Algorithmen und Komplexität
Vorlesung 17

Wolfgang Globke

5. Juni 2019

Problem des Handelsreisenden und Hamilton-Zyklen

Das Problem des Handelsreisenden

Beim Problem des Handelsreisenden, kurz TSP (vom engl. Travelling Salesman Problem), stellt sich die Frage, wie ein Handelsreisender möglichst effizient n Städte bereisen kann.

Beim Problem des Handelsreisenden, kurz TSP (vom engl. Travelling Salesman Problem), stellt sich die Frage, wie ein Handelsreisender möglichst effizient n Städte bereisen kann. Genauer:

- Jede Stadt soll genau einmal besucht werden.
- Am Ende soll der Handelsreisende wieder am Ausgangspunkt ankommen.
- Die Kosten (bzw. zurückgelegten Strecken) sollen minimal sein.

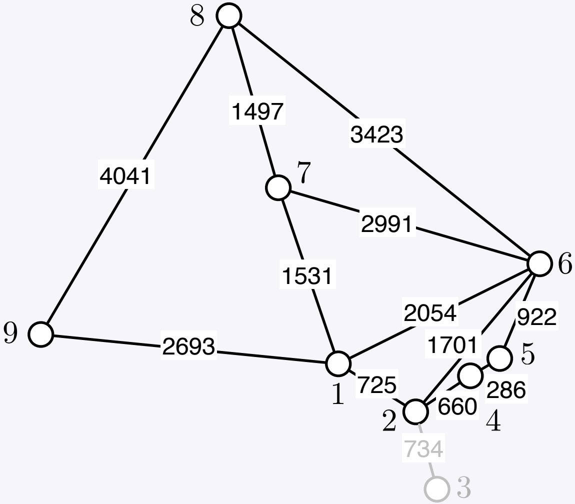
Das Problem des Handelsreisenden

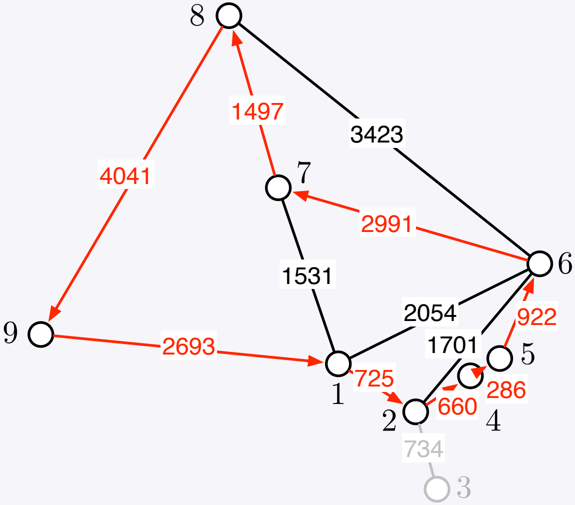
Beim Problem des Handelsreisenden, kurz TSP (vom engl. Travelling Salesman Problem), stellt sich die Frage, wie ein Handelsreisender möglichst effizient n Städte bereisen kann. Genauer:

- Jede Stadt soll genau einmal besucht werden.
- Am Ende soll der Handelsreisende wieder am Ausgangspunkt ankommen.
- Die Kosten (bzw. zurückgelegten Strecken) sollen minimal sein.

Ausgedrückt als graphentheoretisches Problem:

- Die Knoten eines gewichteten Graphen $G = (V, E)$ stellen die Städte dar, und die Kanten stellen die Verbindungen (Straßen, Bahnlinien, Flugrouten $\dots$) zwischen den Städten dar.
- Wir suchen einen Zyklus Z in G, der jeden Knoten in V enthält (wir sprechen auch von einer Tour).
- Die Tour Z soll minimales Gewicht $\mathrm{w}(Z)$ unter allen Touren haben.


8
1497
3423
4041
7
2991
1531
6
2054
922
9
2693
1701
5
725
286
1
660
2
4
734
3


8
1497
3423
4041
7
2991
1531
6
2054
922
9
2693
1701
5
1
725
286
660
2
4
734
3

Für das Problem des Handelsreisenden sind keine effizienten (polynomialen) exakten Lösungsalgorithmen bekannt.

Betrachte etwa den folgenden Brute-Force-Algorithmus:

1. Es sei $V = \{v_1, \ldots, v_n\}$.
2. Für jede Permutation $i_1, \ldots, i_n$ von $1, \ldots, n$, teste ob $Z = (v_{i_1}, \ldots, v_{i_n}, v_{i_1})$ ein Zyklus ist und minimale Kosten hat.

Das Problem des Handelsreisenden

Für das Problem des Handelsreisenden sind keine effizienten (polynomialen) exakten Lösungsalgorithmen bekannt.

Betrachte etwa den folgenden Brute-Force-Algorithmus:

1. Es sei $V = \{v_1, \ldots, v_n\}$.
2. Für jede Permutation $i_1, \ldots, i_n$ von $1, \ldots, n$, teste ob $Z = (v_{i_1}, \ldots, v_{i_n}, v_{i_1})$ ein Zyklus ist und minimale Kosten hat.

Da es $n!$ Permutationen von $1, \ldots, n$ gibt, hat dieser Algorithmus Aufwand $\Omega(n!) = \Omega(2^n)$.

Für das Problem des Handelsreisenden sind keine effizienten (polynomialen) exakten Lösungsalgorithmen bekannt.

Betrachte etwa den folgenden Brute-Force-Algorithmus:

1. Es sei $V = \{v_1, \ldots, v_n\}$.
2. Für jede Permutation $i_1, \ldots, i_n$ von $1, \ldots, n$, teste ob $Z = (v_{i_1}, \ldots, v_{i_n}, v_{i_1})$ ein Zyklus ist und minimale Kosten hat.

Da es $n!$ Permutationen von $1, \ldots, n$ gibt, hat dieser Algorithmus Aufwand $\Omega(n!) = \Omega(2^n)$.

- Das ist vollkommen unakzeptabel!
- Andere bekannte Methoden zur Bestimmung einer exakten Lösung sind effizienter, aber immer noch mit exponentiellem Aufwand $O(a^n)$ mit $a > 2$.
- Es ist nicht bekannt, ob es bessere exakte Lösungen gibt.
- Man muss nach approximativen Lösungen suchen.

Es gibt Unmengen von approximativen Lösungsansätzen für das Handelsreisendenproblem.

Handelsreisen mit Dreiecksungleichung

Es gibt Unmengen von approximativen Lösungsansätzen für das Handelsreisendenproblem.

Ein einfacher und effizienter Ansatz ergibt sich, wenn wir annehmen, dass die Knoten in V Punkte in der Ebene sind, und dass die Gewichte die euklidischen Abstände $d(u,v)$ („Luftlinie”) von v nach u ist,

$$\mathrm{w}(u,v) = d(u,v)$$

für alle Kanten $(u,v) \in E$.

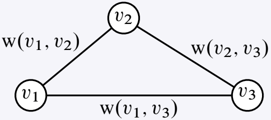
Es gibt Unmengen von approximativen Lösungsansätzen für das Handelsreisendenproblem.

Ein einfacher und effizienter Ansatz ergibt sich, wenn wir annehmen, dass die Knoten in V Punkte in der Ebene sind, und dass die Gewichte die euklidischen Abstände $d(u,v)$ („Luftlinie”) von v nach u ist,

$$\mathrm{w}(u,v) = d(u,v)$$

für alle Kanten $(u,v) \in E$. Dann erfüllen die Gewichte die Dreiecksungleichung,

$$\mathrm{w}(v_1,v_3) \leq \mathrm{w}(v_1,v_2) + \mathrm{w}(v_2,v_3).$$



Handelsreisen mit Dreiecksungleichung

Wir nehmen nun an, dass die Knoten des Graphen $G = (V, E)$ Punkte in der Ebene sind, und die Gewichte der Kanten durch den euklidischen Abstand $d(u, v)$ gegeben sind.

Handelsreisen mit Dreiecksungleichung

Wir nehmen nun an, dass die Knoten des Graphen $G = (V, E)$ Punkte in der Ebene sind, und die Gewichte der Kanten durch den euklidischen Abstand $d(u, v)$ gegeben sind.

Falls eine minimale Tour $Z = (v_0, v_1, \ldots, v_n, v_0)$ existiert, gilt:

- Durch Weglassen der Kante $e = (v_0, v_n)$ von z erhalten wir einen minimalen Spannbaum $T_{\min}$ von G.

Handelsreisen mit Dreiecksungleichung

Wir nehmen nun an, dass die Knoten des Graphen $G = (V, E)$ Punkte in der Ebene sind, und die Gewichte der Kanten durch den euklidischen Abstand $d(u, v)$ gegeben sind.

Falls eine minimale Tour $Z = (v_0, v_1, \ldots, v_n, v_0)$ existiert, gilt:

- Durch Weglassen der Kante $e = (v_0, v_n)$ von z erhalten wir einen minimalen Spannbaum $T_{\min}$ von G. Also: $\mathrm{w}(T_{\min}) \leq \mathrm{w}(Z)$.

Handelsreisen mit Dreiecksungleichung

Wir nehmen nun an, dass die Knoten des Graphen $G = (V, E)$ Punkte in der Ebene sind, und die Gewichte der Kanten durch den euklidischen Abstand $d(u, v)$ gegeben sind.

Falls eine minimale Tour $Z = (v_0, v_1, \dots, v_n, v_0)$ existiert, gilt:

- Durch Weglassen der Kante $e = (v_0, v_n)$ von z erhalten wir einen minimalen Spannbaum T_{min} von G. Also: $\mathrm{w}(T_{\text{min}}) \leq \mathrm{w}(Z)$.
- Umgekehrt erhalten wir offensichtlich die Tour z aus T_{min}, indem wir $e = (v_0, v_n)$ wieder zu T_{min} hinzufügen, weshalb $\mathrm{w}(Z) = \mathrm{w}(T_{\text{min}}) + \mathrm{w}(e)$.

