2.4 Schnelle Matrixmultiplikation

Problem: Matrixmultiplikation

$$C = A \cdot B$$

„Zeile mal Spalte“: Definition der Matrixmultiplikation

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

„Zeile mal Spalte“: $\Theta(n^3)$ Operationen (Addition oder Multiplikation)

n Multiplikationen, n – 1 Additionen pro Zeile/Spalte Paar.

Es gibt n^2 Paare.

Algorithmus von Strassen

Frage: Berechnet der Algorithmus „Zeile mal Spalte“ das Produkt zweier Matrizen mit der minimalen Anzahl Operationen?

(Überraschende) Antwort: Nein!

Algorithmus von Strassen: $O(n^{2.81})$ Operationen

Algorithmus von Strassen

Spezialfall: A und B sind 2x2 Matrizen

Wir berechnen das Produkt auf spezielle Weise

$$C = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} = \begin{pmatrix} m_2 + m_3 & m_1 + m_2 + m_5 + m_6 \\ m_1 + m_2 + m_4 - m_7 & m_1 + m_2 + m_4 + m_5 \end{pmatrix}$$

8 Multiplikationen

7 Multiplikationen

$$\begin{aligned} m_1 &= (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11}) \\ m_2 &= a_{11}b_{11} \\ m_3 &= a_{12}b_{21} \\ m_4 &= (a_{11} - a_{21})(b_{22} - b_{12}) \\ m_5 &= (a_{21} - a_{22})(b_{12} - b_{11}) \\ m_6 &= (a_{12} - a_{21} + a_{11} - a_{22})b_{22} \\ m_7 &= a_{22}(b_{11} + b_{22} - b_{12} - b_{21}) \end{aligned}$$

Korrektheit: einfach nachrechnen … ☺

Anzahl Operationen: 7 Multiplikationen und 21 Additionen (bzw 15 Additionen, bei geschickter Speicherung von Zwischenergebnissen)

Algorithmus von Strassen (II)

Allgemeiner Fall: Annahme: A und B sind $2^k x 2^k$ Matrizen

$$C = A \cdot B$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Rekursive Anwendung des Algorithmus auf die 2x2 Matrix bestehend aus $2^{k-1} x 2^{k-1}$ Untermatrizen

---

Was ist wenn A und B nicht $2^k x 2^k$ Matrizen sind?

A und B durch hinzufügen einer entsprechend dimensionierten Einheitsmatrix zu einer $2^k x 2^k$ Matrix erweitern.

Laufzeit

f(n) := Anzahl der elementaren Operationen (Additionen und Multiplikationen) die für die Multiplikation zweier nxn Matrizen benötigt werden.

