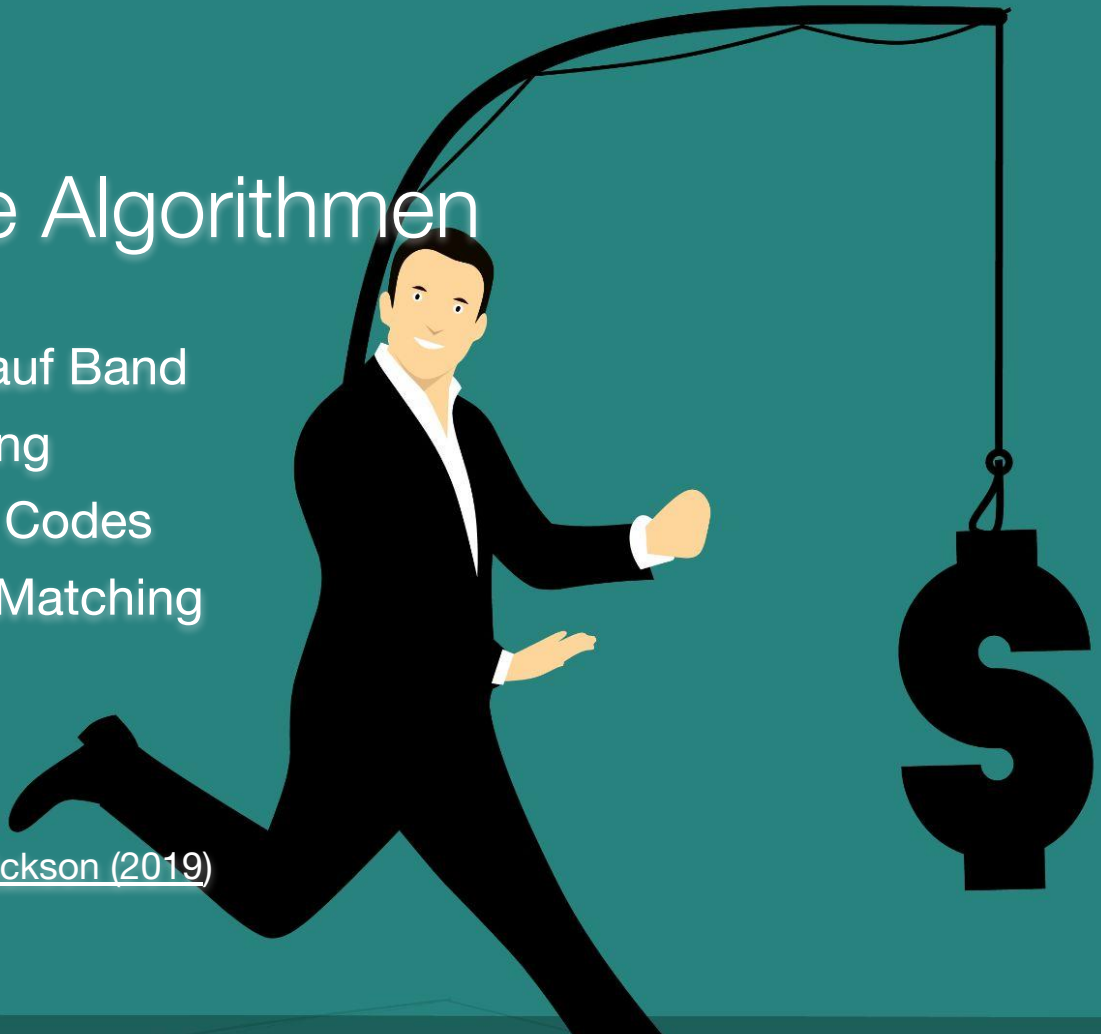
Gierige Algorithmen

- Dateien auf Band
- Scheduling
- Huffman Codes
- Stabiles Matching



Kapitel 4 in Erickson (2019)

CC BY 4.0

Holger Dell

Entwurfsmethoden

- Divide and Conquer
  Merge Sort, Algorithmen auf Bäumen, …
- Dynamische Programmierung
  Fibonacci, Edit Distance, Algorithmen auf Bäumen, …
- Gierige Algorithmen
  Prim/Kruskal/Dijkstra, Huffman-Codierung, ...

- **Warnung!!!**
  - Gieriger Ansatz funktioniert fast nie!
    ⇒ Korrektheitsbeweis enorm wichtig!
  - **Dynamische Programmierung** ist besser!

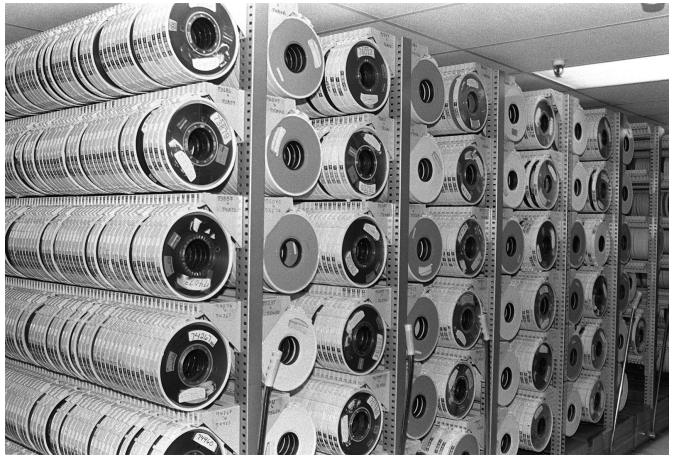
Gierige Algorithmen

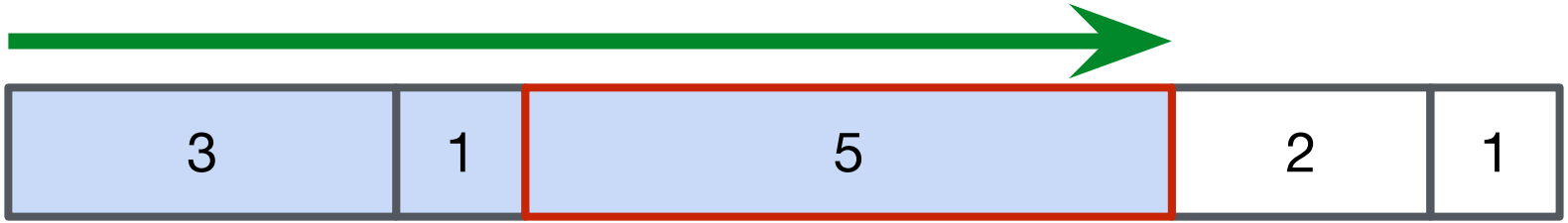
- Dateien auf Band
- Scheduling
- Huffman Codes
- Stabiles Matching

Dateien auf Band

- Magnetband. Speichert n Dateien.
- L[i] = Länge der i-ten Datei.
- Die k-te Datei lesen kostet:

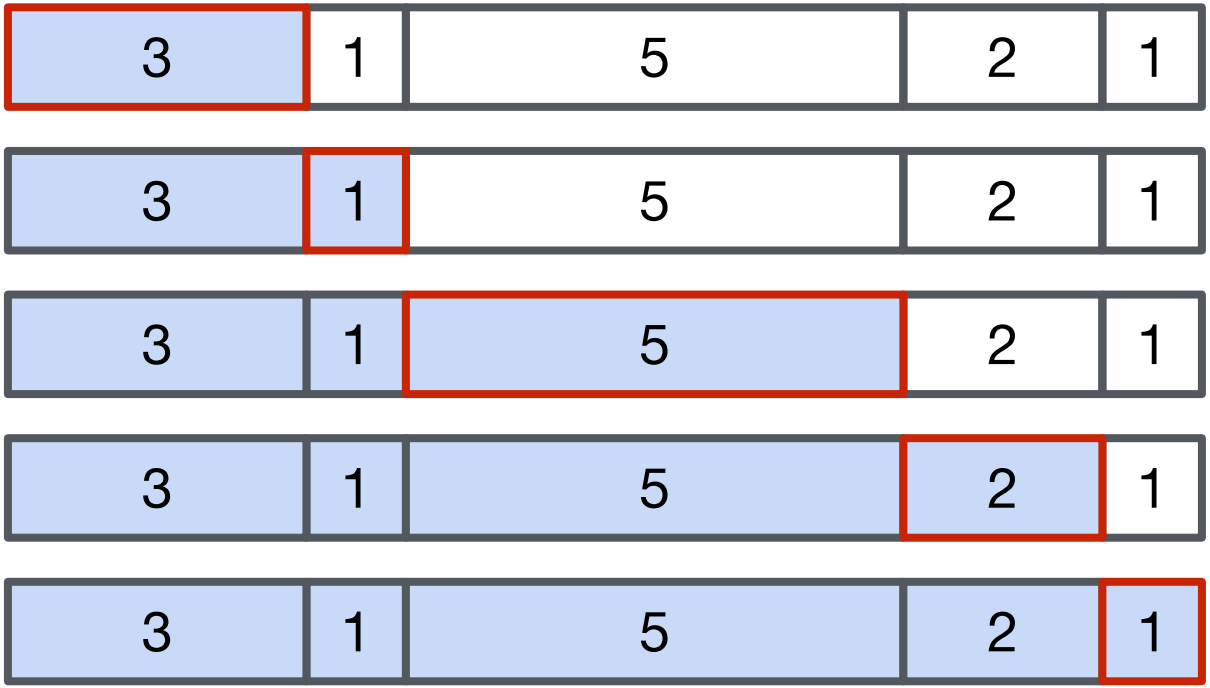
$$cost(k) = \sum_{i=1}^{k} L[i]$$





Zufällige Datei lesen

- Wir lesen die k-te Datei für **zufälliges** k.



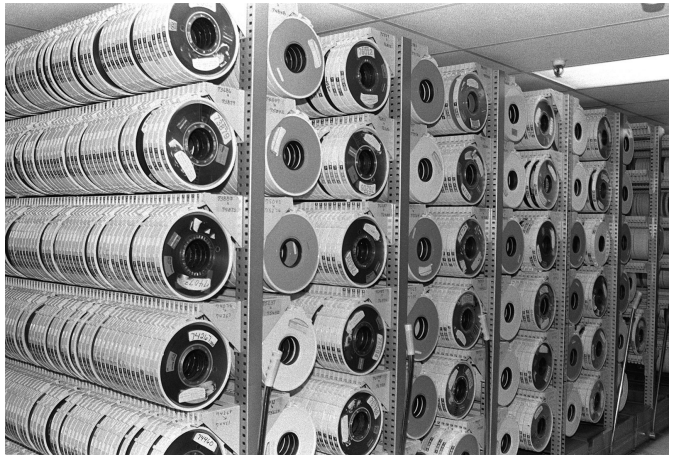
Kosten	WSK
3	1/5
4	1/5
9	1/5
11	1/5
12	1/5

- Erwartete Kosten. ⅕*(3+4+9+11+12) = 7,8

Zufällige Datei lesen

- Erwartete Kosten. Um eine **zufällige** Datei zu lesen, entstehen Kosten:

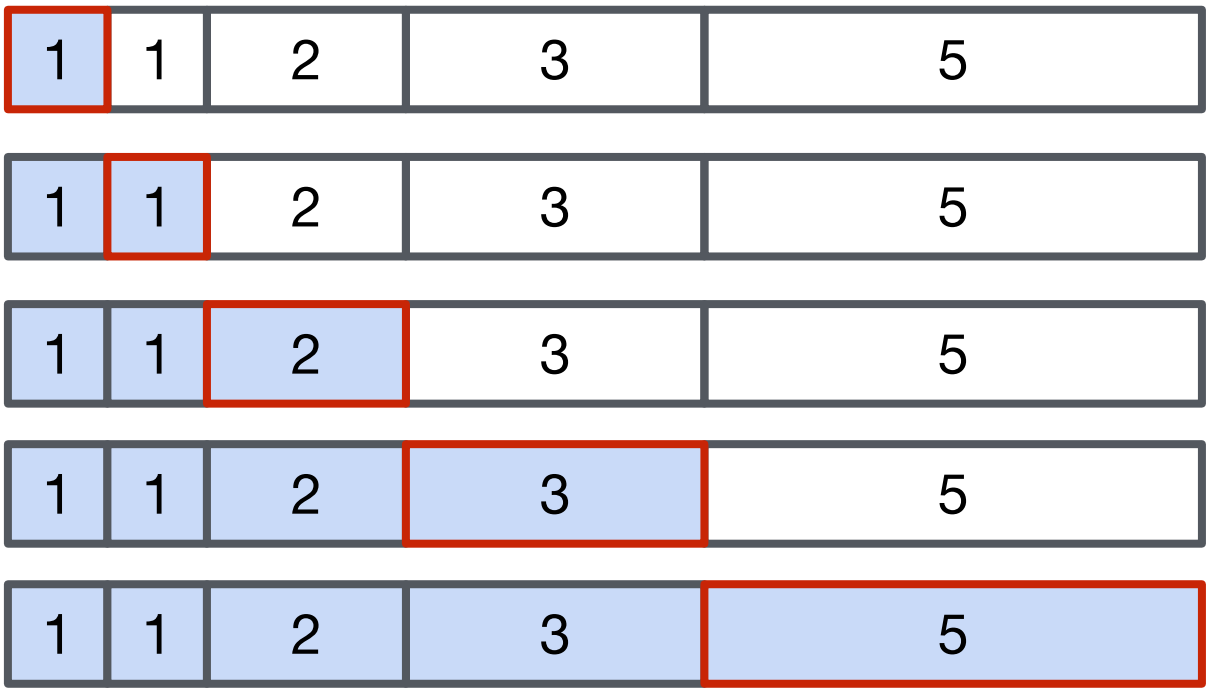
$$\mathrm{E}[cost] = \sum_{k=1}^{n} \frac{cost(k)}{n} = \frac{1}{n} \sum_{k=1}^{n} \sum_{i=1}^{k} L[i]$$



- Frage. In welcher Reihenfolge müssen wir die Dateien speichern, um die erwarteten Kosten zu minimieren?

Zufällige Datei lesen

- Wir lesen die k-te Datei für **zufälliges** k.



- Erwartete Kosten. ⅕*(1+2+4+7+12) = 5,2

Zufällige Datei lesen

- Dateien “gierig” speichern.
  - Sortiere die Dateien der Länge L[i] nach.
  - Speichere die kleinsten Dateien zuerst ab.

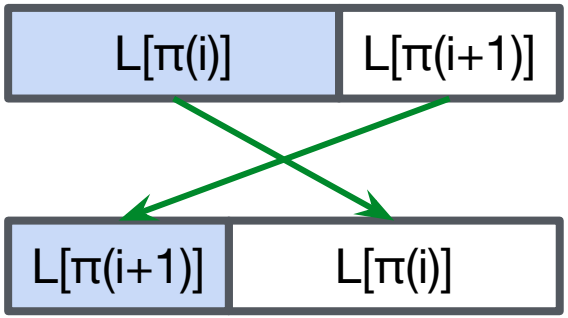
- Diese Methode liefert die kleinsten erwarteten Kosten.
  Aber warum??