Handelsreisen mit Dreiecksungleichung

Wir nehmen nun an, dass die Knoten des Graphen $G = (V, E)$ Punkte in der Ebene sind, und die Gewichte der Kanten durch den euklidischen Abstand $d(u, v)$ gegeben sind.

Falls eine minimale Tour $Z = (v_0, v_1, \ldots, v_n, v_0)$ existiert, gilt:

- Durch Weglassen der Kante $e = (v_0, v_n)$ von z erhalten wir einen minimalen Spannbaum $T_{\min}$ von G. Also: $\mathrm{w}(T_{\min}) \leq \mathrm{w}(Z)$.
- Umgekehrt erhalten wir offensichtlich die Tour z aus $T_{\min}$, indem wir $e = (v_0, v_n)$ wieder zu $T_{\min}$ hinzufügen, weshalb $\mathrm{w}(Z) = \mathrm{w}(T_{\min}) + \mathrm{w}(e)$.
- Wegen der Dreiecksungleichung gilt aber

$$\begin{aligned}
\mathrm{w}(e) = d(v_0, v_n) &\leq d(v_0, v_1) + d(v_1, v_n) \\
&\leq d(v_0, v_1) + d(v_1, v_2) + d(v_2, v_n) \\
&\vdots \\
&\leq d(v_0, v_1) + \ldots + d(v_{n-1}, v_n) \\
&= \mathrm{w}(v_0, v_1) + \ldots + \mathrm{w}(v_{n-1}, v_n) = \mathrm{w}(T_{\min}).
\end{aligned}$$

Handelsreisen mit Dreiecksungleichung

Wir nehmen nun an, dass die Knoten des Graphen $G = (V, E)$ Punkte in der Ebene sind, und die Gewichte der Kanten durch den euklidischen Abstand $d(u, v)$ gegeben sind.

Falls eine minimale Tour $Z = (v_0, v_1, \ldots, v_n, v_0)$ existiert, gilt:

- Durch Weglassen der Kante $e = (v_0, v_n)$ von z erhalten wir einen minimalen Spannbaum $T_{\min}$ von G. Also: $\mathrm{w}(T_{\min}) \leq \mathrm{w}(Z)$.
- Umgekehrt erhalten wir offensichtlich die Tour z aus $T_{\min}$, indem wir $e = (v_0, v_n)$ wieder zu $T_{\min}$ hinzufügen, weshalb $\mathrm{w}(Z) = \mathrm{w}(T_{\min}) + \mathrm{w}(e)$.
- Wegen der Dreiecksungleichung gilt aber

$$\begin{aligned}
\mathrm{w}(e) = d(v_0, v_n) &\leq d(v_0, v_1) + d(v_1, v_n) \\
&\leq d(v_0, v_1) + d(v_1, v_2) + d(v_2, v_n) \\
&\vdots \\
&\leq d(v_0, v_1) + \ldots + d(v_{n-1}, v_n) \\
&= \mathrm{w}(v_0, v_1) + \ldots + \mathrm{w}(v_{n-1}, v_n) = \mathrm{w}(T_{\min}).
\end{aligned}$$

Satz

Sind die Kantengewichte durch den euklidischen Abstand gegeben, so gilt $\mathrm{w}(T_{\min}) \leq \mathrm{w}(Z) \leq 2\mathrm{w}(T_{\min})$.

Handelsreisen mit Dreiecksungleichung

Der vorherige Satz inspiriert den folgenden Approximationsalgorithmus für vollständige Graphen $G = (V, E)$ mit euklidischen Abständen als Gewichte:

Der vorherige Satz inspiriert den folgenden Approximationsalgorithmus für vollständige Graphen $G = (V, E)$ mit euklidischen Abständen als Gewichte:

1. Bestimme einen minimalen Spannbaum T von G.
   Dies kann mit Kruskals Algorithmus oder dem Jarnik-Prim-Algorithmus erfolgen.

Handelsreisen mit Dreiecksungleichung

Der vorherige Satz inspiriert den folgenden Approximationsalgorithmus für vollständige Graphen $G = (V, E)$ mit euklidischen Abständen als Gewichte:

1. Bestimme einen minimalen Spannbaum T von G.
   Dies kann mit Kruskals Algorithmus oder dem Jarnik-Prim-Algorithmus erfolgen.
2. Führe eine Tiefensuche in T vom Knoten v_0 aus durch, und liste dabei die Knoten in der Reihenfolge auf, in der sie abgesucht werden.

Handelsreisen mit Dreiecksungleichung

Der vorherige Satz inspiriert den folgenden Approximationsalgorithmus für vollständige Graphen $G = (V, E)$ mit euklidischen Abständen als Gewichte:

1. Bestimme einen minimalen Spannbaum T von G.
   Dies kann mit Kruskals Algorithmus oder dem Jarnik-Prim-Algorithmus erfolgen.
2. Führe eine Tiefensuche in T vom Knoten v_0 aus durch, und liste dabei die Knoten in der Reihenfolge auf, in der sie abgesucht werden.
3. Streiche jede Wiederholung eines Knotens in der Liste aus Schritt 2.
   Hänge den Startknoten v_0 am Ende der Liste an.

Der vorherige Satz inspiriert den folgenden Approximationsalgorithmus für vollständige Graphen $G = (V, E)$ mit euklidischen Abständen als Gewichte:

1. Bestimme einen minimalen Spannbaum T von G.
   Dies kann mit Kruskals Algorithmus oder dem Jarnik-Prim-Algorithmus erfolgen.
2. Führe eine Tiefensuche in T vom Knoten v_0 aus durch, und liste dabei die Knoten in der Reihenfolge auf, in der sie abgesucht werden.
3. Streiche jede Wiederholung eines Knotens in der Liste aus Schritt 2.
   Hänge den Startknoten v_0 am Ende der Liste an.

Satz

Sind die Kantengewichte durch den euklidischen Abstand gegeben, so liefert der obige Algorithmus mit polynomialem Aufwand für vollständige Graphen einen Zyklus $\tilde{Z}$ in G, dessen Gewicht $\mathrm{w}(\tilde{Z}) \leq 2\mathrm{w}(Z)$ erfüllt, wobei z eine minimale Tour ist.

Der vorherige Satz inspiriert den folgenden Approximationsalgorithmus für vollständige Graphen $G = (V, E)$ mit euklidischen Abständen als Gewichte:

1. Bestimme einen minimalen Spannbaum T von G.
   Dies kann mit Kruskals Algorithmus oder dem Jarnik-Prim-Algorithmus erfolgen.
2. Führe eine Tiefensuche in T vom Knoten v_0 aus durch, und liste dabei die Knoten in der Reihenfolge auf, in der sie abgesucht werden.
3. Streiche jede Wiederholung eines Knotens in der Liste aus Schritt 2.
   Hänge den Startknoten v_0 am Ende der Liste an.

Satz

Sind die Kantengewichte durch den euklidischen Abstand gegeben, so liefert der obige Algorithmus mit polynomialem Aufwand für vollständige Graphen einen Zyklus $\tilde{Z}$ in G, dessen Gewicht $\mathrm{w}(\tilde{Z}) \leq 2\mathrm{w}(Z)$ erfüllt, wobei z eine minimale Tour ist.

Beweisskizze:

- Das Bestimmen eines minimalen Spannbaums kann in polynomialer Zeit erfolgen. Tiefensuche in einem Baum läuft jeden Knoten höchstens zweimal ab, erfolgt also in $\mathrm{O}(|V|)$. Schritt 3 erfolgt offenbar in linearer Zeit, also ist der Gesamtaufwand des Algorithmus polynomial.

Handelsreisen mit Dreiecksungleichung

Der vorherige Satz inspiriert den folgenden Approximationsalgorithmus für vollständige Graphen $G = (V, E)$ mit euklidischen Abständen als Gewichte:

1. Bestimme einen minimalen Spannbaum T von G.
   Dies kann mit Kruskals Algorithmus oder dem Jarnik-Prim-Algorithmus erfolgen.
2. Führe eine Tiefensuche in T vom Knoten v_0 aus durch, und liste dabei die Knoten in der Reihenfolge auf, in der sie abgesucht werden.
3. Streiche jede Wiederholung eines Knotens in der Liste aus Schritt 2.
   Hänge den Startknoten v_0 am Ende der Liste an.

Satz

Sind die Kantengewichte durch den euklidischen Abstand gegeben, so liefert der obige Algorithmus mit polynomialem Aufwand für vollständige Graphen einen Zyklus $\tilde{Z}$ in G, dessen Gewicht $\mathrm{w}(\tilde{Z}) \leq 2\mathrm{w}(Z)$ erfüllt, wobei z eine minimale Tour ist.

Beweisskizze:

- Das Bestimmen eines minimalen Spannbaums kann in polynomialer Zeit erfolgen. Tiefensuche in einem Baum läuft jeden Knoten höchstens zweimal ab, erfolgt also in $\mathrm{O}(|V|)$. Schritt 3 erfolgt offenbar in linearer Zeit, also ist der Gesamtaufwand des Algorithmus polynomial.
- Der Weg q, der bei der Tiefensuche abgelaufen wird, erfüllt $\mathrm{w}(q) \leq 2\mathrm{w}(T)$. Wegen der Dreiecksungleichung gilt $\mathrm{w}(\tilde{Z}) \leq \mathrm{w}(q)$ und somit nach dem vorherigen Satz $\mathrm{w}(\tilde{Z}) \leq 2\mathrm{w}(T) \leq 2\mathrm{w}(Z)$. □

Handelsreisen ohne Dreiecksungleichung

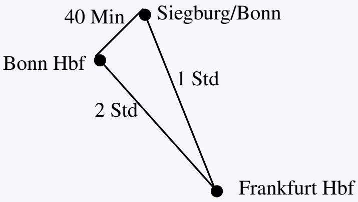
Die Annahme, dass die Gewichte im Graphen euklidischen Abständen entsprechen, ist sehr speziell.

Sie ist bereits nicht erfüllt, wenn wir die Reisezeiten entlang Bahnstrecken als Kantengewichte wählen:

Handelsreisen ohne Dreiecksungleichung

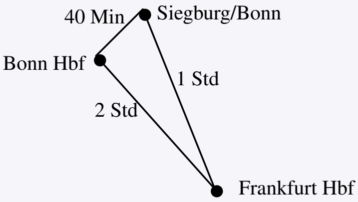
Die Annahme, dass die Gewichte im Graphen euklidischen Abständen entsprechen, ist sehr speziell.

