Ecrin Nazar Aybay

21253043

AN ASYNCHRONOUS SPANNİNG TREE ALGORİTHM WİTH TERMİNATİON DETECTİON

Amacı

Bu algoritma, dağıtık bir ağda, düğümler arasında asenkron iletişim kullanarak bir yayılım ağacı oluşturmayı amaçlar. Algoritma aynı zamanda ağaç oluşturma işleminin tamamlandığını da tespit edebilir (sonlandırma tespiti).

Düğüm Durumları

Her düğüm üç farklı durumda bulunabilir:

1. **IDLE (Boşta)**: Başlangıç durumu. Düğüm henüz hiçbir mesaj almamış.
2. **XPLORD (Keşfedilmiş)**: Düğüm bir PROBE mesajı almış ve ağaca dahil olmuş.
3. **TERM (Sonlanmış)**: Düğüm tüm komşularından yanıt almış ve ağaç oluşturma işlemini tamamlamış.

Mesaj Tipleri

Algoritma üç tip mesaj kullanır:

1. **PROBE (Yoklama)**: Bir düğüm tarafından gönderilen, "ağaca katılmak ister misin?" daveti.
2. **ACK (Onay)**: Bir düğümün PROBE mesajını kabul ettiğini ve alt ağacının tamamen oluşturulduğunu belirten yanıt.
3. **REJECT (Reddetme)**: Bir düğümün zaten ağaca dahil olduğunu ve PROBE mesajını reddettiğini belirten yanıt.

Algoritmanın İşleyişi

1. **Başlangıç**:
    - Tüm düğümler IDLE durumunda başlar.
    - Kök düğüm (genellikle ID=0 olan düğüm) algoritmayı başlatır.
    - Kök düğüm tüm komşularına PROBE mesajı gönderir ve XPLORD durumuna geçer.
2. **PROBE Mesajı Alımı**:
    - Eğer düğüm IDLE durumundaysa:

        - Mesajı gönderen düğümü ebeveyn (parent) olarak belirler.
        - XPLORD durumuna geçer.
        - Ebeveyn dışındaki tüm komşularına PROBE mesajı gönderir.
        - Eğer beklenecek komşu yoksa, hemen ebeveyne ACK mesajı gönderir.
    - Eğer düğüm XPLORD durumundaysa:
        - Gönderene REJECT mesajı döner (çünkü zaten ağaca dahil olmuş).
3. ACK Mesajı Alımı (XPLORD durumundaki düğüm için):
    - Gönderen düğümü çocuklar (children) listesine ekler.
    - Bekleyen yanıtlar listesinden göndereni çıkarır.
    - Görsel olarak ebeveyn-çocuk ilişkisini göstermek için bağlantıyı günceller.
    - Tüm komşulardan yanıt gelip gelmediğini kontrol eder.
4. REJECT Mesajı Alımı (XPLORD durumundaki düğüm için):
    - Gönderen düğümü diğerleri (others) listesine ekler.
    - Bekleyen yanıtlar listesinden göndereni çıkarır.
    - Tüm komşulardan yanıt gelip gelmediğini kontrol eder.
5. Tamamlanma Kontrolü:
    - Eğer bekleyen yanıt kalmadıysa:
        - Ebeveyne ACK mesajı gönderir (kök değilse).
        - TERM durumuna geçer.

Sonlandırma Tespiti

Algoritmanın önemli bir özelliği, düğümlerin ağaç oluşturma işleminin tamamlandığını yerel olarak tespit edebilmesidir. Bir düğüm, tüm komşularından yanıt aldığında ve ebeveynine ACK mesajı gönderdiğinde, kendi alt ağacının tamamen oluşturulduğunu bilir. Kök düğüm tüm komşularından ACK mesajı aldığında, tüm ağın yayılım ağacı oluşturma işleminin tamamlandığını anlar.

