole aj A pozná obe vstupné hodnoty.

---

2 Stačí si zobrať algoritmus z prednášky (algoritmus B z prílohy) a uvedomiť si, že ak sa nejaký proces dostane na level väčší ako $N/2$, tak sa tam už žiaden iný proces nemôže dostať (to priamo vyplýva z Lemy 3). Takže ak je $N > 84$, stačí bezo zmeny spustiť algoritmus B a zopakovať dôkaz z prílohy. Ostáva problém, ako zistiť, či $N > 84$ (procesy nepoznajú N, iba svoj stupeň). Vieme, že ak graf má najviac 84 vrcholov, tak má najviac $\binom{84}{2=3482}$ hrán. Takže najprv každý proces iniciuje nezávislé prehľadávanie do hĺbky, ktoré bude mať TTL (time-to-live) 2.3482. Ak sa podarí prehľadať celý graf, proces pozná maximum, inak vie, že $N > 84$. Keďže prehľadávania majú konštantný TTL, dokopy prispejú lineárne veľa správami.

Optimalita vyplýva priamo zo zadania – trieda takmer úplných grafov pokrýva aj úplné grafy.

---

3 Majme n-rozmernú hyperkocku. Nech $v = (i_{n-1}, i_{n-2}, \ldots, i_0)$ je vrchol tejto hyperkocky, potom d-rozmerná podkocka určená vrcholom v je podkocka pozostávajúca z vrcholov $(i_{n-1}, \ldots, i_{d-1}, 0, \ldots, 0)$ až $(i_{n-1}, \ldots, i_{d-1}, 1, \ldots, 1)$. Susedná podkocka tejto podkocky je tvorená vrcholmi $(i_{n-1}, \ldots, 1 - i_{d-1}, 0, \ldots, 0)$ až $(i_{n-1}, \ldots, 1 - i_{d-1}, 1, \ldots, 1)$.

Hlavná myšlienka riešenia je nasledovná: najskôr sa bude voliť šéf na 1-rozmerných podkockách, potom na 2-rozmerných, atď. Ak sa vrchol stane šéfom d-rozmernej hyperkocky, pošle správu šéfovi susednej d-rozmernej hyperkocky a čaká na správu od neho. Na základe obdržanej správy sa rozhodne, či sa stane šéfom celej $d+1$-rozmernej hyperkocky, alebo prejde do neaktívneho stavu.

Vrcholy v neaktívnom stave musia routovať správy. Toto routovanie však musí byť dosť efektívne na to, aby bol celkový počet správ lineárny. Preto si každý neaktívny vrchol bude pamätať najkratšiu cestu k vrcholu, ktorý ho "zajal", t.j. od ktorého dostal správu, po ktorej prešiel do neaktívneho stavu.

Budeme používať správy, ktoré sú 4-ice $< id, l, path, to\_go >$. Takúto správu pošle aktívny vrchol s identifikátorom id po tom, čo sa stane šéfom l-rozmernej podkocky. Táto správa musí byť doručená šéfovi susednej l-rozmernej podkocky. Premenná $path$ bude zaznamenávať, po hranách ktorých dimenzií správa prešla. Na základe toho adresát zistí, aká je najkratšia cesta k odosielateľovi správy. Premenná to_go určuje, ako treba správu routovať v najbližších krokoch.

Ak dostane neaktívny vrchol správu $< id, l, path, 0 >$, pošle ju po najkratšej ceste k vrcholu, ktorý ho zajal. Na ceste k tomuto vrcholu sa routuje iba podľa premennej to_go. Ak je tento vrchol neaktívny, opäť posiela správu po najkratšej ceste k vrcholu na vyššom leveli, až sa správa dostane

k adresátovi.