Sie ist bereits nicht erfüllt, wenn wir die Reisezeiten entlang Bahnstrecken als Kantengewichte wählen:



Handelsreisen ohne Dreiecksungleichung

Die Annahme, dass die Gewichte im Graphen euklidischen Abständen entsprechen, ist sehr speziell.
Sie ist bereits nicht erfüllt, wenn wir die Reisezeiten entlang Bahnstrecken als Kantengewichte wählen:



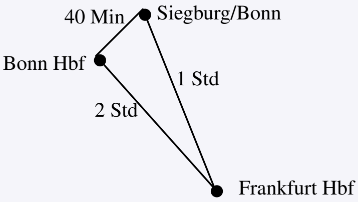
Leider können wir wohl für das allgemeine Handelsreisendenproblem keine effizienten Approximationen finden:

Satz

Es ist „vermutlich" nicht möglich, einen Algorithmus zu finden, der in polynomialer Zeit eine Näherungslösung für das allgemeine Handelsreisendenproblem findet, deren Gewicht nur um einen vorgeschriebene Faktor von dem einer exakten Lösung abweicht.

Handelsreisen ohne Dreiecksungleichung

Die Annahme, dass die Gewichte im Graphen euklidischen Abständen entsprechen, ist sehr speziell.
Sie ist bereits nicht erfüllt, wenn wir die Reisezeiten entlang Bahnstrecken als Kantengewichte wählen:



Leider können wir wohl für das allgemeine Handelsreisendenproblem keine effizienten Approximationen finden:

Satz

Es ist „vermutlich" nicht möglich, einen Algorithmus zu finden, der in polynomialer Zeit eine Näherungslösung für das allgemeine Handelsreisendenproblem findet, deren Gewicht nur um einen vorgeschriebene Faktor von dem einer exakten Lösung abweicht.

(„vermutlich" ?! $\leadsto$ $\mathbf{P} \neq \mathbf{NP}$)

Erinnerung:
Es bezeichne $\mathcal{A}^*$ die Menge aller endlichen Zeichenketten über einem Eingabealphabet $\mathcal{A}$. Ein Entscheidungsproblem besteht darin, für eine Teilmenge $X \subseteq \mathcal{A}^*$ und ein $a \in \mathcal{A}^*$ zu bestimmen, ob $a \in X$ liegt (siehe Vorlesung 4).

Erinnerung:
Es bezeichne $\mathcal{A}^*$ die Menge aller endlichen Zeichenketten über einem Eingabealphabet $\mathcal{A}$. Ein Entscheidungsproblem besteht darin, für eine Teilmenge $X \subseteq \mathcal{A}^*$ und ein $a \in \mathcal{A}^*$ zu bestimmen, ob $a \in X$ liegt (siehe Vorlesung 4).

Eine Variante des Handelsreisendenproblems kann als Entscheidungsproblem formuliert werden:

- Als „Alphabet" nehmen wir die Kanten im Graphen G.
- Die „Zeichenketten" sind dann Folgen von Kanten, insbesondere enthalten sie die Wege und Pfade in G.

Handelsreisen als Entscheidungsproblem

Erinnerung:
Es bezeichne $\mathcal{A}^*$ die Menge aller endlichen Zeichenketten über einem Eingabealphabet $\mathcal{A}$. Ein Entscheidungsproblem besteht darin, für eine Teilmenge $X \subseteq \mathcal{A}^*$ und ein $a \in \mathcal{A}^*$ zu bestimmen, ob $a \in X$ liegt (siehe Vorlesung 4).

Eine Variante des Handelsreisendenproblems kann als Entscheidungsproblem formuliert werden:

- Als „Alphabet" nehmen wir die Kanten im Graphen G.
- Die „Zeichenketten" sind dann Folgen von Kanten, insbesondere enthalten sie die Wege und Pfade in G.
- Das Entscheidungsproblem lautet:
  Gibt es eine Tour Z in G, deren Gewicht $\mathrm{w}(Z) \leq c$ ist für eine gegebene Konstante $c \geq 0$?
- Die Menge X besteht also aus allen Touren vom Gewicht $\leq c$.

Handelsreisen als Entscheidungsproblem

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Handelsreisen als Entscheidungsproblem

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Eine anderes Problem ist es jedoch, für einen gegebenen Lösungsvorschlag Z_0 zu entscheiden, ob Z_0 eine minimale Tour ist oder nicht.

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Eine anderes Problem ist es jedoch, für einen gegebenen Lösungsvorschlag Z_0 zu entscheiden, ob Z_0 eine minimale Tour ist oder nicht.

1. Zunächst prüft man, ob $Z_0 = (v_0, v_1, \ldots, v_n, v_0)$ eine Tour ist, also ob $(v_i, v_{i+1}) \in E$ für alle $i = 0, \ldots, n$, und ob $v_i \neq v_j$ für $i \neq j$.

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Eine anderes Problem ist es jedoch, für einen gegebenen Lösungsvorschlag Z_0 zu entscheiden, ob Z_0 eine minimale Tour ist oder nicht.

1. Zunächst prüft man, ob $Z_0 = (v_0, v_1, \ldots, v_n, v_0)$ eine Tour ist, also ob $(v_i, v_{i+1}) \in E$ für alle $i = 0, \ldots, n$, und ob $v_i \neq v_j$ für $i \neq j$.
   Dies ist mit Aufwand $O(n^2)$ möglich.

Handelsreisen als Entscheidungsproblem

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Eine anderes Problem ist es jedoch, für einen gegebenen Lösungsvorschlag Z_0 zu entscheiden, ob Z_0 eine minimale Tour ist oder nicht.

1. Zunächst prüft man, ob $Z_0 = (v_0, v_1, \ldots, v_n, v_0)$ eine Tour ist, also ob $(v_i, v_{i+1}) \in E$ für alle $i = 0, \ldots, n$, und ob $v_i \neq v_j$ für $i \neq j$.
   Dies ist mit Aufwand $\mathrm{O}(n^2)$ möglich.
2. Dann prüft man, ob das Gewicht von Z_0 zulässig ist, also ob $\mathrm{w}(Z_0) = \sum_{i=0}^{n} \mathrm{w}(v_i, v_{i+1}) \leq c$ gilt.

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Eine anderes Problem ist es jedoch, für einen gegebenen Lösungsvorschlag Z_0 zu entscheiden, ob Z_0 eine minimale Tour ist oder nicht.

1. Zunächst prüft man, ob $Z_0 = (v_0, v_1, \ldots, v_n, v_0)$ eine Tour ist, also ob $(v_i, v_{i+1}) \in E$ für alle $i = 0, \ldots, n$, und ob $v_i \neq v_j$ für $i \neq j$.
   Dies ist mit Aufwand $\mathrm{O}(n^2)$ möglich.
2. Dann prüft man, ob das Gewicht von Z_0 zulässig ist, also ob $\mathrm{w}(Z_0) = \sum_{i=0}^{n} \mathrm{w}(v_i, v_{i+1}) \leq c$ gilt.
   Dies ist mit Aufwand $\mathrm{O}(n)$ möglich.

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Eine anderes Problem ist es jedoch, für einen gegebenen Lösungsvorschlag Z_0 zu entscheiden, ob Z_0 eine minimale Tour ist oder nicht.

1. Zunächst prüft man, ob $Z_0 = (v_0, v_1, \dots, v_n, v_0)$ eine Tour ist, also ob $(v_i, v_{i+1}) \in E$ für alle $i = 0, \dots, n$, und ob $v_i \neq v_j$ für $i \neq j$.
   Dies ist mit Aufwand $\mathrm{O}(n^2)$ möglich.
2. Dann prüft man, ob das Gewicht von Z_0 zulässig ist, also ob $\mathrm{w}(Z_0) = \sum_{i=0}^{n} \mathrm{w}(v_i, v_{i+1}) \leq c$ gilt.
   Dies ist mit Aufwand $\mathrm{O}(n)$ möglich.
3. Sind die Tests 1 und 2 erfolgreich, so ist Z_0 eine Lösung.
   Der Gesamtaufwand hierbei ist in $\mathrm{O}(n^2)$.

Handelsreisen als Entscheidungsproblem

Für die Variante als Entscheidungsproblem sind auch keine effizienten exakten Lösungen bekannt.

Eine anderes Problem ist es jedoch, für einen gegebenen Lösungsvorschlag Z_0 zu entscheiden, ob Z_0 eine minimale Tour ist oder nicht.

1. Zunächst prüft man, ob $Z_0 = (v_0, v_1, \ldots, v_n, v_0)$ eine Tour ist, also ob $(v_i, v_{i+1}) \in E$ für alle $i = 0, \ldots, n$, und ob $v_i \neq v_j$ für $i \neq j$.
   Dies ist mit Aufwand $\mathrm{O}(n^2)$ möglich.
2. Dann prüft man, ob das Gewicht von Z_0 zulässig ist, also ob $\mathrm{w}(Z_0) = \sum_{i=0}^{n} \mathrm{w}(v_i, v_{i+1}) \leq c$ gilt.
   Dies ist mit Aufwand $\mathrm{O}(n)$ möglich.
3. Sind die Tests 1 und 2 erfolgreich, so ist Z_0 eine Lösung.
   Der Gesamtaufwand hierbei ist in $\mathrm{O}(n^2)$.

Fazit

Auch wenn keine effizienten Lösungsmethoden für das Handelsreisendenproblem bekannt sind, so ist es doch effizient möglich, zu überprüfen, ob ein gegebener Weg eine Lösung ist.

Eng verwandt mit dem Problem des Handelsreisenden ist das folgende Problem HC (vom engl. Hamiltonian Cycle Problem):

Existiert in einem Graphen $G = (V, E)$ ein Zyklus Z, der alle Knoten des Graphen enthält?

Ein solcher Zyklus Z wird auch Hamilton-Zyklus genannt.

