6.4 çizge algoritmaları

todo

6.4.1 BFS: Breadth First Search

todo

6.4.1.1 word ladder (kelime merdiveni) problemi

- → bulmaca, kelime oyunu
- → FOOL $\Longrightarrow$??? $\Longrightarrow$ *SAGE*

→ ama nasıl?

→ kural bir anda bir harf değiştir

→ her seferinde anlamlı bir kelime oluşsun

→ keşif: 1878, lewis Carroll (Alice in wonderland'in yazarı)

kelime merdiveni

kelime merdiveni

FOOL
POOL
POLL
POLE
PALE
SALE
SAGE

→ adım sayısı sınırlanabilir

→ amaç: en kısa yoldan hedefe (SAGE) ulaşmak

algoritma

- → kelimeler arası ilişki temsili için: **çizge**
- → başlangıç kelimesinden (FOOL), bitiş kelimesine (SAGE) en kestirme yol: **BFS**

çizge

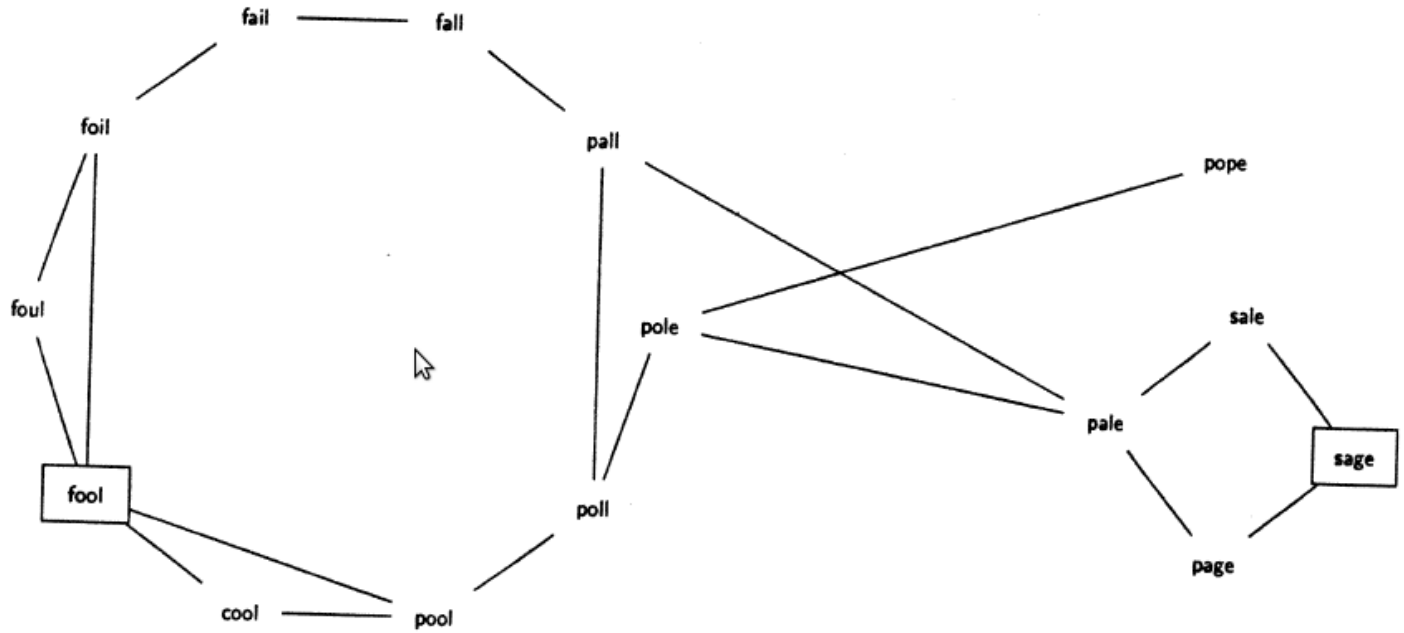


Figure 6.5: A Small Word Ladder Graph

→ kiriş: iki kelimeyi bağlar, bu kelimelerin sadece bir harfi farlıdır

FOOL --> POOL FAIL --> FALL

→ yönsüz çizgedir, kirişler ağırlıksızdır

çözüme doğru

→ elimizde aynı uzunluklu kelime **listesi** var

→ bu listeyi **çizge** ye nasıl aktaracağız?