Program:

```
var active : boolean;
    id : integer;
    level, chief_path : integer;

Initialization:
  active = true;
  level = 1;
  send < id, level, 1, 0 > to 0;

procedure Forward( < attacker_id, l, path, to_go > ):
  let i-th bit is set in to_go
  send < attacker_id, l, path xor 2^i, to_go xor 2^i > to i;

On receipt < attacker_id, l, path, to_go > from q:
  if to_go <> 0 then
    Forward( < attacker_id, l, path, to_go > );
  else if not active then {l > level}
    Forward( < attacker_id, l, path, chief_path > );
  else if l > level then
    process message later
  else {l = level}
    if attacker_id > id then
      active = false;
      chief_path = path;
    else
      level = level+1;
      if level < n then
        send < id, level, 2^(level-1), 0 > to level - 1;
      else
        I am leader
```

Správnosť algoritmu je vidno z platnosti nasledujúcich tvrdení[1]:

- v každej d-rozmernej podkocke je nanajvýš 1 vrchol na leveli d
- v každej d-rozmernej podkocke je aspoň 1 vrchol na leveli aspoň d
- každý vrchol buď postúpi na ďalší level, alebo prejde do neaktívneho stavu
- každý neaktívny vrchol pozná najkratšiu cestu k vrcholu, ktorý ho zajal (k svojmu šéfovi)
- správa na leveli d chodí len v rámci d-rozmernej podkocky

Analýza zložitosti:

Nech N je počet všetkých vrcholov hyperkocky. Na leveli d je $\frac{N}{2^d}$ aktívnych vrcholov. Každý pošle správu, ktorá sa routuje na nanajvýš d^2 hopov. Spolu sa teda pošle $\sum_1^n n\frac{d^2}{2^d} = O(n)$ správ.

---

4 Cieľom je ukázať, že problém generála zo zadania a problém dohody sú ekvivalentné. Na jednej strane, ak by sme mali algoritmus, ktorý vie riešiť problém generálov, vieme pomocou neho

[1] Ktoré sa dajú dokázať indukciou vzhľadom na d.

riešiť aj problém dohody takto: postupne je každý proces i generál a oznámi ostatným svoju hodnotu. Ak je dobrý, všetci dobrí sa zhodnú na jeho hodnote. Ak je chybný, všetci dobrí sa dohodnú na nejakej (rovnakej) hodnote. Nakoniec si všetci urobia hlasovanie – dá sa ukázať, že to je správny výsledok.

Na druhej strane, ak by sme mali algoritmus pre problém dohody, tak vieme riešiť problém generála: v prvom kole generál pošle všetkým svoju hodnotu. Potom spustíme algoritmus na problém dohody.

---

5 V prvom prípade je úloha riešiteľná: každý proces pošle správu s TTL n. Správa obieha po kruhu a pamätá si, koľko jednotiek stretla. Keď obehne n krokov, prišla do procesu, ktorý ju vyslal, takže tento vie odpoveď.

V druhom prípade úloha riešiteľná nie je. Predpokladajme, že by existoval taký algoritmus. Zoberieme nejaké fixné n a pustíme ho na kruhu veľkosti n su samými nulami. Algoritmus bude bežať r kôl a potom všetky procesy terminujú s odpoveďou 'nie'. Zoberme teraz väčší kruh, ktorý bude mať blok $3nr$ núl a blok 42 jednotiek. Proces, ktorý je v strede bloku núl, dostáva počas prvých r kôl rovnaké správy ako v predchádzajúcom výpočte, takže po r kolách terminuje s odpoveďou 'nie', čo je chyba.

---

6 Zoberme si hociakú hranu $u \mapsto v$ a ukážme, že všetky pakety, ktoré cez ňu chcú prejsť, cez ňu aj skutočne prejdú po najviac $n-1$ krokoch. Predstavme si, že by každý vrchol z časti pred u chcel poslať paket do časti za v. V skutočnosti to tak nemusí byť, ale to našu situáciu iba zlepší. Stačí indukciou ukázať, že do u budú pakety prichádzať dosť rýchlo na to, aby v každom kroku mohol niekto prejsť po hrane $u \mapsto v$.

---

7 Dá sa. Keďže nás netrápi počet správ, stačí zobrať algoritmus KKM z prednášky, overiť, že nikde nevyužíva obojsmernosť a použiť hociaké traverzovanie (napr. modifikované "s pravou rukou pri stene"), ktoré funguje v orientovaných grafoch.

