## Übersicht Laufzeiten Algorithmen

| | | |
|---|---|---|
| **Allgemein** | Tiefensuche (DFS) & Breitensuche (BFS) | $\boldsymbol{O(\lvert V\rvert + \lvert E\rvert)}$ |
| | Kruskal | $\boldsymbol{O(\lvert E\rvert \cdot log(\lvert E\rvert)}$ |
| | Prim & Boruvka | $\boldsymbol{O((\lvert E\rvert + \lvert V\rvert) \cdot log(\lvert V\rvert))}$ |
| | Artikulationsknoten finden | $\boldsymbol{O(\lvert E\rvert)}$ |
| | Brücken finden | $\boldsymbol{O(\lvert E\rvert)}$ |
| **Kreise** | Eulertour | $\boldsymbol{O(\lvert E\rvert)}$ |
| | Hamiltonkreis | $\boldsymbol{O(\lvert V\rvert^2 \cdot 2^{\lvert V\rvert})}$ |
| | Hamiltonkreis mit $\forall x \in V: \deg(v) \geq \frac{\lvert V\rvert}{2}$ | $\boldsymbol{O(\lvert V\rvert^2)}$ |
| | Travelling Salesman Problem 2-Approxiamtion | $\boldsymbol{O(\lvert V\rvert^2)}$ |
| | Travelling Salesman Problem 3/2-Approxiamtion | $\boldsymbol{O(\lvert V\rvert^3)}$ |
| **Matching** | Greedy-Matching | $\boldsymbol{O(\lvert E\rvert)}$ $\boldsymbol{mit\ \lvert M_{Greedy}\rvert \geq \frac{1}{2}\lvert M_{Max}\rvert}$ |
| | Augmentierenden Pfad finden | $\boldsymbol{O(\lvert V\rvert + \lvert E\rvert)}$ |
| | Augmenting_Path (nur für bipartite Graphen) | $\boldsymbol{O((\lvert V\rvert + \lvert E\rvert) \cdot \lvert E\rvert)}$ |
| | Hopcroft und Karp Algorithmus (nur für bipartite Graphen) | $\boldsymbol{O\left((\lvert V\rvert + \lvert E\rvert) \cdot \sqrt{\lvert V\rvert}\right)}$ |
| | Blossoms's Algortihmus | $\boldsymbol{O(\lvert E\rvert \cdot \lvert V\rvert^2)}$ |
| | Perfektes Matching in $2^k$-regulären bipartiten Graphen | $\boldsymbol{O(\lvert E\rvert)}$ |
| **Färbung** | Greedy-Färbung *C(G)* auf beliebigen zshg. Graphen G | $\boldsymbol{O(\lvert E\rvert)}$ $\boldsymbol{mit\ \chi(G) \leq C(G) \leq \Delta(G) + 1}$ |
| | Brooks: Färbung eines zshg. Graphen *G* mit $G \neq K_n\ und\ G \neq C_{2n+1}$ | $\boldsymbol{O(\lvert E\rvert)}$ [1] $\boldsymbol{mit\ \chi(G) \leq C(G) \leq \Delta(G)}$ |
| | Färbung eines 3-färbbaren Graphen | $\boldsymbol{O(\lvert E\rvert)}$ $\boldsymbol{mit\ O\left(\sqrt{\lvert V\rvert}\right)\ Farben}$ |
| | Färbung eines Graphen G für den gilt: jeder induzierte Subgraph von G hat einen Knoten v für den gilt $deg(v) \leq k$ (so gilt $x(G) \leq k+1$) | $\boldsymbol{O(\lvert E\rvert)}$ $\boldsymbol{mit\ O(k+1)\ Farben}$ |
| **Randomisierte Algorithmen** | Quickselect *(finde das k-kleinste Element)* | $\boldsymbol{O(n)}$ |
| | Miller-Rabin-Primzahlentest | $\boldsymbol{O(\ln n)}$ [2] $\boldsymbol{'keine\ PZ'\ mit\ p > \frac{3}{4}}$ |
| | Duplikate finden mit Hashmap | $\boldsymbol{O(n \cdot \ln n)}$ $\underbrace{O(n)}_{\text{hashen}} + \underbrace{O(n \cdot log(n))}_{\text{sortieren}} + \underbrace{O(n + \lvert Dupl(S)\rvert)}_{\text{durchlaufen}}$ |
| | Duplikate finden mit Bloomfilter *(k-viele Hashfunktionen)* | $\boldsymbol{O(n \cdot \ln n)}$ [3] |
| | Finde bunten Pfad *Bunt(G, i)* | $O\left(\sum_{v \in V} \deg(v) \cdot \binom{k}{i} \cdot i\right) = \boldsymbol{O\left(\binom{k}{i} \cdot i \cdot m\right)}$ |
| | Colorful-Path Problem *Regenbogen(G, γ)* | $O(\lvert V\rvert + \sum_{i=1}^{k-1} Bunt(G,i) + \lvert V\rvert) = \boldsymbol{O(2^k km)}$ [4] $\boldsymbol{mit\ p_{Erfolg} \geq e^{-k}}$ |
| | Long-Path Problem *(stelle fest, ob es einen Pfad der Länge B gibt)* | $MC: wiederhole\ \lceil \lambda e^k \rceil - Mal\ Regenbogen(G, \gamma)$ $\boldsymbol{O(\lambda(2e)^k km)}$ $\boldsymbol{mit\ p_{Fehler} \leq e^{-\lambda}}$ [5] |

[1] Heuristik in O(|E|) liefert Reihenfolge, für die der Greedy-Algorithmus höchstens Δ(G) Farben benötigt. Heuristik: Knoten dem Grad aufsteigend durchlaufen, färben und dann löschen.

