Algorithmus
Nearest-Neighbor

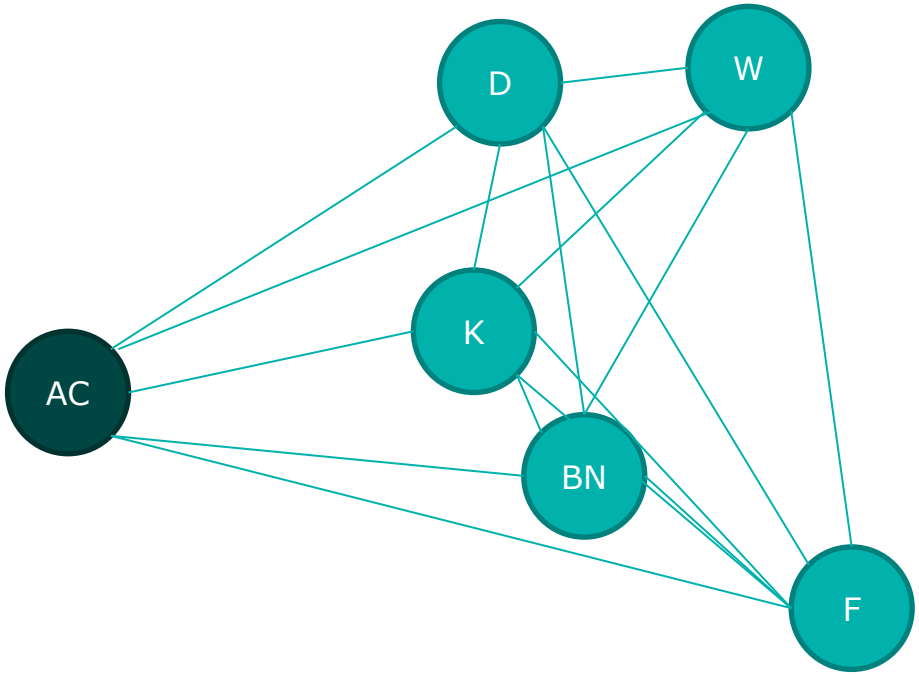
Algorithmus
Nearest-Neighbor

- Heuristisches Verfahren
- Greedy-Algorithmus
- Versucht eine Lösung für das Problem des Handlungsreisenden zu finden

Algorithmus Nearest-Neighbor

	Beschreibung
Eingabe:	Ein vollständiger Graph T mit Kantengewichten $c(e)$
Ausgabe:	Ein Hamilton-Kreis
Schritt 1:	Wähle einen beliebigen Knoten als Startknoten v
Schritt 2:	Ermittle die niedrigste Kante welche den aktuellen Knoten v mit einem unbesuchten Knoten v_u verbindet.
Schritt 3:	Setze $v = v_u$
Schritt 4:	Wenn noch nicht alle Knoten besucht wurden gehe wieder zu Schritt 2
Schritt 5:	Füge die Kante vom letzten besuchten Knoten zum Startknoten hinzu um den Kreis zu schließen.