Eng verwandt mit dem Problem des Handelsreisenden ist das folgende Problem HC (vom engl. Hamiltonian Cycle Problem):

Existiert in einem Graphen $G = (V, E)$ ein Zyklus Z, der alle Knoten des Graphen enthält?

Ein solcher Zyklus Z wird auch Hamilton-Zyklus genannt.

Ein Hamilton-Zyklus ist dabei nichts anderes als eine Tour, wobei wir etwaige Gewichte im Graphen ignorieren.

Reduktion auf das Handelsreisendenproblem

Das Problem HC der Hamilton-Zyklen auf einem Graphen $G = (V, E)$ kann durch Reduktion auf das Handelsreisendenproblem TSP gelöst werden:

Reduktion auf das Handelsreisendenproblem

Das Problem HC der Hamilton-Zyklen auf einem Graphen $G = (V, E)$ kann durch Reduktion auf das Handelsreisendenproblem TSP gelöst werden:

1. Angenommen, wir können durch einen Algorithmus A_{TSP} das Handelsreisendenproblem lösen, d.h.

   $$Z = A_{\mathrm{TSP}}(G')$$

   liefert eine minimale Tour in einem beliebigen gewichteten Graphen G', falls eine solche existiert.

Reduktion auf das Handelsreisendenproblem

Das Problem HC der Hamilton-Zyklen auf einem Graphen $G = (V, E)$ kann durch Reduktion auf das Handelsreisendenproblem TSP gelöst werden:

1. Angenommen, wir können durch einen Algorithmus A_{TSP} das Handelsreisendenproblem lösen, d.h.

$$Z = A_{\text{TSP}}(G')$$

liefert eine minimale Tour in einem beliebigen gewichteten Graphen G', falls eine solche existiert.

2. Es sei $G^* = (V, E^*)$ der zu G gehörige vollständige Graph, d.h. E^* enthält für jedes Paar verschiedener Knoten in G eine Kante.

Reduktion auf das Handelsreisendenproblem

Das Problem HC der Hamilton-Zyklen auf einem Graphen $G = (V, E)$ kann durch Reduktion auf das Handelsreisendenproblem TSP gelöst werden:

1. Angenommen, wir können durch einen Algorithmus A_{TSP} das Handelsreisendenproblem lösen, d.h.

   $$Z = A_{\mathrm{TSP}}(G')$$

   liefert eine minimale Tour in einem beliebigen gewichteten Graphen G', falls eine solche existiert.
2. Es sei $G^* = (V, E^*)$ der zu G gehörige vollständige Graph, d.h. E^* enthält für jedes Paar verschiedener Knoten in G eine Kante.
3. Definiere Gewichte $\mathrm{w}(e)$ für $e \in E^*$ durch

   $$\mathrm{w}(e) = \begin{cases} 0 & \text{falls } e \in E, \\ 1 & \text{falls } e \notin E \end{cases} .$$

Reduktion auf das Handelsreisendenproblem

Das Problem HC der Hamilton-Zyklen auf einem Graphen $G = (V, E)$ kann durch Reduktion auf das Handelsreisendenproblem TSP gelöst werden:

1. Angenommen, wir können durch einen Algorithmus A_{TSP} das Handelsreisendenproblem lösen, d.h.

$$Z = A_{\mathrm{TSP}}(G')$$

   liefert eine minimale Tour in einem beliebigen gewichteten Graphen G', falls eine solche existiert.
2. Es sei $G^* = (V, E^*)$ der zu G gehörige vollständige Graph, d.h. E^* enthält für jedes Paar verschiedener Knoten in G eine Kante.
3. Definiere Gewichte $\mathrm{w}(e)$ für $e \in E^*$ durch

$$\mathrm{w}(e) = \begin{cases} 0 & \text{falls } e \in E, \\ 1 & \text{falls } e \notin E \end{cases} .$$

4. Dann liefert $A_{\mathrm{TSP}}(G^*)$ einen Hamilton-Zyklus für G, sofern einer existiert.

Reduktion auf das Handelsreisendenproblem

Das Problem HC der Hamilton-Zyklen auf einem Graphen $G = (V, E)$ kann durch Reduktion auf das Handelsreisendenproblem TSP gelöst werden:

1. Angenommen, wir können durch einen Algorithmus A_{TSP} das Handelsreisendenproblem lösen, d.h.

$$Z = A_{\mathrm{TSP}}(G')$$

   liefert eine minimale Tour in einem beliebigen gewichteten Graphen G', falls eine solche existiert.

2. Es sei $G^* = (V, E^*)$ der zu G gehörige vollständige Graph, d.h. E^* enthält für jedes Paar verschiedener Knoten in G eine Kante.

3. Definiere Gewichte $\mathrm{w}(e)$ für $e \in E^*$ durch

$$\mathrm{w}(e) = \begin{cases} 0 & \text{falls } e \in E, \\ 1 & \text{falls } e \notin E \end{cases} .$$

4. Dann liefert $A_{\mathrm{TSP}}(G^*)$ einen Hamilton-Zyklus für G, sofern einer existiert.

Wir haben gesehen, dass wir HC lösen können, wenn wir TSP lösen können. Wir sagen, dass wir HC auf TSP reduziert haben.

Allgemeiner seien zwei Probleme $\mathcal{P}$ und $\mathcal{Q}$ gegeben, und wir nehmen an, dass jeder Algorithmus zur Lösung von $\mathcal{P}$ mindestens den Aufwand $T_{\mathcal{P}}$ hat.

Allgemeiner seien zwei Probleme $\mathcal{P}$ und $\mathcal{Q}$ gegeben, und wir nehmen an, dass jeder Algorithmus zur Lösung von $\mathcal{P}$ mindestens den Aufwand $T_{\mathcal{P}}$ hat.

- Existiert ein Algorithmus B, der $\mathcal{P}$ in eine Instanz des Problems $\mathcal{Q}$ umwandelt, so kann jeder Algorithmus A, der $\mathcal{Q}$ löst, auch $\mathcal{P}$ lösen (durch $A(B(\mathcal{P}))$).

Allgemeiner seien zwei Probleme $\mathcal{P}$ und $\mathcal{Q}$ gegeben, und wir nehmen an, dass jeder Algorithmus zur Lösung von $\mathcal{P}$ mindestens den Aufwand $T_{\mathcal{P}}$ hat.

- Existiert ein Algorithmus B, der $\mathcal{P}$ in eine Instanz des Problems $\mathcal{Q}$ umwandelt, so kann jeder Algorithmus A, der $\mathcal{Q}$ löst, auch $\mathcal{P}$ lösen (durch $A(B(\mathcal{P}))$).
- Wir sagen dann, dass $\mathcal{P}$ auf $\mathcal{Q}$ reduzierbar ist, und schreiben $\mathcal{P} \preceq \mathcal{Q}$.

Allgemeiner seien zwei Probleme $\mathcal{P}$ und $\mathcal{Q}$ gegeben, und wir nehmen an, dass jeder Algorithmus zur Lösung von $\mathcal{P}$ mindestens den Aufwand $T_{\mathcal{P}}$ hat.

- Existiert ein Algorithmus B, der $\mathcal{P}$ in eine Instanz des Problems $\mathcal{Q}$ umwandelt, so kann jeder Algorithmus A, der $\mathcal{Q}$ löst, auch $\mathcal{P}$ lösen (durch $A(B(\mathcal{P}))$).
- Wir sagen dann, dass $\mathcal{P}$ auf $\mathcal{Q}$ reduzierbar ist, und schreiben $\mathcal{P} \preceq \mathcal{Q}$.
- Hinter diesen Bezeichnungen steckt die Intuition, dass $\mathcal{Q}$ mindestens so schwer lösbar ist wie $\mathcal{P}$, also dass jeder Algorithmus A zur Lösung von $\mathcal{Q}$ auch mindestens Aufwand $T_{\mathcal{P}}$ hat.

Allgemeiner seien zwei Probleme $\mathcal{P}$ und $\mathcal{Q}$ gegeben, und wir nehmen an, dass jeder Algorithmus zur Lösung von $\mathcal{P}$ mindestens den Aufwand $T_{\mathcal{P}}$ hat.

- Existiert ein Algorithmus B, der $\mathcal{P}$ in eine Instanz des Problems $\mathcal{Q}$ umwandelt, so kann jeder Algorithmus A, der $\mathcal{Q}$ löst, auch $\mathcal{P}$ lösen (durch $A(B(\mathcal{P}))$).
- Wir sagen dann, dass $\mathcal{P}$ auf $\mathcal{Q}$ reduzierbar ist, und schreiben $\mathcal{P} \preceq \mathcal{Q}$.
- Hinter diesen Bezeichnungen steckt die Intuition, dass $\mathcal{Q}$ mindestens so schwer lösbar ist wie $\mathcal{P}$, also dass jeder Algorithmus A zur Lösung von $\mathcal{Q}$ auch mindestens Aufwand $T_{\mathcal{P}}$ hat.
- Es gilt allerdings $T(A) = T_{\mathcal{P}} + T(B)$.

Allgemeiner seien zwei Probleme $\mathcal{P}$ und $\mathcal{Q}$ gegeben, und wir nehmen an, dass jeder Algorithmus zur Lösung von $\mathcal{P}$ mindestens den Aufwand $T_{\mathcal{P}}$ hat.

- Existiert ein Algorithmus B, der $\mathcal{P}$ in eine Instanz des Problems $\mathcal{Q}$ umwandelt, so kann jeder Algorithmus A, der $\mathcal{Q}$ löst, auch $\mathcal{P}$ lösen (durch $A(B(\mathcal{P}))$).
- Wir sagen dann, dass $\mathcal{P}$ auf $\mathcal{Q}$ reduzierbar ist, und schreiben $\mathcal{P} \preceq \mathcal{Q}$.
- Hinter diesen Bezeichnungen steckt die Intuition, dass $\mathcal{Q}$ mindestens so schwer lösbar ist wie $\mathcal{P}$, also dass jeder Algorithmus A zur Lösung von $\mathcal{Q}$ auch mindestens Aufwand $T_{\mathcal{P}}$ hat.
- Es gilt allerdings $T(A) = T_{\mathcal{P}} + T(B)$. Unsere Intuition ist also nur dann gerechtfertigt, wenn der Aufwand des Reduktionsalgorithmus B nicht (wesentlich) $T_{\mathcal{P}}$ übersteigt.
  (Andernfalls kann die Reduktion die Sache sogar verschlimmern!)

