

Diskrete Optimierung  
Prof. Dr. Sándor P. Fekete  
Sommersemester 2003

# Algorithmen

Andreas Baude  
[a.baude@tu-braunschweig.de](mailto:a.baude@tu-braunschweig.de)

19. März 2005



## Inhaltsverzeichnis

<b>1</b>	<b>Graphen und Grundbegriffe</b>	<b>1</b>
1.3	Zusammenhang . . . . .	1
<b>2</b>	<b>Aufspannende Bäume und Arboreszenzen</b>	<b>2</b>
2.1	Aufspannende Bäume . . . . .	2
2.2	Gewichtsmaximale Branchings . . . . .	3
<b>3</b>	<b>Kürzeste Wege</b>	<b>4</b>
3.2	Kürzeste Wege von einer Quelle . . . . .	4
3.3	Kürzeste Wege zwischen allen Knoten . . . . .	5
<b>4</b>	<b>Netzwerkflüsse</b>	<b>6</b>
4.2	Maximaler Netzwerkfluss . . . . .	6
4.3	Der Preflow-Push-Algorithmus von GOLDBERG und TARJAN . . . . .	7
<b>5</b>	<b>Kostenminimale Flüsse</b>	<b>9</b>
5.3	Min-Cost-Flow-Algorithmen . . . . .	9
<b>6</b>	<b>Maximale Matchings</b>	<b>11</b>
6.3	Matching in allgemeinen Graphen . . . . .	11
<b>7</b>	<b>Gewichtsminimales Matching</b>	<b>13</b>
7.1	Bipartite Graphen . . . . .	13
7.2	Allgemeine Graphen . . . . .	14

## Liste der Algorithmen

1.17	Graphen-Scan . . . . .	1
2.2	KRUSKAL . . . . .	2
2.5	PRIM . . . . .	2
2.7	EDMONDS . . . . .	3
3.3	DIJKSTRA . . . . .	4
3.5	MOORE-BELLMAN-FORD . . . . .	4
3.8	FLOYD, WARSHALL . . . . .	5
4.5	FORD, FULKERSON . . . . .	6
4.11	EDMONDS-KARP . . . . .	6
4.17	Push-Relabel . . . . .	7
4.23	GOLDBERG und TARJAN . . . . .	8
5.8	Cycle Cancelling . . . . .	9
5.9	Minimum Mean Cycle Cancelling . . . . .	9
5.13	Minimum Mean Cycle . . . . .	10
6.8	Maximales Matching in bipartiten Graphen . . . . .	11
6.13	Blossom-Algorithmus für perfektes Matching . . . . .	12
7.1	Gewichtsminimales perfektes Matching in bipartiten Graphen . . . . .	13
7.3	Blossom-Algorithmus für gewichtsminimales perfektes Matching . . . . .	14

# 1 Graphen und Grundbegriffe

## 1.3 Zusammenhang

---

### Algorithmus 1.17 : Graphen-Scan

---

**Eingabe:** Graph  $G$ , Startknoten  $s$ .

**Ausgabe:** Knotenmenge  $R \subseteq V(G)$ , die von  $s$  aus erreichbar ist, sowie eine Kantenmenge  $T \subseteq E(G)$ , die die Erreichbarkeit von  $R$  sicherstellt.

- Im gerichteten Fall ist  $(R, T)$  eine in  $s$  verwurzelte *Arboreszenz*.
- Im ungerichteten Fall ist  $(R, T)$  ein *aufspannender Baum*.

**Komplexität:** Der Algorithmus kann so implementiert werden, dass seine Laufzeit linear ist, d.h.  $\mathcal{O}(\max(n, m))$ .

