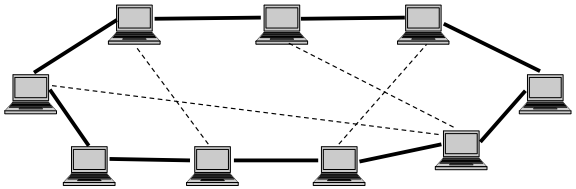
Netværksalgoritmer

Netværksalgoritmer

Netværksalgoritmer er algoritmer, der udføres på et netværk af computere

Deres udførelse er **distribueret**

Omfatter algoritmer for, hvorledes routere sender pakker igennem netværket



Ringnetværk

Routing

Formål: at sende meddelelser (pakker) imellem computere i et netværk

Mål: forsendelse skal ske
- hurtigt
- sikkert
- "retfærdigt"

Forsendelsestyper:
1. **Broadcast** - en pakke sendes til alle computere
2. **Unicast** - en pakke sende til en specifik computer
3. **Multicast** - en pakke sendes til en gruppe af computere

Meddelelses-modellen

Netværket modelleres som en graf, hvor knuder svarer til computere, og kanter svarer til faste forbindelser imellem computere

Hver kant muliggør forsendelse af en meddelelse imellem de to computere, der svarer til kantens endepunkter

Hver computer har en unik identifikator (f.eks. en IP-adresse)

Hver computer kender sine naboer og kan kun kommunikere direkte med dem

I det følgende antages, at netværket er **statisk**

Synkronisering

Synkron model: Computerne ”går i takt”. Hver computer har et internt ur, som er synkroniseret med alle ure i de øvrige computere. Det antages, at enhver operation tager samme tid på alle computere, og at det tager samme tid at sende en meddelelse igennem en forbindelse

Asynkron model: Computerne behøver ikke at arbejde med samme hastighed. Igangsættelse af trin er bestemt af betingelser, hændelser (ikke af et ur). Det antages, at alle meddelelser modtages i samme rækkefølge, som de er sendt

Kompleksitetsmål

Antallet af runder: Det globale antal runder. I den synkrone model er en runde bestemt af urets takstslag. I den asynkrone model igangsættes en runde ofte ved at udsende en “bølge” af meddelelser igennem netværket

Plads: Kan være angivet *globalt* eller *lokalt*

Køretid: Oftest analyseres den *lokale* køretid

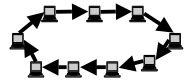
Meddelelseskompleksitet: Det totale antal meddelelser, eller den samlede længde (f.eks. målt i antal ord) af disse meddelelser

Kompleksitetsmål (fortsat)

Kompleksiteten udtrykkes ofte som funktion af følgende parametre:

1. Længden af input (målt i ord)
2. Antallet af computere
3. Antallet af forbindelser

Lederudvælgelse i en ring



Givet: et orienteret ringnetværk bestående af *n* processorer (grafen er en cykel)

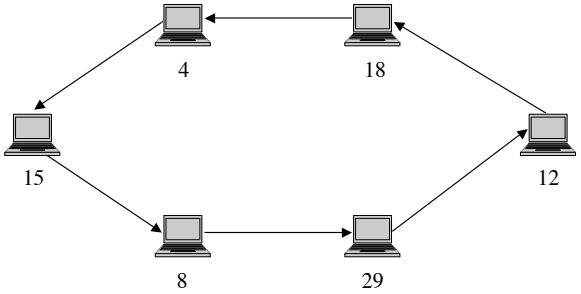
Mål: identificér en af processorerne som “leder” og meddel resultatet til alle processorer. Lederen kan f.eks. være den af processorerne, der har mindst identifikator

Formål: mange distribuerede opgaver løses simplest, hvis der er udpeget en leder blandt processorerne

Synkron løsning

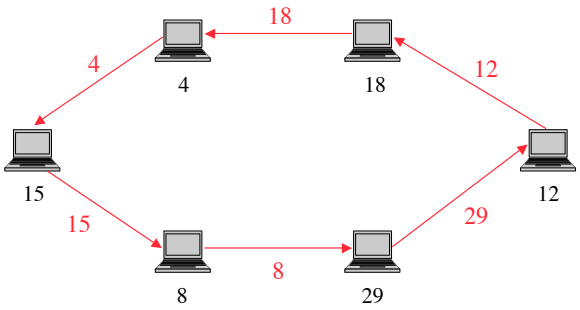
- I første runde sender hver processor sin identifikator til sin efterfølger
- I de efterfølgende runder udfører hver processor følgende:
  1. Modtag en identifikator fra forgængeren
  2. Sammenlign med egen identifikator
  3. Send den mindste af disse to videre til efterfølgeren
- Hvis en processor modtager sin egen identifikator, må den have den mindste identifikator af alle, og den må derfor være lederen.
  Send besked herom til alle de øvrige processorer

