Verteilte Systeme

Graphenalgorithmen

Allgemeine Netzwerke

- Reale Computernetze sind meist keine Ringe
- Beliebige Netze lassen sich als Graph modellieren: $G=(V,E)$
  Knoten V (Prozessen, Stationen)
  Kanten E (Kanälen, …)
    - ungerichtet: Kanäle sind bidirektional
    - gerichtet: Kanäle sind unidirektional

Definitionen:

Weg:

Eine Folge aus Knoten $(v_1,\ldots,v_n)$, wobei $(v_i, v_{i+1}) \in E$

Pfad:

Knotendisjunkter Weg

Baum:

Kreisfreier Graph

Wald:

Menge von Bäumen

Definitionen:

Gewicht:

Funktion aus der Kantenmenge in die Reellen Zahlen,
üblicherweise um Kosten auszudrücken.

Abstand:

Länge des kürzesten Pfades zwischen zwei Knoten
In gewichteten Graphen: kleinste Summe der Kantengwichte
auf Pfaden zwischen zwei Knoten

Durchmesser (diam):

Größter Abstand zwischen zwei Knoten in einem Graphen

Problem: Rundruf

- Eine Nachricht soll an alle Stationen des Netzes gesendet werden.
- Es sollen so wenig wie möglich Nachrichten versendet werden
- Es soll so schnell wie möglich gehen

Rundruf: Fluten

Idee:

Die Rundrufnachricht wird an alle Nachbarn weitergeleitet. Schon einmal gesehene Nachrichten werden verworfen.

Kommunikationskomplexität:

$2*|E| = O(|E|) \leq O(|V|^2)$

Zeitkomplexität:

$O(diam(G))$

=> Sehr “teuer” in dichten Netzen

Breitensuche

Die einfachste Methode einen aufspannenden Baum zu erhalten ist Breitensuche.

Ablauf:

- Die schon im Baum enthaltenen werden Knoten markiert. Initial ist nur Knoten i_0 markiert.
- Alle Knoten senden in der ersten Runde, in der sie markiert sind “search” an alle Nachbarn
- Ein Knoten, der noch nicht markiert ist und “search” empfängt, sucht sich einen der Absender als “parent” aus und setzt sich selbst auf markiert.

Kommunikationskomplexität: $|E| = O(|E|) = O(|V|^2)$

Zeitkomplexität: $O(diam)$

Breitensuche mit Kindern

Variante: Bidirektionaler Kanal

Knoten sendet “parent” oder “non-parent” zurück an alle, von denen er “search” empfangen hat.

Kommunikationskomplexität: $2|E| = O(|E|) = O(|V|^2)$

Zeitkomplexität: $O(diam(G))$

Variante: Unidirektionaler Kanal

Knoten sendet “parent ID PARENT_ID” und “non-parent ID PARENT_ID” mit einer neuen Breitensuchrunde huckepack (piggybacked) an alle, von denen er “search” empfangen hat

Kommunikationskomplexität: $O(|E|^2 b)$

Zeitkomplexität: $O(diam(G))$

Breitensuche (asynchron)

Im asynchronen Fall erzeugt der Algorithmus zwar einen aufspannenden Baum, aber keinen Breitensuchbaum!

Anpassung:

Die “search” Meldungen erhalten zusätzlich die Entfernung von i_0.

Nach einiger Zeit stabilisiert sich der Zustand des Systems so, dass wir einen Breitensuchbaum erhalten.

Kommunikationskomplexität: $O(|V| * |E|) \leq O(|V|^3)$

Zeitkomplexität: $O(diam * |V| * (l+d))$

Breitensuche (asynchron, ebenenweise)

Idee:

- Wir bauen den BFS-Baum in Ebenen auf.
- In jeder Phase wird eine Ebene konstruiert.
- Erkennung, ob eine Phase abgeschlossen ist, durch explizite Bestätigungen.

Ablauf:

1. Phase: - i_0 sendet “search” an alle Nachbarn
   - Nachbarn Antworten mit “parent” oder “non-parent”
   - wenn alle Nachbarn geantwortet haben sendet i_0 “new phase” an alle Kinder.

n. Phase: - Knoten in ebene n sendet “search” an alle Nachbarn
   - Nachbarn Antworten mit “parent” oder “non-parent”
   - wenn alle Nachbarn geantwortet haben sendet i_0 “new phase” an alle Kinder.

