Netzwerkalgorithmen

June 30, 2016

1 Topologisches Sortieren

1.1 BFS

```
bool TOPSORT (const graph& G, node_array<int>& topnum){
        int count = 0
        list<node> zero
        node_array<int> indeg(G)
        node v
        forall_nodes(v,G){
                indeg[v] = G.indeg(v)
                if (indeg[v] == 0) zero.append(v)
        }
        while (!zero.empty()){
                node v = zero.pop()
                topnum[v] = ++count
                edge e
                forall_out_edges(e,v){
                        node w = G.target(e)
                        if(--indeg[w] == 0) zero.append(w)
                }
        }
        return count == G.number_of_nodes()
}
```

1.2 DFS

```
bool DFS_TOPSORT (const graph& G, node_array<int>& dfsnum, node_array<int>& compnum){
        int count1 = 0
        int count2 = 0
        node_array<bool> visited(G,false)
        node v
        forall_nodes(v,G){
                if (!visited[v]) dfs(G,v,count1,count2,dfsnum,compnum)
        }
}
void dfs(const graph& G, node v, int& count1, int& count2, node_array<int>& dfsnum, node_a
        dfsnum[v] = ++count1
        visited[v] = true
        edge e
        forall_out_edges(e,v){
                edge w = G.target(e)
                if(!visited[w]) dfs(G,w,count1,count2,dfsnum,compnum)
        }
        compnum[v] = ++count2
```

```
}
```

1.3 Starke Zusammenhangskomponenten

2 Kürzeste Wege

2.1 $\mathcal{O}(n+m)$

```
forall v in V do
        DIST[v] = unendlich
od
DIST[s] = 0
U = {s}
while U nicht leer
        wähle das u aus U mit minimaler topologischen Nummer
        forall v in V mit (u,v) in E do
                c = DIST[u]+cost(u,v)
                if c < DIST[v] then
                        DIST[v] = c
                        U = U + {v}
                fi
        od
od
```

2.2 Dijkstra

1x Konstruktor, n*(insert+delmin+empty), mx*decrease
Binärer Heap/Balancierter Baum: $\mathcal{O}((n+m)*log(n))$
Fib-Heap: $\mathcal{O}(n*log(n)+m)$

```
void DIJKSTRA(const graph& G, node s, const edge_array<int>& cost,
        _ const node_array<int>& dist, node_array<edge>& pred){
        node_pq<int> PQ(G);
        node v
        forall_nodes(v,G){
                dist[v] = unendl
                pred[v] = null
        }
        dist[s] = 0
        PQ.insert(s,0)
        while(!PQ.empty()){
                node u = PQ.delmin()
                edge e
                forall_out_edges(e,u){
                        node v = G.target(e)
                        int c = DIST[u]+cost(e)
                        if(c < dist[v]){
                                if(dist[v] == MAXINT){
                                        PQ.insert(v,c)
                                }else{
                                        PQ.decrease(v,c)
                                }
                                dist[v] = c
                                pred[v] = u
                        }
                }
        }
}
```

2.3 Bellman-Ford

```
bool BELLMAN(const graph& G, node s, const edge_array<int>& cost, node_array<int>& DIST){
        queue<node> Q;
        node_array<bool> inQ(G,false)
        node_array<int> count
        DIST[s] = 0
        Q.append(s)
        inQ[s] = true
        while (!Q.empty()){
                node u = Q.pop()
                inQ[u] = false
                if (++count[u] > G.number_of_nodes()){
                        return false
                }
                edge e
                forall_out_edges(e,u){
                        node v = G.target(e)
                        int c = DIST[u] + cost[e]
                        if (c<DIST[v]) {
                                DIST[v] = c
                                //Setze PRED verweis hier falls nötig
                                if (!inQ[v]){
                                        Q.append(v)
                                        inQ[v] = true
                                }
                        }
                }
        }
        return true
}
```

3 Maximaler Fluss

3.1 MF-Labeling

Labeling: $\mathcal{O}(n+m)$, Gesamt: $\mathcal{O}(n^2U + nmU)$, da max n-1 Kanten über den Schnitt laufen und damit $F_{max} \leq n * U$ mit U der Kapazutät der mächtigsten Kante.
Bei zusammenhängendem Graphen:$\mathcal{O}(nmU)$

```
void MF_Labeling (const graph& G, node s, node t, cost edge_array<int>& cap,
        _ edge_array<int>& flow){
        list<node> L
        node_array<bool> labeled(G,false)
        node_arrray<edge> pred(G,null)
        while(true){
                labeled[s] = true
                L.append(s)
                while(!L.empty()) {
                        node v = L.pop()
                        edge e
                        forall_out_edges(e,v){
                                if ( flow[e]==cap[e]) continue
                                node w = G.target(e)
                                if (labeled[w]) true
                                labeled[w]=true
                                pred[w]=e
```

```
                        L.append(w)
                }
                forall_in_edges(e,v){
                        if (flow[e]==0) continue
                        node w = G.source(e)
                        if (labeled[w]) continue
                        labeled w = true
                        pred[w] = e
                        L.append(w)
                }
                if (labeled[t]) L.clear()
        }
        if (labeled[t]) AUGMENT(G,s,t,pred,cap,flow)
        else break
    }
}

void AUGMENT (const graph& G, node s, node t, cost node_array<edge>& pred, edge_array<int>
    int delta = MAXINT
    node v = t
    while (v != s){
        int r
        edge e = pred[v]
        if(v==G.source(e)){
            r= flow[e]
            v = G.target[w]
        }else{
            r = cap[e]-flow[e]
            v = G.source(e)
        }
        if (r<delta) delta =r
    }
    v = t
    while (v!=s){
        edge e = pred[v]
        if(v==G.source(e)){
            flow[e] -= delta
            v = G.target(e)
        }else{
            flow[e] += delta
            v = G.source(e)
        }
    }
}
```

3.2 Capacity-Scaling

Anzahl der Phasen $\leq logU$
Jede Phase führt max 2m Erhöhungen aus + labeling: $\mathcal{O}(2mm)$
$\mathcal{O}(m^2 * log(U))$

```
In Labeling:
    if(cap[e]-flow[e] < delta) continue
    statt: if(flow[e] == cap[e]) continue

    if (flow[e] < delta) continue
```

```
        statt: if (flow[e]==0) continue

CAPACITY_SCALING (G,s,t,cap,flow,U){
        flow = 0 //alle Flüsse = 0
        delta = 2^(log(U))
        while delta > 0 do
                MF-LABELING(G,s,t,cap,flow,delta)
                delta = delta / 2
        od
}
```