Visualisering af algoritmen



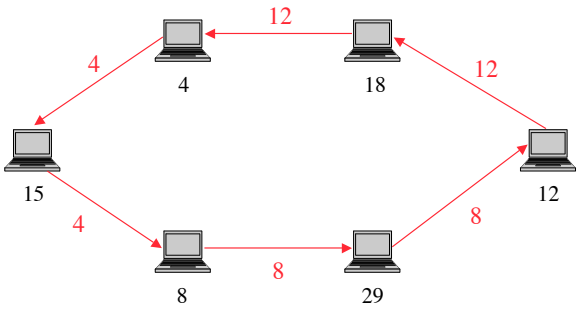
runde 0

Visualisering af algoritmen



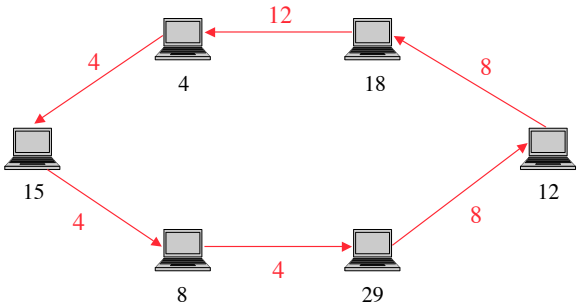
runde 1

Visualisering af algoritmen



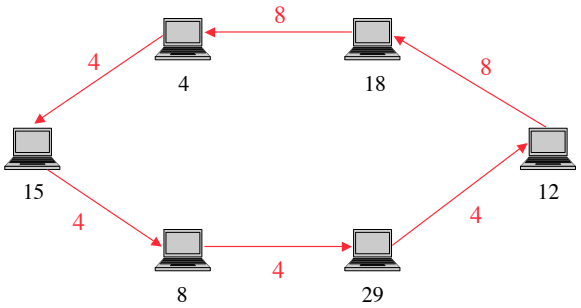
runde 2

Visualisering af algoritmen



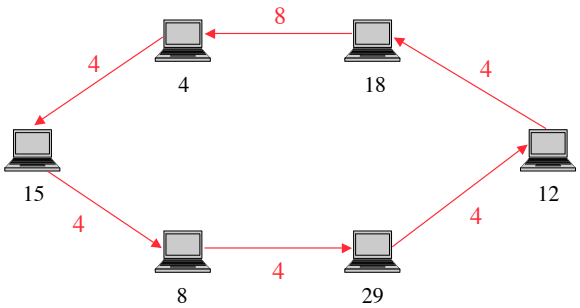
runde 3

Visualisering af algoritmen



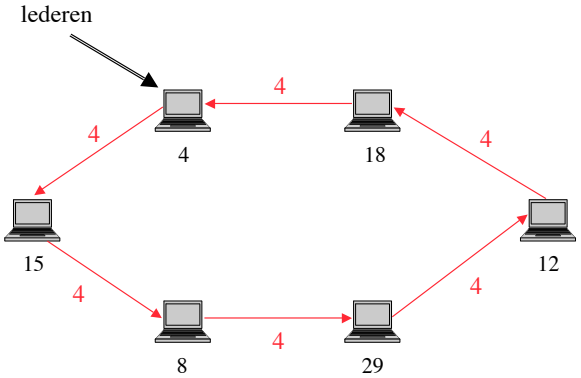
runde 4

Visualisering af algoritmen



runde 5

Visualisering af algoritmen



runde 6

```
Algorithm RingLeader(id)
    Input The unique identifier id for the processor running this algorithm
    Output The smallest identifier of a processor in the ring
    M ← [Candidate is id]
    Send message M to the successor processor in the ring
    done ← false
    repeat
        Receive message M from the predecessor in the ring
        if M = [Candidate is i] then
            if i = id then
                M ← [Leader is id]
                done ← true
            else
                m ← min{i, id}
                M ← [Candidate is m]
        else
            {M is a "Leader is" message}
            done ← true
        Send message M to the next processor in the ring
    until done
    return M        {M is a "Leader is" message"}
```