Brute force

- → iki kelimeyi karşılaştır
- → sadece bir harfi farklıysa kiriş oluştur
- → az sayıda kelime için kolay/verimli
- → liste 5.700 kelimeden oluşuyorsa. ne yapacağız?
- → karşılaştırma maliyeti: O(n^2), n=5.700, 32 milyon karşılaştırma
- → daha pratik/efektif çözüme ihtiyacımız var

gerçekleme

```
1       def buildGraph():
2           d = {}
3           g = Graph()
4           wfile = file('words.dat')
5           # create buckets of words that differ by one letter.
6           for line in wfile:
7               word = line[0:5]
8               for i in range(5):
9                   bucket = word[0:i] + '_' + word[i+1:5]
10                  if d.has_key(bucket):
11                      d[bucket].append(word)
12                  else:
13                      d[bucket] = [word]
14          # add vertices and edges for words in the same bucket.
15          for i in d.keys():
16              for j in d[i]:
17                  for k in d[i]:
18                      if j != k:
19                          g.addEdge(j,k)
20          return g
```

açıklama

→ her bir kelime için (word)

→ bir harfini değiştir (bucket)

→ s13: bucket'i d-sözlüğüne liste olarak gir

örnek

word = “crave”

```
bucket_0 = "_rave"  --> d["_rave"] = ["crave"]
bucket_1 = "c_ave"  --> d["c_ave"] = ["crave"]
bucket_2 = "cr_ve"
bucket_3 = "cra_e"
bucket_4 = "crav_"
```

word = “brave”

```
bucket_0 = "_rave"  --> d["_rave"] = ["crave", "brave"]
bucket_1 = "b_ave"  --> d["b_ave"] = ["brave"]
bucket_2 = "br_ve"
bucket_3 = "bra_e"
bucket_4 = "brav_"
```

→ dolayısıyla d[“_xyz”] gösterimi ilk harfi farklı olanlar listesidir

→ d[“_rave”] = [“crave”, “brave”, “zrave”]

açıklama

→ satır14:19

→ buildGraph: sözlük+liste $\Longrightarrow$ ç*izge*

→ sayfa36

karşılaştırma

n=5.757 için

- → komşuluk matrisi: n^2= 33 milyon hücre
- → buildGraph ile üretilen komşuluk listesi: 28.810 kiriş
- → demek ki komşuluk matrisi 28.810/33 milyon x 100= %0.086 doludur
- → OLDUKÇA OLDUKÇA SEYREK

BFS

→ şimdi sırada en kısa sürede kelime merdivenini kurmak var

→ bunun için BFS kullanacağız

gerçekleme

gerçekleme

```
1       def bfs(g,vertKey):
2           s = g.getVertex(vertKey)
3           s.setDistance(0)
4           s.setPred(None)
5           s.setColor('gray')
6           Q = Queue()
7           Q.enqueue(s)
8           while (Q.size() > 0):
9               w = Q.dequeue()
10              for v in w.getAdj():
11                  if (v.getColor() == 'white'):
12                      v.setColor('gray')
13                      v.setDistance( w.getDistance() + 1 )
14                      v.setPred(w)
15                      Q.enqueue(v)
16              w.setColor('black')
```

açıklama

- → s2: g çizgesindeki vertKey'li s düğümüyle başla
- → s3: setDistance, adım sayacı
- → s'den k uzaklıktaki tüm düğümleri bul
- → sonra (k+1) uzaklıktakileri, k-uzaklıktakilerden bul
- → ama nasıl?
- → s5 ve s10: aynı bucket'tekiler "white", sadece s'nin ki "gray"
- → s9 ve s16: w ile işin bitince, tüm komşularını ziyaret ettiysen karart-"black"