Príloha: algoritmus na voľbu šéfa v úplných grafoch

Majme N procesov zapojených do úplného grafu. Každý z nich má jednoznačné ID a vie rozlíšiť medzi svojimi linkami. Okrem rôznych ID sú procesy identické a pracujú asynchrónne. Na začiatku je zobudených niekoľko (aspoň jeden) procesov. Keď proces dostane správu, zobudí sa. Uvažujeme obojsmerné full-duplex FIFO linky. Každá poslaná správa v konečnom čase príde do cieľa. Algoritmus je zapísaný tak, že pri zobudení (spontánnom alebo ľubovoľnou správou) sa najprv vykoná sekcia Init a potom sa vykonáva kód špecifikovaný v sekcii Code. Pri príchode správy nastane "prerušenie" a spracuje sa príslušný odsek On receipt; potom sa pokračuje vo vykonávaní v sekcii Code alebo (pomocou príkazu **goto**) v inej sekcii. Pre určenie časovej zložitosti predpokladáme, že doručenie správy po linke trvá maximálne 1 časovú jednotku a všetky interné výpočty trvajú 0 časových jednotiek.

Algoritmus A

Naivný algoritmus, v ktorom každý vrchol pošle všetkým ostatným správu, že chce byť šéf a čaká, či mu to dovolia. Iba procesu s najvyšším ID všetky ostatné procesy schvália kandidatúru. Algoritmus pre proces vo vrchole v:

const: N : integer
ID : integer
$Neigh$: [1...N-1] link
var: $leader$: boolean
$count$: integer
i : integer

Init:

$count := 0$
$leader :=$ **false**

Code:

for $i = 1$ **to** $N - 1$ **do**
  send $\langle$**elect**$, ID\rangle$ **to** $Neigh[i]$
while $count < N - 1$ **wait**
for $i = 1$ **to** $N - 1$ **do**
  send $\langle$**leader**$, ID\rangle$ **to** $Neigh[i]$
$leader :=$ **true**

On receipt $\langle$**elect**$, id_i\rangle$ from $Neigh[i]$:

if $id_i > ID$ **send** $\langle$**accept**$\rangle$ **to** $Neigh[i]$

On receipt $\langle$**accept**$\rangle$ from $Neigh[i]$:

$count++$

On receipt $\langle$**leader**$, id_i\rangle$ from $Neigh[i]$:

Skonči algoritmus

Aby sme dokázali správnosť, treba ukázať, že v ľubovoľnom výpočte sa práve jeden proces stane šéfom. Predpokladajme sporom, že sa ani jeden nestane šéfom. Každý zobudený proces pošle správy všetkým ostatným procesom, teda aj procesu s najvyšším ID a tým ho zobudí (ak už nebol zobudený). Proces s najvyšším ID pošle **elect** všetkým ostatným procesom a dostane naspäť všetky **accept** správy (lebo žiaden z procesov neskončil – to by musel dostať správu **leader** alebo sa sám vyhlásiť za šéfa) a vyhlási sa za šéfa. Ďalej predpokladajme, že sa za šéfa vyhlásia aspoň dvaja. V tom prípade musel ten s menším ID niekedy dostať **accept** správu od toho s väčším ID, čo je spor.

Po každej hrane sa posielajú v jednom smere maximálne 3 správy, teda počet správ je $O(n^2)$. Proces s maximálnym ID dostane za 2 časové jednotky naspäť všetky **accept** správy, celková zložitosť je teda $O(1)$.