Kompleksitet

Antal runder: $2n$

Den første "Candidate is" fra lederen bruger n runder
Meddelelsen "Leader is" fra lederen bruger n runder

Antal meddelelser: $O(n^2)$

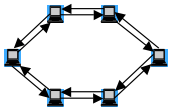
første fase: I hver af de n runder sender enhver af de n processorer en meddelelse ("Candidate is")
anden fase: Lederen udsender en "Leader is"-meddelelse. Enhver anden processor udsender en "Candidate is"-meddelelse, indtil den modtager "Leader is"-meddelelsen, som den sender videre. Antal meddelelser $1 + 2 + ... + n$, som er $O(n^2)$

Asynkron løsning

Svarer til den synkrone løsning

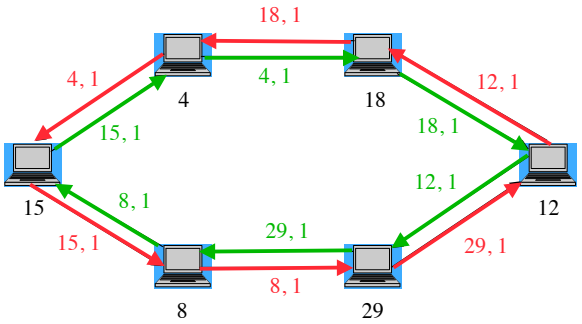
Den synkrone løsning er nemlig ikke funderet på synkronisering, kun på, at meddelelser modtages i den rækkefølge, de er sendt. Men det gælder også i en asynkron model

Løsning i en tovejs-ring



- I starten (runde 0) er alle knuder potentielle ledere
- I runde i sender alle potentielle ledere en prøvemeddelelse i retning af sine nærmeste 2^i knuder på begge sider
- Hvis knuden stadig er potentiel leder (ingen knude i omegnen har en mindre indentifikator), vil der komme besked herom fra begge sider
- Hver ny prøvemeddelelse forsynes med et *hop-tæller* med startværdi 2^i, som sænkes med 1, hver gang meddelelsen videresendes af en knude. Når hop-tælleren bliver 0, sendes besked tilbage i modsat retning

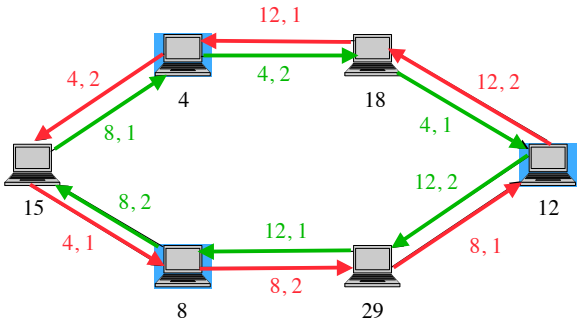
Visualisering af algoritmen



runde 0

potentiel leder

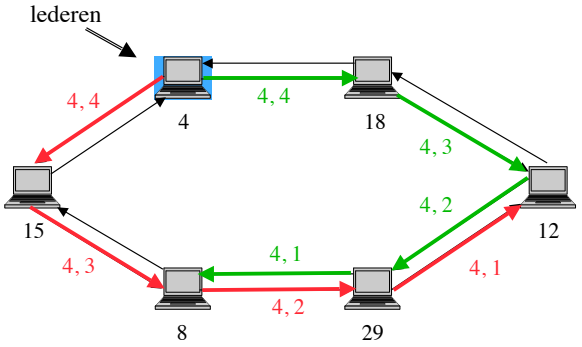
Visualisering af algoritmen



runde 1

potentiel leder

Visualisering af algoritmen



runde 2

potentiel leder

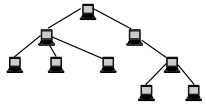
Kompleksitet

Antal runder: $O(\log n)$
antallet af potentielle ledere halveres i hver runde

Antal meddelelser: $O(n \log n)$

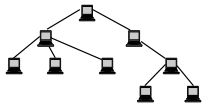
antal potentielle ledere i runde i:	$n/2^i$
antal hop fra hver potentielle leder:	$O(2^i)$
antal runder:	$O(\log n)$