**Varianten:** Breitensuche (BFS), Tiefensuche (DFS).

```

1  $R \leftarrow \{s\}; Q \leftarrow \{s\}; T \leftarrow \emptyset;$ 
2 if  $Q == \emptyset$  then
  | stopp ;
  else
  | Wähle  $v \in Q;$ 
3 if es gibt kein  $w \in V(G) \setminus R$  mit  $e = (v, w) \in E(G)$  then
  |  $Q \leftarrow Q \setminus \{v\};$ 
  | goto (2);
  else
  | Wähle ein  $w \in V(G) \setminus R$  mit  $e = (v, w) \in E(G);$ 
4  $R \leftarrow R \cup \{w\}; Q \leftarrow Q \cup \{w\}; T \leftarrow T \cup \{e\};$ 

```

---

## 2 Aufspannende Bäume und Arboreszenzen

### 2.1 Aufspannende Bäume

---

**Algorithmus 2.2 : KRUSKAL**

---

**Eingabe:** Zusammenhängender, ungerichteter Graph  $G$ ,  
Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ .

**Ausgabe:** Aufspannender Baum  $T$  minimalen Gewichts.

**Komplexität:**  $\mathcal{O}(m \cdot n)$  (mit „Teamchefs“  $\mathcal{O}(m \cdot \log n)$ ).

- 1 Sortieren der Kanten nach  $c(e_1) \leq c(e_2) \leq \dots \leq c(e_m)$ ;
  - 2  $T \leftarrow (V(G), \emptyset)$ ;
  - 3 **for**  $i = 1, \dots, m$  **do**
    - └ **if**  $T + e_i$  *enthält keinen Kreis* **then**
      - └  $T \leftarrow T + e_i$ ;
- 

---

**Algorithmus 2.5 : PRIM**

---

**Eingabe:** Zusammenhängender, ungerichteter Graph  $G$ ,  
Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ .

**Ausgabe:** Aufspannender Baum  $T$  minimalen Gewichts.

**Komplexität:**  $\mathcal{O}(n^2)$ .

- 1 Wähle  $v \in V(G)$ ;  $T \leftarrow \{\{v\}, \emptyset\}$ ;
  - 2 **while**  $V(T) \neq V(G)$  **do**
    - └ Wähle eine Kante  $e \in \delta(V(T))$  minimalen Gewichts;
    - └  $T \leftarrow T + e$ ;
-

## 2.2 Gewichtsmaximale Branchings

---

**Algorithmus 2.7** : EDMONDS
 

---

**Eingabe:** Digraph  $G$ , Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ .**Ausgabe:** Gewichtsmaximales Branching.**Komplexität:**  $\mathcal{O}(n \cdot m)$ .

```

1  $i \leftarrow 0$ ;  $G_0 \leftarrow G$ ;  $c_0 \leftarrow c$ ;
2 Sei  $B_i$  ein gewichtsmaximaler Teilgraph von  $G_i$  mit  $|\delta_{B_i}^-(v)| \leq 1$  für alle  $v \in V(B_i)$ ;
3 if  $B_i$  enthält keinen Kreis then
  |  $B \leftarrow B_i$ ;
  | goto (5);
4  $(G_{i+1}, c_{i+1}) \leftarrow (G_i, c_i)$ ;
   foreach Kreis  $C$  von  $B_i$  do
     Kontrahiere  $C$  zu einem einzigen Knoten  $v_C$  in  $G_{i+1}$ ;
     foreach Kante  $e = (z, y) \in E(G_i)$  mit  $z \notin V(C)$ ,  $y \in V(C)$  do
       |  $\text{pred}(e, C) \leftarrow (x, y) \in E(C)$ ;
       |  $e' \leftarrow (z, v_C)$ ;
       |  $e_C$  ist billigste Kante in  $C$ ;
       |  $c_{i+1}(e') \leftarrow c_i(e) - c_i(\text{pred}(e, C)) + c_i(e_C)$ ;
       |  $\Phi(e') \leftarrow e$ ;
     |  $i \leftarrow i + 1$ ;
   goto (2);
5 if  $i == 0$  then
  | stopp ;
6 foreach Kreis  $C$  von  $B_{i+1}$  do
  | if es gibt eine Kante  $e' = (z, v_C) \in E(B)$  then
  | |  $E(B) \leftarrow (E(B) \setminus \{e'\}) \cup \Phi(e') \cup (E(C) \setminus \text{pred}(\Phi(e'), C))$ ;
  | else
  | |  $E(B) \leftarrow E(B) \cup (E(C) \setminus \{e_C\})$ ;
  | |  $V(B) \leftarrow V(G_{i-1})$ ;
  | |  $i \leftarrow i - 1$ ;
  | goto (5);