KOD AÇIKLAMASI

İçe Aktarma İşlemleri ve Sabitler

```
import random
import sys

sys.path.insert(1, '.')
from source import DawnSimVis
```

- random modülünü düğümlerin pozisyonlarına rastgele değerler eklemek için içe aktarıyoruz.
- sys.path.insert(1, '.') ile Python'a mevcut dizini modül arama yoluna ekliyoruz.
- DawnSimVis modülünü, ağ simülasyonu için içe aktarıyoruz.

```
# Root node that will initiate the spanning tree construction
ROOT = 0
```

- Yayılım ağacını başlatacak kök düğümün ID'sini belirliyoruz (0 numaralı düğüm).

```
# States
IDLE = "IDLE"
XPLORD = "XPLORD"
TERM = "TERM"
```

Düğümlerin sahip olabileceği üç durumu (state) tanımlıyoruz:

- IDLE: Başlangıç durumu, düğüm henüz ağaca dahil olmamış
- XPLORD: Keşfedilmiş durumu, düğüm ağaca katılmış ve komşularından yanıt bekliyor
- TERM: Sonlanmış durumu, düğüm tüm komşularından yanıt almış

```
# Message types
PROBE = "probe"
ACK = "ack"
REJECT = "reject"
```

Düğümler arasında gönderilecek üç mesaj tipini tanımlıyoruz:

- PROBE: "Ağaca katılmak ister misin?" daveti
- ACK: "Evet, ağaca katılıyorum ve alt ağacım tamam" yanıtı
- REJECT: "Hayır, zaten başka bir ebeveyne katıldım" yanıtı

```
############################################################
class Node(DawnSimVis.BaseNode):
```

- DawnSimVis.BaseNode sınıfından türetilen Node sınıfını tanımlıyoruz.

```
def init(self):
    # Initialize node state and variables
    self.currstate = IDLE  # Current state of the node
    self.parent = -1  # Parent node in the spanning tree
```

```
    self.childs = set()  # Children nodes in the spanning tree
    self.others = set()  # Other neighbors (not parent or children)
    self.pending_responses = set()  # Neighbors we're waiting to hear from

    # Color the node based on its initial state (IDLE)
    self.change_color(0.8, 0.8, 1.0)  # Light blue for IDLE
```

init() metodu, düğüm oluşturulduğunda çağrılır ve başlangıç değerlerini atar:

- self.currstate: Düğümün mevcut durumunu takip eder, IDLE ile başlar
- self.parent: Düğümün ağaçtaki ebeveynini tutar, -1 ile başlar (ebeveyn yok)
- self.childs: Düğümün ağaçtaki çocuklarını tutan küme
- self.others: Ağaçta ebeveyn veya çocuk olmayan komşuları tutan küme
- self.pending_responses: Yanıt beklenilen komşuları tutan küme
- self.change_color(0.8, 0.8, 1.0): Düğümü açık mavi renge boyar (IDLE durumu için)

```
def run(self):
    # If this is the root node, start the algorithm
    if self.id == ROOT:
        self.log("Root node starting algorithm")
        # Change state to XPLORD
        self.currstate = XPLORD
        self.change_color(0.0, 0.7, 0.0)  # Green for XPLORD

        # Broadcast probe to all neighbors
        self.broadcast_probe()
```

run() metodu, simülasyon başladığında her düğüm için çağrılır:

- if self.id == ROOT: Sadece kök düğüm (ID=0) için işlem yapar
- self.log(...): Günlüğe mesaj yazar
- self.currstate = XPLORD: Kök düğümün durumunu XPLORD olarak değiştirir
- self.change_color(0.0, 0.7, 0.0): Düğümü yeşil renge boyar (XPLORD durumu için)
- self.broadcast_probe(): Tüm komşulara PROBE mesajı gönderir

```
def on_receive(self, pck):
    msg_type = pck["type"]
    sender = pck["sender"]

    self.log(f"Received {msg_type} from {sender}, current state:
{self.currstate}")
```

on_receive() metodu, düğüm bir mesaj aldığında çağrılır:

- msg_type = pck["type"]: Mesajın tipini alır (PROBE, ACK veya REJECT)
- sender = pck["sender"]: Mesajı gönderen düğümün ID'sini alır
- self.log(...): Alınan mesajı ve mevcut durumu günlüğe kaydeder.

```
# Handle message based on current state
if self.currstate == IDLE:
```

```
    if msg_type == PROBE:
        # First time receiving a probe
        self.parent = sender
        self.log(f"Setting parent to {self.parent}")

        # Move to XPLORD state
        self.currstate = XPLORD
        self.change_color(0.0, 0.7, 0.0)  # Green for XPLORD

        # Broadcast probe to all neighbors except parent
        self.broadcast_probe()

        # If no neighbors to wait for, immediately send ACK to parent
        if len(self.pending_responses) == 0:
            self.send_ack_to_parent()
```

IDLE durumunda bir PROBE mesajı alındığında:

- self.parent = sender: Mesajı gönderen düğümü ebeveyn olarak belirler
- self.currstate = XPLORD: Durumu XPLORD'a değiştirir
- self.change_color(0.0, 0.7, 0.0): Düğümü yeşil renge boyar
- self.broadcast_probe(): Ebeveyn dışındaki tüm komşulara PROBE mesajı gönderir
- Eğer beklenecek yanıt yoksa, hemen ebeveyne ACK mesajı gönderir

```
elif self.currstate == XPLORD:
    if msg_type == PROBE:
        # Already received a probe before, reject this one
        self.send(sender, {"type": REJECT, "sender": self.id})
```

- XPLORD durumunda bir PROBE mesajı alındığında:
    - Düğüm zaten ağaca dahil olduğu için REJECT mesajı gönderir.

```
elif msg_type == ACK:
    # Add sender to children set
    self.childs.add(sender)
    self.pending_responses.remove(sender)
    self.log(f"Added {sender} to children, pending:
{self.pending_responses}")

    # Visual indication of parent-child relationship
    self.sim.scene.dellink(self.id, sender, "edge")
    self.sim.scene.addlink(self.id, sender, "prev")

    # Check if we've heard from all neighbors
    self.check_completion()
```

XPLORD durumunda bir ACK mesajı alındığında:

- self.childs.add(sender): Göndereni çocuklar kümesine ekler
- self.pending_responses.remove(sender): Göndereni bekleyen yanıtlar kümesinden çıkarır
- Ebeveyn-çocuk ilişkisini görsel olarak gösterir (standart bağlantıyı silip yönlü ok ekler)

- self.check_completion(): Tüm komşulardan yanıt alınıp alınmadığını kontrol eder

```
elif msg_type == REJECT:
        # Add sender to others set
        self.others.add(sender)
        self.pending_responses.remove(sender)
        self.log(f"Added {sender} to others, pending:
{self.pending_responses}")

        # Check if we've heard from all neighbors
        self.check_completion()

# In TERM state, we ignore all messages
```

XPLORD durumunda bir REJECT mesajı alındığında:

- self.others.add(sender): Göndereni diğerleri kümesine ekler
- self.pending_responses.remove(sender): Göndereni bekleyen yanıtlar kümesinden çıkarır
- self.check_completion(): Tüm komşulardan yanıt alınıp alınmadığını kontrol eder

```
def broadcast_probe(self):
    """Broadcast probe message to all neighbors except parent"""
    # Get all neighbors within transmission range
    neighbors = []
    for dist, node in self.neighbor_distance_list:
        if dist <= self.tx_range and node.id != self.parent:
            neighbors.append(node.id)

    # Set up pending responses
    self.pending_responses = set(neighbors)
    self.log(f"Waiting for responses from: {self.pending_responses}")

    # Broadcast probe message
    if neighbors:
        message = {"type": PROBE, "sender": self.id}
        for neighbor in neighbors:
            self.send(neighbor, message)
```

broadcast_probe() metodu:

- İletim menzili içindeki ve ebeveyn olmayan tüm komşuları bulur
- Bu komşuları bekleyen yanıtlar kümesine ekler
- Her komşuya PROBE mesajı gönderir

```
def check_completion(self):
    """Check if we've heard from all neighbors and can send ACK to
parent"""
    if not self.pending_responses:
        self.log(f"All neighbors responded: children={self.childs},
others={self.others}")
```

```
        # Send ACK to parent if not root
        self.send_ack_to_parent()

        # Move to TERM state
        self.currstate = TERM
        self.change_color(1.0, 0.5, 0.0)  # Orange for TERM
        self.log("Node terminated")
```

check_completion() metodu:

- Bekleyen yanıt olup olmadığını kontrol eder
- Eğer tüm yanıtlar alındıysa:
    - Ebeveyne ACK mesajı gönderir
    - Düğümün durumunu TERM'e değiştirir
    - Düğümü turuncu renge boyar (TERM durumu için)

```
def send_ack_to_parent(self):
    """Send ACK message to parent if not root"""
    if self.parent != -1:
        self.send(self.parent, {"type": ACK, "sender": self.id})
        self.log(f"Sent ACK to parent {self.parent}")
```

send_ack_to_parent() metodu:

- Eğer düğümün bir ebeveyni varsa, ona ACK mesajı gönderir

```
def finish(self):
    """Print final state information"""
    self.log(f"Final state: {self.currstate}")
    self.log(f"Parent: {self.parent}")
    self.log(f"Children: {self.childs}")
    self.log(f"Others: {self.others}")
```

finish() metodu, simülasyon sonunda çağrılır:

- Düğümün son durumunu (state, parent, children, others) günlüğe kaydeder

```
def create_network():
    # place nodes over 10x10 grids
    for x in range(10):
        for y in range(10):
            px = 50 + x * 60 + random.uniform(-20, 20)
            py = 50 + y * 60 + random.uniform(-20, 20)
            sim.add_node(Node, pos=(px, py), tx_range=75)
```

create_network() fonksiyonu:

- 10x10 grid üzerinde toplam 100 düğüm oluşturur
- Her düğüm için bir pozisyon hesaplar (biraz rastgele sapma ile)
- Her düğümün iletim menzilini 75 birim olarak ayarlar

```
# setting the simulation
sim = DawnSimVis.Simulator(
    duration=100,
    timescale=1,
```

```
    visual=True,
    terrain_size=(650, 650),
    title='Flooding')

# creating network
create_network()

# start the simulation
sim.run()
```

Simülasyon nesnesi oluşturulur:

- duration=100: Simülasyon 100 zaman birimi sürecek
- timescale=1: Gerçek zamanlı çalışacak
- visual=True: Görsel arayüz aktif olacak
- terrain_size=(650, 650): Simülasyon alanı 650x650 birim
- title='...': Simülasyon penceresinin başlığı
- Ağ oluşturulur ve simülasyon başlatılır

Her Adımın İşlevi Açısından Algoritma Özeti:

1. **Başlangıç**:
    - Tüm düğümler IDLE durumunda başlar.
    - Kök düğüm (ID=0) algoritmayı başlatır, XPLORD durumuna geçer ve tüm komşularına PROBE mesajı gönderir.
2. **PROBE Mesajı İşleme**:
    - IDLE durumdaki düğüm PROBE aldığında:
        - Göndereni ebeveyn olarak belirler.
        - XPLORD durumuna geçer.
        - Ebeveyn dışındaki tüm komşularına PROBE gönderir.
    - XPLORD durumdaki düğüm PROBE aldığında:
        - REJECT mesajı ile yanıt verir.
3. **ACK Mesajı İşleme** (XPLORD durumdaki düğüm için):
    - Göndereni çocuklar kümesine ekler.
    - Görsel olarak ebeveyn-çocuk bağlantısını gösterir.
    - Tüm yanıtlar alındıysa, TERM durumuna geçer.
4. **REJECT Mesajı İşleme** (XPLORD durumdaki düğüm için):

- Göndereni diğerleri kümesine ekler.
- Tüm yanıtlar alındıysa, TERM durumuna geçer.

5. Tamamlanma:
    - Tüm yanıtlar alındığında:
        - Ebeveyne ACK gönderilir.
        - TERM durumuna geçilir.
        - Düğümün rengi turuncuya değişir.