[2] Die Fehlerwahrscheinlichkeit kann mit Satz 2.74 beliebig klein gemacht werden.

[3] wählen wir k und m gross, reduziert sich die #falscherEinträge, jedoch: k gross → Algo. langsamer, m gross → mehr Speicher

[4] D.h., für $k \leq \log n$ ist die Laufzeit $O(mn \log n)$ und für $k = O(\log n)$ ist die Laufzeit $O(\text{poly}(n))$

[5] Antwortet der Algorithmus «Graph enthält keinen Pfad der Länge k», so stimmt diese Aussage mit einer Wahrscheinlichkeit von mindestens $1 - e^{-\lambda}$.

| | | |
|---|---|---|
| **Flüsse in Netzwerke** | Augmentierender-Pfad finden (mit BFS) | $\boldsymbol{O(m)}$ |
| | Ford-Fulkerson [6] *(suche $O(nU) - Mal$ einen augm. Pfad)* | $\boldsymbol{O(mnU)}$ wobei U := obere Schranke für Kap. |
| | Bestimme MaxFlow mit Capacity-Scaling | $\boldsymbol{O(mn(1 + \log U))}$ |
| | Bestimme MaxFlow mit Dynamic Trees | $\boldsymbol{O(mn \log n)}$ |
| | Finde ein kardinalitätsmax. Matching in bipartiten Graph | $\boldsymbol{O(mn)}$ |
| | Finde alle Kantendisjunkte u-v-Pfade | $\boldsymbol{O(mn)}$ |
| **Minimale Schnitte** | Fixiere $s \in V$ und berechne MaxFlow zu allen $t \in V$ (nehme Min.) | $(n-1) \cdot O(mn \log n) = \boldsymbol{O(n^4 \log n)}$ |
| | *Cut(G)* *(wähle zufällig Kante aus G und kontrahiere sie, bis $V(G) = 2$ )* | $\boldsymbol{O(n^2)}$ $mit\ Pr[Cut(G) gibt\ \mu(G) aus] \geq \frac{1}{\binom{n}{2}}$ [7] |
| | MonteCarlo-Algo: wiederhole $\boldsymbol{\lambda}\binom{n}{2}$-mal $Cut(G)$ | $\boldsymbol{O(\lambda n^4)}$ $\boldsymbol{mit\ p_{Erfolg} \geq 1 - e^{-\lambda}}$ [8] |
| | Verbesserung durch Bootstrapping | $\boldsymbol{O(\lambda n^3)}$ |
| | Grenzwert des Bootstrapping | $\boldsymbol{O(n^2 \cdot poly(\log n))}$ |
| **Kleinst. umschl. Kreis** | *CompleteEnumeration(P)*<br>→ durchlaufe $\binom{n}{3}$ Mengen Q, berechne C(Q) in O(1), prüfe ob $P \subseteq C(Q)$ in O(n) | $\boldsymbol{O(n^4)}$ |
| | *CompleteEnumerationSmart(P)*<br>→ durchlaufe $\binom{n}{3}$ Mengen Q, berechne C(Q) in O(1), merke max. Radius in O(1) | $\boldsymbol{O(n^3)}$ |
| | *Randomised_PrimitiveVersion(P)*<br>→ Wähle Q (bzw. 3 Punkte) zufällig gleichverteilt: $p_{korrektes\ Q} \geq 1/\binom{n}{3}$ | $\boldsymbol{O(n^4)}$ |
| | *Randomised_CleverVersion(P)*<br>→ wähle 11 Punkte zufällig gleichvrtl., bestimme C(Q), verdopple Punkte ausserhalb | $\underbrace{O(\log n)}_{\#Runden} \cdot \underbrace{O(n)}_{\text{Laufzeit/Runde}} = \boldsymbol{O(n \log n)}$ |
| **Konvexe Hülle** | *FindNext(q)* *(finde rechtesten Punkt von q aus)* | $\boldsymbol{O(n)}$ |
| | JarvisWrap | $\boldsymbol{O(nh) \leq O(n^2)}$ |
| | Untere Schranke für ConvexHull Algorithmen | $\boldsymbol{O(n \log n)}$ |
| | LocalRepair | $\underbrace{\boldsymbol{O(n \log n)}}_{\boldsymbol{sortieren}} + \underbrace{\boldsymbol{O(n)}}_{\boldsymbol{\#Tests}} = \boldsymbol{O(n \log n)}$ |

[6] Achtung: alle Kapazitäten müssen aus $\mathbb{Q}$ sein. Mit Kapazitäten aus $\mathbb{R}$ können wir nicht garantieren, dass der Algo. terminiert.

[7] D.h., der Erwartungswert der #Wiederholungen bis wir das erste Mal μ(G) ausgeben ist $\leq \binom{n}{2}$.

[8] Wählen wir $\lambda = \ln n$, so erhalten wir eine Laufzeit von $O(n^4 \log n)$ und $p_{Fehler} \geq \frac{1}{n}$ – also sogar schlechter wie vorhin