```

---

### 3 Kürzeste Wege

#### 3.2 Kürzeste Wege von einer Quelle

---

**Algorithmus 3.3 : DIJKSTRA**


---

**Eingabe:** Digraph  $G$ , Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ , Startknoten  $s$ .

**Ausgabe:** Für jeden Knoten  $v \in V(G)$  die Angaben

$l(v)$ : Länge des kürzesten  $s$ - $v$ -Pfad.

$p(v)$ : Vorgänger von  $v$  in einem kürzesten  $s$ - $v$ -Pfad.

(falls  $v$  nicht erreichbar ist, gilt  $l(v) = \infty$ ,  $p(v) = \text{NIL}$ ).

**Komplexität:**  $\mathcal{O}(n^2)$ .

**Verbesserung:** Wahl von geschickteren Datenstrukturen, z.B. FIBONACCI-Heaps.

```

1  $l(v) \leftarrow \infty$  für alle  $v \in V(G) \setminus \{s\}$ ;
    $l(s) \leftarrow 0$ ;  $R \leftarrow \emptyset$ ;
2 Finde einen Knoten  $v \in V(G) \setminus R$  mit  $l(v) = \min_{w \in V(G) \setminus R} l(w)$ ;
3  $R \leftarrow R \cup \{v\}$ ;
4 forall  $w \in V(G) \setminus R$  mit  $(v, w) \in E(G)$  do
   |   if  $l(w) > l(v) + c((v, w))$  then
   |   |    $l(w) \leftarrow l(v) + c((v, w))$ ;
   |   |    $p(w) \leftarrow v$ ;
5 if  $R \neq V(G)$  then
   |   goto (2);

```

---



---

**Algorithmus 3.5 : MOORE-BELLMAN-FORD**


---

**Eingabe:** Digraph  $G$ , konservative Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ , Startknoten  $s$ .

**Ausgabe:** Für jeden Knoten  $v \in V(G)$  die Angaben

$l(v)$ : Länge des kürzesten  $s$ - $v$ -Pfad.

$p(v)$ : Vorgänger von  $v$  in einem kürzesten  $s$ - $v$ -Pfad.

(falls  $v$  nicht erreichbar ist, gilt  $l(v) = \infty$ ,  $p(v) = \text{NIL}$ ).

**Komplexität:**  $\mathcal{O}(n \cdot m)$ .

```

1  $l(v) \leftarrow \infty$  für alle  $v \in V(G) \setminus \{s\}$ ;
    $l(s) \leftarrow 0$ ;
2 for  $i = 1, \dots, n - 1$  do
   |   foreach Kante  $(v, w) \in E(G)$  do
   |   |   if  $l(w) > l(v) + c((v, w))$  then
   |   |   |    $l(w) \leftarrow l(v) + c((v, w))$ ;
   |   |   |    $p(w) \leftarrow v$ ;