görsellik

→ görsellik için ağaç kullan

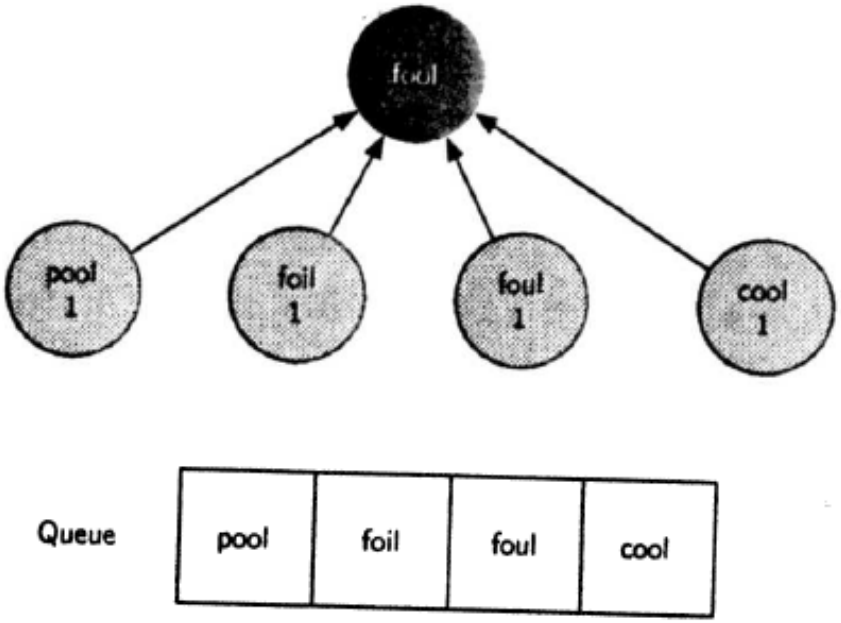


Figure 6.6: Fist Step in the Breadth First Search

→ kök, dal, çocuk?

→ k nedir?

görsellik

- ağacın çocukları bir harfi farklı kelimeler
- ilerlemeyi görselleştirmede renkleri kullan
  - *white: başlangıç rengi, henüz ziyaret edilmemiş*
  - *gray: ziyaret başlangıcı bfs:s5*
  - *black: ziyaret bitimi bfs:s16*
- komşu tüm düğümleri ziyaret ettiğinde (hiç beyaz kalmadığında) (bfs:s10–11)
- düğümü karart - black (bfs:s16)
- ziyaret devam ediyorken gray tut (bfs:s12), komşu beyazlarla çalış (bfs:s11)

gerçekleme

BFS: gerçekleme

```
1       def bfs(g,vertKey):
2           s = g.getVertex(vertKey)
3           s.setDistance(0)
4           s.setPred(None)
5           s.setColor('gray')
6           Q = Queue()
7           Q.enqueue(s)
8           while (Q.size() > 0):
9               w = Q.dequeue()
10              for v in w.getAdj():
11                  if (v.getColor() == 'white'):
12                      v.setColor('gray')
13                      v.setDistance( w.getDistance() + 1 )
14                      v.setPred(w)
15                      Q.enqueue(v)
16              w.setColor('black')
```