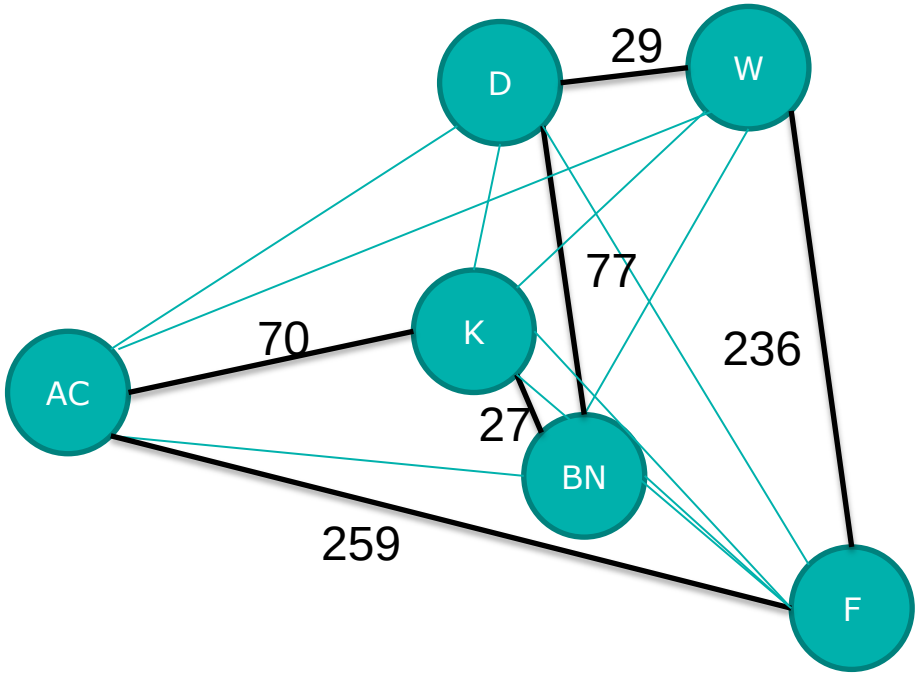
Algorithmus Nearest-Neighbor



	AC	BN	D	F	K	W
AC		91	80	259	70	121
BN	91		77	175	27	84
D	80	77		232	47	29
F	259	175	232		189	236
K	70	27	47	189		55
W	121	84	29	236	55	

	Beschreibung
Eingabe:	Ein vollständiger Graph T mit Kantengewichten $c(e)$
Ausgabe:	Ein Hamilton-Kreis
Schritt 1:	Wähle einen beliebigen Knoten als Startknoten v
Schritt 2:	Ermittle die niedrigste Kante welche den aktuellen Knoten v mit einem unbesuchten Knoten v_u verbindet.
Schritt 3:	Setze $v = v_u$
Schritt 4:	Wenn noch nicht alle Knoten besucht wurden gehe wieder zu Schritt 2
Schritt 5:	Füge die Kante vom letzten besuchten Knoten zum Startknoten hinzu um den Kreis zu schließen.

Algorithmus Nearest-Neighbor



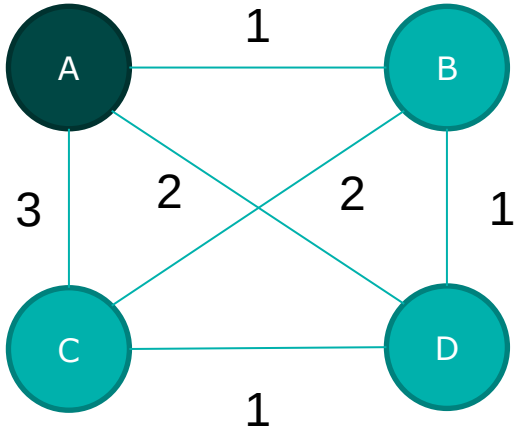
- Gesamtkosten: 698
- Keine optimale Lösung
  - AC-K-BN-F-W-D-AC
    Gesamtkosten: 617

	AC	BN	D	F	K	W
AC		91	80	259	70	121
BN	91		77	175	27	84
D	80	77		232	47	29
F	259	175	232		189	236
K	70	27	47	189		55
W	121	84	29	236	55	

Algorithmus Nearest-Neighbor

- Wie schlecht kann das Ergebnis werden?

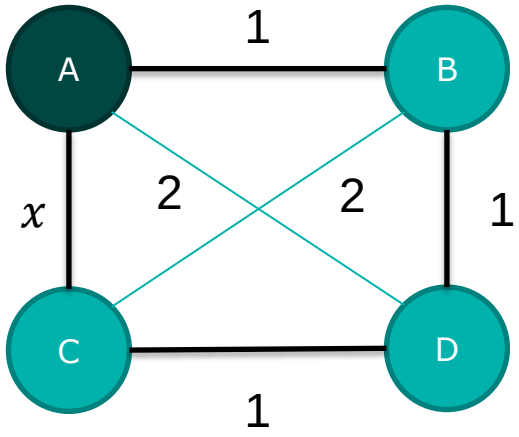
- ***Aufgabe:*** *Wende den Algorithmus auf den Graphen an.*
  Wähle A als Startknoten



	Beschreibung
Schritt 1:	Wähle einen beliebigen Knoten als Startknoten v
Schritt 2:	Ermittle die niedrigste Kante welche den aktuellen Knoten v mit einem unbesuchten Knoten v_u verbindet.
Schritt 3:	Setze $v = v_u$
Schritt 4:	Wenn noch nicht alle Knoten besucht wurden gehe wieder zu Schritt 2
Schritt 5:	Füge die Kante vom letzten besuchten Knoten zum Startknoten hinzu um den Kreis zu schließen.

