Tutorium 12

Florian Weber

8. Juli 2014

Wiederholung MST

Schnitteigenschaft

- $S \subset V$
- $C = \{(u, v) \in E : u \in S, v \in \overline{S}\}$

Satz: Die leichteste Kante $e \in C$ kann in einem MST verwendet werden.

- Beweis:
  - Sei T ein MST
  - Fall 1: e ist Teil von T: Beweis fertig
  - Sonst: T $\cup\{e\}$ enthält Kreis $K \Rightarrow \exists$ Kante $(u, v) \neq e \in C$
  - Dann ist $T \cup \{e\} \backslash \{(u, v)\}$ ein Spannbaum der nicht schwerer ist (aka MST)

Kreiseigenschaft

Satz: Die schwerste Kante auf einem Kreis wird für einen MST nicht benötigt.

- Beweis:
  - Angenommen der MST T benutzt die schwerste Kante e auf einem Kreis c
  - Wähle $f \in C$ mit $f \neq e$
  - Dann gilt $\texttt{weight}(f) \leq \texttt{weight}(e)$
  - Dann ist $T \cup \{f\} \backslash \{e\}$ auch ein MST

Algorithmus von Kruskal

Prinzip

- Idee: Nehme solange die kürzeste Kante des Graphen die keinen Kreis mit bisherigen Kanten bildet bis diese Kanten einen MST bilden.
- Probleme:
  - Wie findet man die leichteste Kante?
  - Wie findet man schnell heraus, ob ein Kreis gebildet wird?

Die Union-Find-Datenstruktur

- Knotensammlung für jede eigenständige Partition, mit eindeutigem Wurzelknoten

```
class UnionFind:
    parents: Array<Index> = [0, 1, 2, 3, ..., n]

    fun link(i: index, j: index):
        parents[i] := j

    fun find(i: index): index
        if parents[i] = i: return i
        else: return find(parents[i])

    fun union(i: index, j: index):
        if find(i) != find(j):
            link(find(i), find(j))
```

Pfadkompression

```
fun find(i: index): index
    if parents[i] = i:
        return i
    root := find(parents[i])
    parents[i] := root
    return root
```

Pfadkompression

```
fun find(i: index): index
    if parents[i] = i:
        return i
    root := find(parents[i])
    parents[i] := root
    return root
```

- `find` nun $\in O(\log n)$ (Beweis im Buch)

Union-By-Rank

```
rank: Array<UInt> = [0, 0, ..., 0]

fun link(i: index, j: index):
    if rank[i] < rank[j]:
        parents[i] := j
    else:
        parent[j] := i
        if rank[i] = rank[j]:
            rank[i] += 1
```

Union-By-Rank

```
rank: Array<UInt> = [0, 0, ..., 0]

fun link(i: index, j: index):
    if rank[i] < rank[j]:
        parents[i] := j
    else:
        parent[j] := i
        if rank[i] = rank[j]:
            rank[i] += 1
```

- `find` (ohne Pfadkompression) nun $\in O(\log n)$ (ohne Beweis)

Pfadkompression + Union-By-Rank

- $A(\mathbb{N}, \mathbb{N})$ Ackermannfunktion: wächst **absurd schnell**
- $\alpha_T(m, n) = \min i \geq 1 : A(i, \mathrm{ceil}(m/n)) \geq \log n$

Pfadkompression + Union-By-Rank

- $A(\mathbb{N}, \mathbb{N})$ Ackermannfunktion: wächst **absurd schnell**
- $\alpha_T(m, n) = \min i \geq 1 : A(i, \mathrm{ceil}(m/n)) \geq \log n$

- $m \times \texttt{find} + n \times \texttt{link} \in O(m\alpha_T(m, n)) \approx O(m)$

Kruskal revisited

```
fun kruskal(edges: Array<Edge>): List<Edge>
    sort(edges) // by weight, ascending
    forests := UnionFind()
    for ein edges:
        if forests.find(e[0]) != forests.find(e[1]):
            yield e
            forests.union(e[0], e[1])
```

Kruskal revisited

```
fun kruskal(edges: Array<Edge>): List<Edge>
    sort(edges) // by weight, ascending
    forests := UnionFind()
    for ein edges:
        if forests.find(e[0]) != forests.find(e[1]):
            yield e
            forests.union(e[0], e[1])
```

- Laufzeit: $O(m \log m)$ (sortieren der Kanten)
- Nahezu Linearzeit, wenn sortieren unnötig oder $\in O(m)$

Kreativaufgabe