Zufällige Datei lesen

- Reihenfolge π. Wir speichern Datei i an Stelle π(i) ∈ {1,...,n} ab.
- Erwartete Kosten für π: $\mathrm{E}[cost(\pi)] = \frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{k} L[\pi(i)]$

Lemma. E[cost(π)] ist minimal, wenn ∀i. L[π(i)] ≤ L[π(i+1)].

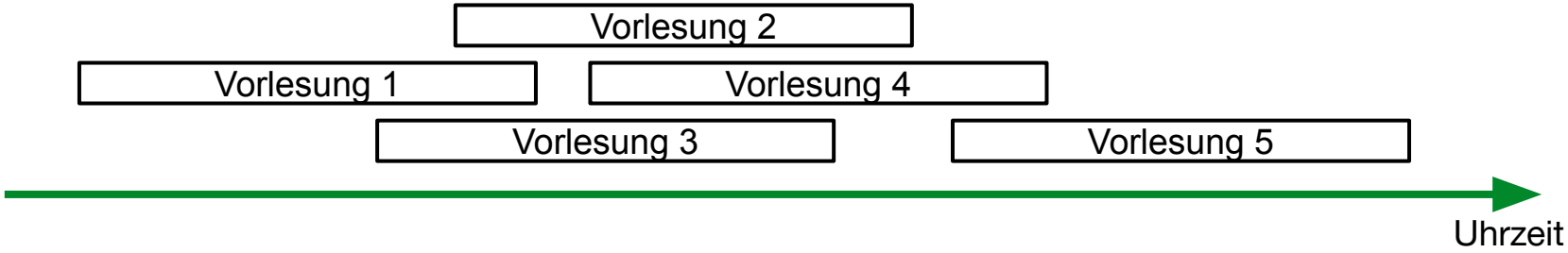
Beweis durch Widerspruch.
Angenommen L[π(i)] > L[π(i+1)].
Tauschen π(i) und π(i+1), und erhalten π'.
⇒ E[cost(π)] - E[cost(π')]
= (L[π(i)] - L[π(i+1)]) / n > 0.
⇒ Widerspruch zur Minimalität. ↯



Gierige Algorithmen

- Dateien auf Band
- Scheduling
- Huffman Codes
- Stabiles Matching

Vorlesungen belegen



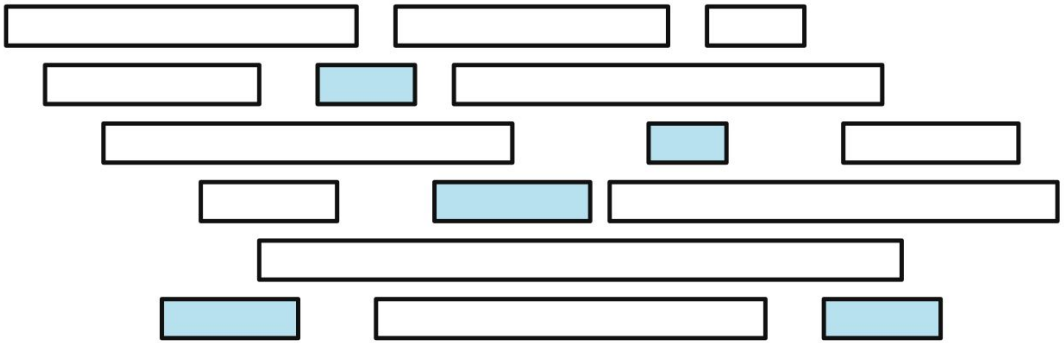
- Vorlesungen. Vorlesung i beginnt zum Zeitpunkt S[i] und endet zum Zeitpunkt F[i]. Es gilt $0 \le S[i] < F[i] \le M$ für irgendein M.

- Konflikt. Zeiten von Vorlesung i und j überlappen sich.

- Stundenplan (*schedule*). Eine Menge $X \subseteq \{1, \ldots, n\}$.

- **Ziel.** Belege so viele Vorlesungen wie möglich, ohne Konflikte.

Optimaler Stundenplan



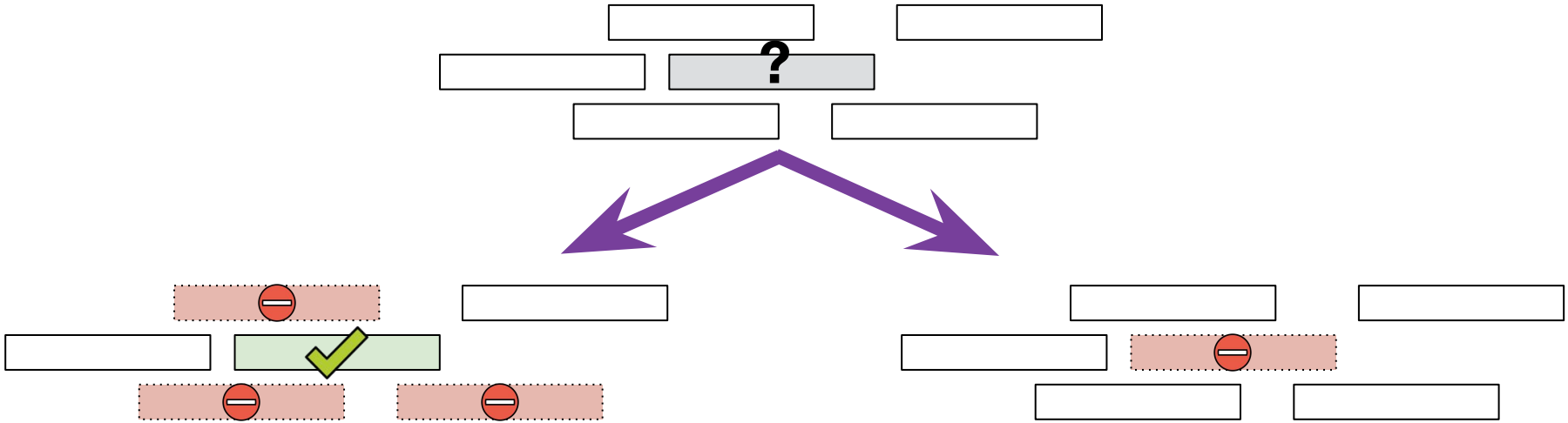
Ein maximaler konfliktfreier Stundenplan
(*schedule*) für eine Menge von Vorlesungen

Anwendungen

- Enorm viele Anwendungen und Varianten
- Betriebssystem. Prozesse auf Prozessoren zuweisen
- High-performance computing. Jobs auf Maschinen zuweisen
- Satelliten und Rover. Energiemanagement
- Verkehrsplanung. Fahrzeiten für Bus, Bahn, Flugzeug
- ...

Rekursiver Ansatz

- Frage. Wie würden wir das Problem rekursiv lösen?
- Beobachtung. Wir belegen entweder Vorlesung 1 oder nicht.

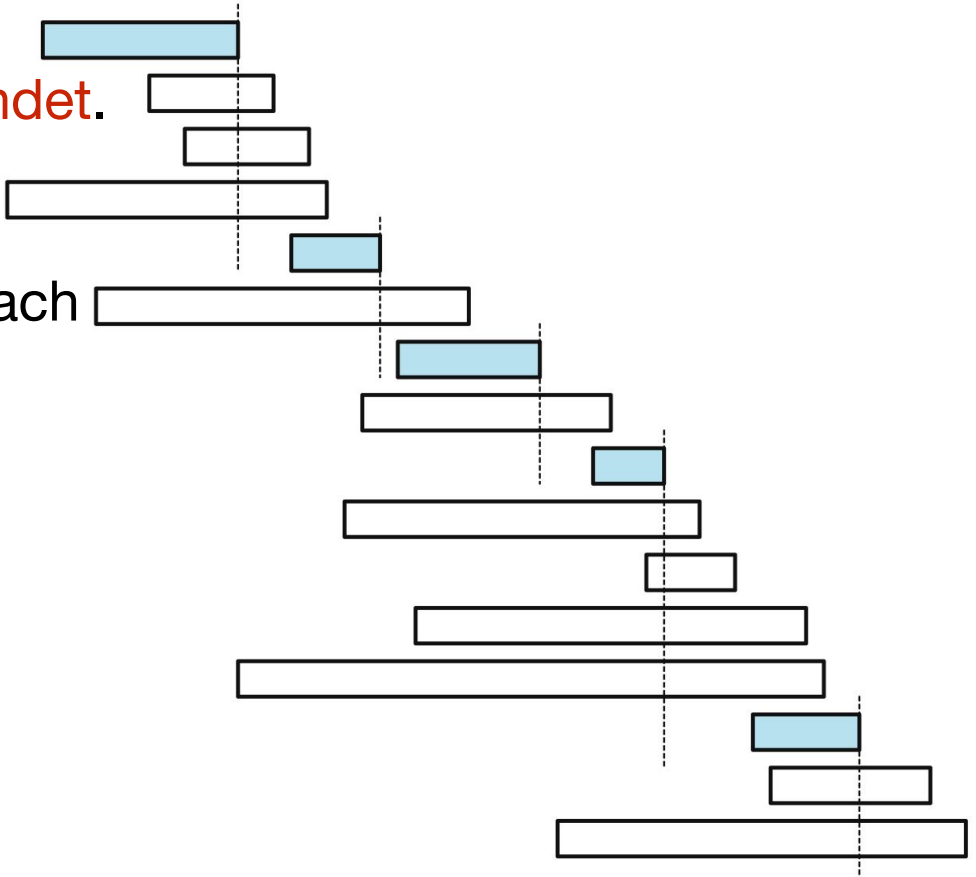


- Übung. Dynamische Programmierung braucht Zeit $O(n^3)$.
- Geht's besser?

Gierige Intuition

- Idee. Wir belegen die Vorlesung, die als erstes endet.

- Gieriger Algorithmus.
  - Sortiere Vorlesungen nach ihrer Endzeit.
  - Belege die nächste konfliktfreie Vorlesung.
  - Wiederhole



Pseudocode

```
GreedySchedule(S[1..n], F[1..n]):
sortiere F und permutiere S passend
count ← 1
X[count] ← 1
for i ← 2 to n:
    if S[i] > F[X[count]]:
        count ← count + 1
        X[count] ← i
return X[1..count]
```

Laufzeit.

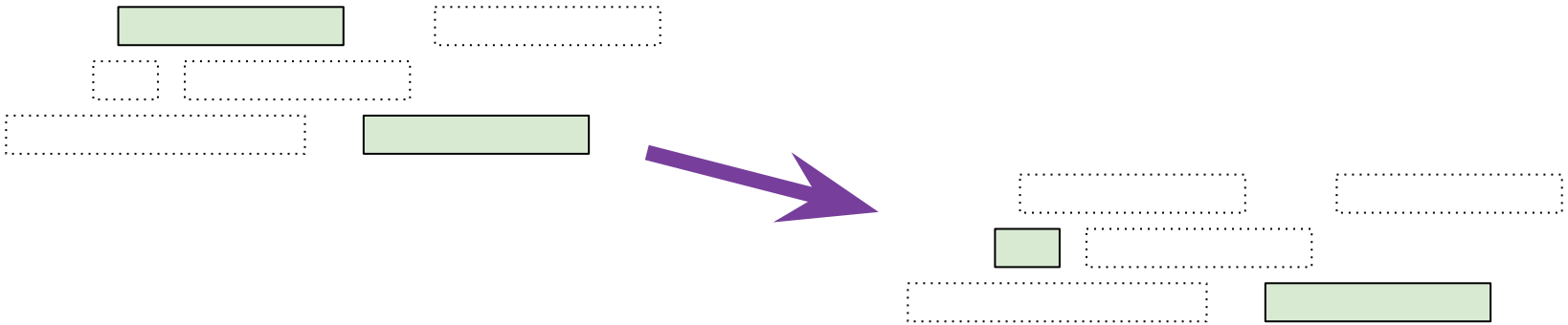
- Sortieren
  in O(n log n) Zeit.
- Rest in O(n) Zeit.
- Gesamt: **O(n log n)**

Korrektheit

- **Warum** ist `GreedySchedule` korrekt??!!

Lemma. Mindestens ein optimaler Stundenplan enthält die Vorlesung, die als erstes endet.

Beweisidee: Falls sie nicht belegt wird, können wir sie anstatt der ersten belegten Vorlesung einfügen (denn sie endet früher):

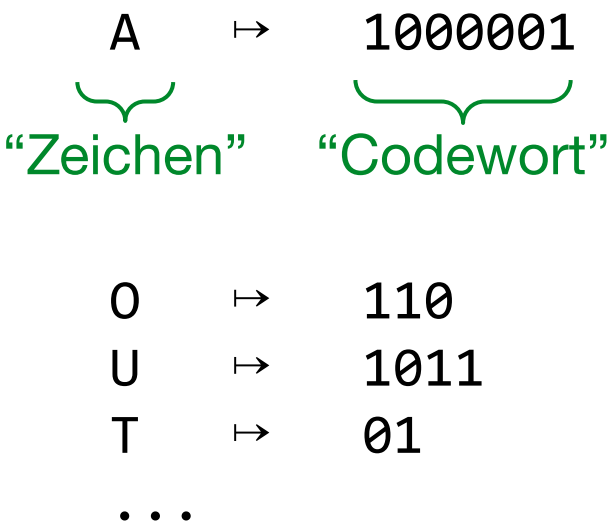


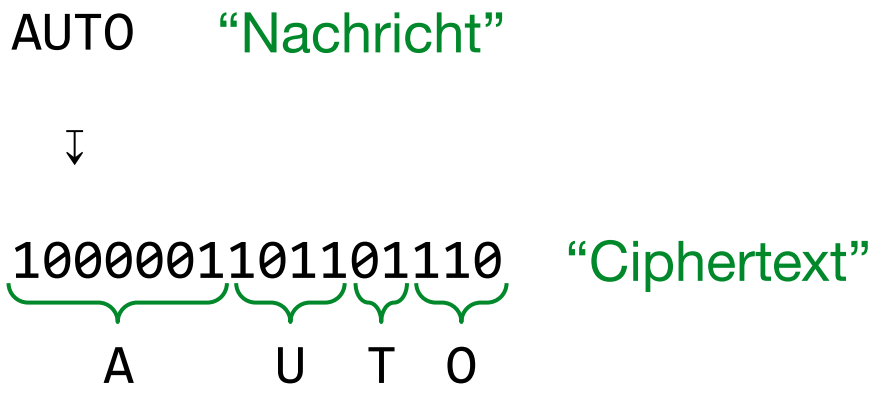
Gierige Algorithmen

- Dateien auf Band
- Scheduling
- Huffman Codes
- Stabiles Matching

Binäre Codes

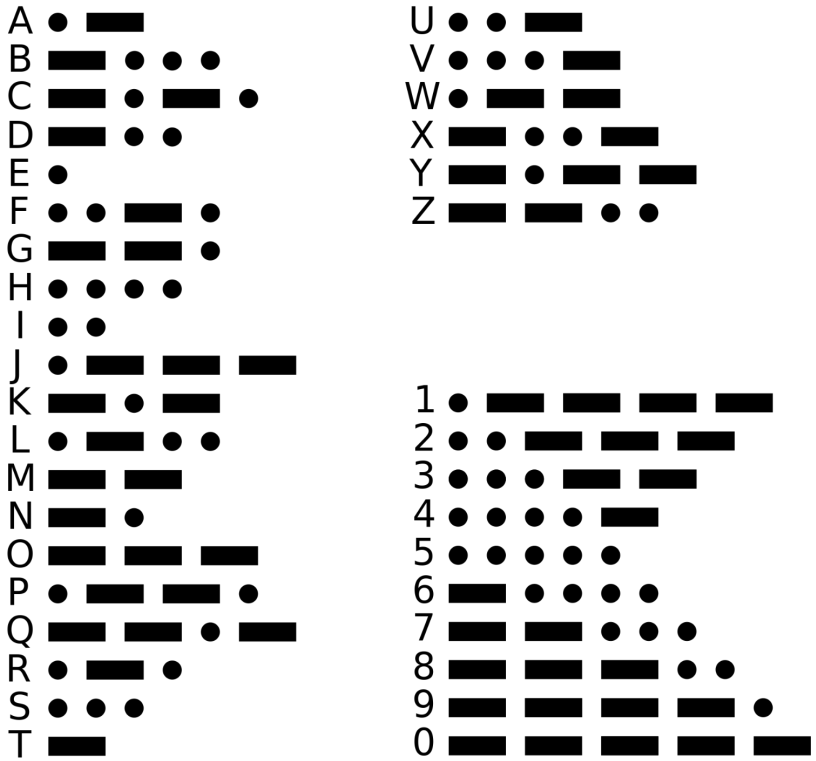
- Jedem Zeichen wird ein binäres Codewort zugewiesen.