```

---

## 3.3 Kürzeste Wege zwischen allen Knoten

---

**Algorithmus 3.8** : FLOYD, WARSHALL

---

**Eingabe:** Digraph  $G$ , konservative Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ .**Ausgabe:** Für jedes Knotenpaar  $i, j \in V(G)$  die Angaben $l_{ij}$ : Länge eines kürzesten  $i$ - $j$ -Pfad. $p_{ij}$ : Vorgänger von  $j$  in einem kürzesten  $i$ - $j$ -Pfad.(Betrachte  $V(G) = \{1, 2, \dots, n\}$  statt  $V(G) = \{v_1, v_2, \dots, v_n\}$ ).**Komplexität:**  $\mathcal{O}(n^3)$ .

```

1  $l_{ij} \leftarrow c((i, j))$  für alle  $(i, j) \in E(G)$ ;
   $l_{ij} \leftarrow \infty$  für alle  $(i, j) \in V(G)^2 \setminus E(G)$ ,  $i \neq j$ ;
   $l_{ii} \leftarrow 0$  für alle  $i \in V(G)$ ;
   $p_{ij} \leftarrow i$  für alle  $(i, j) \in E(G)$ ;
2 for  $j = 1, \dots, n$  do
  | for  $i = 1, \dots, n$  do
  | | if  $i \neq j$  then
  | | | for  $k = 1, \dots, n$  do
  | | | | if  $k \neq j$  then
  | | | | | if  $l_{ik} > l_{ij} + l_{jk}$  then
  | | | | | |  $l_{ik} \leftarrow l_{ij} + l_{jk}$ ;
  | | | | | |  $p_{ik} \leftarrow p_{jk}$ ;

```

---

## 4 Netzwerkflüsse

### 4.2 Maximaler Netzwerkfluss

---

**Algorithmus 4.5** : FORD, FULKERSON

---

**Eingabe:** Netzwerk  $(G, u, s, t)$ .

**Ausgabe:**  $s$ - $t$ -Fluss mit maximalem Wert.

**Komplexität:** Schlecht Laufzeit (abhängig von den Kapazitäten) und terminiert für irrationale Kapazitäten mitunter nicht, bzw. konvergiert dabei nicht unbedingt gegen den richtigen Wert.

- 1  $f(e) \leftarrow 0$  für alle  $e \in E(G)$ ;
  - 2 Finde einen  $f$ -augmentierenden Pfad  $P$  in  $G_f$ ;  
**if** *es existiert keiner* **then**  
     $\perp$  **stopp** ;
  - 3 Bestimme  $\gamma \leftarrow \min_{e \in E(P)} u_f(e)$ ;  
    Erhöhe  $f$  entlang  $P$  um  $\gamma$ ;  
    **goto** (2);
- 

---

**Algorithmus 4.11** : EDMONDS-KARP

---

**Eingabe:** Netzwerk  $(G, u, s, t)$ .

**Ausgabe:**  $s$ - $t$ -Fluss mit maximalem Wert.

**Komplexität:**  $\mathcal{O}(m^2 \cdot n)$ . Unabhängig von den Kapazitäten stoppt er nach  $\frac{mn}{2}$  Augmentierungen.

- 1  $f(e) \leftarrow 0$  für alle  $e \in E(G)$ ;
  - 2 Finde einen *kürzesten*  $f$ -augmentierenden Pfad  $P$  in  $G_f$ ;  
**if** *es existiert keiner* **then**  
     $\perp$  **stopp** ;
  - 3 Bestimme  $\gamma \leftarrow \min_{e \in E(P)} u_f(e)$ ;  
    Erhöhe  $f$  entlang  $P$  um  $\gamma$ ;  
    **goto** (2);
-

## 4.3 Der Preflow-Push-Algorithmus von Goldberg und Tarjan

---

**Algorithmus 4.17** : Push-Relabel

---

**Eingabe:** Netzwerk  $(G, u, s, t)$ .**Ausgabe:**  $s$ - $t$ -Fluß mit maximalem Wert.**Komplexität:** Hängt von der Implementierung ab (Reihenfolge der PUSH- und RELABEL-Operationen).**Anmerkung:** Während des Algorithmus ist  $f$  immer ein  $s$ - $t$ -Präfluss und  $\Psi$  ein Distanzlabeling bezüglich  $f$ .