algoritma

→ Vertex sınıfı (listing_6_1), Graph sınıfı (listing_6_2)'den yararlanır

→ [A] sonraki ziyaret düğümünü belirlemek için kuyruk (queueu) kullanılır

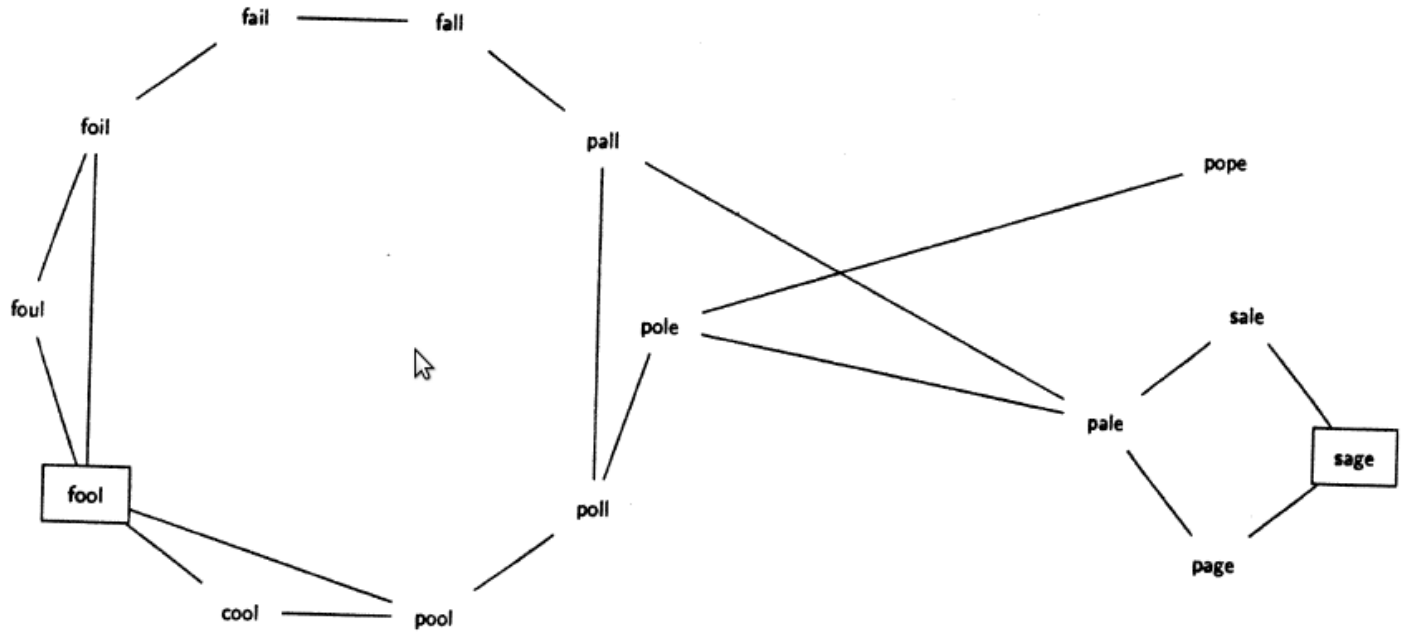
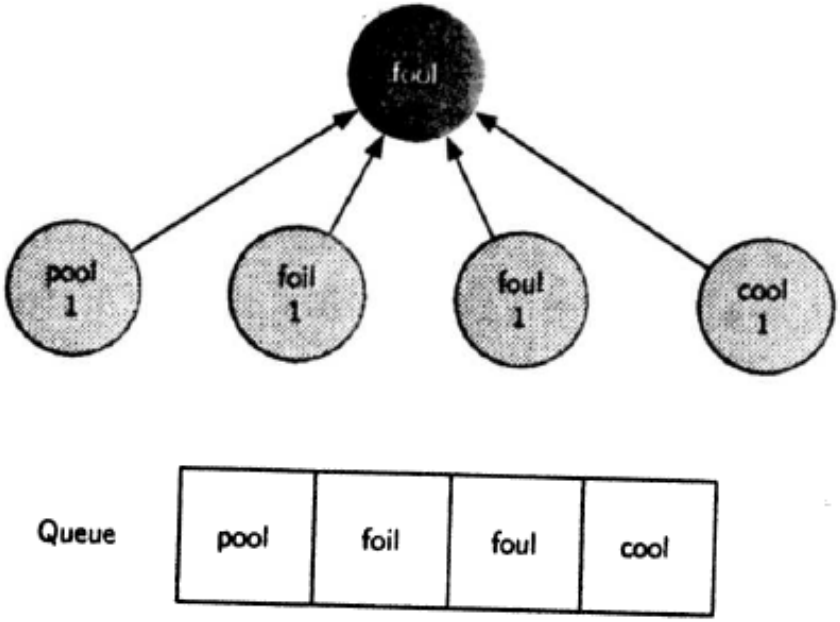