Algorithmus Nearest-Neighbor

- Wie schlecht kann das Ergebnis werden?



Gesamtkosten: $1 + 1 + 1 + x$

Algorithmus
Doppelter-Baum

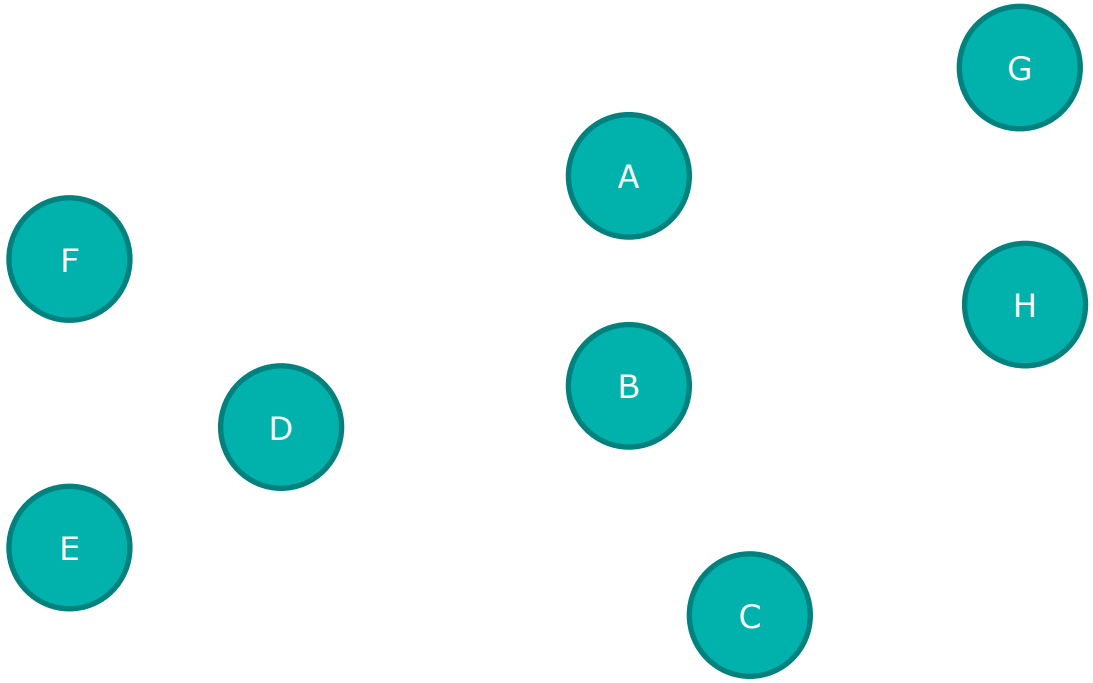
Algorithmus
Doppelter Baum

- Von Rosenkranz, Stearns und Lewis (1977)
- Berechnet einen Hamilton-Kreis

Algorithmus
Doppelter Baum

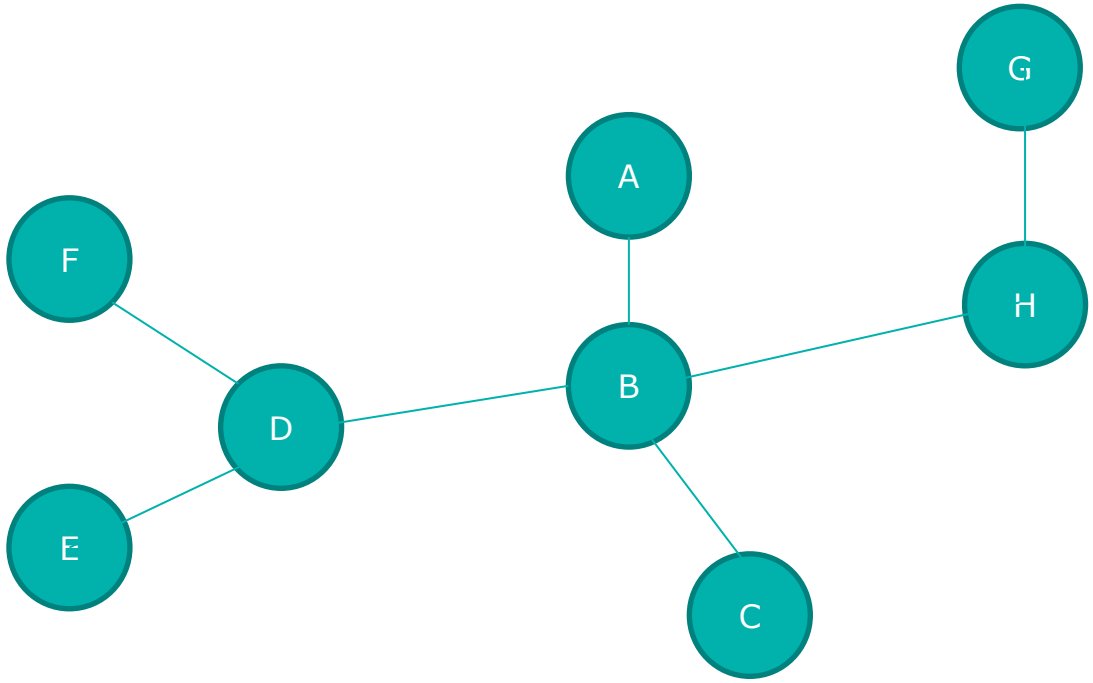
	Beschreibung
Eingabe:	Ein vollständiger Graph K_n mit Kantengewichten $c(e)$
Ausgabe:	Ein Hamilton-Kreis
Schritt 1:	Konstruiere einen minimal spannenden Baum T von K_n