International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



7-Bit ASCII

Binary	Oct	Dec	Hex	Glyph		
				1963	1965	1967
010 0000	040	32	20	space		
010 0001	041	33	21	!		
010 0010	042	34	22	"		
010 0011	043	35	23	#		
010 0100	044	36	24	$		
010 0101	045	37	25	%		
010 0110	046	38	26	&		
010 0111	047	39	27	'		
010 1000	050	40	28	(		
010 1001	051	41	29	)		
010 1010	052	42	2A	*		
010 1011	053	43	2B	+		
010 1100	054	44	2C	,		
010 1101	055	45	2D	-		
010 1110	056	46	2E	.		
010 1111	057	47	2F	/		
011 0000	060	48	30	0		
011 0001	061	49	31	1		
011 0010	062	50	32	2		
011 0011	063	51	33	3		

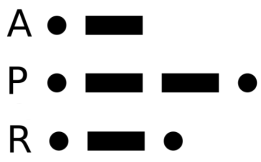
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T

https://en.wikipedia.org/wiki/ASCII

Präfixfreie Codes

- Ein Code ist **präfixfrei**, wenn kein Codewort Präfix eines anderen Codewort ist.

- Morse Code ist **nicht** präfixfrei

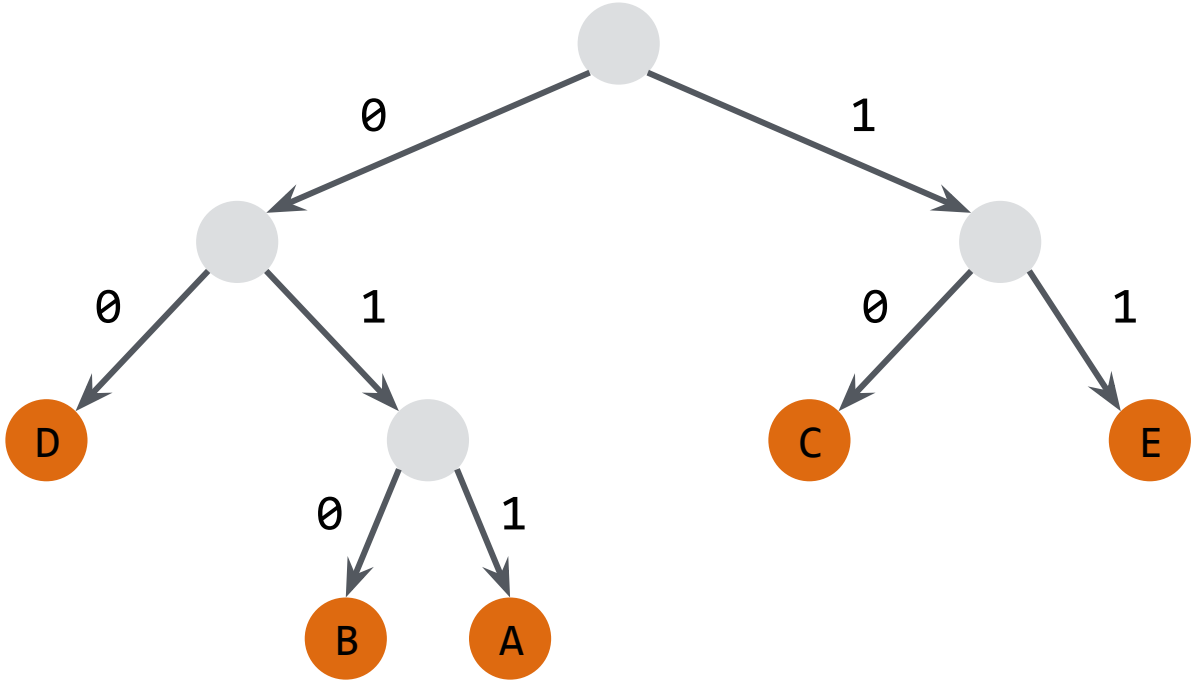


- Übung. 7-Bit ASCII ist präfixfrei
- UTF-8 ist präfixfrei

Präfixfreie Codes in Baumdarstellung

- Jeder präfixfreie binäre Code kann als binärer Baum dargestellt werden.

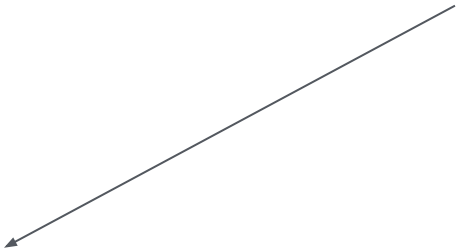
```
A ↦ 011
B ↦ 010
C ↦ 10
D ↦ 00
E ↦ 11
```



Ein Satz von Sallow

This sentence contains three a’s, three c’s, two d’s, twenty-six e’s, five f’s, three g’s, eight h’s, thirteen i’s, two l’s, sixteen n’s, nine o’s, six r’s, twenty-seven s’s, twenty-two t’s, two u’s, five v’s, eight w’s, four x’s, five y’s, and only one z.

```
THISSENTENCECONTAINSTHREEASTHREECS
TWODSTWENTYSIXESFIVEFSTHREEGSEIGHT
HSTHIRTEENISTWOLSSIXTEENNSNINEOSSI
XRSTWENTYSEVENSSTWENTYTWOTSTWOUSFI
VEVSEIGHTWSFOURXSFIVEYSANDONLYONEZ
```



A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

Optimale Codes

- Ziel. Text so kurz wie möglich codieren!

- Idee. Kurze Codeworte für häufige Zeichen.

- Häufigkeit. i-tes Zeichen im Alphabet kommt f[i] mal vor

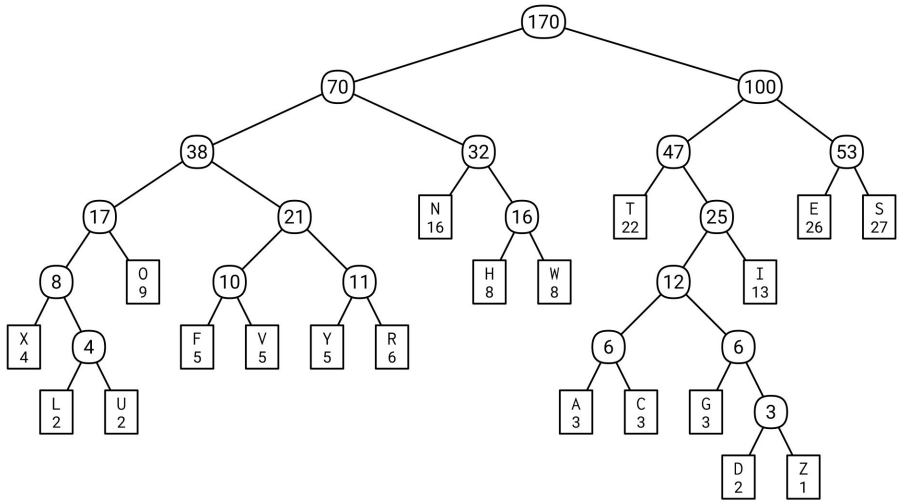
- Optimaler Code. Binärbaum mit n Blättern,
  der die **Code-Länge** minimiert:

$$\sum_{i=1}^{n} f[i] \cdot depth(i)$$

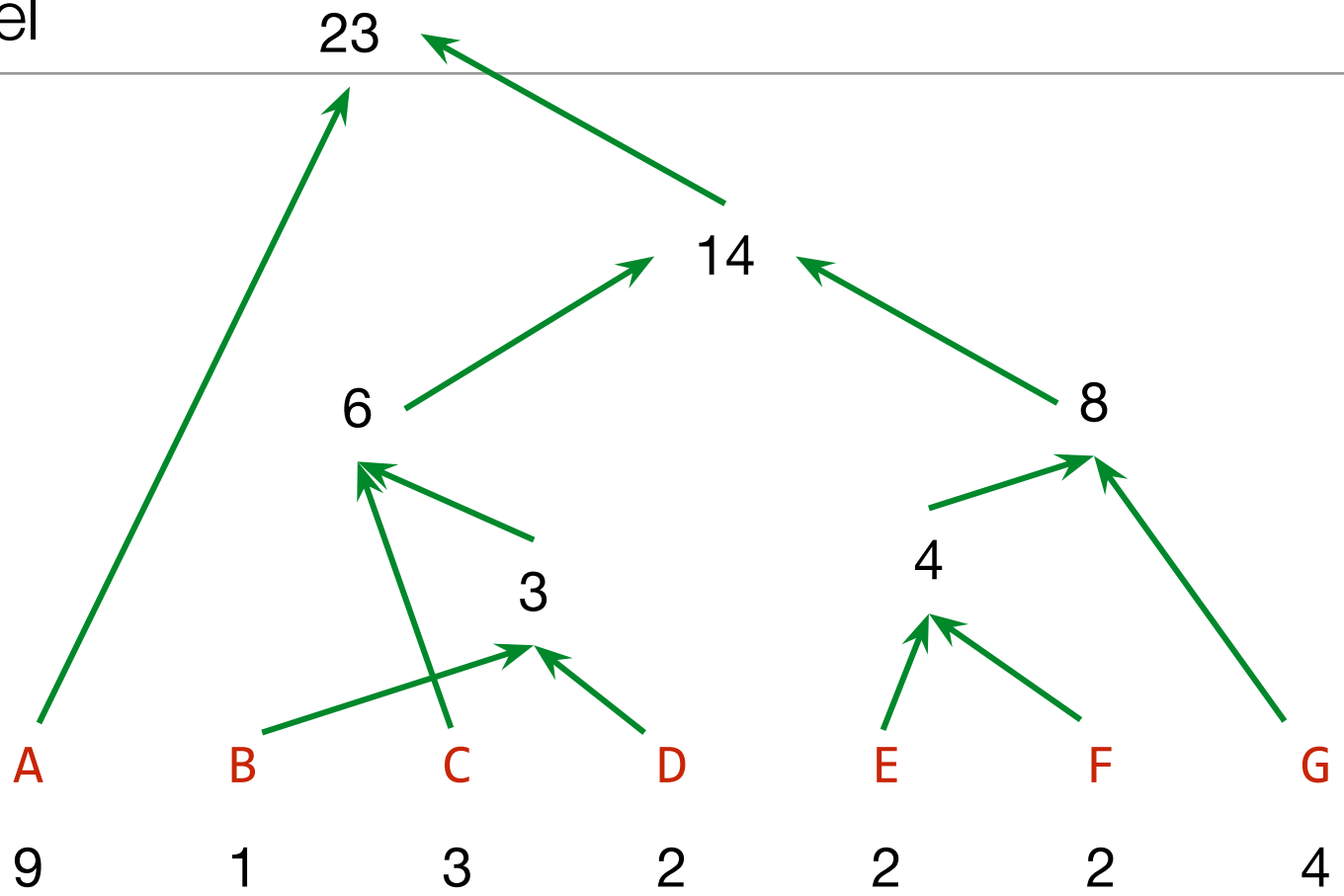
Huffman-Codierung

- Algorithmus. Verschmelze die zwei seltensten Buchstaben und gehe in Rekursion.