Figure 6.5: A Small Word Ladder Graph

algoritma

- → fool: “black”
- → kuyruk fool’un bir harfi farklı olanları
- → kuyruktakiler “gray”
- → kuyruk başında “pool” var
- → ağacın en tepesi (=fool), “0” değerinde
- → çocukları (ör. pool) “1” değerinde



algoritma

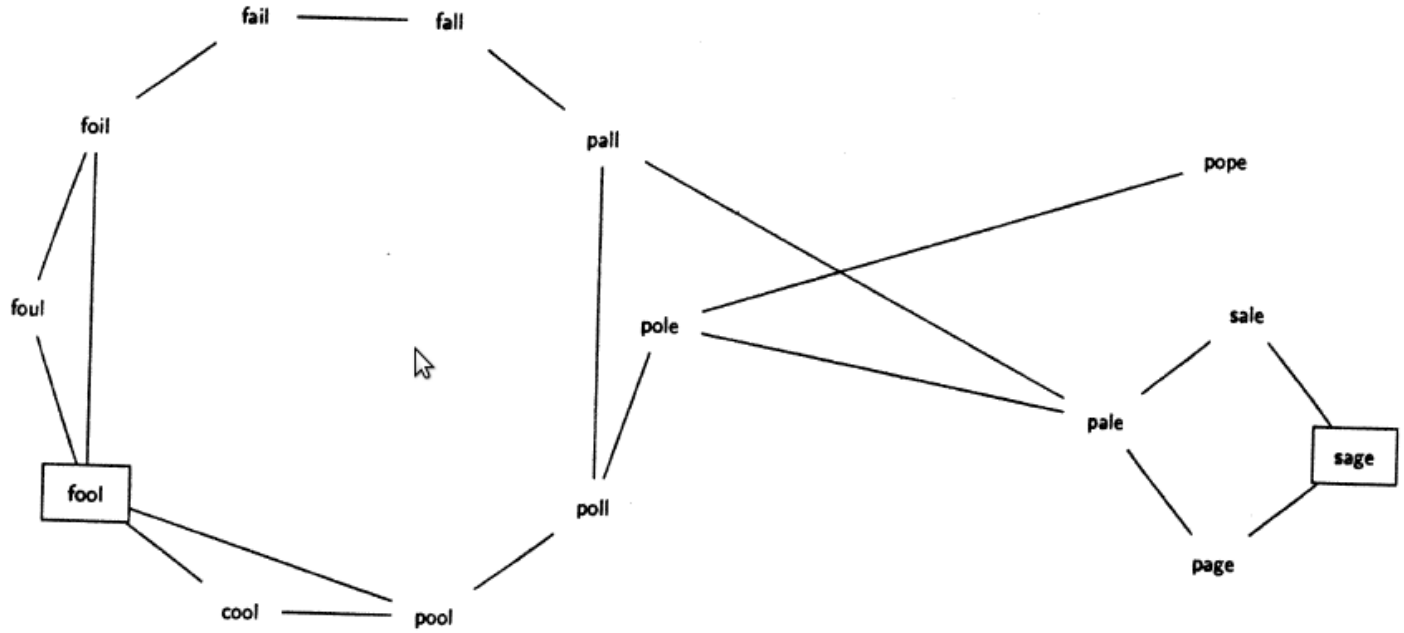


Figure 6.5: A Small Word Ladder Graph

→ [B] kuyruk başındakine (=pool) hizmet ver

→ kuyruk= [foil foul cool] oldu

algoritma

- → kuyruğa ziyaret edilenler (=gray) eklenmiyor
- → sadece ziyaret edilmemişler (=white) eklenecek (=poll)
- → yeni durum, 0:fool --> 1:pool --> 2:poll
- → 1:pool'un çocuklarından bahsettiğimize ve kuyruktan çıkardığımıza göre
- → karart, 1:pool=black