Lederudvælgelse i et træ



Givet: et trænetværk bestående af n processorer (grafen er et frit træ)

Lederudvælgelse er simplere end i et ringnetværk. Vi kan starte beregningerne i de eksterne knuder



Asynkron løsning

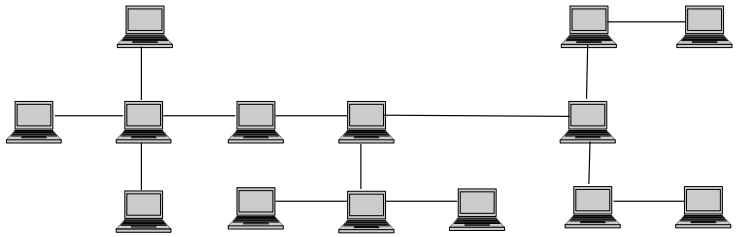
Benyt to faser:

Akkumulationsfasen: Indentifikatorer strømmer fra de eksterne knuder. Hver knude holder rede på l, minimum af dens egen identifikator og den mindste identifikator, den har modtaget. Når den har modtaget indentifikatorer fra alle sine naboer på nær en, sender den l til denne nabo. På et tidspunkt har en knude modtaget en meddelelse fra alle sine naboer. Denne knude, der kaldes **akkumulationsknuden**, bestemmer lederen.

Rundsendingsfasen: Akkumulationsknuden udsender en meddelelse, om hvilken knude, der er leder, imod de eksterne knuder

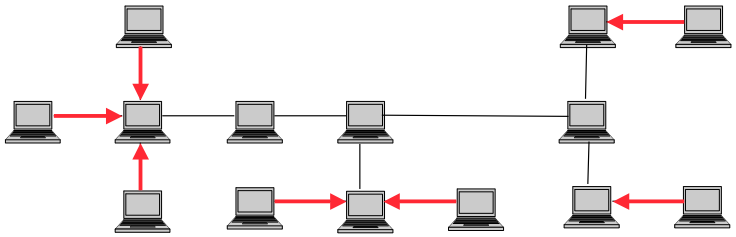
Bemærk, at to naboknuder begge kan blive akkumulationsknuder. I så fald rundsender de meddelelsen til hver deres "halvdel" af træet

Visualisering af algoritmen



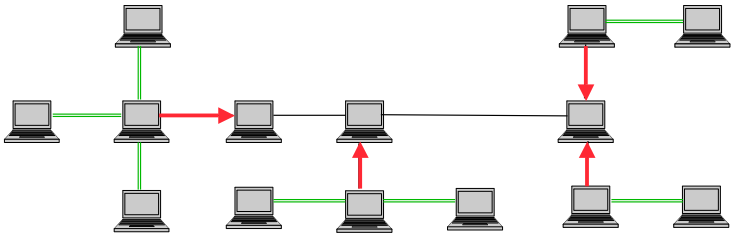
runde 0

Visualisering af algoritmen



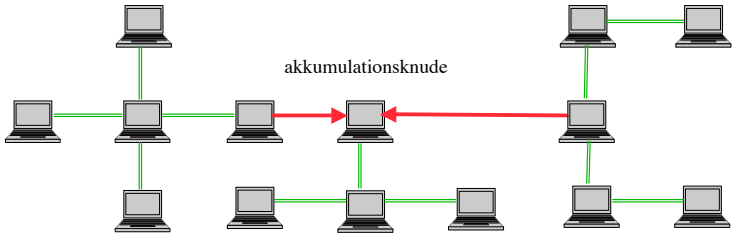
runde 1

Visualisering af algoritmen



runde 2

Visualisering af algoritmen



runde 3


```
Algorithm TreeLeader(id)
   Input The unique identifier id for the processor running this algorithm
   Output The smallest identifier of a processor in the tree
   {Accumulation phase}
   let d be the number of neighbors of processor id    {d ≥ 1}
   m ← 0        {counter for messages received}
   l ← id       {tentative leader}
   repeat
      { begin a new round }
      for each neighbor j do
         check if a message from processor j has arrived
         if message M = [Candidate is i] from j has arrived then
            l ← min{i, l}
            m ← m + 1
   until m ≥ d - 1
   if  m = d then
      M ← [Leader is l]
      for each neighbor j ≠ k do
         send message M to processor j
      return M       {M is a "Leader is" message}
   else
      M ←[Candidate is l]
      send M to the neighbor k that has not sent a message yet
```

fortsættes


```
      {Broadcast phase}
   repeat
      { begin a new round }
      check if a message from processor k has arrived
      if message M from k has arrived then
         m ← m + 1
         if M = [Candidate is i] then
            l ← min{i, l}
            M ← [Leader is l]
            for each neighbor j do
               send message M to processor j
         else
            { M is a "Leader is" message }
            for each neighbor j ≠ k do
               send message M to processor j
   until m = d
return M       {M is a "Leader is" message}
```