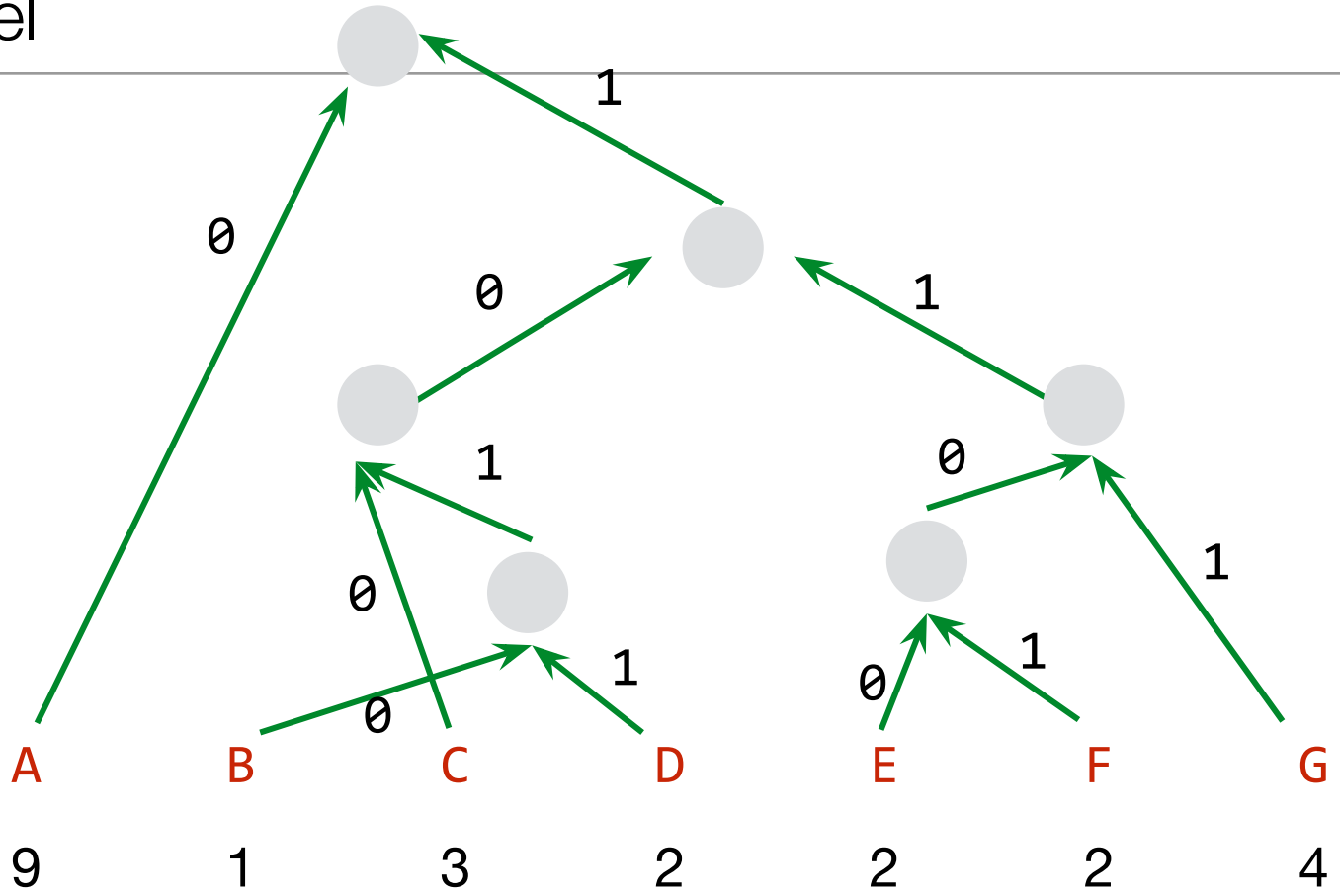
A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1



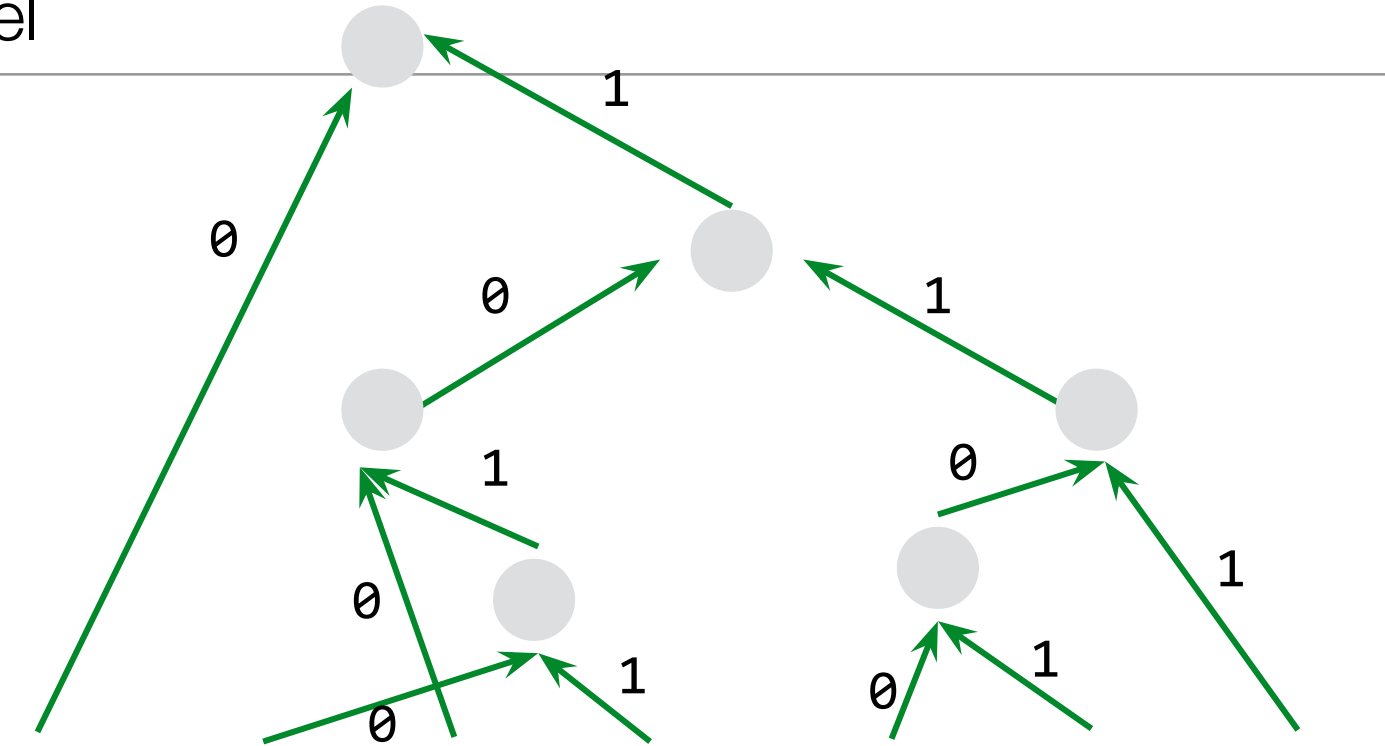
Beispiel



Beispiel



Beispiel

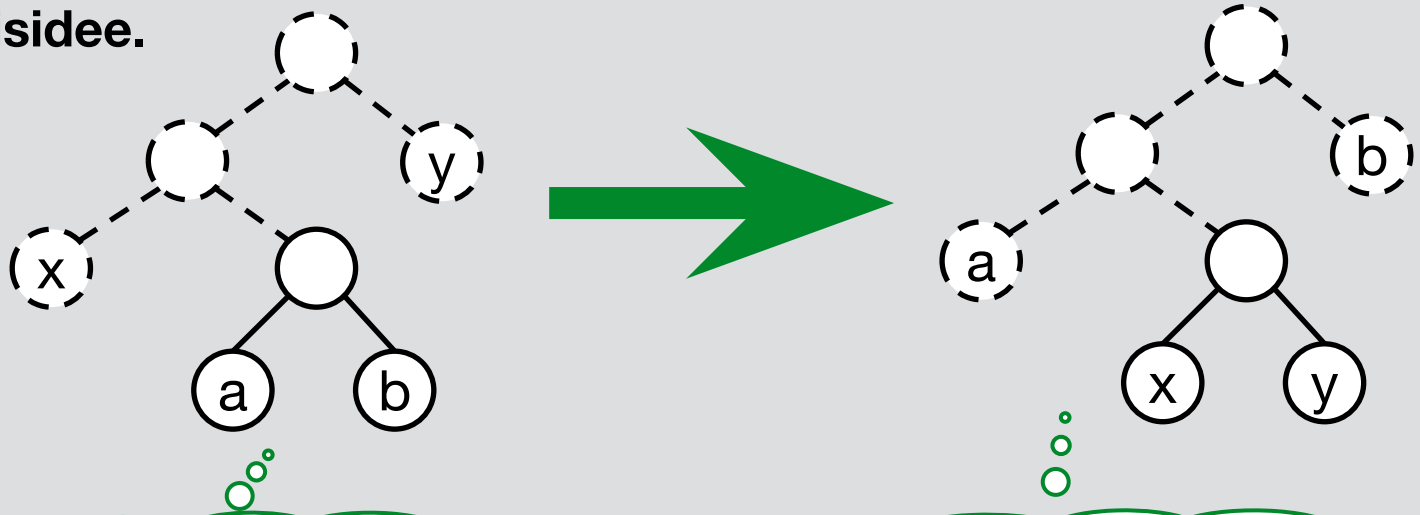


Korrektheit

- Frage. Warum ist Huffmans Algorithmus korrekt?

Lemma. Wenn x und y die zwei seltensten Zeichen sind, gibt es einen optimalen Codebaum, in dem x und y Geschwister sind.

Beweisidee.



Pseudocode

- Eingabe. f[1..n] für die Häufigkeiten
- Ausgabe. L[1..2n], R[1..2n] für linkes/rechtes Kind und P[1..2n] für Eltern
- Nutzen Prioritätswarteschlange, um seltenste Zeichen zu finden.

```
BuildHuffman(f[1..n]):
  for i ← 1 to n
      L[i] ← 0;  R[i] ← 0
      Insert(i, f[i])
  for i ← n to 2n − 1
      x ← ExtractMin( )      ⟨⟨find two rarest symbols⟩⟩
      y ← ExtractMin( )
      f[i] ← f[x] + f[y]     ⟨⟨merge into a new symbol⟩⟩
      Insert(i, f[i])
      L[i] ← x;  P[x] ← i    ⟨⟨update tree pointers⟩⟩
      R[i] ← y;  P[y] ← i
  P[2n − 1] ← 0
```

Laufzeit.

Mit Min-Heap: O(n log n)

Platz. O(n)

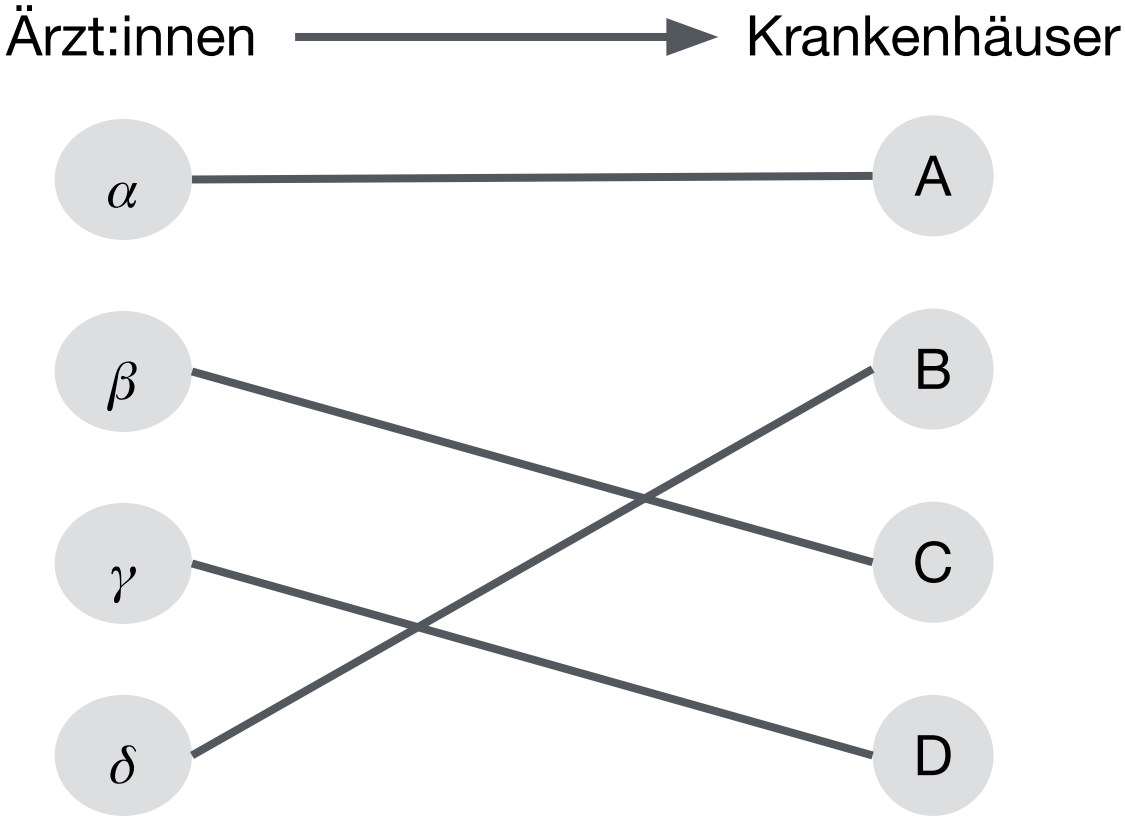
Gierige Algorithmen

- Dateien auf Band
- Scheduling
- Huffman Codes
- Stabiles Matching

National Resident Matching Program (NRMP)

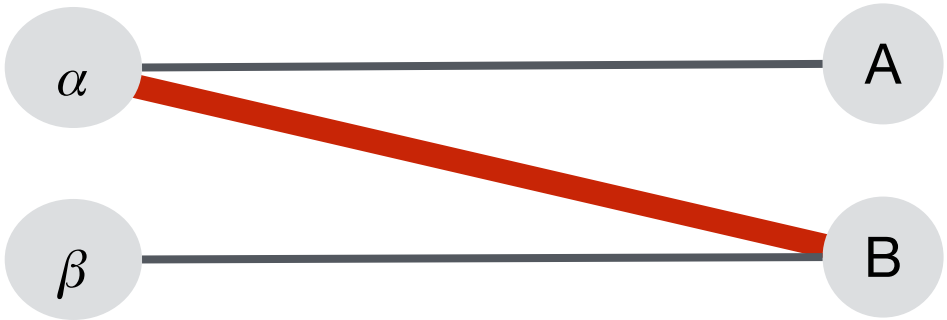
- Eingeführt 1952 in USA.
- Frage. Welches Krankenhaus stellt welche angehende Ärzt:in ein?
- Angehende Ärzt:innen geben Präferenzliste für Krankenhäuser an
- Krankenhäuser geben Präferenzliste für angehende Ärzt:innen an
- NRMP berechnet ein **stabiles Matching.**

Stabiles Matching



Stabiles Matching

Def. Ein Matching ist **stabil**, wenn es kein instabiles Paar gibt.



αB ist ein **instabiles Paar**, wenn

- α ist mit A gematcht, aber bevorzugt B, und
- B ist mit β gematcht, aber bevorzugt α

Gale-Shapley Algorithmus

- Algorithmisches Problem. Gegeben Präferenzlisten aller Ärzt:innen und aller Krankenhäuser, finde ein stabiles Matching.

- Gale-Shapley Algorithmus.
  - Beliebiges ungematchtes Krankenhaus A macht seiner Topkanditat:in α ein Stellenangebot.
  - Wenn α ungematcht ist:
    α sagt A vorläufig zu.
  - Wenn α zwar mit B gematcht ist, aber A bevorzugt:
    α sagt B ab und A zu.
  - Wenn α mit B gematcht ist und A schlechter findet:
    α sagt A ab.