algoritma

→ 1:pool'un ziyaret edilmemiş komşularının hepsini(=white olanları) kuyruğa ekle

→ kuyruk= [foil foul cool **pool**] olur

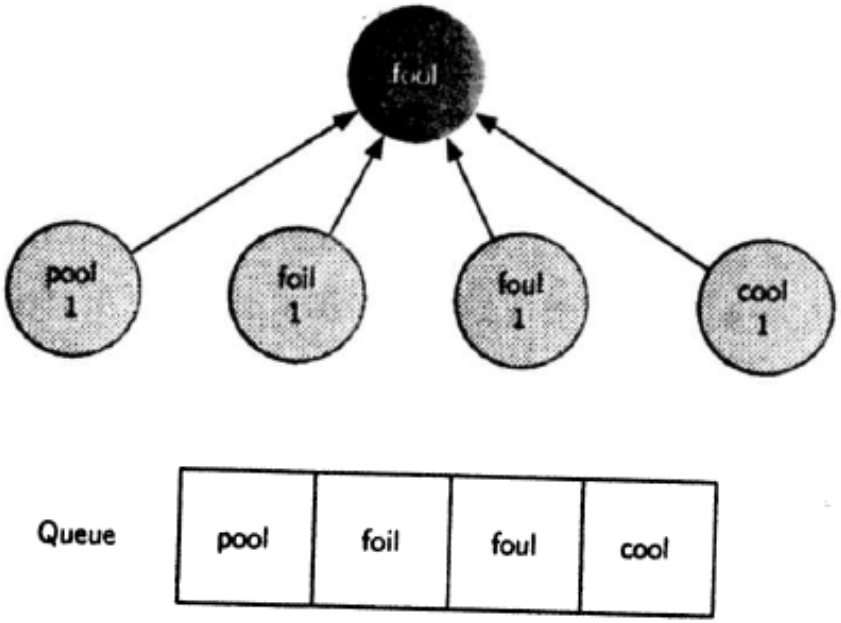


Figure 6.6: Fist Step in the Breadth First Search

algoritma

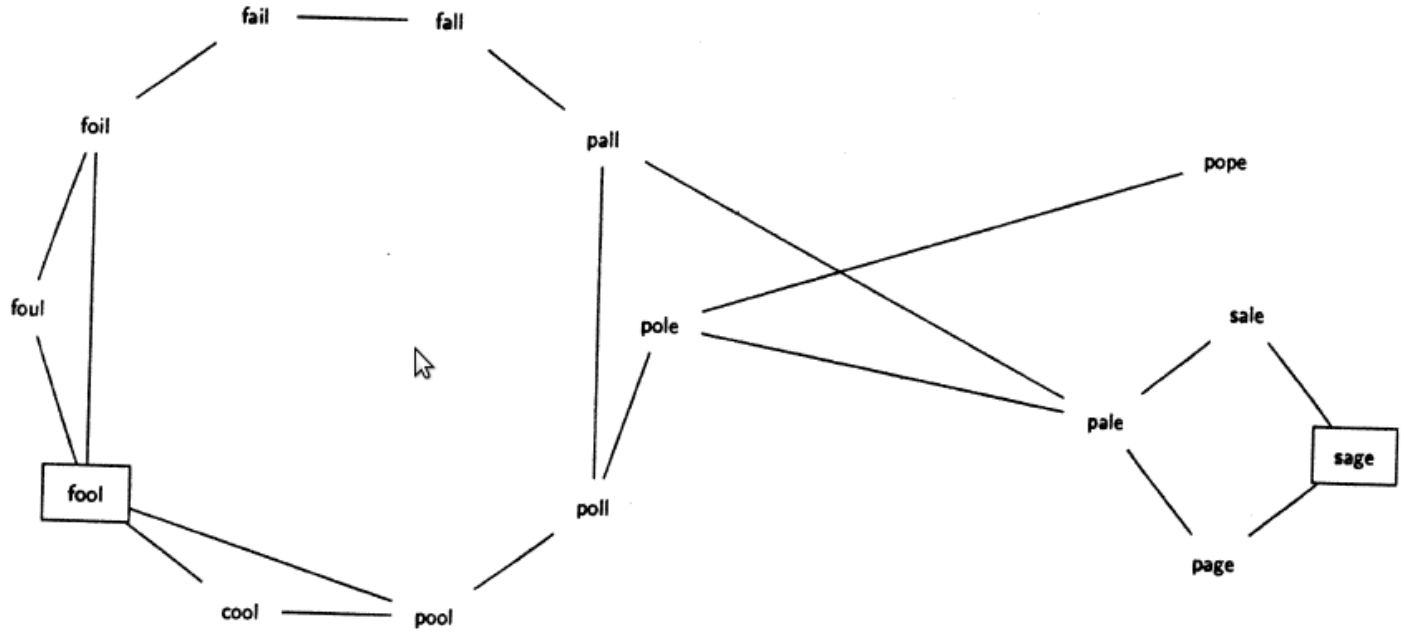