- 1  $f(e) \leftarrow u(e)$  für jedes  $e \in \delta^+(s)$ ;  
 $f(e) \leftarrow 0$  für jedes  $e \in E(G) \setminus \delta^+(s)$ ;
- 2  $\Psi(s) \leftarrow n$  und  $\Psi(v) \leftarrow 0$  für  $v \in V(G) \setminus \{s\}$ ;
- 3 **while** es gibt einen aktiven Knoten  $v$  **do**

<b>if</b> es gibt eine zulässige Kante $e \in \delta^+(v)$ in $G_f$ <b>then</b>	PUSH( $e$ );
<b>else</b>	RELABEL( $v$ );

---

**Funktion** PUSH( $e \in E$ )

---

- 1  $\gamma \leftarrow \min\{\text{ex}_f(v), u_f(e)\}$ , wobei  $v$  der Knoten mit  $e \in \delta^+(v)$  ist;
- 2 Augmentiere  $f$  um  $\gamma$  entlang  $e$ ;

---

**Funktion** RELABEL( $v \in V$ )

---

- 1  $\Psi(v) \leftarrow \min\{\Psi(w) + 1 \mid e = (v, w) \in E(G_f)\}$ ;
-

---

**Algorithmus 4.23** : GOLDBERG und TARJAN

---

**Eingabe:** Netzwerk  $(G, u, s, t)$ .Eine Liste  $\text{list}(v)$  der Kanten, die zu  $v$  inzident sind.Ein Pointer  $\text{curr}(v)$  auf das aktuelle Element von  $\text{list}(v)$ .**Ausgabe:**  $s$ - $t$ -Fluß mit maximalem Wert.**Komplexität:**  $\mathcal{O}(n^3)$ .**Anmerkung:** Während des Algorithmus ist  $f$  immer ein  $s$ - $t$ -Präfluss und  $\Psi$  ein Distanzlabeling bezüglich  $f$ .

```

1  $f(e) \leftarrow u(e)$  für jedes  $e \in \delta^+(s)$ ;
   $f(e) \leftarrow 0$  für jedes  $e \in E(G) \setminus \delta^+(s)$ ;
2  $\Psi(s) \leftarrow n$  und  $\Psi(v) \leftarrow 0$  für  $v \in V(G) \setminus \{s\}$ ;
3 forall  $v \in V(G)$  do
  | Setze  $\text{curr}(v)$  auf das erste Element von  $\text{list}(v)$ ;
4  $Q \leftarrow \{v \in V(G) \mid v \text{ ist aktiv}\}$ ;
  if  $Q == \emptyset$  then
  | stopp ;
5 forall  $v \in Q$  do
  | ERLEDIGE( $v$ );
goto (4);

```

---



---

**Funktion** ERLEDIGE( $v \in V$ )

---

```

1 Sei  $e$  die Kante, auf die  $\text{curr}(v)$  zeigt;
2 if  $e$  ist zulässig then
  | PUSH( $e$ );
else
  | if  $e$  ist die letzte Kante in  $\text{list}(v)$  then
  | | RELABEL( $v$ );
  | | Setze  $\text{curr}(v)$  auf das erste Element in  $\text{list}(v)$ ;
  | | return ;
  | else
  | | Setze  $\text{curr}(v)$  auf das nächste Element in  $\text{list}(v)$ ;
3 if  $ex_f > 0$  then
  | goto (1);

```

---



---

**Funktion** PUSH( $e \in E$ )

---

```

1  $\gamma \leftarrow \min\{ex_f(v), u_f(e)\}$ , wobei  $v$  der Knoten mit  $e \in \delta^+(v)$  ist;
2 Augmentiere  $f$  um  $\gamma$  entlang  $e$ ;

```

---



---

**Funktion** RELABEL( $v \in V$ )

---

```

1  $\Psi(v) \leftarrow \min\{\Psi(w) + 1 \mid e = (v, w) \in E(G_f)\}$ ;

```

---

## 5 Kostenminimale Flüsse

### 5.3 Min-Cost-Flow-Algorithmen

---

#### Algorithmus 5.8 : Cycle Cancelling

---

**Eingabe:** Digraph  $G$ , Kapazitäten  $u : E(G) \rightarrow \mathbb{R}_+$ ,  
 Bilanzen  $b : V(G) \rightarrow \mathbb{R}$  mit  $\sum_{v \in V(G)} b(v) = 0$ .