Gale-Shapley Beispiel

ABCD q

ADCB r

BACD s

DBCA t

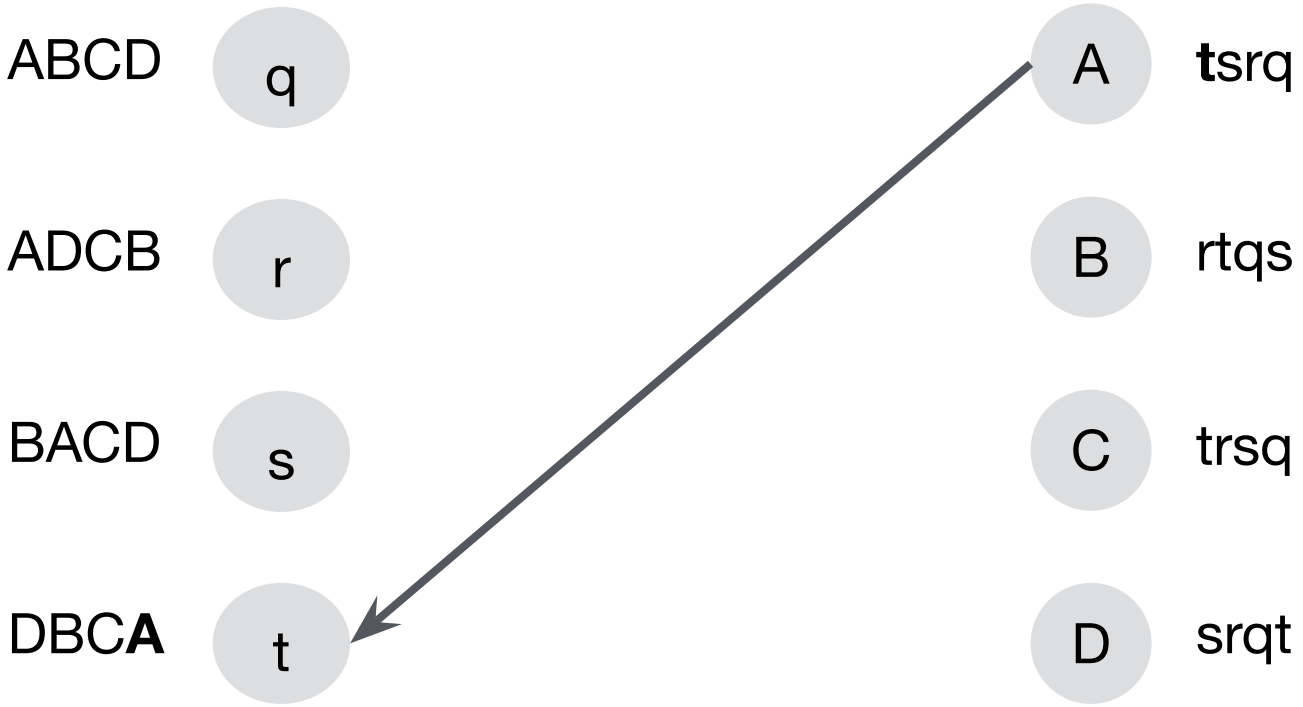
A tsrq

B rtqs

C trsq

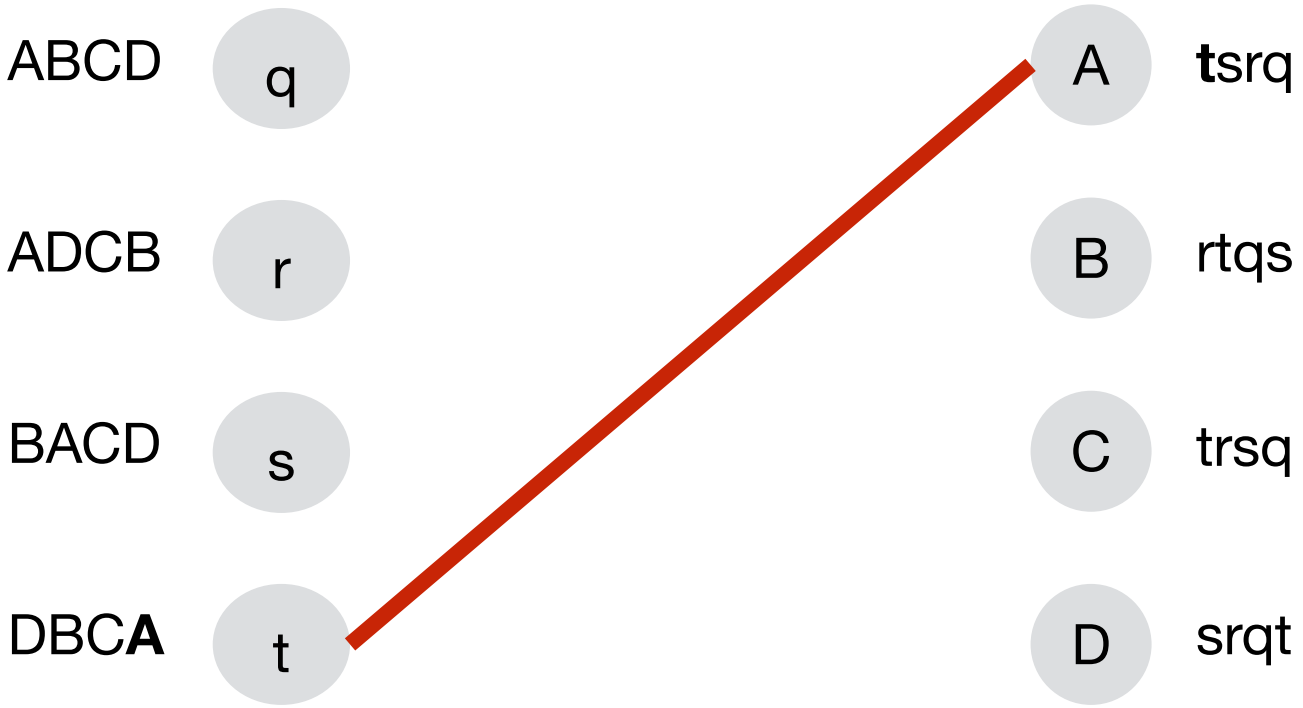
D srqt

Gale-Shapley Beispiel



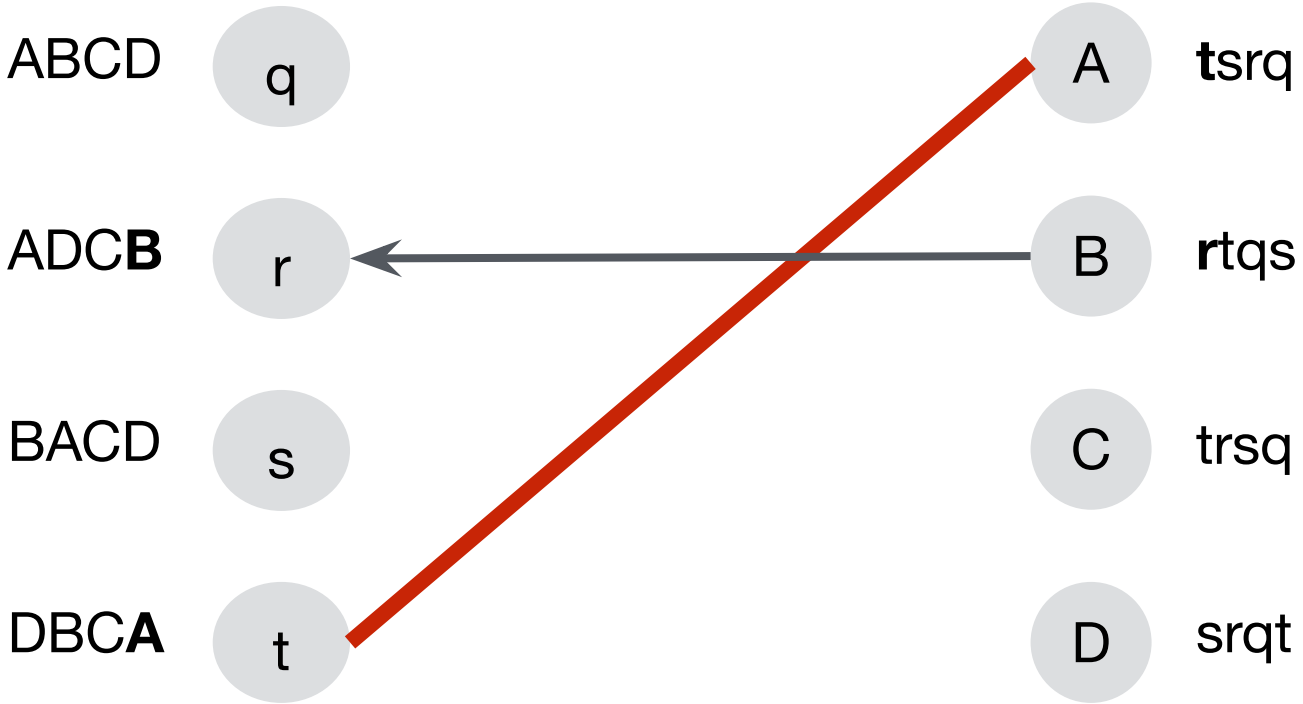
Schritt 1. A macht t ein Angebot

Gale-Shapley Beispiel



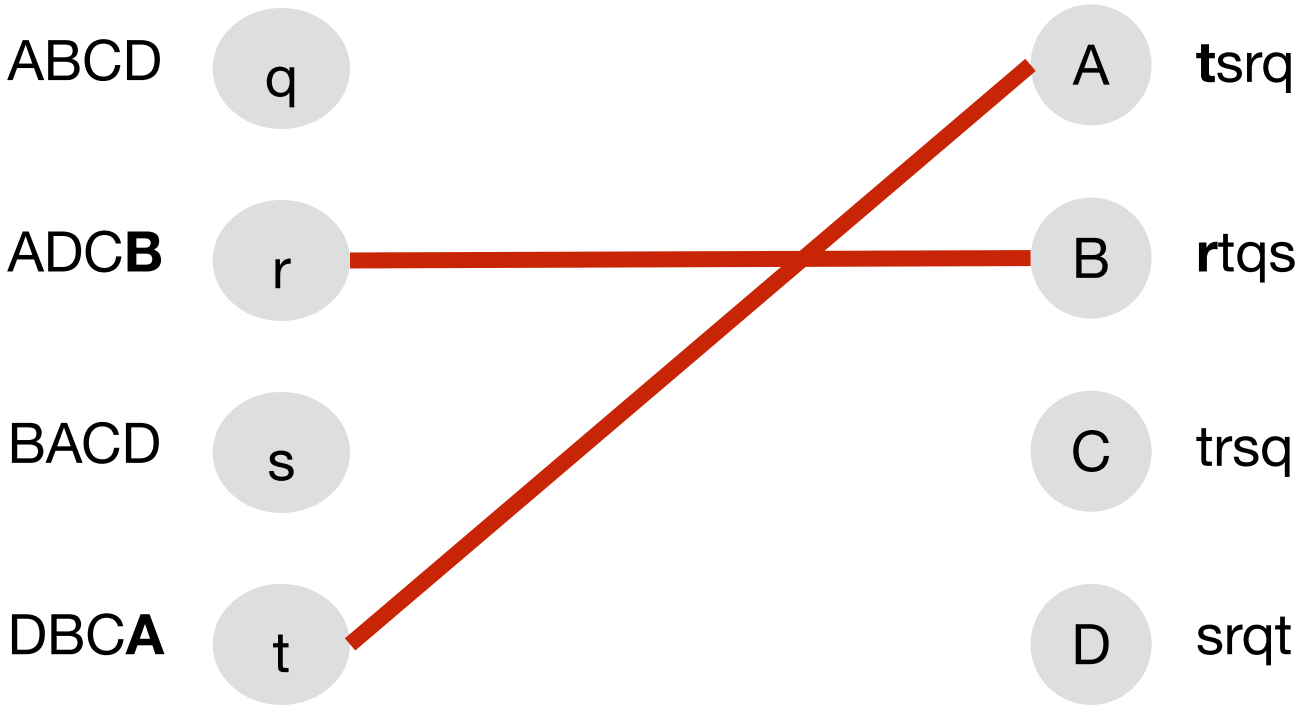
t sagt vorläufig zu.