Allgemeiner seien zwei Probleme $\mathcal{P}$ und $\mathcal{Q}$ gegeben, und wir nehmen an, dass jeder Algorithmus zur Lösung von $\mathcal{P}$ mindestens den Aufwand $T_{\mathcal{P}}$ hat.

- Existiert ein Algorithmus B, der $\mathcal{P}$ in eine Instanz des Problems $\mathcal{Q}$ umwandelt, so kann jeder Algorithmus A, der $\mathcal{Q}$ löst, auch $\mathcal{P}$ lösen (durch $A(B(\mathcal{P}))$).
- Wir sagen dann, dass $\mathcal{P}$ auf $\mathcal{Q}$ reduzierbar ist, und schreiben $\mathcal{P} \preceq \mathcal{Q}$.
- Hinter diesen Bezeichnungen steckt die Intuition, dass $\mathcal{Q}$ mindestens so schwer lösbar ist wie $\mathcal{P}$, also dass jeder Algorithmus A zur Lösung von $\mathcal{Q}$ auch mindestens Aufwand $T_{\mathcal{P}}$ hat.
- Es gilt allerdings $T(A) = T_{\mathcal{P}} + T(B)$. Unsere Intuition ist also nur dann gerechtfertigt, wenn der Aufwand des Reduktionsalgorithmus B nicht (wesentlich) $T_{\mathcal{P}}$ übersteigt.
  (Andernfalls kann die Reduktion die Sache sogar verschlimmern!)
- Wir gehen davon aus, dass der Reduktionsaufwand $T(B)$ vernachlässigbar ist, wenn $T(B)$ polynomial ist. Dann schreiben wir $\mathcal{P} \preceq_{\mathrm{p}} \mathcal{Q}$.

Beispiel 1
Wir haben gesehen, dass das Problem HC auf TSP reduzierbar ist.

Beispiel 1

Wir haben gesehen, dass das Problem HC auf TSP reduzierbar ist.

- Der Aufwand für die Reduktion liegt dabei im Bilden des zu G gehörigen vollständigen Graphen G^* und im Festlegen der Gewichte auf G^*.
- Dies ist in $O(|V|^2)$ möglich.
- Also gilt HC $\preceq_p$ TSP. Somit ist TSP mindestens so schwer wie HC.

Beispiel 1

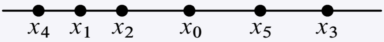
Wir haben gesehen, dass das Problem HC auf TSP reduzierbar ist.

- Der Aufwand für die Reduktion liegt dabei im Bilden des zu G gehörigen vollständigen Graphen G^* und im Festlegen der Gewichte auf G^*.
- Dies ist in $\mathrm{O}(|V|^2)$ möglich.
- Also gilt HC $\preceq_{\mathrm{p}}$ TSP. Somit ist TSP mindestens so schwer wie HC.
- Leider wissen wir aber nicht, wie effizient HC gelöst werden kann (Vermutung: nur mit exponentiellem Aufwand).

Beispiel 2

Betrachte das Problem, für n Punkte $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n$ in der Ebene die konvexe Hülle zu finden. Dieses Problem hat mindestens die Komplexität $\Omega(n \log(n))$.

Beweis: Reduziere Sortieren auf das Bilden der konvexen Hülle:

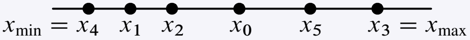


Wir wollen reelle Zahlen $x_0, \ldots, x_n$ sortieren

Beispiel 2

Betrachte das Problem, für n Punkte $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n$ in der Ebene die konvexe Hülle zu finden. Dieses Problem hat mindestens die Komplexität $\Omega(n \log(n))$.

Beweis: Reduziere Sortieren auf das Bilden der konvexen Hülle:

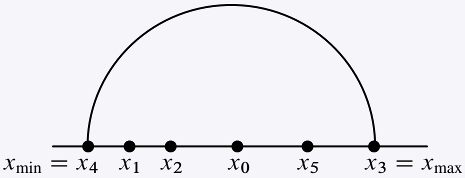


Bestimme $x_{\min}$ und $x_{\max}$ in $O(n)$

Beispiel 2

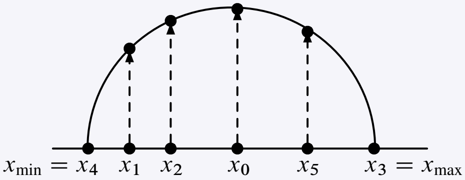
Betrachte das Problem, für n Punkte $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n$ in der Ebene die konvexe Hülle zu finden. Dieses Problem hat mindestens die Komplexität $\Omega(n \log(n))$.

Beweis: Reduziere Sortieren auf das Bilden der konvexen Hülle:



Betrachte Halbkreis mit Durchmesser $x_{\max} - x_{\min}$

Beispiel 2
Betrachte das Problem, für n Punkte $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n$ in der Ebene die konvexe Hülle zu finden. Dieses Problem hat mindestens die Komplexität $\Omega(n \log(n))$.
Beweis: Reduziere Sortieren auf das Bilden der konvexen Hülle:

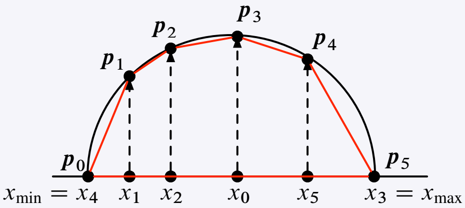


Projiziere alle Punke
auf den Halbkreis in $O(n)$

Beispiel 2

Betrachte das Problem, für n Punkte $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n$ in der Ebene die konvexe Hülle zu finden. Dieses Problem hat mindestens die Komplexität $\Omega(n \log(n))$.

Beweis: Reduziere Sortieren auf das Bilden der konvexen Hülle:

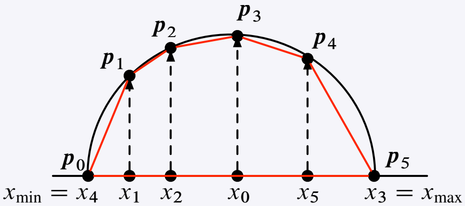


Bestimme konvexe Hülle $\rightsquigarrow$
x-Koordinaten ablesen in $\mathrm{O}(n)$
liefert Sortierung

Beispiel 2

Betrachte das Problem, für n Punkte $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n$ in der Ebene die konvexe Hülle zu finden. Dieses Problem hat mindestens die Komplexität $\Omega(n \log(n))$.

Beweis: Reduziere Sortieren auf das Bilden der konvexen Hülle:



Bestimme konvexe Hülle $\rightsquigarrow$
x-Koordinaten ablesen in $O(n)$
liefert Sortierung

Wenn das Bestimmen der konvexen Hülle schneller als $\Omega(n \log(n))$ mögliche wäre, könnten wir also auch schneller sortieren. Wir wissen aber, dass dies unmöglich ist! Es gilt Sortieren $\preceq_p$ Konvexe Hülle.

Komplexitätsklassen

Wir haben nun

- einerseits gesehen, dass es mutmaßlich sehr schwere Probleme gibt (also nicht effizient lösbare),
- und andererseits, dass man Informationen über die Schwierigkeit von Problemen erhalten kann, indem man sie aufeinander zurückführt.

Wir haben nun

- einerseits gesehen, dass es mutmaßlich sehr schwere Probleme gibt (also nicht effizient lösbare),
- und andererseits, dass man Informationen über die Schwierigkeit von Problemen erhalten kann, indem man sie aufeinander zurückführt.

Dies legt es nahe, Probleme entsprechend ihrer Schwierigkeit in Komplexitätsklassen einzuteilen.

Wir haben nun

- einerseits gesehen, dass es mutmaßlich sehr schwere Probleme gibt (also nicht effizient lösbare),
- und andererseits, dass man Informationen über die Schwierigkeit von Problemen erhalten kann, indem man sie aufeinander zurückführt.

Dies legt es nahe, Probleme entsprechend ihrer Schwierigkeit in Komplexitätsklassen einzuteilen.

Beispiel

Wir kennen bereits die Klasse **P** der Entscheidungsprobleme, die durch effiziente Algorithmen (d.h. in polynomialer Zeit) gelöst werden.

Wir haben nun

- einerseits gesehen, dass es mutmaßlich sehr schwere Probleme gibt (also nicht effizient lösbare),
- und andererseits, dass man Informationen über die Schwierigkeit von Problemen erhalten kann, indem man sie aufeinander zurückführt.

Dies legt es nahe, Probleme entsprechend ihrer Schwierigkeit in Komplexitätsklassen einzuteilen.

Beispiel

Wir kennen bereits die Klasse **P** der Entscheidungsprobleme, die durch effiziente Algorithmen (d.h. in polynomialer Zeit) gelöst werden.

Erinnerung: Jedes Problem, das die Berechnung einer Funktion $y = f(x)$ verlangt, kann mit einem Entscheidungsproblem $\mathcal{E}_f$ identifiziert werden.

- Wir können also insbesondere auch Probleme wie Sortieren, kürzeste Wege, minimale Spannbäume, Suchen, … mit Entscheidungsproblemen identifizieren.
- Entscheidungsprobleme stellen also eine für uns sehr interessante Klasse von Problemen dar.

Um Überlegungen über die Komplexität von Problemen anzustellen, brauchen wir ein geeignetes (abstraktes) Berechnungsmodell.

Turing-Maschinen

Um Überlegungen über die Komplexität von Problemen anzustellen, brauchen wir ein geeignetes (abstraktes) Berechnungsmodell.

Für theoretische Betrachtungen sind Turing-Maschinen sehr geeignet.

Um Überlegungen über die Komplexität von Problemen anzustellen, brauchen wir ein geeignetes (abstraktes) Berechnungsmodell.