**Ausgabe:** Kostenminimaler  $b$ -Fluss.

**Komplexität:** Laufzeit kann schlecht sein.

- 1 Finde einen  $b$ -Fluss  $f$  oder stelle fest, dass keiner existiert;
  - 2 Finde einen negativen Kreis  $C$  in  $G_f$ ;  
 if  $G_f$  hat keinen negativen Kreis then  
   └ **stopp** ;
  - 3 Berechne  $\gamma \leftarrow \min_{e \in E(C)} u_f(e)$ ;  
 Augmentiere  $f$  entlang  $C$  um  $\gamma$ ;  
**goto** (2);
- 

---

#### Algorithmus 5.9 : Minimum Mean Cycle Cancelling

---

**Eingabe:** Digraph  $G$ , Kapazitäten  $u : E(G) \rightarrow \mathbb{R}_+$ ,  
 Bilanzen  $b : V(G) \rightarrow \mathbb{R}$  mit  $\sum_{v \in V(G)} b(v) = 0$ .

**Ausgabe:** Kostenminimaler  $b$ -Fluss.

**Komplexität:**  $\mathcal{O}(m^3 \cdot n^2 \cdot \log n)$ .

- 1 Finde einen  $b$ -Fluss  $f$  oder stelle fest, dass keiner existiert;
  - 2 Finde einen negativen Kreis  $C$  in  $G_f$  von minimalem Durchschnittsgewicht;  
 if  $G_f$  hat keinen negativen Kreis then  
   └ **stopp** ;
  - 3 Berechne  $\gamma \leftarrow \min_{e \in E(C)} u_f(e)$ ;  
 Augmentiere  $f$  entlang  $C$  um  $\gamma$ ;  
**goto** (2);
-

---

**Algorithmus 5.13** : Minimum Mean Cycle
 

---

**Eingabe:** Digraph  $G$ , konservative Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ .

**Ausgabe:** Kreis  $C$  minimalen Durchschnittsgewichts.

**Komplexität:**  $\mathcal{O}(n \cdot (m + n))$ .

- 1 Setze  $F_0(s) \leftarrow 0$  und  $F_0(x) \leftarrow \infty$  für alle  $x \in V(G) \setminus \{s\}$ ;
  - 2 **for**  $k = 1, \dots, n$  **do**
    - forall**  $x \in V(G)$  **do**
      - $F_u(x) \leftarrow \infty$ ;
      - foreach** Kante  $(w, x) \in \delta^-(x)$  **do**
        - if**  $F_{k-1}(w) + c((w, x)) < F_k(x)$  **then**
          - $F_k(x) \leftarrow F_{k-1}(w) + c((w, x))$ ;
          - //Setze Vorgänger auf kürzeste Kantenfolge mit  $k$  Kanten
          - $p_k(x) \leftarrow w$ ;
  - 3 **if**  $F_n(x) == \infty$  für alle  $x \in V(G)$  **then**
    - stopp** ;
  - 4 Sei  $x$  mit  $\max_{\substack{0 \leq k \leq n-1 \\ F_k(x) < \infty}} \frac{F_n(x) - F_k(x)}{n-k}$  minimal;
  - 5 Setze  $C$  auf irgendeinen Kreis in der Knotenfolge
    - $p_n(x), p_{n-1}(p_n(x)), p_{n-2}(p_{n-1}(p_n(x))), \dots$ ;
-

## 6 Maximale Matchings

### 6.3 Matching in allgemeinen Graphen

---

**Algorithmus 6.8** : Maximales Matching in bipartiten Graphen

---

**Eingabe:** Zusammenhängender, bipartiter Graph  $G$ .

**Ausgabe:** (Perfektes) Matching  $M$  maximaler Kardinalität.

**Komplexität:** ?

```

1  $M \leftarrow \emptyset$ ;
2 Wähle einen ungematchten Knoten  $r$ ;
   $T \leftarrow (\{r\}, \emptyset)$  und  $W(T) \leftarrow \{r\}$ ;
3 while es gibt eine Kante  $\{v, w\} \in E$  mit  $v \in W(T)$ ,  $w \notin V(T)$  do
  | if w ist ungematcht then
  | | Benutze  $\{v, w\}$ , um einen augmentierenden Pfad zu komplettieren;
  | | Augmentiere  $M$ ;
  | | if es gibt keinen ungematchten Knoten mehr then
  | | | return (perfektes Matching);
  | | else
  | | | Ersetze  $T$  durch  $(\{r\}, \emptyset)$ , wobei  $r$  ungematcht ist;
  | else
  | | Benutze  $\{v, w\}$ , um  $T$  zu erweitern;
4 return (maximales Matching);
```