Gale-Shapley Beispiel



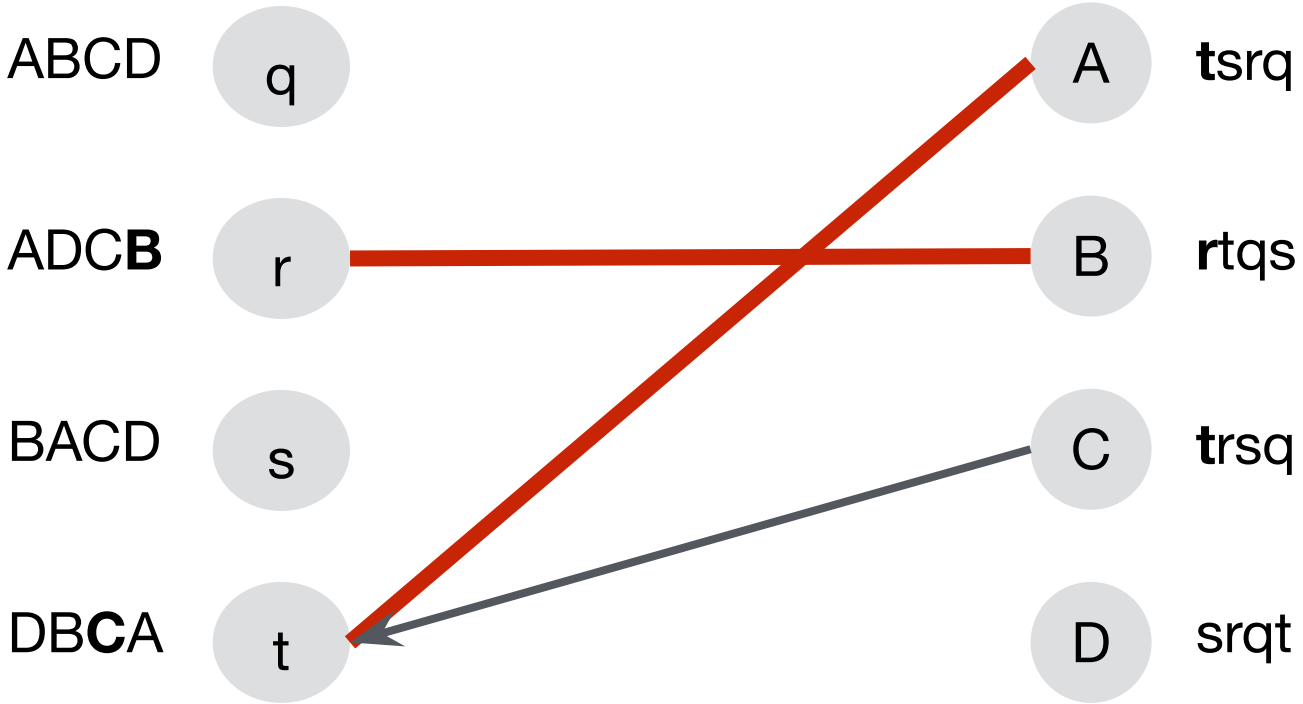
Schritt 2. B macht r ein Angebot

Gale-Shapley Beispiel



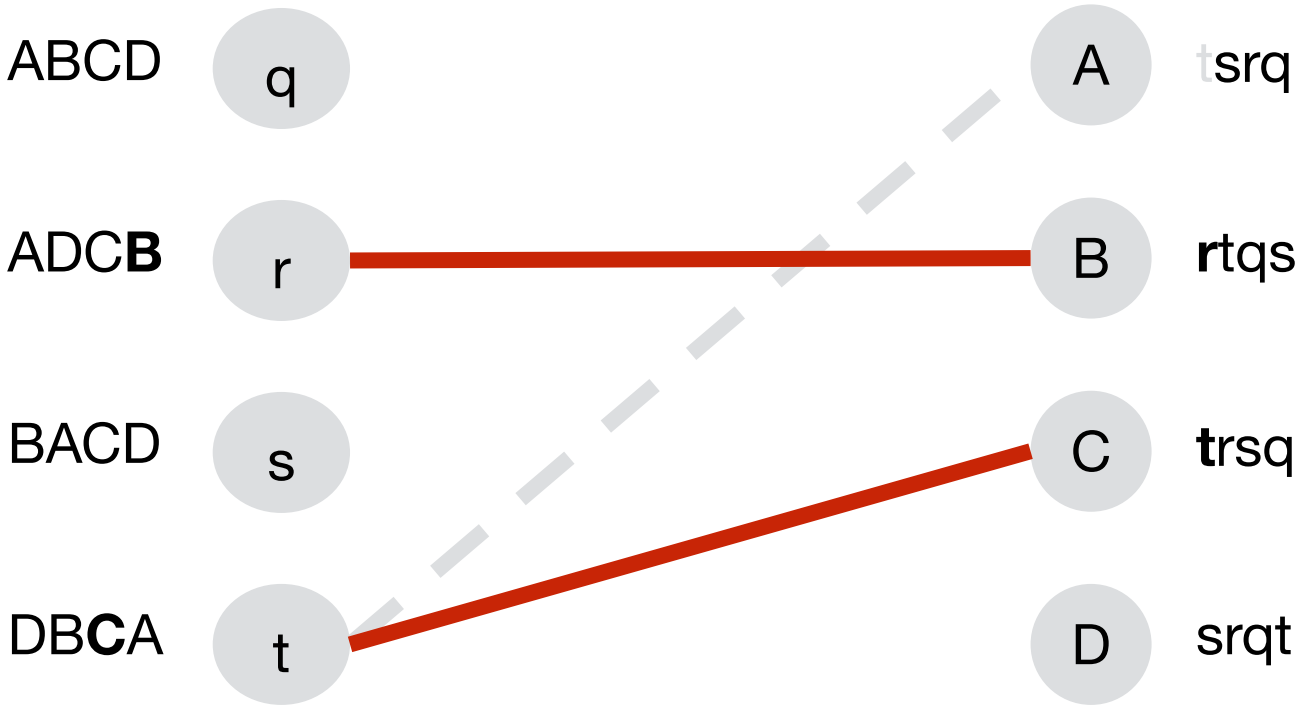
r sagt vorläufig zu.

Gale-Shapley Beispiel



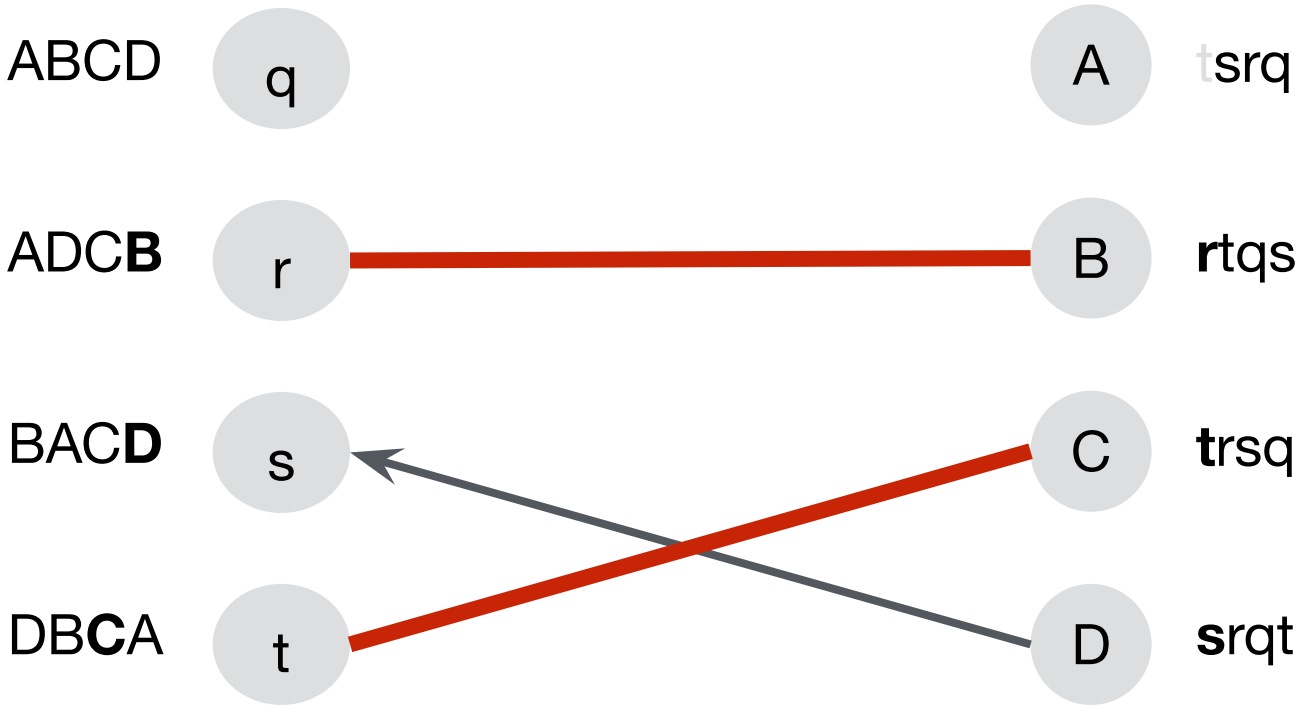
Schritt 3. C macht t ein Angebot.

Gale-Shapley Beispiel



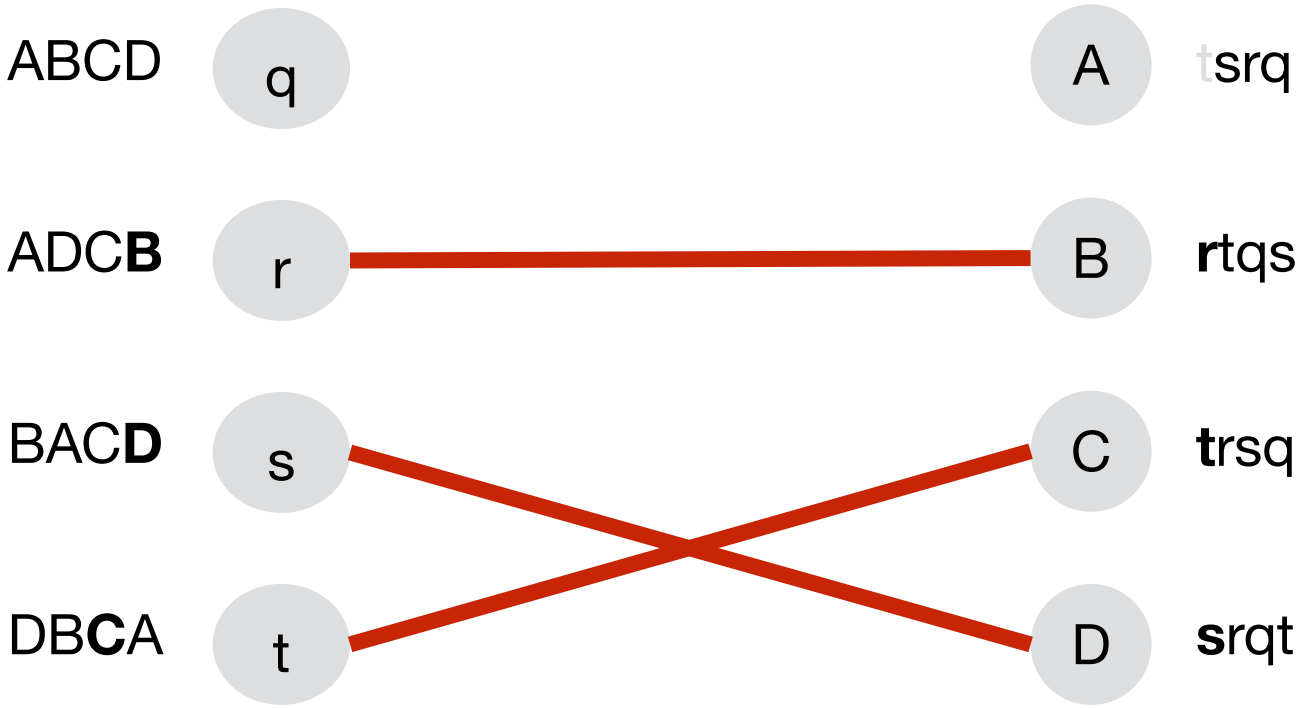
t bevorzugt C über A. Sagt A nun ab und C zu.

Gale-Shapley Beispiel



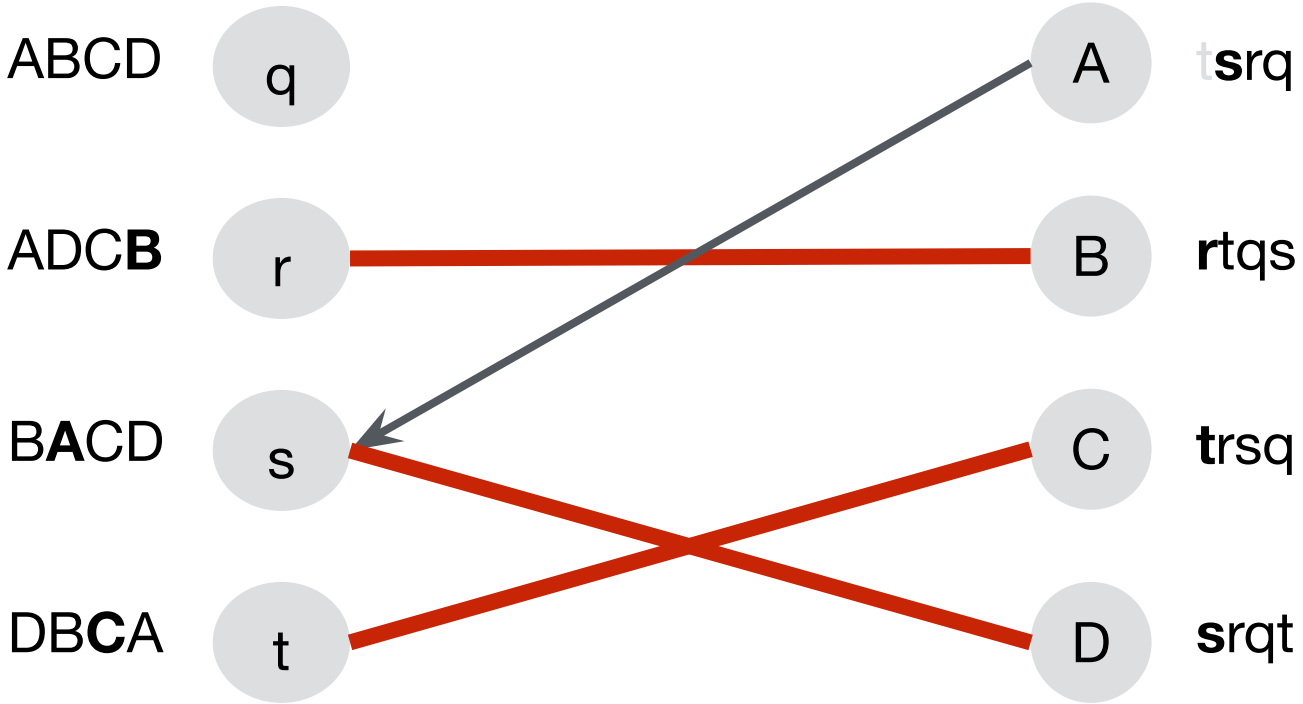
Schritt 4. D macht s ein Angebot.

Gale-Shapley Beispiel



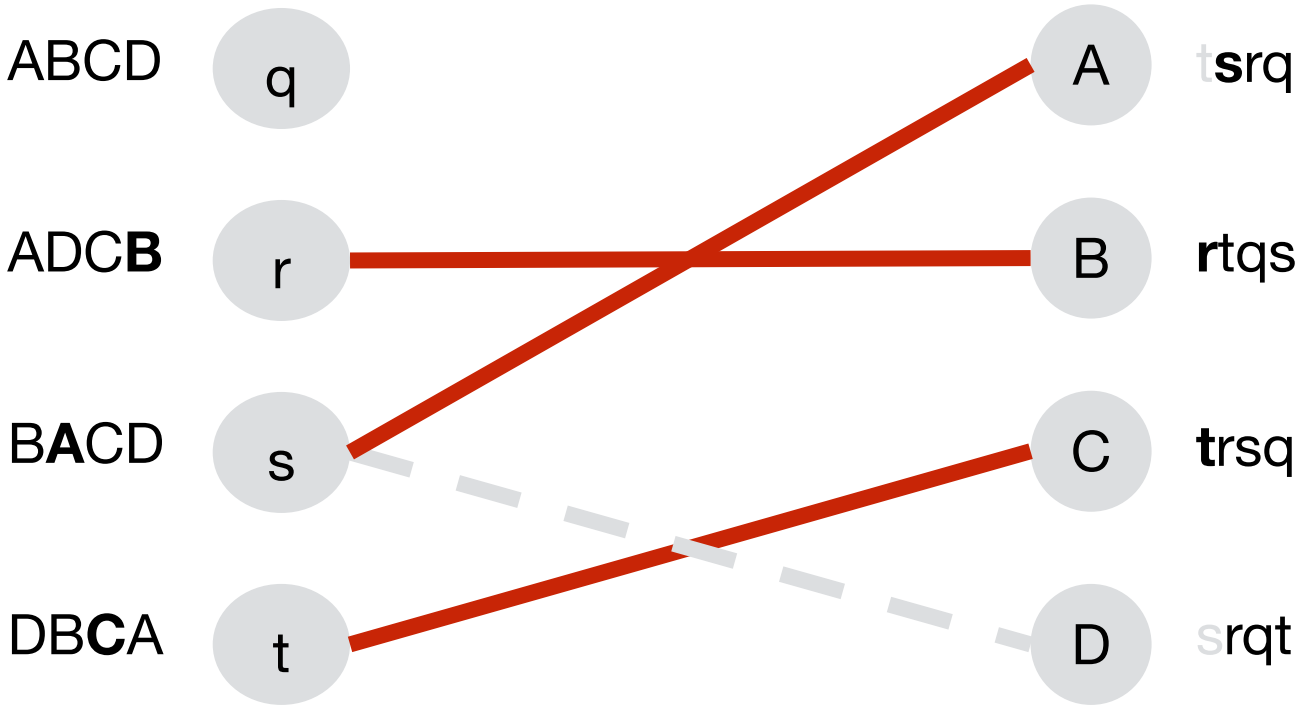
s sagt vorläufig zu.