Für theoretische Betrachtungen sind Turing-Maschinen sehr geeignet.

- Sehr einfache Struktur gut für logische Analyse.

Um Überlegungen über die Komplexität von Problemen anzustellen, brauchen wir ein geeignetes (abstraktes) Berechnungsmodell.

Für theoretische Betrachtungen sind Turing-Maschinen sehr geeignet.

- Sehr einfache Struktur gut für logische Analyse.
- Eines der ältesten Modelle.

Turing-Maschinen

Um Überlegungen über die Komplexität von Problemen anzustellen, brauchen wir ein geeignetes (abstraktes) Berechnungsmodell.

Für theoretische Betrachtungen sind Turing-Maschinen sehr geeignet.

- Sehr einfache Struktur gut für logische Analyse.
- Eines der ältesten Modelle.
- Kann mindestens so viel berechnen, wie alle anderen gängigen deterministischen Rechenmodelle.

Eine Turing-Maschine T besteht aus:

Eine Turing-Maschine T besteht aus:

- Einem (beidseitig unendlich langen) Band als Speicher. Das Band ist in Felder eingeteilt, die jeweils ein Zeichen enthalten können.

Eine Turing-Maschine T besteht aus:

- Einem (beidseitig unendlich langen) Band als Speicher. Das Band ist in Felder eingeteilt, die jeweils ein Zeichen enthalten können.
- Einem Bandalphabet $\mathcal{B}$ der zulässigen Zeichen auf dem Band. Das Bandalphabet selbst ist die Vereinigung $\mathcal{B} = \mathcal{A} \cup \{\textvisiblespace\}$ aus einem Eingabealphabet $\mathcal{A}$ und dem Leerzeichen „$\textvisiblespace$”.

Eine Turing-Maschine T besteht aus:

- Einem (beidseitig unendlich langen) Band als Speicher. Das Band ist in Felder eingeteilt, die jeweils ein Zeichen enthalten können.
- Einem Bandalphabet $\mathcal{B}$ der zulässigen Zeichen auf dem Band. Das Bandalphabet selbst ist die Vereinigung $\mathcal{B} = \mathcal{A} \cup \{\textvisiblespace\}$ aus einem Eingabealphabet $\mathcal{A}$ und dem Leerzeichen „␣”.
- Einem Lese-/Schreibkopf, der sich entlang des Bandes bewegen kann und die Einträge ändern kann.

Eine Turing-Maschine T besteht aus:

- Einem (beidseitig unendlich langen) Band als Speicher. Das Band ist in Felder eingeteilt, die jeweils ein Zeichen enthalten können.
- Einem Bandalphabet $\mathcal{B}$ der zulässigen Zeichen auf dem Band. Das Bandalphabet selbst ist die Vereinigung $\mathcal{B} = \mathcal{A} \cup \{\textvisiblespace\}$ aus einem Eingabealphabet $\mathcal{A}$ und dem Leerzeichen „␣”.
- Einem Lese-/Schreibkopf, der sich entlang des Bandes bewegen kann und die Einträge ändern kann.
- Mehrere Zustände Q, in denen T sich befinden kann, darunter ein Startzustand Q_0 und mindestens ein Endzustand Q_{end}.

Eine Turing-Maschine T besteht aus:

- Einem (beidseitig unendlich langen) Band als Speicher. Das Band ist in Felder eingeteilt, die jeweils ein Zeichen enthalten können.
- Einem Bandalphabet $\mathcal{B}$ der zulässigen Zeichen auf dem Band. Das Bandalphabet selbst ist die Vereinigung $\mathcal{B} = \mathcal{A} \cup \{\text{␣}\}$ aus einem Eingabealphabet $\mathcal{A}$ und dem Leerzeichen „␣”.
- Einem Lese-/Schreibkopf, der sich entlang des Bandes bewegen kann und die Einträge ändern kann.
- Mehrere Zustände Q, in denen T sich befinden kann, darunter ein Startzustand Q_0 und mindestens ein Endzustand Q_{end}.
- Einer Menge von Übergängen der Form $(Q, z) \mapsto (Q', z', \alpha)$, wobei $\alpha \in \{\rightarrow, \leftarrow, \downarrow\}$. Solch ein Übergang bedeutet:

Eine Turing-Maschine T besteht aus:

- Einem (beidseitig unendlich langen) Band als Speicher. Das Band ist in Felder eingeteilt, die jeweils ein Zeichen enthalten können.
- Einem Bandalphabet $\mathcal{B}$ der zulässigen Zeichen auf dem Band. Das Bandalphabet selbst ist die Vereinigung $\mathcal{B} = \mathcal{A} \cup \{\textvisiblespace\}$ aus einem Eingabealphabet $\mathcal{A}$ und dem Leerzeichen „␣".
- Einem Lese-/Schreibkopf, der sich entlang des Bandes bewegen kann und die Einträge ändern kann.
- Mehrere Zustände Q, in denen T sich befinden kann, darunter ein Startzustand Q_0 und mindestens ein Endzustand Q_{end}.
- Einer Menge von Übergängen der Form $(Q, z) \mapsto (Q', z', \alpha)$, wobei $\alpha \in \{\rightarrow, \leftarrow, \downarrow\}$. Solch ein Übergang bedeutet:
  Befindet sich T im Zustand Q und zeigt der Kopf auf das Zeichen z, so
  - geht die Maschine in den Zustand Q' über,
  - ersetzt das Zeichen z durch z',
  - bewegt danach den Kopf ein Feld in Richtung α entlang des Bandes (wobei $\downarrow$ bedeutet, dass der Kopf seine Position nicht ändert).

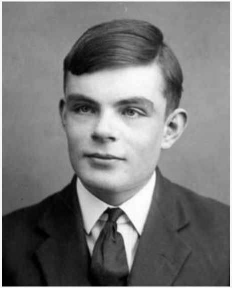
Eine Turing-Maschine T besteht aus:

- Einem (beidseitig unendlich langen) Band als Speicher. Das Band ist in Felder eingeteilt, die jeweils ein Zeichen enthalten können.
- Einem Bandalphabet $\mathcal{B}$ der zulässigen Zeichen auf dem Band. Das Bandalphabet selbst ist die Vereinigung $\mathcal{B} = \mathcal{A} \cup \{\llcorner\!\lrcorner\}$ aus einem Eingabealphabet $\mathcal{A}$ und dem Leerzeichen „$\llcorner\!\lrcorner$".
- Einem Lese-/Schreibkopf, der sich entlang des Bandes bewegen kann und die Einträge ändern kann.
- Mehrere Zustände Q, in denen T sich befinden kann, darunter ein Startzustand Q_0 und mindestens ein Endzustand Q_{end}.
- Einer Menge von Übergängen der Form $(Q, z) \mapsto (Q', z', \alpha)$, wobei $\alpha \in \{\rightarrow, \leftarrow, \downarrow\}$. Solch ein Übergang bedeutet:
  Befindet sich T im Zustand Q und zeigt der Kopf auf das Zeichen z, so
  - geht die Maschine in den Zustand Q' über,
  - ersetzt das Zeichen z durch z',
  - bewegt danach den Kopf ein Feld in Richtung α entlang des Bandes (wobei $\downarrow$ bedeutet, dass der Kopf seine Position nicht ändert).
- Die Maschine beginnt im Zustand Q_0, die Eingabe ist, was zu Beginn auf dem Band steht. Die Berechnung endet, wenn ein Endzustand erreicht wird.

Wenige Leute haben soviel Einfluss auf die Entwicklung der Grundlagen der Informatik gehabt wie Alan Turing (1912-1954).

Wenige Leute haben soviel Einfluss auf die Entwicklung der Grundlagen der Informatik gehabt wie Alan Turing (1912-1954).

- Die nach ihm benannten Rechenmodelle erdachte er, um David Hilberts Vermutung, dass man die Korrektheit mathematischer Sätze automatisch entscheiden könne, zu widerlegen.
- Erlangte Ruhm durch seine Dechiffrierarbeit im zweiten Weltkrieg. Seine Ideen und die von ihm entworfenen Rechner führten zu den ersten Entschlüsselungen der Enigma-Codes.
- Beschäftigte sich außerdem mit vielen weiteren Problemen der Logik und Mathematik, u.a. wie Leoparden zu ihren Flecken kommen.
- Tod durch Gift, nachdem er aufgrund seiner Homosexualität zur Hormonbehandlung gezwungen wurde und von sensiblen Forschungen ausgeschlossen wurde.



Beispiel 1:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\textvisiblespace, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\textvisiblespace$	Q_{end}	$\vert$	$\downarrow$

Beispiel 1:
Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\llcorner\!\lrcorner, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\llcorner\!\lrcorner$	Q_{end}	$\vert$	$\downarrow$

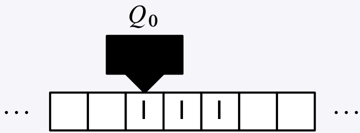
Interpretieren wir n-fache Wiederholung von $|$ als die Zahl n, so erhöht diese Maschine n auf $n+1$ (Kopf beginnt auf dem $|$ am weitesten links).

Beispiel 1:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\textvisiblespace, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\textvisiblespace$	Q_{end}	$\vert$	$\downarrow$

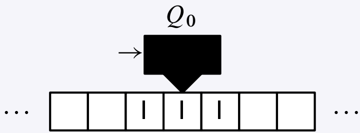
Interpretieren wir n-fache Wiederholung von $|$ als die Zahl n, so erhöht diese Maschine n auf $n+1$ (Kopf beginnt auf dem $|$ am weitesten links).



Beispiel 1:
Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\llcorner\!\lrcorner, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\llcorner\!\lrcorner$	Q_{end}	$\vert$	$\downarrow$

Interpretieren wir n-fache Wiederholung von $|$ als die Zahl n, so erhöht diese Maschine n auf $n+1$ (Kopf beginnt auf dem $|$ am weitesten links).