Algoritmus B

Mierne rafinovanejší algoritmus, v ktorom sa každý proces snaží získavať povolenia sekvenčne. Pri porovnávaní sa neberie do úvahy iba ID, ale aj "level" (t.j. počet porazených procesov). Proces posiela správy **capture** a vždy čaká na odpoveď **accept**. Tá však príde iba vtedy, keď vyhral. Odpoveď na správu **capture** závisí od toho, či už proces bol raz porazený. Ak nie, rozhoduje veľkosť $[Level, id]$ (lexikograficky). Ak áno, porovnanie nerobí on, ale jeho "rodič" (t.j. proces, ktorý ho poslednýkrát "zajal"). Algoritmus pre proces vo vrchole v:

const: N : integer
ID : integer
$Neigh$: [1...N-1] link
var: $leader$: boolean
$state$: {active, captured, killed}
$level$: integer
$parent$: link
msg : {victory, defeat}
i : integer

Init:

$state :=$ active
$level := 0$
$leader :=$ **false**

Code:

for $i = 1$ **to** $N - 1$ **do**
  send $\langle$**capture**, $[level, ID]\rangle$ **to** $Neigh[i]$
  receive $\langle$**accept**$\rangle$ **from** $Neigh[i]$
  $level++$
$leader :=$ **true**
for $i = 1$ **to** $N - 1$ **do**
  send $\langle$**leader**, $ID\rangle$ **to** $Neigh[i]$

Dead:

loop forever

On receipt $\langle$**capture**, $[level_i, id_i]\rangle$ from $Neigh[i]$:

if $state \in$ {active, killled} **and** $[level_i, id_i] > [level, ID]$
  $state :=$ captured
  $parent := Neigh[i]$
  send $\langle$**accept**$\rangle$ **to** $parent$
  goto Dead
else if $state =$ captured
  send $\langle$**help**, $[level_i, id_i]\rangle$ **to** $parent$
  receive msg **from** $parent$
  if $msg =$ defeat
    send $\langle$**accept**$\rangle$ **to** $Neigh[i]$
    $parent := Neigh[i]$

On receipt $\langle$**help**, $[level_i, id_i]\rangle$ from $Neigh[i]$:

if $[level_i, id_i] < [level, ID]$
  send $\langle$victory$\rangle$ **to** $Neigh[i]$
else
  send $\langle$defeat$\rangle$ **to** $Neigh[i]$
  if $state =$ active
    $state :=$ killed
    goto Dead

On receipt $\langle$**leader**, $id_i\rangle$ from $Neigh[i]$:

Skonči algoritmus

Opäť najprv ukážeme, že v ľubovoľnom výpočte sa práve jeden proces stane šéfom. Dokážeme takúto lemu:

Lema 1 *V ľubovoľnom výpočte existuje pre každý level* $l = 0, ..., N - 1$ *aspoň jeden proces, ktorý bol počas výpočtu na leveli* l.

Dôkaz: Pre $l = 0, 1$ je tvrdenie triviálne. Ďalej postupujme sporom. Zoberme maximálny level l taký, že v priebehu výpočtu bol nejaký proces na leveli l ale žiaden proces nebol na leveli $l + 1$. Zoberme proces v s maximálnym ID i spomedzi procesov na leveli l. Keďže v nepostúpil o level, nastala jedna z troch možností: niekto mu zmenil stav na "captured" (resp. "killed") alebo v poslal **capture** a nedostal odpoveď. Zmeniť stav je možné iba správami **capture** alebo **help**, ak obsahujú väčší (lexikograficky) proces. To je ale spor s tým, že v je (v tomto usporiadaní) maximálny. Takže v poslal správu **capture** povedzme procesu v'. Ak v' bol aktívny alebo zabitý, pošle okamžite odpoveď **accept** (lebo je lexikograficky menší). Ak v' je zajatý, opýta sa svojho rodiča; ten je však podľa predpokladu tiež lexikograficky menší ako v, takže v postúpi o level. □

Keďže v priebehu výpočtu existoval (apoň jeden) proces s levelom $N-1$, aspoň jeden proces sa vyhlási za šéfa. Teraz dokážeme ešte jednu lemu:

Lema 2 *Nech v je aktívny proces (state =* active*) s levelom l. Potom existuje l zajatých procesov ktoré patria v (t.j. ich premenná parent ukazuje na v).*

Dôkaz: Proces postúpi o level iba vtedy, keď dostal **accept**. Poslanie správy **accept** je vždy doprevádzané zmenou stavu na "captured" a nastavením *parent*. Takže pri každom postúpení o level sa počet zajatých procesov zväčší. Tento počet sa môže zmenšiť iba ak si niektorý z nich nastaví *parent* inam. Z kódu ale vyplýva, že potom v prejde do stavu "killed" a nie je viac aktívny. □