Für eine $2^k x 2^k$ Matrix sind

18 $2^{k-1} x 2^{k-1}$ Matrixadditionen und

7 $2^{k-1} x 2^{k-1}$ Matrixmultiplikationen nötig

$$f(n) = f(2^k) = 7 \cdot f(2^{k-1}) + 18 \cdot 2^{2(k-1)}$$

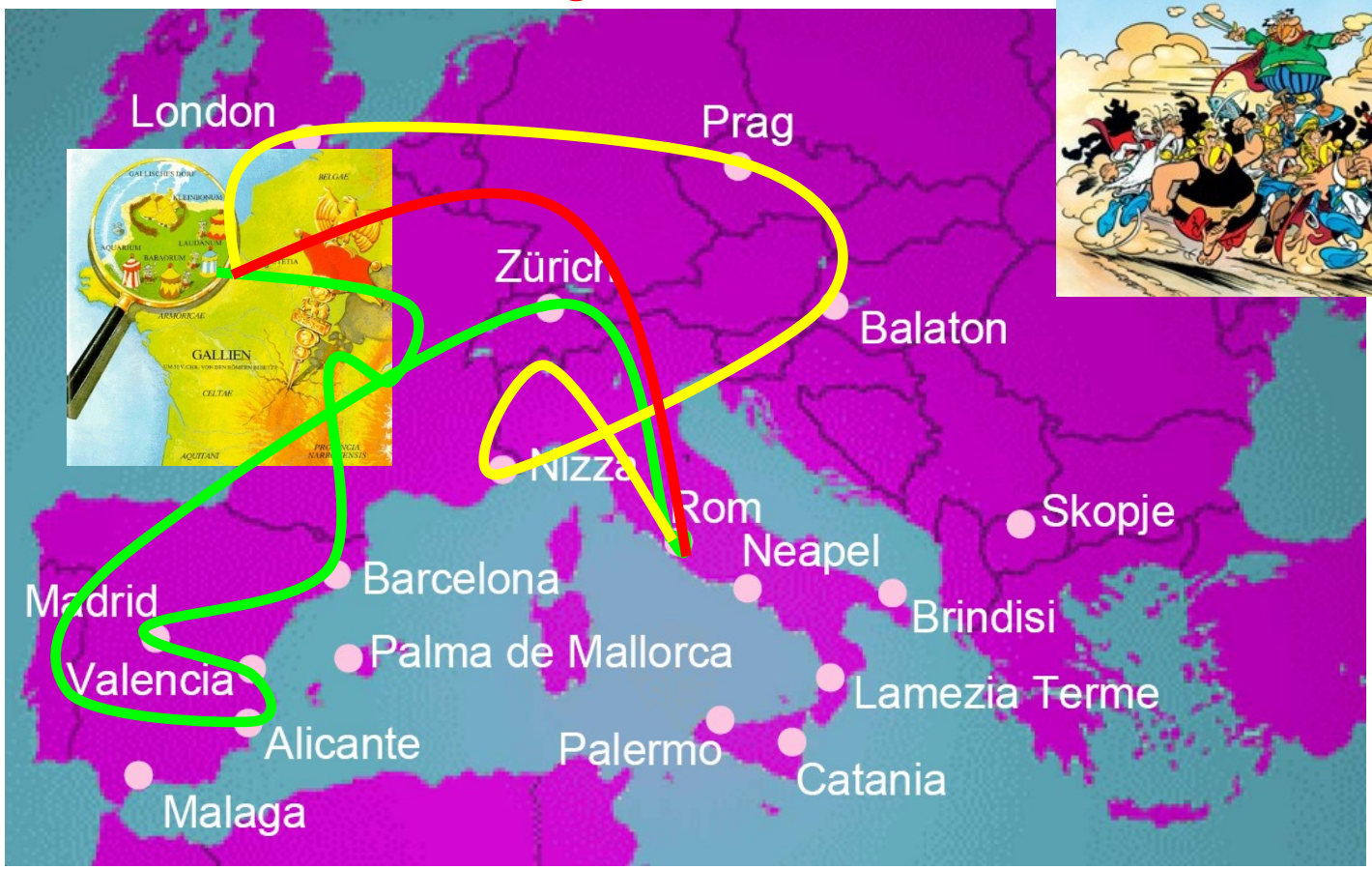
Satz

Das Produkt zweier nxn Matrizen kann in Zeit $O(n^{2.81})$ berechnet werden.

Coppersmith & Winograd: $O(n^{2.3727})$ bester bekannter Exponent (aber Algorithmus eher von theoretischem Interesse)

2.5 Kürzeste Pfade in Graphen

Alle Wege führen nach Rom!



Aber welcher ist der kürzeste?

Problemstellung

Gegeben

Netzwerk: Graph G = (V, E), Gewichtsfunktion w: E → **N**

Zwei Knoten: s, t

Kantenzug

Folge von Knoten $v_0, v_1, v_2, \ldots, v_{n-1}, v_n$ wobei

$\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{n-1}, v_n\}$ Kanten des Graphen sind.

Pfad

Ein Kantenzug, bei dem jeder Knoten höchstens einmal vorkommt.

Gewicht eines Pfades

Summe der Kantengewichte des Pfades

Gesucht

Kürzester Pfad (d.h. mit geringstem Gewicht) zwischen s und t.

Idee des Algorithmus von Dijkstra

Grundlegende Idee

Besuche in jedem Schritt den Knoten, der den kürzesten Abstand von den schon besuchten Knoten hat.

Die Abstände zu s werden in jedem Knoten gespeichert.