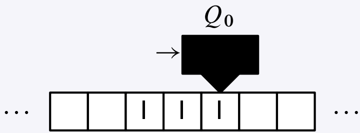


$$(Q_0, |) \mapsto (Q_0, |, \rightarrow)$$

Beispiel 1:
Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\sqcup, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\sqcup$	Q_{end}	$\vert$	$\downarrow$

Interpretieren wir n-fache Wiederholung von $|$ als die Zahl n, so erhöht diese Maschine n auf $n+1$ (Kopf beginnt auf dem $|$ am weitesten links).



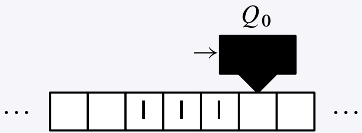
$$(Q_0, |) \mapsto (Q_0, |, \rightarrow)$$

Beispiel 1:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{␣, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	␣	Q_{end}	$\vert$	$\downarrow$

Interpretieren wir n-fache Wiederholung von $|$ als die Zahl n, so erhöht diese Maschine n auf $n+1$ (Kopf beginnt auf dem | am weitesten links).



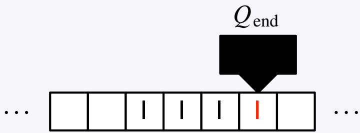
$$(Q_0, |) \mapsto (Q_0, |, \rightarrow)$$

Beispiel 1:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\sqcup, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\sqcup$	Q_{end}	$\vert$	$\downarrow$

Interpretieren wir n-fache Wiederholung von $|$ als die Zahl n, so erhöht diese Maschine n auf $n+1$ (Kopf beginnt auf dem $|$ am weitesten links).

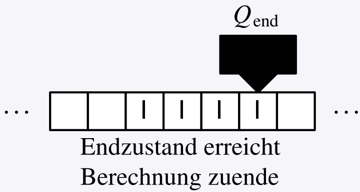


$$(Q_0, \sqcup) \mapsto (Q_{\text{end}}, |, \downarrow)$$

Beispiel 1:
Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{␣, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	␣	Q_{end}	$\vert$	$\downarrow$

Interpretieren wir n-fache Wiederholung von $|$ als die Zahl n, so erhöht diese Maschine n auf $n+1$ (Kopf beginnt auf dem | am weitesten links).



Beispiel 2:
Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\textvisiblespace, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\#$	Q_1	$\vert$	$\rightarrow$
Q_1	$\vert$	Q_1	$\vert$	$\rightarrow$
Q_1	$\textvisiblespace$	Q_2	$\textvisiblespace$	$\leftarrow$
Q_2	?	Q_{end}	$\textvisiblespace$	$\downarrow$

Hier steht „?” für ein beliebiges Zeichen.

Beispiel 2:
Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\llcorner, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\|$	Q_0	$\|$	$\rightarrow$
Q_0	$\#$	Q_1	$\|$	$\rightarrow$
Q_1	$\|$	Q_1	$\|$	$\rightarrow$
Q_1	$\llcorner$	Q_2	$\llcorner$	$\leftarrow$
Q_2	$?$	Q_{end}	$\llcorner$	$\downarrow$

Hier steht „?" für ein beliebiges Zeichen.
T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).

Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\sqcup, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\#$	Q_1	$\vert$	$\rightarrow$
Q_1	$\vert$	Q_1	$\vert$	$\rightarrow$
Q_1	$\sqcup$	Q_2	$\sqcup$	$\leftarrow$
Q_2	?	Q_{end}	$\sqcup$	$\downarrow$

Hier steht „?" für ein beliebiges Zeichen.
T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).

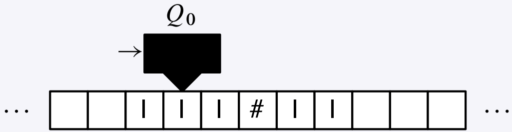
Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\text{␣}, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\|$	Q_0	$\|$	$\rightarrow$
Q_0	#	Q_1	$\|$	$\rightarrow$
Q_1	$\|$	Q_1	$\|$	$\rightarrow$
Q_1	␣	Q_2	␣	$\leftarrow$
Q_2	?	Q_{end}	␣	$\downarrow$

Hier steht „?” für ein beliebiges Zeichen.

T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



$$(Q_0, |) \mapsto (Q_0, |, \rightarrow)$$

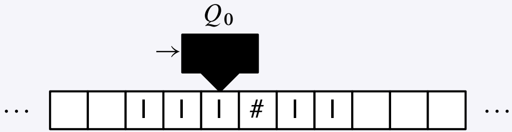
Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\textvisiblespace, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\#$	Q_1	$\vert$	$\rightarrow$
Q_1	$\vert$	Q_1	$\vert$	$\rightarrow$
Q_1	$\textvisiblespace$	Q_2	$\textvisiblespace$	$\leftarrow$
Q_2	$?$	Q_{end}	$\textvisiblespace$	$\downarrow$

Hier steht „?" für ein beliebiges Zeichen.

T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



$$(Q_0, |) \mapsto (Q_0, |, \rightarrow)$$

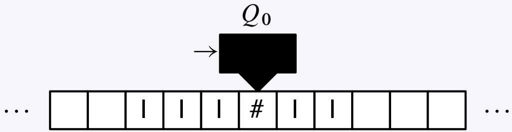
Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\sqcup, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\|$	Q_0	$\|$	$\rightarrow$
Q_0	#	Q_1	$\|$	$\rightarrow$
Q_1	$\|$	Q_1	$\|$	$\rightarrow$
Q_1	$\sqcup$	Q_2	$\sqcup$	$\leftarrow$
Q_2	?	Q_{end}	$\sqcup$	$\downarrow$

Hier steht „?" für ein beliebiges Zeichen.

T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



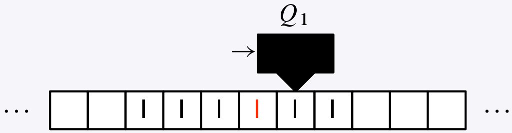
$$(Q_0, |) \mapsto (Q_0, |, \rightarrow)$$

Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\text{␣}, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\#$	Q_1	$\vert$	$\rightarrow$
Q_1	$\vert$	Q_1	$\vert$	$\rightarrow$
Q_1	␣	Q_2	␣	$\leftarrow$
Q_2	?	Q_{end}	␣	$\downarrow$

Hier steht „?” für ein beliebiges Zeichen.
T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



$$(Q_0, \#) \mapsto (Q_1, |, \rightarrow)$$

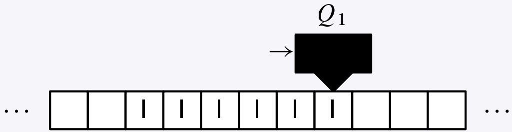
Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\sqcup, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\|$	Q_0	$\|$	$\rightarrow$
Q_0	$\#$	Q_1	$\|$	$\rightarrow$
Q_1	$\|$	Q_1	$\|$	$\rightarrow$
Q_1	$\sqcup$	Q_2	$\sqcup$	$\leftarrow$
Q_2	?	Q_{end}	$\sqcup$	$\downarrow$

Hier steht „?" für ein beliebiges Zeichen.

T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



$$(Q_1, |) \mapsto (Q_1, |, \rightarrow)$$

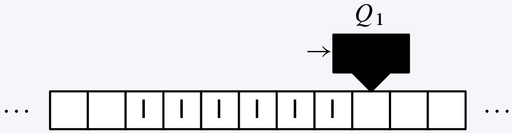
Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{␣, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\#$	Q_1	$\vert$	$\rightarrow$
Q_1	$\vert$	Q_1	$\vert$	$\rightarrow$
Q_1	␣	Q_2	␣	$\leftarrow$
Q_2	?	Q_{end}	␣	$\downarrow$

Hier steht „?" für ein beliebiges Zeichen.

T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



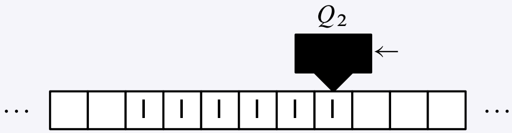
$$(Q_1, |) \mapsto (Q_1, |, \rightarrow)$$

Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{␣, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\#$	Q_1	$\vert$	$\rightarrow$
Q_1	$\vert$	Q_1	$\vert$	$\rightarrow$
Q_1	␣	Q_2	␣	$\leftarrow$
Q_2	?	Q_{end}	␣	$\downarrow$

Hier steht „?” für ein beliebiges Zeichen.
T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



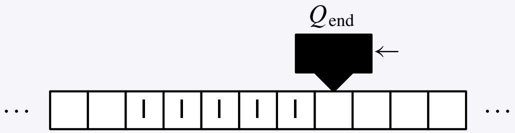
$(Q_1, ␣) \mapsto (Q_2, ␣, \leftarrow)$

Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\sqcup, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\|$	Q_0	$\|$	$\rightarrow$
Q_0	$\#$	Q_1	$\|$	$\rightarrow$
Q_1	$\|$	Q_1	$\|$	$\rightarrow$
Q_1	$\sqcup$	Q_2	$\sqcup$	$\leftarrow$
Q_2	$?$	Q_{end}	$\sqcup$	$\downarrow$

Hier steht „?” für ein beliebiges Zeichen.
T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



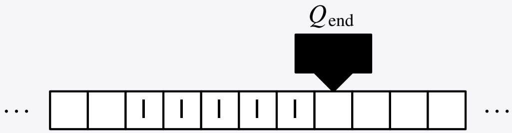
$$(Q_2, ?) \mapsto (Q_{\text{end}}, \sqcup, \downarrow)$$

Beispiel 2:

Eine Turing-Maschine T habe Bandalphabet $\mathcal{B} = \{\sqcup, \#, |\}$ und Übergänge, die durch die folgende Tabelle gegeben sind:

Q	z	Q'	z'	α
Q_0	$\vert$	Q_0	$\vert$	$\rightarrow$
Q_0	$\#$	Q_1	$\vert$	$\rightarrow$
Q_1	$\vert$	Q_1	$\vert$	$\rightarrow$
Q_1	$\sqcup$	Q_2	$\sqcup$	$\leftarrow$
Q_2	?	Q_{end}	$\sqcup$	$\downarrow$

Hier steht „?" für ein beliebiges Zeichen.
T addiert zwei Zahlen, die durch Strichfolgen gegeben sind und durch das Trennzeichen # getrennt werden (Kopf beginnt auf dem | am weitesten links).