Schritt 2:	Verdopple alle Kanten von T (daraus resultiert ein eulerscher Graph T_d).
Schritt 3:	Berechne eine Euler-Tour in T_d
Schritt 4:	Durchlaufe die Euler Tour von einem Startknoten aus. Falls dabei ein Knoten schon besucht wurde, nehme die Abkürzung zum nächsten unbesuchten Knoten auf der Tour.

Algorithmus
Doppelter Baum



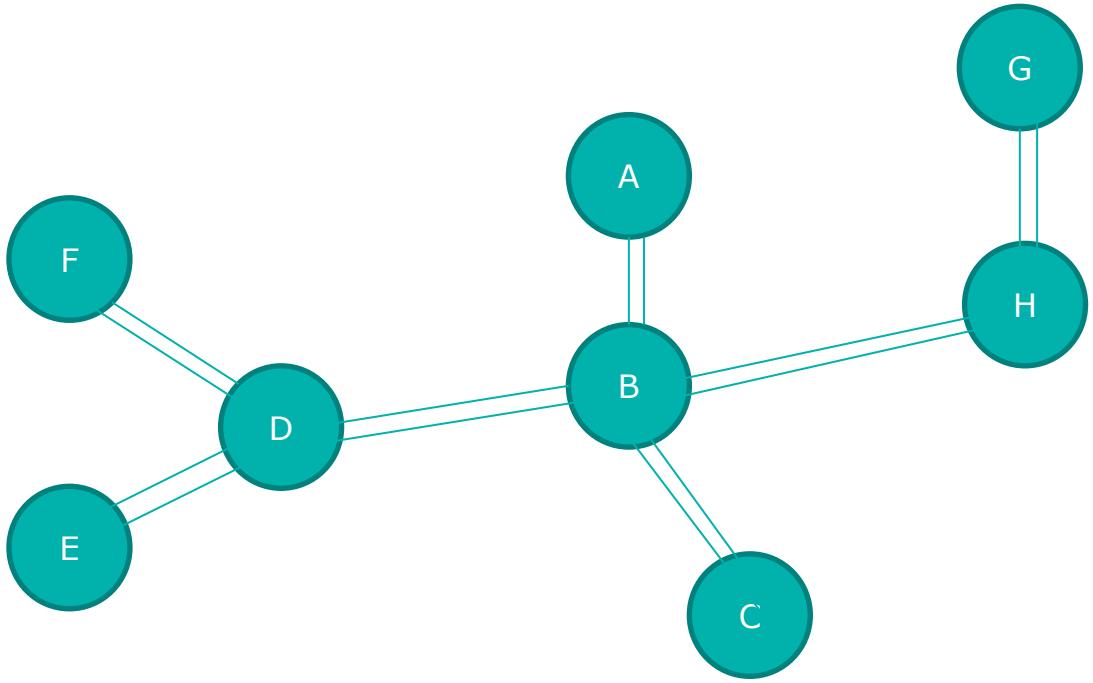
Eingabe:	Ein vollständiger Graph K_n mit euklidischen Abstand der Knoten als Kantengewicht (Kante und Gewicht hier nicht eingezeichnet)