Gale-Shapley Beispiel



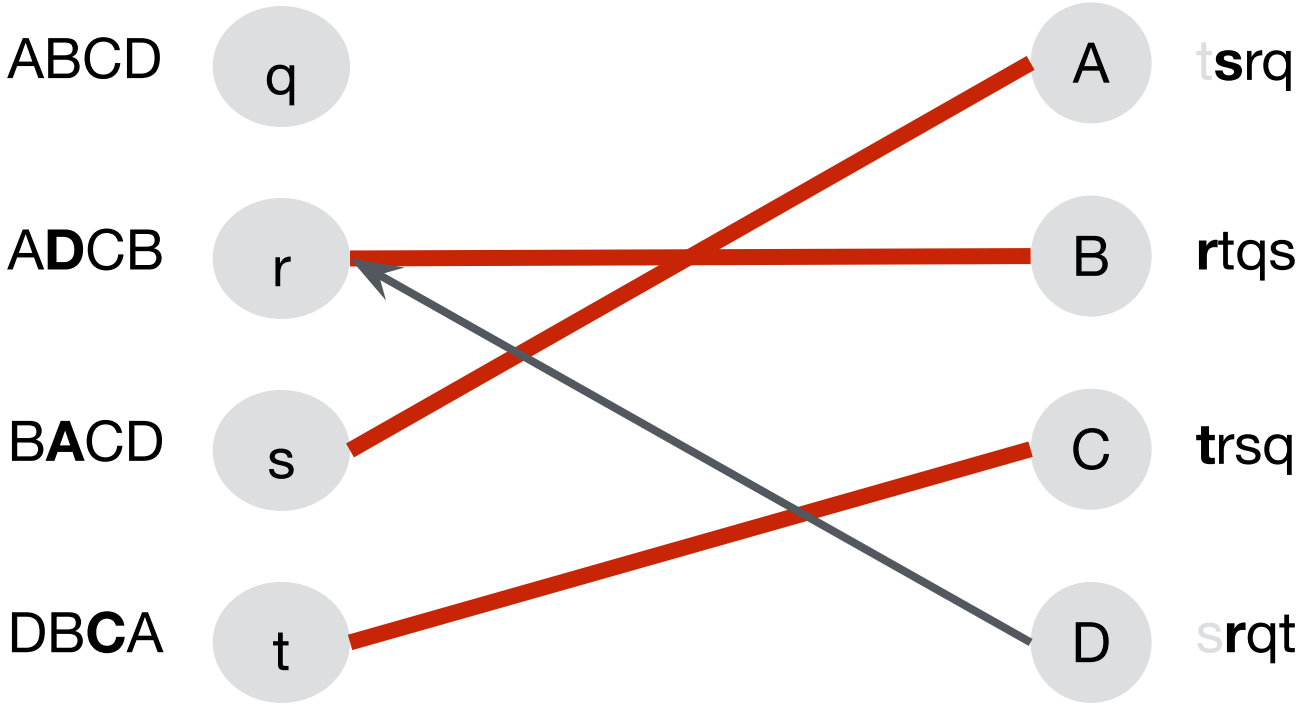
Schritt 5. A macht s ein Angebot

Gale-Shapley Beispiel



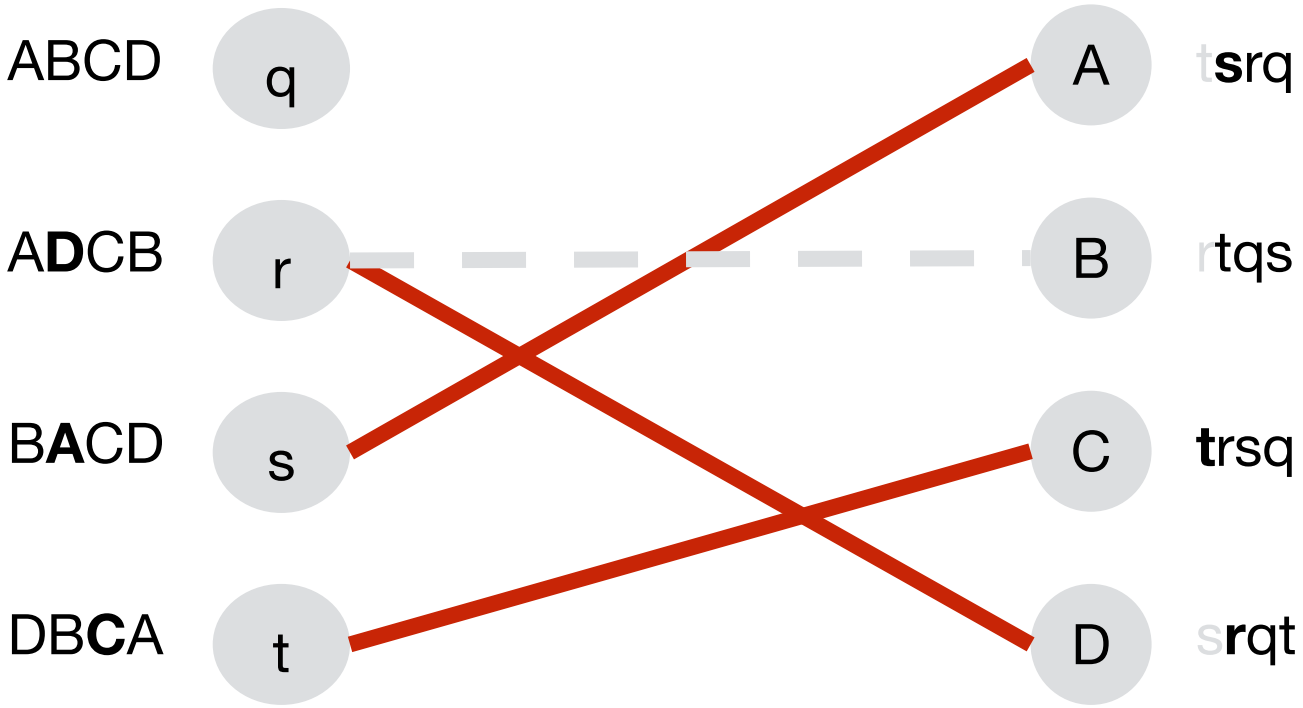
s bevorzugt A und sagt D ab.

Gale-Shapley Beispiel



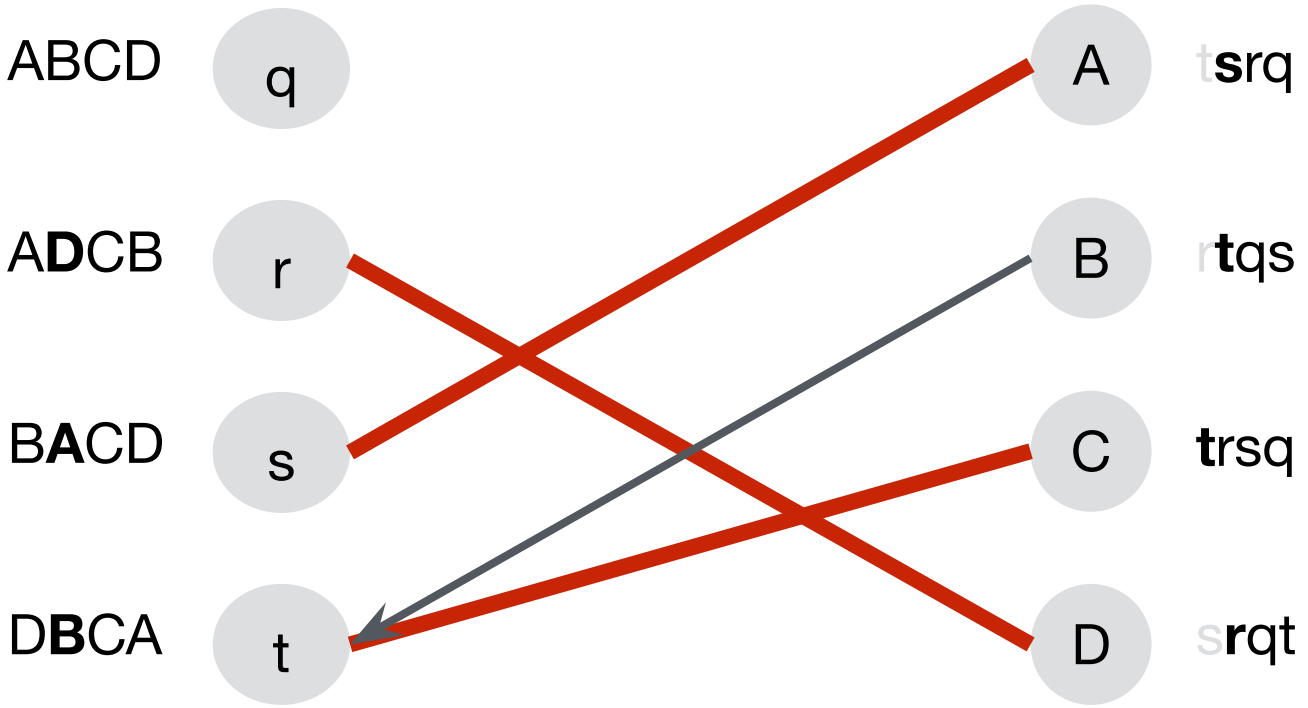
Schritt 6. D macht r ein Angebot

Gale-Shapley Beispiel



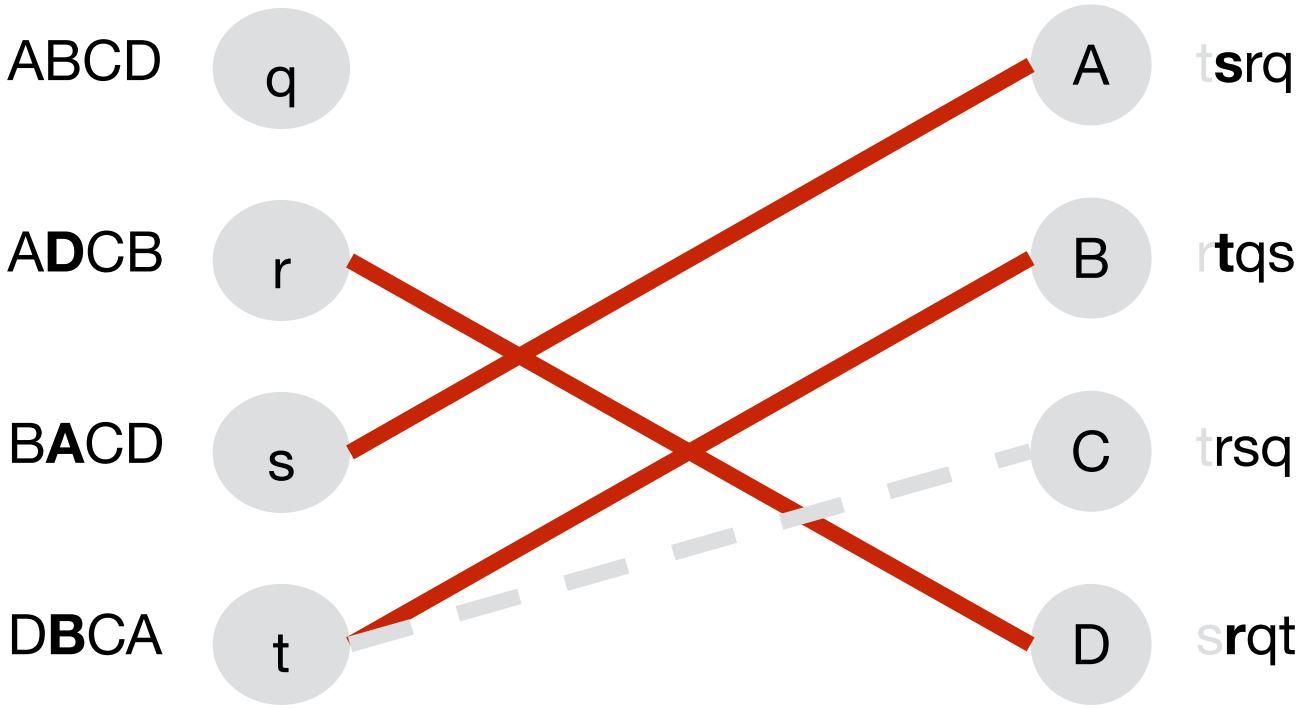
r bevorzugt D und sagt B ab.