Endzustand erreicht
Berechnung zuende

Die Klasse **P** der deterministisch polynomialen Entscheidungsprobleme besteht aus allen (entscheidbaren) Prädikaten, die von einer Turing-Maschine in polynomialer Zeit entschieden werden können.

Die Klasse **P** der deterministisch polynomialen Entscheidungsprobleme besteht aus allen (entscheidbaren) Prädikaten, die von einer Turing-Maschine in polynomialer Zeit entschieden werden können.

- T entscheidet ein Prädikat p, wenn T für Eingaben aus der Menge $M_p = \{x \mid p(x) = \text{wahr}\}$ in einem Endzustand Q_{wahr} endet, und für $x \notin M_p$ in einem Endzustand Q_{falsch} endet.

Die Klasse **P** der deterministisch polynomialen Entscheidungsprobleme besteht aus allen (entscheidbaren) Prädikaten, die von einer Turing-Maschine in polynomialer Zeit entschieden werden können.

- T entscheidet ein Prädikat p, wenn T für Eingaben aus der Menge $M_p = \{x \mid p(x) = \text{wahr}\}$ in einem Endzustand Q_{wahr} endet, und für $x \notin M_p$ in einem Endzustand Q_{falsch} endet.
- „Zeit" wird für Turing-Maschinen in der Anzahl der Zustandsübergänge gemessen.

Die Klasse **P** der deterministisch polynomialen Entscheidungsprobleme besteht aus allen (entscheidbaren) Prädikaten, die von einer Turing-Maschine in polynomialer Zeit entschieden werden können.

- T entscheidet ein Prädikat p, wenn T für Eingaben aus der Menge $M_p = \{x \mid p(x) = \text{wahr}\}$ in einem Endzustand Q_{wahr} endet, und für $x \notin M_p$ in einem Endzustand Q_{falsch} endet.
- „Zeit" wird für Turing-Maschinen in der Anzahl der Zustandsübergänge gemessen.
- Fakt:
  Programme, die durch Turing-Maschinen gegeben sind, lassen sich mit polynomialem Aufwand in äquivalente Programme in gleichmächtigen Rechenmodellen (z.B. RAM-Modell) übertragen.

Die Klasse **P** der deterministisch polynomialen Entscheidungsprobleme besteht aus allen (entscheidbaren) Prädikaten, die von einer Turing-Maschine in polynomialer Zeit entschieden werden können.

- T entscheidet ein Prädikat p, wenn T für Eingaben aus der Menge $M_p = \{x \mid p(x) = \text{wahr}\}$ in einem Endzustand Q_{wahr} endet, und für $x \notin M_p$ in einem Endzustand Q_{falsch} endet.
- „Zeit" wird für Turing-Maschinen in der Anzahl der Zustandsübergänge gemessen.
- Fakt:
  Programme, die durch Turing-Maschinen gegeben sind, lassen sich mit polynomialem Aufwand in äquivalente Programme in gleichmächtigen Rechenmodellen (z.B. RAM-Modell) übertragen. Somit ist diese Definition von **P** für alle gebräuchlichen deterministischen Rechenmodelle die selbe.

Nicht in **P**?

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Nicht in **P**?

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Was können wir hier tun?

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Was können wir hier tun?

- Wir haben jedoch gesehen, dass wir für TSP mit polynomialem Aufwand geprüft werden kann, ob ein Lösungsvorschlag korrekt ist.

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Was können wir hier tun?

- Wir haben jedoch gesehen, dass wir für TSP mit polynomialem Aufwand geprüft werden kann, ob ein Lösungsvorschlag korrekt ist.
- Dies gilt ebenso für HC, da ja HC $\preceq_p$ TSP.

Nicht in **P**?

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Was können wir hier tun?

- Wir haben jedoch gesehen, dass wir für TSP mit polynomialem Aufwand geprüft werden kann, ob ein Lösungsvorschlag korrekt ist.
- Dies gilt ebenso für HC, da ja HC $\preceq_p$ TSP.
- Dies liefert einen Algorithmus/eine Turing-Maschine, die diese Probleme exakt lösen kann:
  - Erzeuge einen neuen Lösungsvorschlag Z (in polynomialer Zeit möglich).
  - Teste, ob Z eine Lösung für TSP (bzw. HC) ist.
  - Wiederhole, bis alle Lösungvorschläge abgearbeitet wurden oder eine Lösung gefunden wurde.

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Was können wir hier tun?

- Wir haben jedoch gesehen, dass wir für TSP mit polynomialem Aufwand geprüft werden kann, ob ein Lösungsvorschlag korrekt ist.
- Dies gilt ebenso für HC, da ja HC $\preceq_{\mathrm{p}}$ TSP.
- Dies liefert einen Algorithmus/eine Turing-Maschine, die diese Probleme exakt lösen kann:
  - Erzeuge einen neuen Lösungsvorschlag Z (in polynomialer Zeit möglich).
  - Teste, ob Z eine Lösung für TSP (bzw. HC) ist.
  - Wiederhole, bis alle Lösungvorschläge abgearbeitet wurden oder eine Lösung gefunden wurde.
- Obwohl der Test effizient ist, haben wir das Problem, dass in der Regel exponentiell viele Lösungsmöglichkeiten getestet werden müssen.

Nicht in **P**?

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Was können wir hier tun?

- Wir haben jedoch gesehen, dass wir für TSP mit polynomialem Aufwand geprüft werden kann, ob ein Lösungsvorschlag korrekt ist.
- Dies gilt ebenso für HC, da ja HC $\preceq_{\mathrm{p}}$ TSP.
- Dies liefert einen Algorithmus/eine Turing-Maschine, die diese Probleme exakt lösen kann:
  - Erzeuge einen neuen Lösungsvorschlag Z (in polynomialer Zeit möglich).
  - Teste, ob Z eine Lösung für TSP (bzw. HC) ist.
  - Wiederhole, bis alle Lösungvorschläge abgearbeitet wurden oder eine Lösung gefunden wurde.
- Obwohl der Test effizient ist, haben wir das Problem, dass in der Regel exponentiell viele Lösungsmöglichkeiten getestet werden müssen.
- Wäre es nicht toll, wenn wir eine Turing-Maschine hätten, die uns alle Möglichkeiten gleichzeitig überprüfen ließe?

Für Entscheidungsprobleme wie HC oder TSP kennen wir keine Turing-Maschine, die sie entscheiden kann. Wir wissen also nicht, ob sie in **P** liegen.

Was können wir hier tun?

- Wir haben jedoch gesehen, dass wir für TSP mit polynomialem Aufwand geprüft werden kann, ob ein Lösungsvorschlag korrekt ist.
- Dies gilt ebenso für HC, da ja HC $\preceq_p$ TSP.
- Dies liefert einen Algorithmus/eine Turing-Maschine, die diese Probleme exakt lösen kann:
  - Erzeuge einen neuen Lösungsvorschlag Z (in polynomialer Zeit möglich).
  - Teste, ob Z eine Lösung für TSP (bzw. HC) ist.
  - Wiederhole, bis alle Lösungvorschläge abgearbeitet wurden oder eine Lösung gefunden wurde.
- Obwohl der Test effizient ist, haben wir das Problem, dass in der Regel exponentiell viele Lösungsmöglichkeiten getestet werden müssen.
- Wäre es nicht toll, wenn wir eine Turing-Maschine hätten, die uns alle Möglichkeiten gleichzeitig überprüfen ließe? Oder ein Orakel, dass uns sagt, welche Möglichkeit wir überprüfen müssen?

Nicht-deterministische Turing-Maschinen

Bei einer nicht-deterministische Turing-Maschine NT hat jeder Übergang mehrere mögliche Ausgänge,

$$(Q, z) \mapsto \begin{cases} (Q_1, z_1) \\ (Q_2, z_2) \\ \vdots \\ (Q_k, z_k) \end{cases}$$

Bei einer nicht-deterministische Turing-Maschine NT hat jeder Übergang mehrere mögliche Ausgänge,

$$(Q, z) \mapsto \begin{cases} (Q_1, z_1) \\ (Q_2, z_2) \\ \vdots \\ (Q_k, z_k) \end{cases}$$

NT entscheidet ein Prädikat p, wenn wir für eine Eingabe (Bandinschrift) x aus allen möglichen Übergängen endlich viele auswählen können, so dass wir in einen Endzustand Q_{wahr} gelangen falls $x \in M_p = \{x \mid p(x) = \text{wahr}\}$, und in einem Endzustand Q_{falsch} gelangen, falls $x \notin M_p$.

Nicht-deterministische Turing-Maschinen

Bei einer nicht-deterministische Turing-Maschine NT hat jeder Übergang mehrere mögliche Ausgänge,

$$(Q, z) \mapsto \begin{cases} (Q_1, z_1) \\ (Q_2, z_2) \\ \vdots \\ (Q_k, z_k) \end{cases}$$

NT entscheidet ein Prädikat p, wenn wir für eine Eingabe (Bandinschrift) x aus allen möglichen Übergängen endlich viele auswählen können, so dass wir in einen Endzustand Q_{wahr} gelangen falls $x \in M_p = \{x \mid p(x) = \text{wahr}\}$, und in einem Endzustand Q_{falsch} gelangen, falls $x \notin M_p$.

Dies kann man sich so vorstellen,

- als könnte man in jedem Schritt alle möglichen Übergänge gleichzeitig ausführen, um auf irgendeinem Wege zu einem Endzustand zu gelangen,
- oder als hätte man ein Orakel, das einem bei jedem Übergang den richtigen Ausgang vorhersagt.

Die Klasse **NP** der nicht-deterministisch polynomialen Entscheidungsprobleme besteht aus allen (entscheidbaren) Prädikaten, die von einer nicht-deterministischen Turing-Maschine in polynomialer Zeit entschieden werden können.

Nachlesen

T.H. Cormen, C.E. Leiserson, R. Rivest, C. Stein,
Algorithmen – Eine Einführung,
Abschnitte 34.2, 34.5.4, 35.2