Breitensuche (asynchron, ebenenweise)

Idee:

- Wir bauen den BFS-Baum in Ebenen auf.
- In jeder Phase wird eine Ebene konstruiert.
- Erkennung, ob eine Phase abgeschlossen ist, durch explizite Bestätigungen.

Kommunikationskomplexität:

diam+1 Phasen, maximal |V| Nachrichten pro Phase

$O(diam*|V|)$

Zeitkomplexität:

Jede Phase dauert $O(diam*(l+d))$

$O(diam^2*(l+d))$

Breitensuche (asynchron, hybrid)

Idee:

Wir konstruieren m Ebenen parallel.

Kommunikationskomplexität:

- maximal O(m*|E|) Nachrichten, zur Bestimmung der Kind/Eltern Beziehungen.
- Um das Ende eine Phase beginn der nächsten zu kommunizieren, werden O(|V|*diam/m) Nachrichten benötigt

$$O(\ m*|E| + |V|*diam/m\ )$$

Zeitkomplexität: $O(diam^2*(l+d)/m)$

Kürzeste Wege

Problem:

Finde die kürzesten Wege und ihre Länge zwischen Knoten in einem gewichteten Graphen.

Eine Lösung: Bellman-Ford-Algorithmus

Idee: Jede Runde wird überprüft ob es einen kürzeren Weg zum Startknoten i_0 über einen Nachbarn gibt und die eigene Entfernung aktualisiert.

Bellman-Ford

Vor: Jeder Prozess i kennt

- seine Nachbarn $j \in neigh_i$
- die Gewichte der Kanten zu seinen Nachbarn $weight_{i,j}$
- die Anzahl n der Prozesse

Initialisierung:

$dist_i = \infty$ für alle $i \neq i_0$

$dist_i = 0$ für i_0

Bellman-Ford

```
send:
    sende send an alle j aus neigh_i
recv:
    dist_i,j = recv from j für alle j aus neigh_i
state:
    send = null
    falls weight_i,j + dist_i,j < dist_i
        dist_i := weight_i,j + dist_i,j
        parent_i = j
        send = dist_i
term:
    nach n-1 Runden ist dist_i minimal und parent_i enthält
    den Nachbarn über den der kürzeste Weg führt
```

Bellman-Ford

Kommunikationskomplexität:

$$O(\ (|V|-1) * |E|\ )$$

Zeitkomplexität:

$$O(|V| - 1)$$

Bellman-Ford (async)

```
init:
  dist_i = ∞ für alle i ≠ i_0
  dist_i = 0 für i_0
  send(dist_i)_j an alle j aus neigh_i0

recv(w)_i,j:
   if w + weight_i,j < dist_i
      dist_i   = w + weight_i,j
      parent_i = j
      send(dist_i)_j an alle j aus neigh_i


terminierung:
   problematisch, wir brauchen Bestätigungen
   (analog zur Breitensuche)
```

Bellman-Ford

Kommunikationskomplexität:

$$O(|V|^{|V|})$$

Zeitkomplexität:

$$O(|V|^{|V|+1} * (l+d))$$

Bew: siehe Lynch, Seite. 508

Bellman-Ford

- Der Bellman-Ford wurde am Anfang als Routing-Protokoll im Internet verwendet

- Die Familie der Rouingprotokolle, die auf Varianten von Bellman-Ford basiert, heisst Distanz-Vektor-Protokolle

Distanz-Vektor-Protokolle

Wir modifizieren Bellman-Ford wefolgt:

- wir speichern die Liste der $dist_{i,j}$ für alle i,j aus V statt nur $dist_i$
- wir senden in regelmäßigen Abständen die Liste an alle Nachbarn
- Die Nachbarn aktualisieren ihre Liste analog zur Relaxion in Bellman-Ford
- Falls i von einem Nachbarn j lange keine Nachricht bekommen hat, erklären er $weight_{i,j} := \infty$

Problem: “count to infinity”

Nicht mehr verfügbare Kanten können lange überleben