Gale-Shapley Beispiel



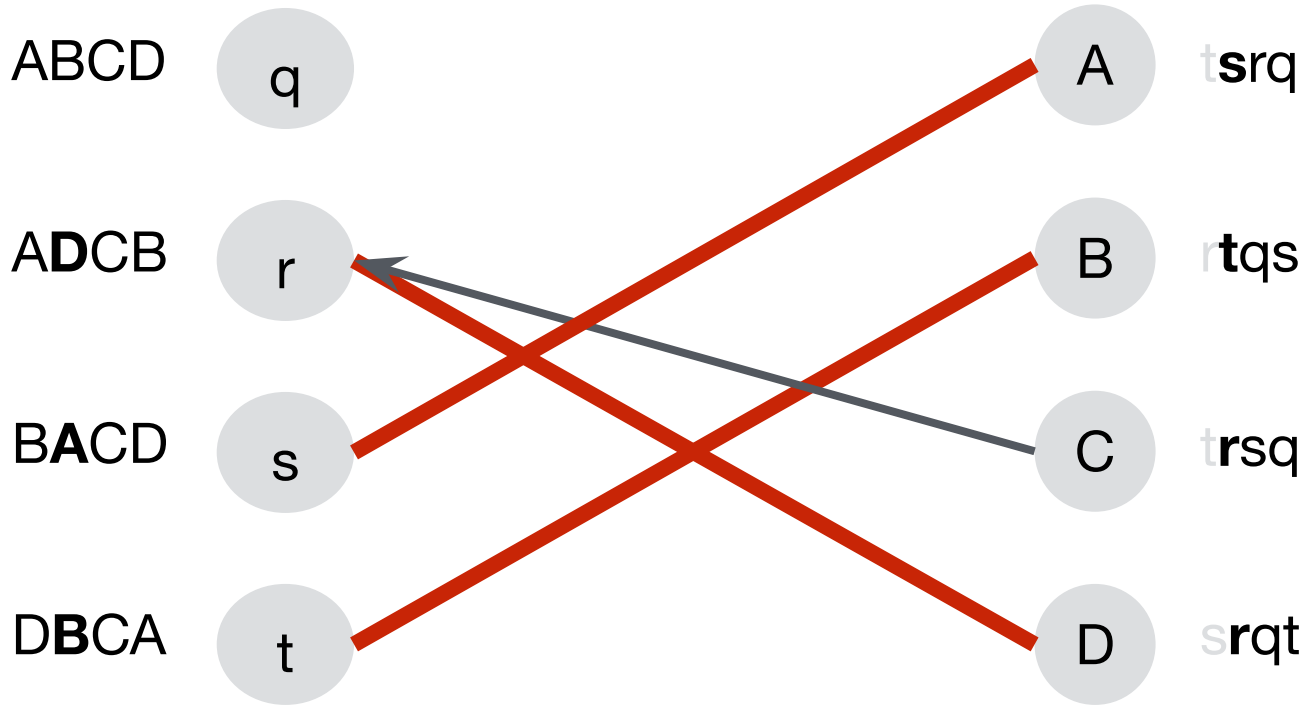
Schritt 7. B macht t ein Angebot

Gale-Shapley Beispiel



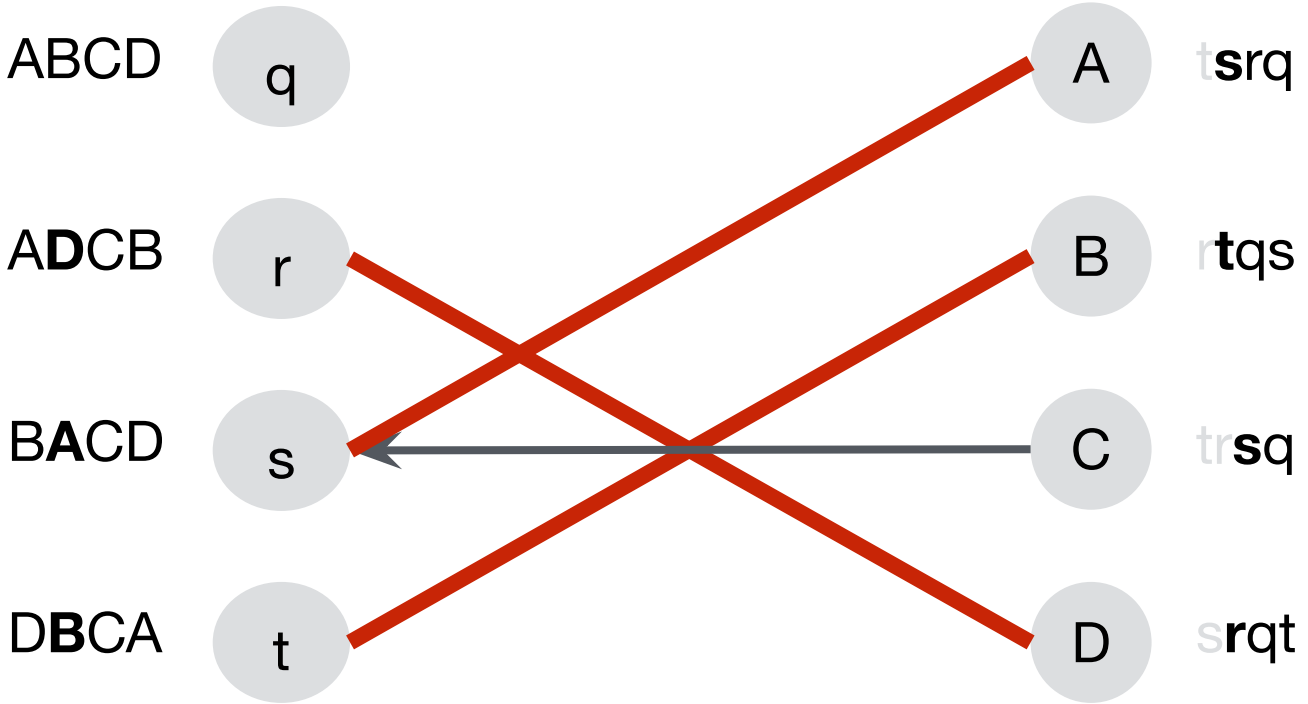
t bevorzugt B und sagt C ab.

Gale-Shapley Beispiel



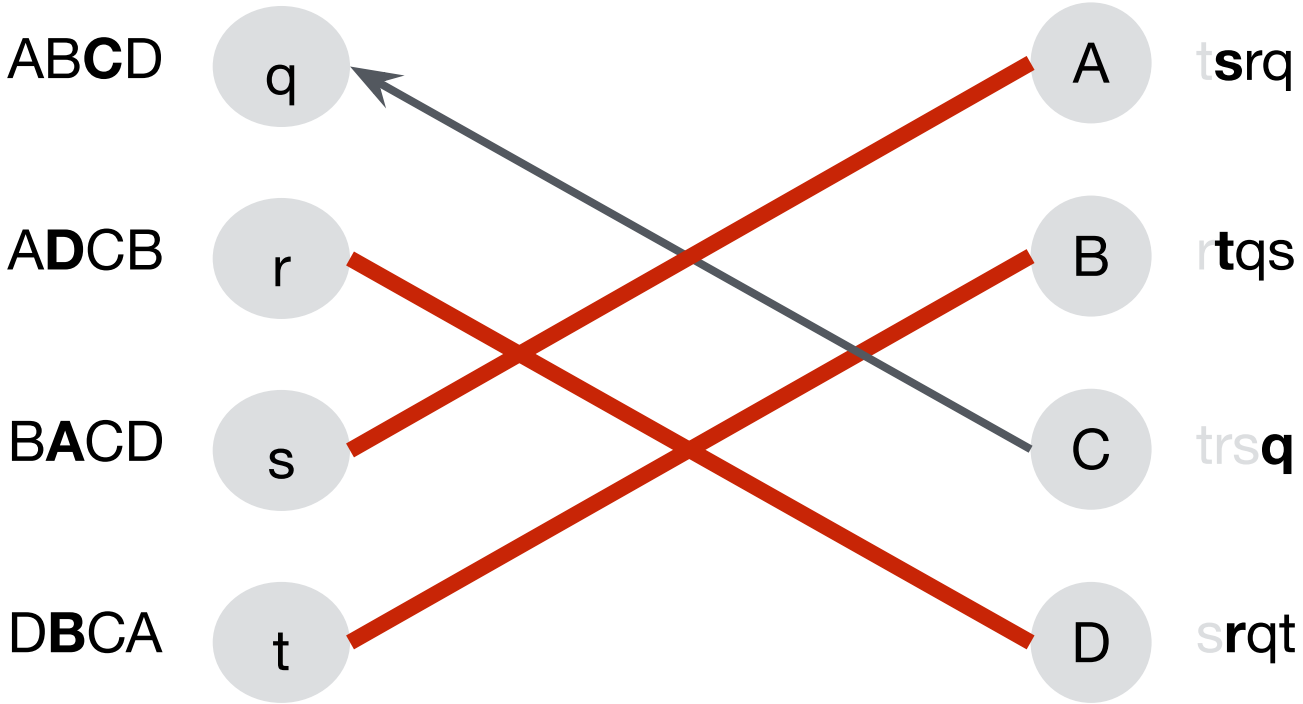
Schritt 8. C macht r ein Angebot, aber r lehnt ab

Gale-Shapley Beispiel



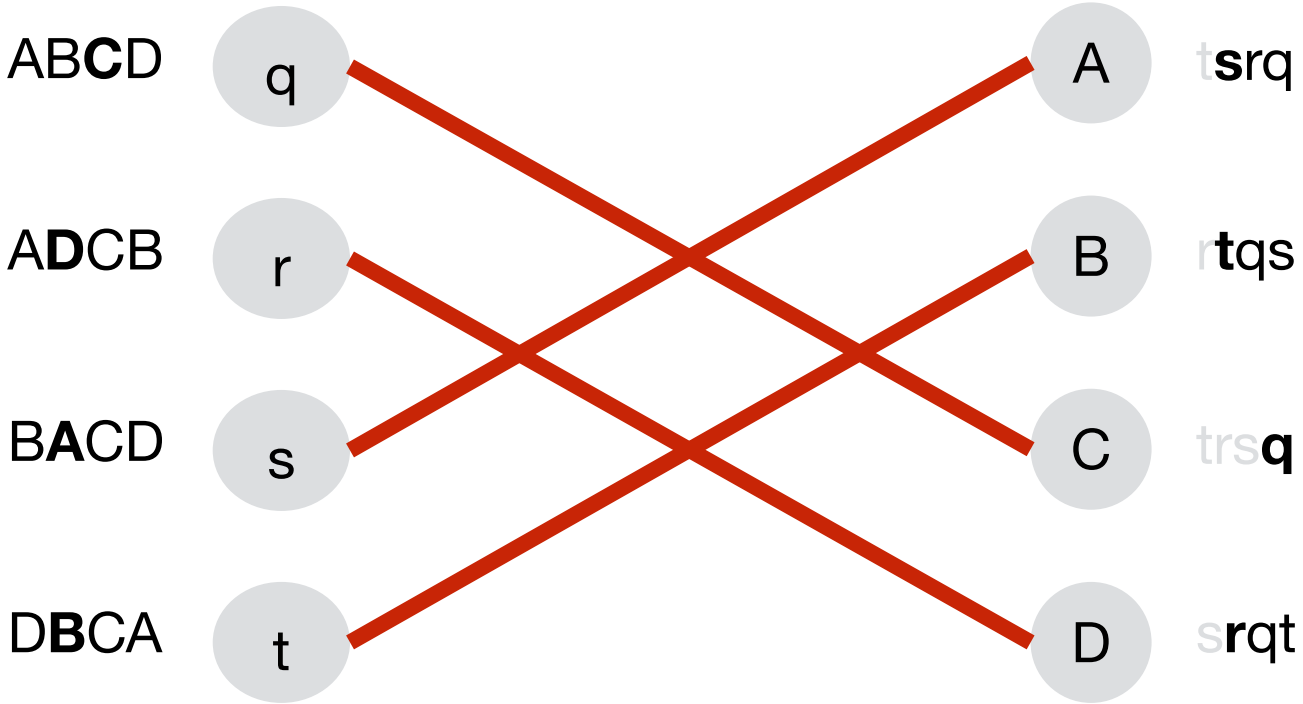
Schritt 9. C macht s ein Angebot, aber s lehnt ab

Gale-Shapley Beispiel



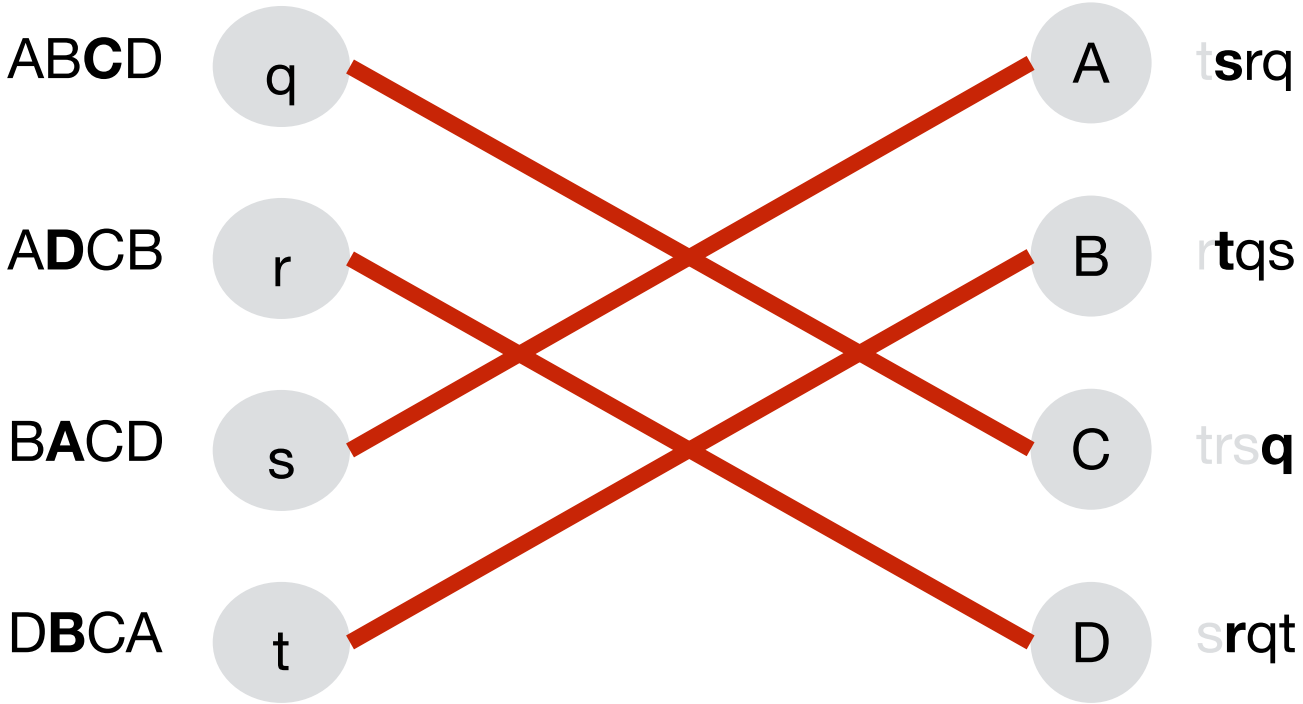
Schritt 9. C macht q ein Angebot

Gale-Shapley Beispiel



q nimmt das Angebot von C an.

Gale-Shapley Beispiel



Alle gematcht ⇒ Gale-Shapley Algorithmus terminiert.

Laufzeit des Gale-Shapley Algorithmus

- n Ärzt:innen und n Krankenhäuser.
- Beobachtung.
  Jedes Krankenhaus macht höchstens ein Angebot je Ärzt:in.
- ⇒ höchstens n^2 Angebote

- Laufzeit.
  - Hängt davon ab, wie wir Ärzt:innen, Krankenhäuser und Präferenzlisten speichern.
  - Einfache Darstellung.
    - Ärzt:innen 1..n, Krankenhäuser 1..n
    - Präferenzlisten Äpref[1..n][1..n] und Kpref[1..n][1..n]
    - Laufzeit $O(n^2)$

Korrektheit

- Fragen.
  - Berechnet Gale-Shapley immer ein **perfektes** Matching?

    Ja, denn wenn eine Ärzt:in nicht gematcht wäre, dann gäbe es auch ein ungematchtes Krankenhaus, das ihr außerdem kein Angebot gemacht hätte und also seine Präferenzliste nicht abgearbeitet hat.

  - Ist das berechnete Matching immer **stabil**?

    Ja, denn es gibt kein Instabiles Paar αB: Wenn α mit A gematcht ist, aber B bevorzugt, dann ist B mit einer für B besseren Kandidat:in gematcht.

Optimalität

- Eindeutigkeit. Der Gale-Shapley Algorithmus berechnet immer dasselbe stabile Matching, egal in welcher Reihenfolge die Krankenhäuser dran sind.

- Wer macht Angebote? Es macht einen Unterschied, ob Krankenhäuser die Angebote machen oder Ärzt:innen. Wer anbietet ist im Vorteil!

- Seit 1998 hat NRMP den Algorithmus invertiert, sodass Ärzt:innen die Angebote machen.

Andere Anwendungen

- Referendar:innen — Schulen
- Studieninteressierte in zentral organisierten Fächern — Unis
- Client — Server
- Bachelorarbeit-Schreiben-Wollende — Professor:innen

Gierige Algorithmen

- Dateien auf Band
- Scheduling
- Huffman Codes
- Stabiles Matching