Z uvedenej lemy vyplýva, že na leveli $N-1$ môže byť počas celého výpočtu najviac jeden proces. (dôkaz: zoberme prvý proces, ktorý sa dostane na level $N-1$; všetky zvyšné procesy sú v stave "captured" a na menšom leveli; keďže v stave "captured" vždy vykonávajú sekciu Dead, žiaden z nich už nikdy viac nepostúpi o level). Takže za šéfa sa vyhlási práve jeden proces.

Ideme ukazovať počet správ. Zrejme ak nejaký proces postúpi o level, spotrebuje na to konštantný počet správ (**capture**, príp. **help**+defeat, **accept**). Ak nejaký proces pošle **capture** a napriek tomu nepostúpi o level (t.j. nedostane **accept**) prestane byť aktívny a vymení sa tiež konštantý počet správ (**capture**, príp. **help**+victory). Neaktívne procesy neposielajú spontánne správy. Z toho vyplýva, že proces na každom leveli spôsobí poslanie konštantného počtu správ. Teraz dokážeme takúto lemu:

Lema 3 *V ľubovoľnom výpočte je najviac $N/(l+1)$ procesov, ktoré niekedy dosiahli level l.*

Dôkaz: Pre každý proces v, ktorý dosiahol niekedy level l označme C_v tie procesy, ktoré patrili v (t.j. boli zajaté a *parent* mali nastavené na v) vtedy, keď dosiahol level l. Zrejme $|C_v| = l$. Ukážeme, že množiny C_v sú po dvoch disjunktné. Dokážeme to indukciou na čas, kedy proces dosiahol level l. Prvý proces v_1 dosiahol level l a mal množinu C_{v_1}. Predpokladajme, že procesy $v_1, ..., v_k$ dosiahli level l a množiny $C_{v_1}, ..., C_{v_k}$ sú navzájom disjunktné. Majme proces v_{k+1}, ktorý je na leveli $l-1$ a ide postúpiť na level l. Zrejme mu teraz patrí $l-1$ procesov, ktoré nepatria do $C_{v_1} \cup ... \cup C_{v_k}$ (v_{k+1} nemohol získať žiadny proces, ktorý už raz mal rodiča na leveli l). Takisto na postup na level l nemôže v_{k+1} použiť proces z $C_{v_1} \cup ... \cup C_{v_k}$ (to by ho zabilo), takže musí pridať iný proces. □

Z uvedeného vyplýva, že v ľubovoľnom výpočte sa maximálne $\sum_{l=1}^{N-1} \frac{N}{l+1}$ krát posunie nejaký proces o level. Keďže $\sum_{l=1}^{N-1} \frac{N}{l+1} = N(\mathbf{H}_N - 1) \approx N \log N$, v celom algoritme sa použije $O(N \log N)$ správ.

Na určenie časovej zložitosti najprv ukážeme, že proces, ktorý bude zvolený za šéfa sa zobudí najneskôr $O(N)$ časových jednotiek po začiatku. Stačí ukázať, že najneskôr $O(N)$ časových jednotiek po začiatku niektorý proces postúpi aspoň na level 1 (potom už nemôže byť zvolený šéf s levelom 0). Ako dlho môže trvať situácia, že všetky procesy sú na leveli 0? Zrejme tak dlho, pokiaľ všetky **capture** správy idú do procesov s väčším ID. To zrejme môže byť najviac $N-1$ (N správ už nutne vytvorí cyklus). Takže najneskôr po $O(N)$ časových jednotkách je zobudený šéf. Šéf vždy v konštantnom čase postúpi o level, takže celková zložitosť je $O(N)$.

Poznámka: Časová zložitosť $O(N)$ ostane zachovaná, aj keď použijeme model kedy po jednej linke v jednom smere môže ísť vždy iba jedna správa. V tom prípade ale neplatí, že šéf zajme každý vrchol v konštantnom čase.