Kompleksitet

Antal runder: $2h$, hvor h er træets højde

Antal meddelelser: $O(n)$
- **akkumulationsfasen**: Hver processor sender en meddelelse ("Candidate is")
- **rundsendingsfasen**: Hver processor sender højst en meddelelse ("Leader is")

Synkron løsning

Svarer til den asynkrone løsning

Alle processorer begynder en ny runde på samme tid

Antallet af runder er lig med træets diameter (længden af den længste vej imellem to knuder)

Lokal køretid: $O(d_iD)$, hvor d_i er antallet af naboer, og D er grafens diameter

Lokalt pladsforbrug: $O(d_i)$

Bredde-først søgning

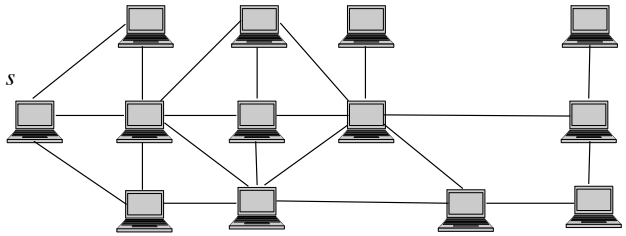
Givet: et sammenhængende netværk bestående af n processorer, hvor en knude s er udpeget som kilde

Mål: Foretag en bredde-først søgning startende i s

Synkron løsning

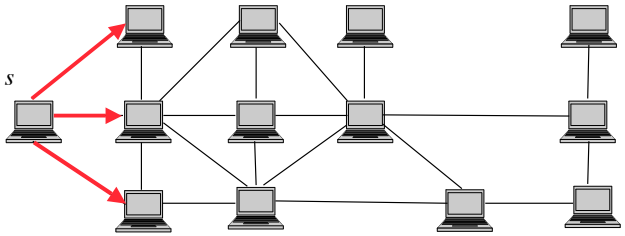
- s identificeres som en "ekstern" knude i det aktuelle BFS-træ
- I hver runde sender hver eksterne knude v en meddelelse til alle sine naboer, der endnu ikke har kontaktet v for at fortælle dem, at v gerne vil have dem som børn i BFS-træet. Disse gør v til deres forælder, hvis de ikke allerede har valgt en forælder