Algorithmus
Doppelter Baum



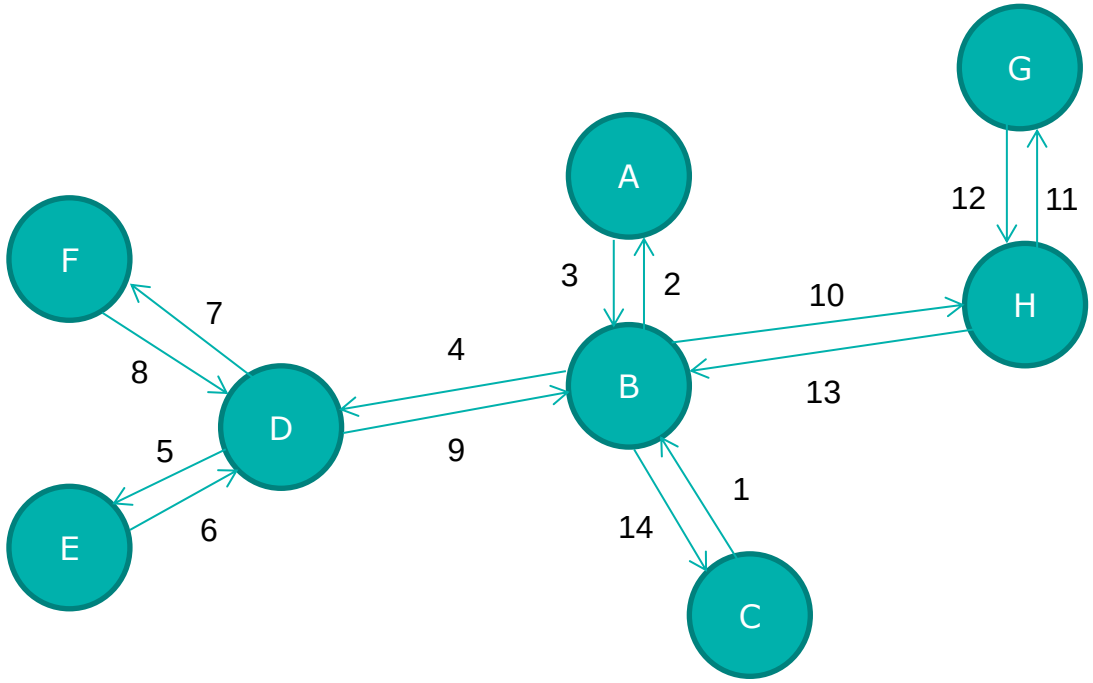
Schritt 1:	Konstruiere einen minimal spannenden Baum T von K_n

