Anwendung DIJKSTRA

```
void Dijkstra(const graph& G, node s, const edge_array<
   int>& cost, node_array<int>& DIST){
   p_queue<node,int> PQ;
   node_array<pq_item> I(G,NULL);
   I[s] = PQ.insert(s,0); // I ist Menge der Indexe der
                          //Knoten
   node v;
   forall_nodes(v,G){
      DIST[v] = MAXINT;
   }
   DIST[s] = 0;

   while(!PQ.empty()){
      node u = PQ.del_min();
      egde e;
      forall_out_edges(e,u){
         node v = G.target(e);
         int d = DIST[u] + cost[e];
         if(d < DIST[v]){ // Dreiecks-Ungleichung verletzt
            if(DIST[v] == MAXINT)
               I[v] = PQ.insert(v,d); // zum ersten Mal
                                      //besucht
            else // v ist in PQ
               PQ.decrease_p(I[v],d)
            DIST[v] = d;
         }
      }
   }
}
```

Erklärung:
Zuerst wird s in die Priority Queue PQ aufgenommen. Die *while*-Schleife läuft nun so lange, bis PQ leer ist. Ist dies der Fall, ist zu jedem Knoten ein kürzester Weg bekannt, falls der Graph azyklisch ist. In der *while*-Schleife wird zuerst aus der PQ ein Knoten u gewählt. Dabei wird darauf geachtet, dass es eine perfekte Wahl ist, d.h. das der Knoten mit dem kleinsten, in der Priority Queue vorkommenden, DIST-Wert gewählt wird. Dann werden alle von u ausgehenden Kanten betrachtet, die jeweilige Δ-Ungleichung überprüft und falls sie verletzt wurde, der erreichte Knoten

entweder in PQ aufgenommen, falls er noch nicht in PQ drin ist oder sonst seine Priorität auf seinen neuen DIST-Wert gesetzt. Danach wird noch sein $DIST$-Wert angepasst.

Analyse
$\left.\begin{array}{l} n \times delmin \\ n \times insert \end{array}\right\} n \times log_2(n)$
$m \times decrease\_p$ ($\mathcal{O}(1)$ amortisiert) $\}\mathcal{O}(m)$

Gesamtlaufzeit: $\mathcal{O}(m + n * log(n))$

Weiterer Aspekt: Korrektheit der Implementierung

Idee: Füge Programmcode hinzu, der für die Konkrete Eingabe testet, ob das Ergebnis korrekt ist.
→ (Programm Checker) Verifying Algorithms

hier: Teste für jede Kante am Ende die Δ-Ungleichung.

```
edge e;
forall_edges(e,G){
   node v = G.source(e);
   node w = G.target(e);
   ASSERT(DIST[v] + cost[e] >= DIST[w])
}
```

5 Netzwerkflussprobleme

Ein Unternehmer möchte täglich möglichst viele seiner Waren von einem seiner Standorte zu einem anderen Standort transportieren. Er nutzt dazu LKW‘s anderer Unternehmen, die zwischen verschiedenen Städten hin und her fahren. Jeder hat eine maximale Kapazität, die er von einer zu einer andern Stadt transportieren kann. Jeder dieser LKW‘s hat allerdings eine maximale Kapazität an Waren, die er transportieren kann. Da es in den einzelnen Städten keine Lagerhäuser gibt, können die Waren nicht zwischengelagert werden, sondern müssen sofort weiter transportiert werden.
Der Ausgangsstandort in diesem Szenario stellt den Startknoten s und der zweite Standort den Endknoten t dar. Die LKW ‘s fahren entlang der Kanten und ihre Kapazität entspricht der Kapazität der jeweiligen Kante.

Ziele

i) MaxFlow-Problem
Maximiere den Transport von einem Knoten s (source, Quelle) zu einem Knoten t (target, Senke)
Beispiel: Transport von Tonnen Stahl pro Tag über das Eisenbahnnetz.

ii) MinCostFlow
Konkretes Transportproblem (z.B. zwischen Produzenten und Konsumenten)
Ziel: Minimiere die Transportkosten.
Dazu wird zusätzlich du der Kapazität jeder Kante auch Kosten zugeordnet.

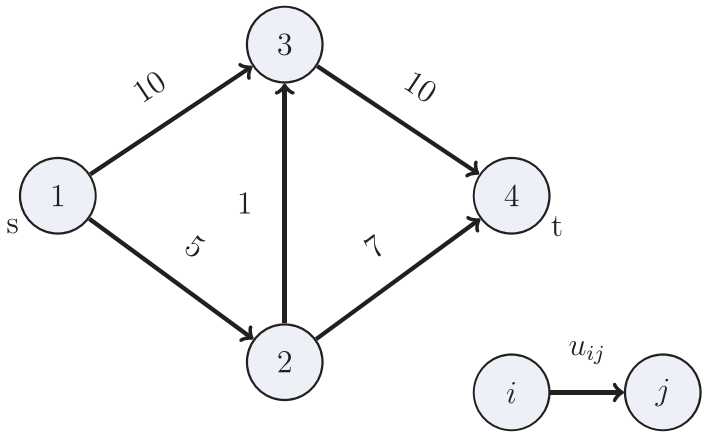
5.1 MaxFlow-Problem

5.1.1 Formale Definition

Gegeben sei

- ein gerichteter Graph $G = (V, E)$
- eine Kapazitätsfunktion $u : E \to \mathbb{R}_0^+$ und
- zwei Knoten $s, t \in V$ mit $s \neq t$.