Algorithmus von Dijkstra

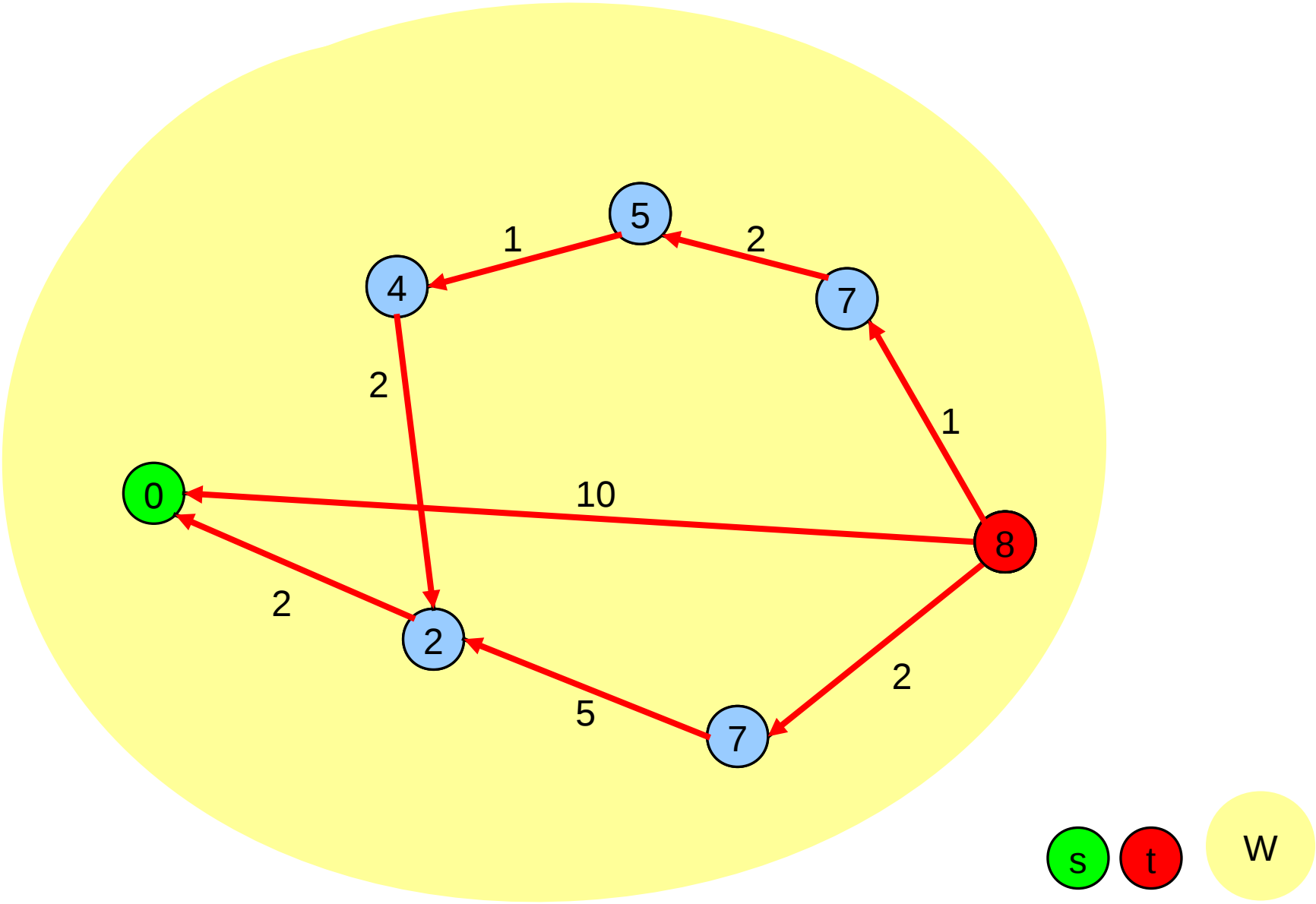
Algorithmus 2.5 Algorithmus von Dijkstra

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$
Ausgabe: Ein kürzester s–t-Pfad in N

{ *Initialisierung* }
$W := \emptyset$; $\rho[s] := 0$; pred$[s] := nil$;
for all $v \in V \setminus \{s\}$ **do** $\rho[v] := \infty$;

{ *Traversieren und Markieren* }
while $t \notin W$ **do**
    { *Traversieren* }
    Finde ein $x_0 \in V \setminus W$ mit $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\}$;
    $W := W \cup \{x_0\}$;
    { *Markieren* }
    for all $v \in \Gamma(x_0) \cap (V \setminus W)$ mit $\rho[v] > \rho[x_0] + \ell(x_0, v)$ **do**
        $\rho[v] := \rho[x_0] + \ell(x_0, v)$; pred$[v] := x_0$;

Ablauf des Algorithmus

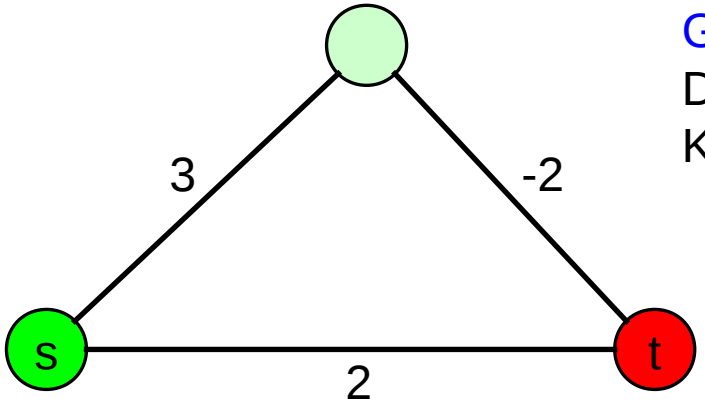


Korrektheit

Satz

Für nicht-negative Gewichtsfunktionen berechnet der Algorithmus von Dijkstra alle kürzesten Pfadlängen ab einem Startknoten s und benötigt höchstens $O(n^2)$ Schritte.