---

---

**Algorithmus 6.13** : Blossom-Algorithmus für perfektes Matching
 

---

**Eingabe:** Zusammenhängender Graph  $G$ .

**Ausgabe:** Perfektes Matching  $M$  oder ein Beleg, dass keines existiert.

**Komplexität:**  $\mathcal{O}(n \cdot m \cdot \log n)$ .

```

1  $M' \leftarrow M \leftarrow \emptyset; G' \leftarrow G;$ 
2 Wähle einen ungematchten Knoten  $r$  in  $G'$ ;
   $T \leftarrow (\{r\}, \emptyset); W(T) \leftarrow \{r\}; S(T) \leftarrow \emptyset;$ 
3 while es gibt eine Kante  $\{v, w\} \in E'$  mit  $v \in W(T), w \notin S(T)$  do
  | case  $w$  ist ungematcht
  |   Benutze  $\{v, w\}$ , um  $M'$  zu augmentieren;
  |   Erweitere  $M'$  zu einem Matching  $M$  von  $G$ ;
  |   Ersetze  $M'$  durch  $M$ ,  $G'$  durch  $G$ ;
  |   if es gibt keinen ungematchten Knoten in  $G$  then
  |     | return (perfektes Matching in  $G$ );
  |   else
  |     | Ersetze  $T$  durch  $(\{r\}, \emptyset)$ , wobei  $r$  in  $M$  ungematcht ist;
  |   case  $w \notin V(T)$ ,  $w$  ist in  $M'$  gematcht
  |     | Benutze  $\{v, w\}$ , um  $T$  zu erweitern;
  |   case  $w \in V(T)$ , genauer  $w \in W(T)$ 
  |     | Benutze  $\{v, w\}$  zum Schrumpfen;
  |     | Aktualisieren von  $M'$  und  $T'$ ;
4 return ( $G', M', T$ :  $G$  hat kein perfektes Matching);

```

---

## 7 Gewichtsminimales Matching

### 7.1 Bipartite Graphen

---

**Algorithmus 7.1** : Gewichtsminimales perfektes Matching in bipartiten Graphen

---

**Eingabe:** Zusammenhängender, bipartiter Graph  $G = (V, E)$ , Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ .

**Ausgabe:** Perfektes Matching  $M$  minimalen Gewichts, Optimalitätsbeleg durch passende Dualvariable,  
oder die Erkenntnis (samt Beleg), dass  $G$  kein perfektes Matching hat.

**Komplexität:**  $\mathcal{O}(n^2 \cdot m)$ .

```

1 Sei  $y$  eine zulässige Duallösung;
  Sei  $M$  ein Matching in  $G^=$ ;
2  $T \leftarrow (\{r\}, \emptyset)$ , wobei  $r$  in  $M$  ungematcht ist;
3 loop
  while es gibt  $\{v, w\} \in E^=$  mit  $v \in W(T)$ ,  $w \notin V(T)$  do
    if  $w$  ist ungematcht then
      Benutze  $\{v, w\}$ , um  $M$  zu augmentieren;
      if es gibt keinen ungematchten Knoten mehr then
        | return (perfektes Matching, Duallösung  $y$ );
      else
        | Ersetze  $T$  durch  $(\{r\}, \emptyset)$ , wobei  $r$  ungematcht ist;
    else
      | Benutze  $\{v, w\}$ , um  $T$  zu erweitern;
  if für jedes  $\{v, w\} \in E$  mit  $v \in W(T)$  gilt  $w \in S(T)$  then
    | stopp ( $G$  hat kein perfektes Matching (Beleg ist  $S(T)$ ));
  else
    |  $\varepsilon \leftarrow \min\{\bar{c}_{\{v,w\}} \mid v \in W(T), w \notin V(T)\}$ ;
    | Ersetze  $y_v$  durch  $y_v + \varepsilon$  für  $v \in W(T)$  und durch  $y_v - \varepsilon$  für  $v \in S(T)$ ;