$u((v,w))$ heißt Kapazität von (v,w). s heißt Quelle (source) und t Senke (target).



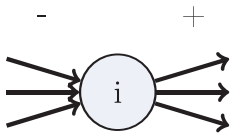
Gesucht ist eine Flussfunktion $x : E \to \mathbb{R}_0^+$ mit folgenden Eigenschaften:
(x_{ij} ist der Fluss über die Kante (i,j)) mit:

i) Kapazitätsbedingung:
$\forall\ e \in E : 0 \leq x(e) \leq u(e)$

ii) Massenbalance-Bedingung:
Sei $v \in V$ (beliebig aber fest!)

$$\underbrace{\sum_{(v,u)\in E}}_{\substack{\text{gehe über alle}\\ \textbf{ausgehenden}\\ \text{Kanten von v}}} x((v,u)) - \underbrace{\sum_{(w,v)\in E}}_{\substack{\text{gehe über alle}\\ \textbf{eingehenden}\\ \text{Kanten von v}}} x((w,v)) = \begin{cases} F, & v = s \\ 0, & \forall\ v \notin \{s,t\} \\ -F, & v = t \end{cases}$$

$F \geq 0$



$$\delta(i) = \sum x((i,j)) - \sum x((k,i))$$

ii) Optimalitätsbedingung:
F soll maximal sein.

Erklärungen:

i) Für alle Kanten muss gelten, dass ihr Flusswert zwischen Null und ihrer maximalen Kapazität liegt.

ii) Für alle Knoten die nicht s oder t sind, muss gelten, dass alles was in sie rein fließt auch wieder abfließt. Es kann also nichts in diesen Knoten gelagert werden. F entspricht dem Fluss der fließt. Aus dem Knoten s fließt der komplette Fluss F heraus, aber nichts herein. Beim Knoten t ist es genau umgekehrt.

iii) Ohne diese Bedingung wäre das Problem trivial, da der Null-Fluss ($x = 0$) die anderen beiden Bedingungen immer erfüllt.

Notation:
Knoten: i, j
Die Kapazität wird anstatt mit $u((i, j))$ jetzt nur noch mit u_{ij} und die Flussfunktion anstatt mit $x((i, j))$ mit x_{ij} bezeichnet.

Beobachtung

$\delta(t) = -\delta(s)$

1. Idee für einen Algorithmus: Erhöhe Pfade

i) Starte mit $x = 0(\forall\ (i, j) \in E\ x_{ij} \leftarrow 0)$

ii) Erhöhe x entlang von Pfaden von s nach t

5.1.2 Das Restnetzwerk

(residual network)
Ein Restnetzwerk beschreibt mögliche Flussrichtungen.