Figure 6.5: A Small Word Ladder Graph

→ kuyruk= [foil foul cool **pool**] idi

→ [C] [B] adımını kuyruk başındaki ile (=foil) yinele

algoritma

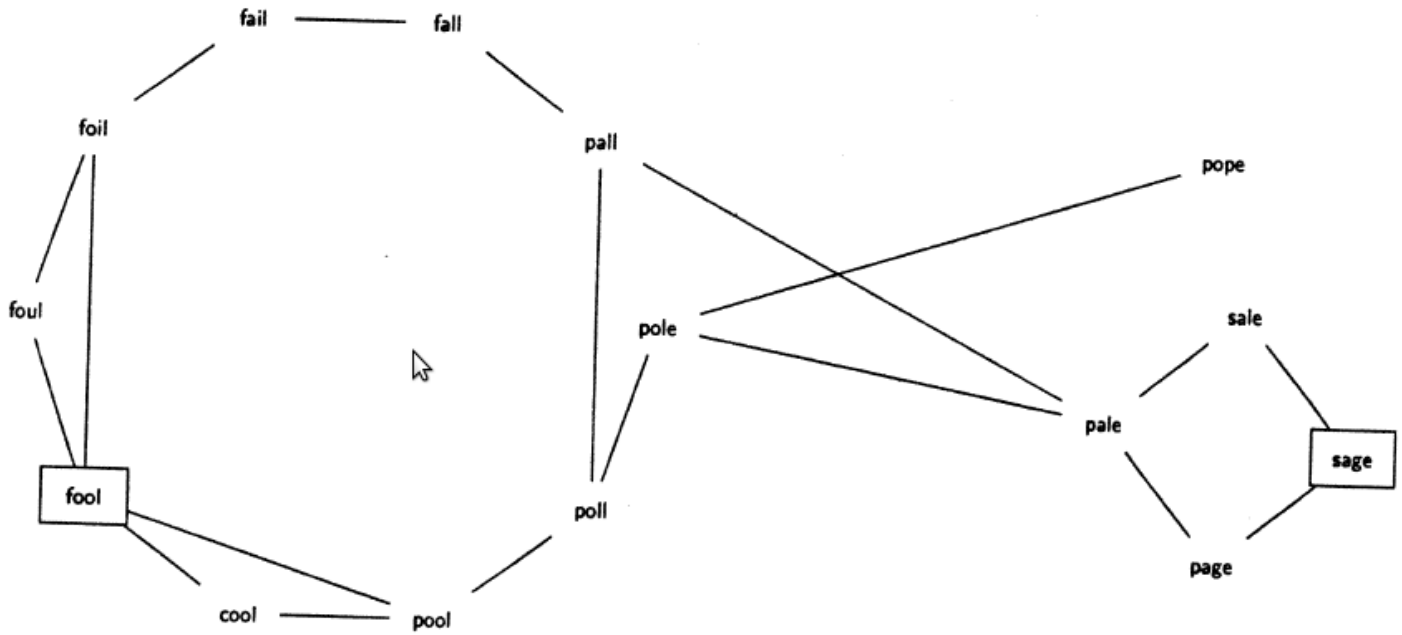
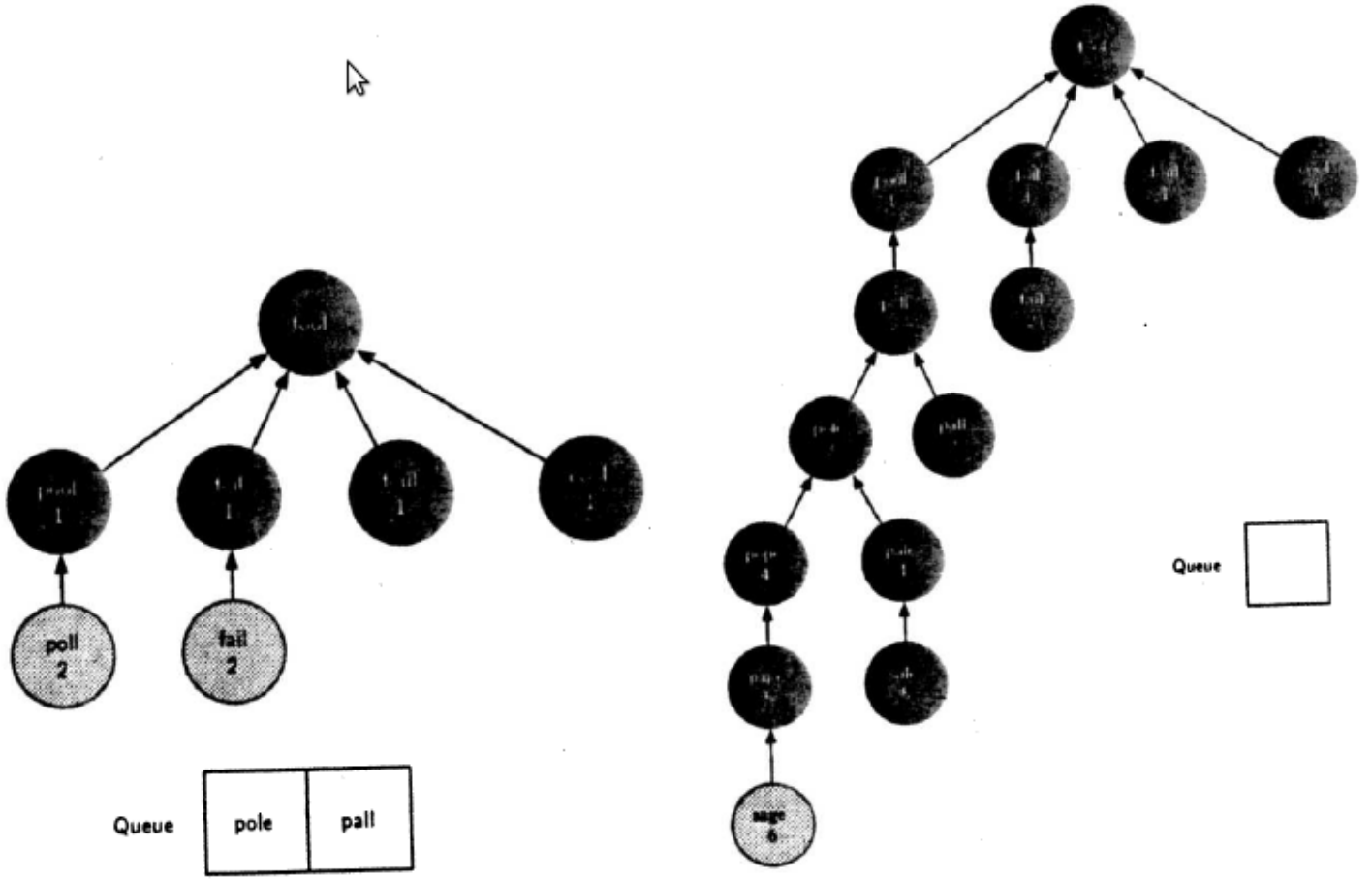


Figure 6.5: A Small Word Ladder Graph

algoritma

→ tüm kirişler tarandığında



(a) Breadth First Search Tree After Completing One Level

(b) Final Breadth First Search Tree

Figure 6.8: Constructing the Breadth First Search Tree

özet

→ BFS, ağırlıksız çizge problemidir

→ iki düğüm arasındaki en kısa yolun bulunması