Visualisering af algoritmen



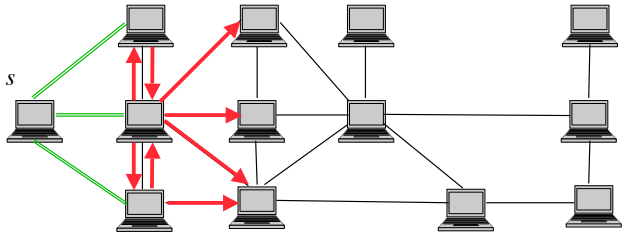
runde 0

Visualisering af algoritmen



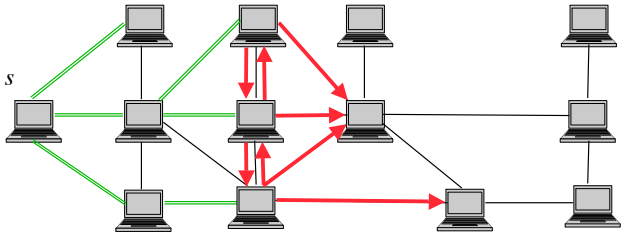
runde 1

Visualisering af algoritmen



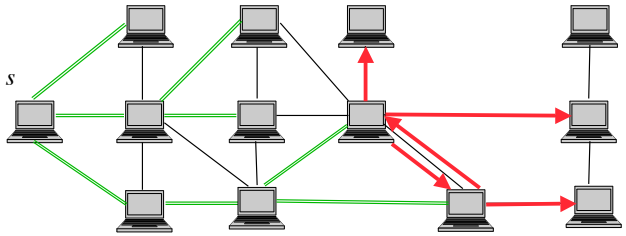
runde 2

Visualisering af algoritmen



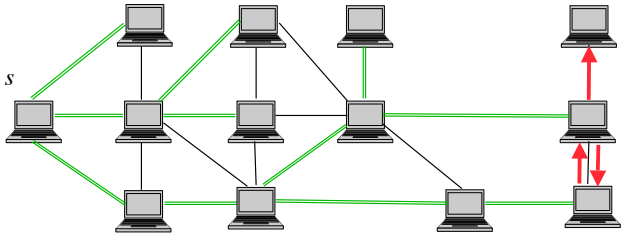
runde 3

Visualisering af algoritmen



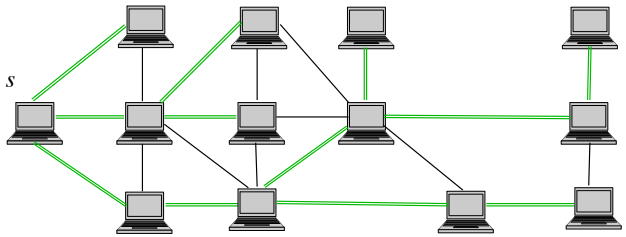
runde 4

Visualisering af algoritmen



runde 5

Visualisering af algoritmen



runde 6

Algorithm *SynchronousBFS*(v, s)
  Input The identifier v of the node (processor) executing this algorithm and the indentifier s of the start node of the BFS traversal
  Output For each node v, its parent in a BFS tree rooted at s
  repeat
    {begin a new round}
    if $v = s$ **or** v has received a message from one of its neighbors **then**
      set parent(v) to a node requesting v to become its child (or null, if $v = s$)
      for each node w adjacent to v that has not contacted v yet **do**
        send a message to w asking w to become a child of v
  until $v = s$ **or** v has received a message

Kompleksitet

Antal runder: højden af BFS-træet

Antal meddelelser: $O(n + m)$
Der sendes højst en meddelelse på hver kant

Asynkron løsning

Algoritmen opererer i et antal runder

I hver runde usender s en `pulse-down`-meddelelse til alle knuder i det aktuelle BFS træ

Når meddelelsen når til de eksterne knuder i træet, forsøger de eksterne knuder at udvide træet med endnu et niveau ved at udsende et `make-child`-meddelelse til sine kandidatbørn

Når disse kandidatbørn svarer ved enten at acceptere (`accept-child`) eller forkaste invitationen (`reject-child`), sendes en `pulse-up`-meddelelse tilbage til s, som så kan påbegynde en ny runde

Algorithm *AsynchronousBFS*(v, s, n)
Input The identifier v of the processor running this algorithm, the indentifier s of the start node of the BFS traversal, and the number n of nodes of the network
Output For each node v, its parent in a BFS tree rooted at s
$C \leftarrow \emptyset$ {verified BFS children for v}
set A to be the set of neighbors of v
repeat
{ begin a new round }
if parent(v) is defined **or** $v = s$ **then**
if parent(v) is defined **then**
wait for a `pulse-down` message from parent(v)
if C is not empty **then**
{v is an internal node in the BFS tree}
send a `pulse-down` message to all nodes in C
wait for a `pulse-up` message from all nodes in C
else
for each node u in A **do**
send a `make-child` message to u
for each node u in A **do**
get a message M from u and remove u from A
if M is an `accept-child` message **then**
add u to C
send a `pulse-up` message to parent(v)
else

fortsættes

else
{$v \neq s$ has no parent yet}
for each node w in A **do**
if w has sent a `make-child` message then
remove w from A {w is no longer a candidate child for v}
if parent(v) is undefined **then**
parent(v) $\leftarrow w$
send an `accept-child` message to w
else
send a `reject-child` message to w
until (v has received message `done`) **or** ($v = s$ and has pulsed-down n-1 times)
send a `done` message to all nodes in C

Kompleksitet

Antal runder: n - 1, den maksimale højde af BFS-træet
(for en sikkerheds skyld)

Antal meddelelser: $O(n^2)$
`accept-child` og `reject-child`: m
(1 per kant)
`pulse-down` og `push-up`: $O(n^2)$
(n - 1 runder med højst n meddelelser i hver)

Kan forbedres til O($nh + m$)
(se opgave C-11.4)