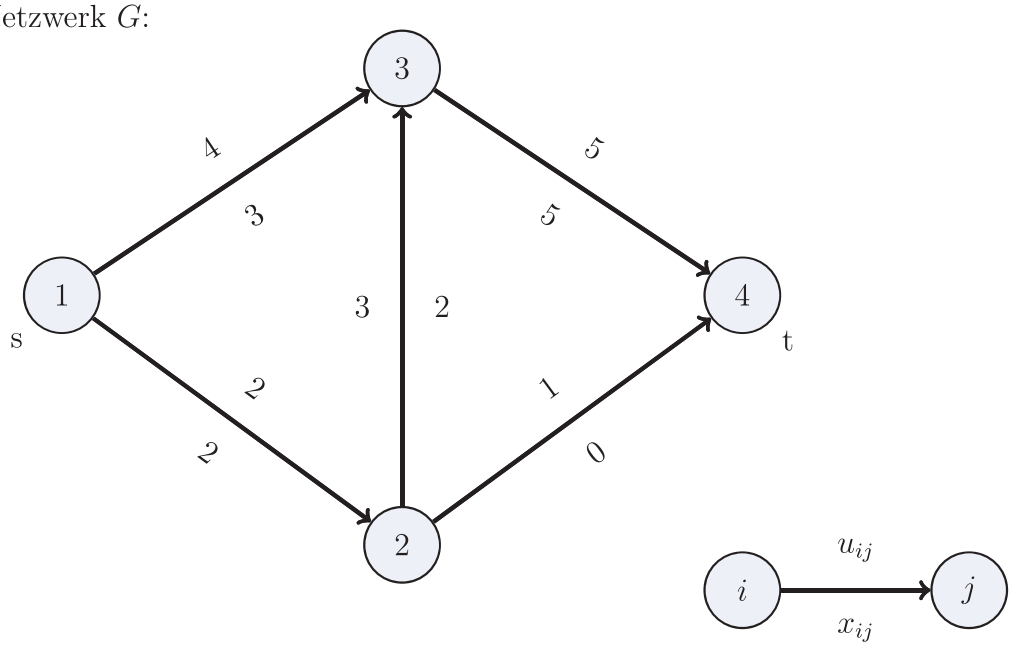
Definition

Sei x eine aktuelle Flussfunktion, dh. sie erfüllt die Kapazitätsbedingung (z.B. der Nullfluss ($\forall\ i, j\ x_{ij} = 0$)).
Das Restnetzwerk $G(x)$ besteht aus allen Kanten von G und deren Gegenkante. Es existieren also für jede Kante (i, j) aus G zwei Kanten in $G(x)$, nämlich die Kante (i, j) und ihre Gegenkante (j, i). Sie besitzen die Restkapazitäten $r_{ij} = u_{ij} - x_{ij}$ und $r_{ji} = x_{ij}$.

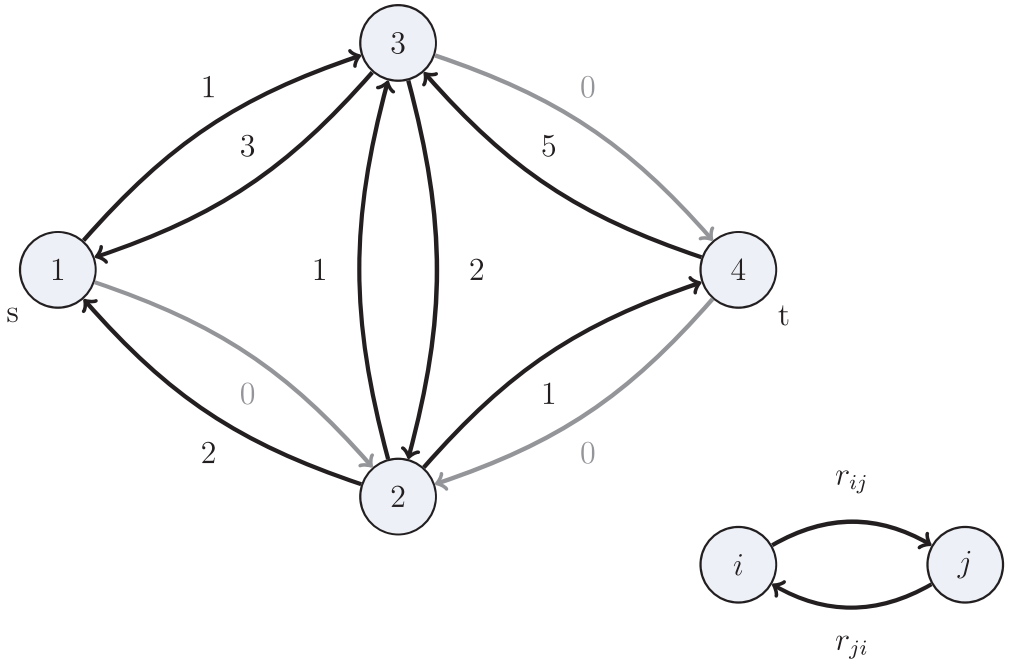
Die Restkapazitäten beschreiben mögliche Änderungen an der Flussfunktion. Eine Erhöhung geschieht in Richtung der Kante (i, j) (max $r_{ij} = u_{ij} - x_{ij}$) und eine Verminderung in die Gegenrichtung (max $r_{ji} = x_{ij}$).

Beispiel 5.1

Netzwerk G:



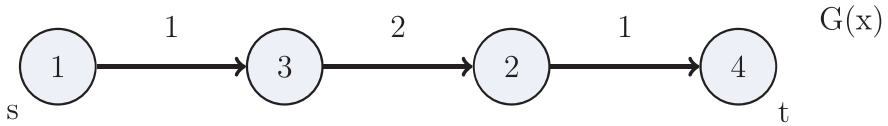
Aktueller Flusswert: $F = 5$
Restnetzwerk $G(x)$:



Da über Kanten mit einer Restkapazität von Null, nichts mehr fließen kann, werden sie weggelassen. Dadurch ist die folgende Beobachtung möglich.

Beobachtung: (ohne 0-Kanten)
Wenn alle Null-Kanten im Graphen weggelassen werden, entspricht die Frage ob ein Pfad existiert der Frage ob ein erhöhender Pfad existiert.
Jeder Pfad in $G(x)$ von s nach t beschreibt eine mögliche Erhöhung des Gesamtflusses F. ($\rightarrow$ Pfaderhöhung)

Im Beispiel 5.1 existiert nur ein solcher Pfad:



Eine Erhöhung um $\delta = min\{r_{ij}|(i,j) \in P\} = 1$ ist möglich.