Algorithmus
Doppelter Baum



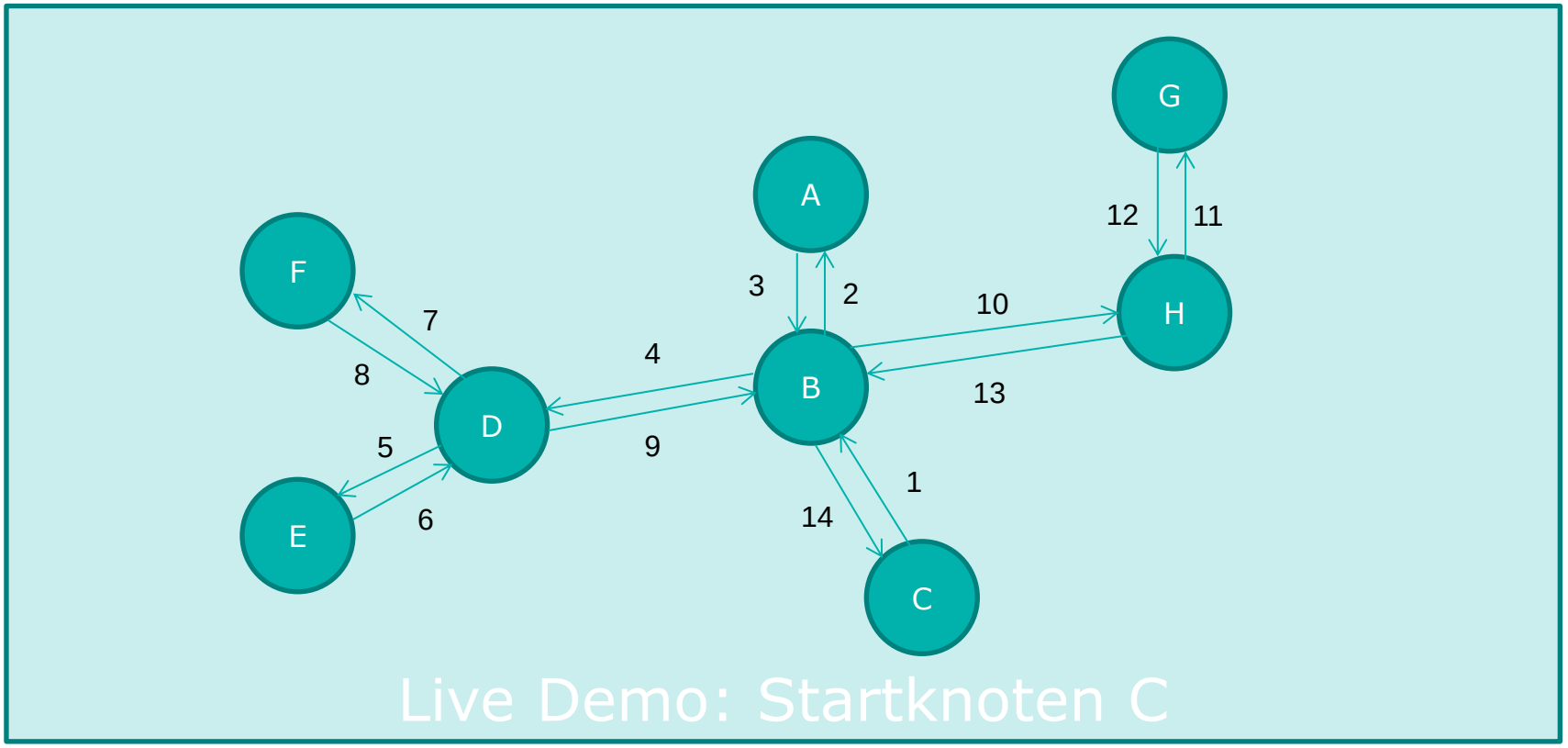
Schritt 2:	Verdopple alle Kanten von T (daraus resultiert ein eulerscher Graph T_d).

Algorithmus
Doppelter Baum



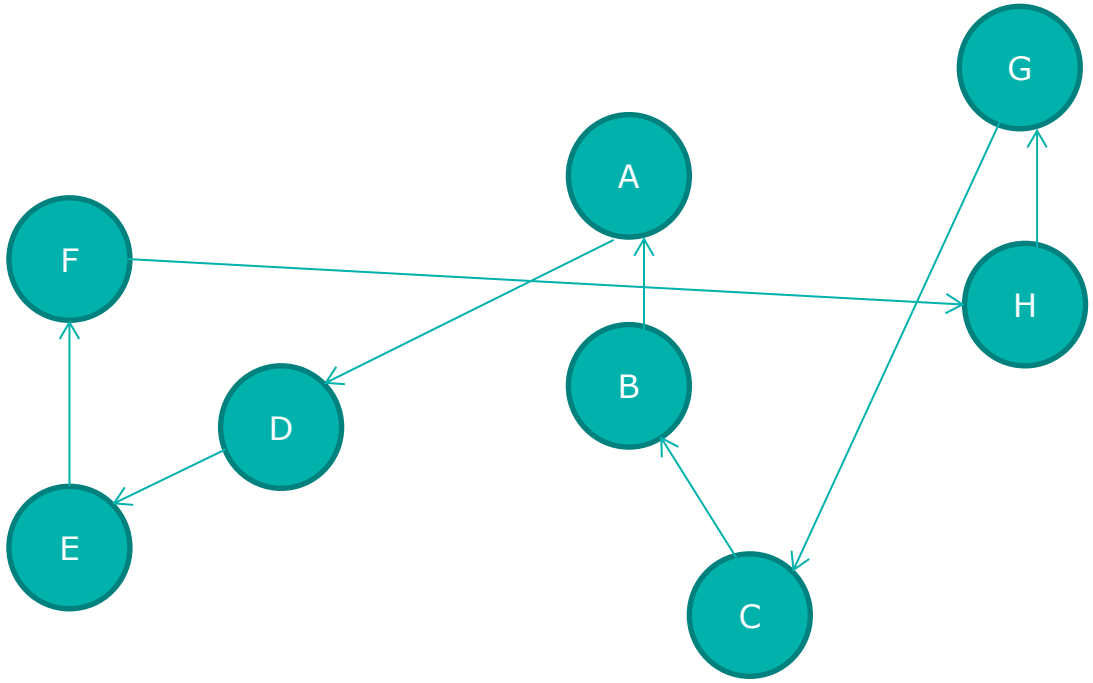
Schritt 3:	Berechne eine Euler-Tour in T_d