```

---

## 7.2 Allgemeine Graphen

---

**Algorithmus 7.3** : Blossom-Algorithmus für gewichtsminimales perfektes Matching

---

**Eingabe:** Zusammenhängender Graph  $G = (V, E)$ , Kantengewichte  $c : E(G) \rightarrow \mathbb{R}$ .**Ausgabe:** Kostenminimales perfektes Matching in  $G$  samt Optimalitätsbeleg,  
oder die Angabe (mit Beleg), dass  $G$  kein perfektes Matching hat.**Komplexität:** Polynomielle Laufzeit.

```

1 Sei  $y$  eine zulässige Lösung des Dualproblems;
  Sei  $M'$  ein Matching in  $G^=$ ;  $G' \leftarrow G$ ;
2  $T \leftarrow (\{r\}, \emptyset)$ , wobei  $r$  ein in  $M'$  ungematchter Knoten in  $G'$  ist;
3 loop
  | case es gibt  $e = \{v, w\} \in E^=$  mit  $v \in W(T)$ ,  $w \notin V(T)$  und  $w$  ungematcht in  $M'$ 
  |   | Benutze  $\{v, w\}$ , um  $M'$  zu augmentieren;
  |   | if es gibt keinen ungematchten Knoten in  $M'$  then
  |   |   | Erweitere  $M'$  zu einem perfekten Matching  $M$  von  $G$ ;
  |   |   | stopp ;
  |   | else
  |   |   | Ersetze  $T$  durch  $(\{r\}, \emptyset)$ , wobei  $r$  in  $M'$  ungematcht ist;
  | case es gibt  $e = \{v, w\} \in E^=$  mit  $v \in W(T)$ ,  $w \notin V(T)$  und  $w$  gematcht in  $M'$ 
  |   | Benutze  $\{v, w\}$ , um  $T$  zu erweitern;
  | case es gibt  $e = \{v, w\} \in E^=$  mit  $v \in W(T)$ ,  $w \in V(T)$ 
  |   | Benutze  $\{v, w\}$ , um einen ungeraden Kreis zu schrumpfen;
  |   | Aktualisieren von  $M'$ ,  $T'$  und  $c'$ ;
  | case es gibt einen Pseudoknoten  $v \in S(T)$  mit  $y_v = 0$ 
  |   | Erweitere  $v$  und aktualisiere  $M'$ ,  $T'$  und  $c'$ ;
  | case sonst
  |   | if für jede Kante  $e = \{v, w\} \in E$  mit  $v \in W(T)$  hat  $w \in S(T)$  then
  |   |   | stopp ( $G$  hat kein perfektes Matching);
  |   | else
  |   |   | ÄNDERE( $y$ );

```

---

**Funktion** ÄNDERE( $y$ )

---

```

1  $\varepsilon_1 \leftarrow \min\{\bar{c}_e \mid e \text{ verbindet } v \in W(T) \text{ mit } w \notin V(T)\};$ 
   $\varepsilon_2 \leftarrow \min\{\frac{\bar{c}_e}{2} \mid e \text{ verbindet in } G' \text{ zwei Knoten in } W(T)\};$ 
   $\varepsilon_3 \leftarrow \min\{y_v \mid v \in S(T), v \text{ ist Pseudoknoten in } G'\};$ 
2  $\varepsilon \leftarrow \min\{\varepsilon_1, \varepsilon_2, \varepsilon_3\};$ 
   $y_v \leftarrow \begin{cases} y_v + \varepsilon & \text{für } v \in W(T) \\ y_v - \varepsilon & \text{für } v \in S(T) \\ y_v & \text{sonst} \end{cases} ;$ 
3

```

---