Fluss: Funktion f: V^2 -> R
(1) f <= k
(2) A x,y e V: f(x,y) = -f(y,x)
(3) A x e V\{q,s}: 0 = Sum(f(x,V))

bei präfluss statt (3):
A x e V\{q}: ü(x) = Sum(f(V,x)) >= 0

Bem Residualkapazität k_f = k - f
k_f(x,y) + k_f(y,x) = k(x,y) + k(y,x)

An die Spitze
```
generiere L
x = head(L)
A x e V: i_x = Grad(x)
While x != NIL
        h_alt = h(x)
        LEERE(x)
        if h_alt < h(x)
                x in front of L
        x = nachfolger von x in L
```

Höhenfunktion h: V -> N0:
(1) h(q) = |V|
(2) h(s) = 0
(3) A (x,y) e E_f: h(x) - h(y) <= 1

LEERE
```
solange ü(x) > 0
        if i_x > 0
                y = n_x(i_x)
                if (x,y) puschbar, pushe es
                else i_x --
        else Lifte(x)
                i_x = Grad(x)
```

mit n_x array der nachbarn von x

Pusch(x,y) bis zu kapazität des überschusses aus x nach y nur wenn,
ü(x) > 0
x e V\{q,s}
h(x) - h(y) = 1

SCHNITT
```
While |V| >= 3
        wähle zufällig Kante (x,y) e E
        entferne alle (x,y) aus E
        verschmelze x mit y
        gib S = E aus
```

Lifte(x)
h(x) = 1 + min{h(y) | (x,y) e E_f} nur wenn,
x e V\{q,s}
ü(x) > 0
(3) h(x) <= min {h(y) | (x,y) e E_f}
zu (3): keine tiefergelegenen nachbarknoten im residualgraph

Sym DIff D = (P\Q) u (Q\P)
besteht aus Wegen, deren Kanten alternierend in P und Q liegen
mit P eine Paarung und Q eine maximale Paarung. Zyklen haben daher gerade Länge

E[Sum(X_i)] = Sum(E[X_i])
Pr[E1 | E2] = Pr[E1 ^ E2] / Pr[E2] bedingte wahrscheinlichkeit für eintreten von Ereignis1 unter Voraussetzung E2
Wenn: Pr[E_1 | E_2] = Pr[E_1] oder Pr[E_1] * Pr[E_2] = Pr[E_1 ^ E_2] heißen E_1 und E_2 unabhängig

Folgerung: Pr[Cut(E_i)] = Pr[E_1 | E_2 ^ ... ^ E_n] * ... * Pr[E_n-1 | E_n] * Pr[E_n]

VonNeumannMinMax W := max_p(min_q(p^t*M*q)) = min_q(max_p(p^t*M*q)) gilt immer
und ein paar optimaler Strategien (p,q) heißt Lösung

Für festes p wird p^t*M*q minimal, wenn q = e_j <- d.h. wähle minimalen Eintrag aus p^t*M
damit Loomis Satz: W := max_p(min_j(p^t*M*e_j)) = min_q(max_i(e_i^t*M*q))

Strategien
-Zeile i ist optimale Strategie von A, falls sie A's Mindestgewinn: min_j(m_ij) maximiert
-Spalte j ist optimale Strategie von B, falls sie B's Maximalverlust: max_i(m_ij) minimiert
-Wählen also beide optimale Strategien gilt:
    max_i(min_j(m_ij)) <= Gewinn(A) <= min_j(max_i(m_ij))

Bei gemischten Strategien A verfolt p und B verfolgt q:
-E[Gewinn(A)] = p^t*M*q
-Strategie von A optimal, wenn min_q(p^t*M*q) maximal
-Strategie von B optimal, wenn max_p(p^t*M*q) minimal
-Wählen beide optimal:
    max_p(min_q(p^t*M*q)) <= E[Gewinn(A)] <= min_q(max_p(p^t*M*q))

WERT
1: (y,z) = Paar der Kinder von x in zufälliger Reihenfolge
2:if (WERT(y) == 1) return 0
3: else return 0 NOR WERT(z)

Wert und Lösung bei reinen Strategien
falls W := min_j(max_i(m_ij)) = max_i(min_j(m_ij))
heißt W wert des spiels und das dazugehörige (i,j) Lösung

Yaos Technik Für alle p und q:
min_j p^t*M*e_j <= K_v <= max_i e_i^t*M*q
damit kann die worst-case Laufzeit aller stochastischen Algorithmen durch Laufzeit des schnellsten deterministischen nach unten abgeschätzt werden

Komplexitäten
m_ij ist Laufzeit von A_j(E_i)
max_i(m_ij) ist worst-case Laufzeit von A_j
min_j(max_i(m_ij)) ist deterministische komplexität von P bzgl A =: K_d
min_j(m_ij) ist Laufzeit des schnellsten Algos für E_i
max_i(min_j(m_ij)) ist stochastische Komplexität von P bzgl A =: K_s

K_s <= K_d

gemischte Strategien:
    E[Laufzeit] = p^t*M*q
    max_p(min_q(p^t*M*q)) heißt Verteilungskomplexität =: K_v
    K_s <= K_v <= K_d (mit Loomi und Neumann)