Algorithmus
Doppelter Baum



Schritt 4:	Durchlaufe die Euler Tour von einem Startknoten aus. Falls dabei ein Knoten schon besucht wurde, nehme die Abkürzung zum nächsten unbesuchten Knoten auf der Tour.

Algorithmus
Doppelter Baum



Ausgabe:	Ein Hamilton-Kreis

Algorithmus
Doppelter Baum

- Wie gut ist das Ergebnis des Algorithmus?
- Forderung der Dreiecksungleichung

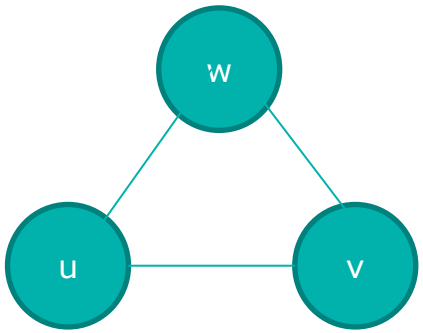
Algorithmus
Einschub: Dreiecksungleichung

- Kosten c zweier Knoten:
  Kantengewicht der verbindenden Kante

Definition: Dreiecksungleichung

Die Dreiecksungleichung garantiert bei einem vollständigen Graphen, dass für alle Knoten u, v und w gilt:

$$c(u,v) \leq c(u,w) + c(w,v)$$



Algorithmus
Doppelter Baum - Abschätzung

- Wie gut ist das Ergebnis des Algorithmus?

	Beschreibung
Eingabe:	Ein vollständiger Graph K_n mit Kantengewichten $c(e)$, **die die Dreiecksungleichung erfüllen.**

Satz

K_n sei ein vollständiger Graph mit Kantengewichten welche die Dreiecksungleichung erfüllen.
Ferner sei T' das Ergebnis des Doppelten-Baum-Algorithmus und OPT eine optimale Lösung.
Dann gilt:

$$c(T') \leq 2 * c(OPT)$$

„Die durch den Algorithmus bestimmte Tour ist maximal doppelt so lange wie eine optimale Tour"

Algorithmus
Doppelter Baum – Abschätzung (Beweis)

Wie kommt man darauf?

1. Der minimale Spannbaum T wird verdoppelt und daraus wird eine Euler-Tour gebildet (Schritt 2 & 3 im Algorithmus).

   Länge der Euler-Tour: $2 * c(T)$

2. Im vierten Schritt wird die Euler-Tour verfolgt oder eine direkte Kante gewählt.
   Die direkte Kante ist günstiger als ein Umweg (Dreiecksungleichung)

3. Für die resultierende Hamilton-Tour T' gilt:

$$c(T') \leq 2 * c(T)$$ (obere Schranke)

Algorithmus
Doppelter Baum – Abschätzung (Beweis)

4. Entfernt man eine Kante aus der optimalen Tour OPT, erhält man einen spannenden Baum

   Dieser ist aber nicht billiger als der minimalspannende Baum T.

   Es gilt also: $c(T) \leq c(OPT)$ (untere Schranke)

Algorithmus
Doppelter Baum – Abschätzung (Beweis)

$$c(T') \leq 2 * c(T)$$ (obere Schranke)

$$c(T) \leq c(OPT)$$ (untere Schranke)

$$c(T') \leq 2 * c(OPT)$$ (Satz)