Die Voraussetzung nicht-negativer Gewichte ist wesentlich.



Gewicht eines kürzesten s-t-Pfades
Dijkstra: 2
Korrekt: 1

Beweis

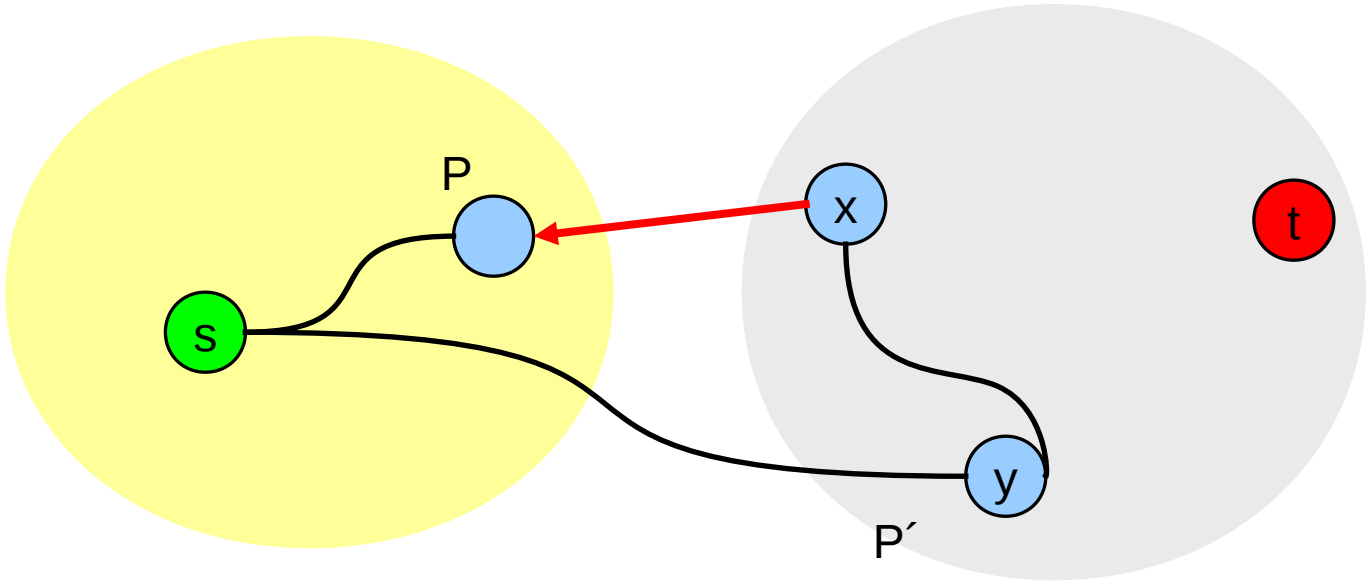
Zu zeigende Invarianten

1. Für alle Knoten v aus W ist der Wert der Markierung gleich der Länge eines kürzesten s-v Pfades; und es gibt einen solchen Pfad, der mit der Kante {pred[v], v} endet.
2. Für alle Knoten v aus V – W mit Markierung kleiner ∞ ist diese gleich einem kürzesten Pfad in dem Netzwerk auf den Knoten W U {v}; und es gibt einen solchen Pfad, der mit der Kante {pred[v], v} endet.

Beweis per Induktion

Verankerung: 1. und 2. offensichtlich bei der Initialisierung erfüllt

Induktionsschritt



- x sei Knoten der zu W hinzugefügt wird.
- Sei P´ ein anderer s-x Pfad
- Falls P‘ in W U {x}, dann folgt w(P) ≤ w(P´) aus Invariante 2.
- Ansonsten sei y erster Knoten der nicht in W U {x} ist.
- y hat Markierung ρ(y) ≥ ρ(x) wegen Definition des Algorithmus.
- Kanten sind nicht-negativ:

$$w(P) = \rho(x) \leq \rho(y) \leq w(P´)$$

- Invarianten bleiben erfüllt … ☺

Laufzeit

Initialisierung: O(n)

while-Schleife

Speicherung der Markierungen in einem Array.
O(n) mal wird ein Element mit kleinster Markierung gesucht.
Jede solche Suche kostet O(n) Vergleichsoperationen.
Ändern der Markierungen der Nachbarn kostet O(n) Speicherzugriffe.

Insgesamt: $O(n^2)$ Operationen.

Ausblick

Verwendung von Fibonacci Heaps Laufzeit $O(n \log n + m)$ möglich.