technische universität
dortmund

LS 2 /
Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Gierige Algorithmen

Einführendes Beispiel

- Preisausschreiben gewonnen
- 3 Preise auswählen

Mögliche Preise:

- Haus
- Auto
- Topfpflanze
- Reise nach Australien
- Abendessen

Gierige Algorithmen

Einführendes Beispiel

- Preisausschreiben gewonnen
- 3 Preise auswählen

Mögliche Preise:

- Haus 1
- Auto
- Topfpflanze
- Reise nach Australien
- Abendessen

Gierige Algorithmen

Einführendes Beispiel

- Preisausschreiben gewonnen
- 3 Preise auswählen

Mögliche Preise:

- Haus 1
- Auto 2
- Topfpflanze
- Reise nach Australien
- Abendessen

Gierige Algorithmen

Einführendes Beispiel

- Preisausschreiben gewonnen
- 3 Preise auswählen

Mögliche Preise:

- Haus 1
- Auto 2
- Topfpflanze
- Reise nach Australien 3
- Abendessen

Gierige Algorithmen

Was ist unser Ziel?

- Wollen Gewinn maximieren

Wie wollen wir dieses Ziel erreichen?

- In jedem Auswahlschritt wählen wir den teuersten noch nicht gewählten Gewinn

Frage:

- Maximieren wir dadurch den Gesamtgewinn?

Gierige Algorithmen

Was ist unser Ziel?

- Wollen Gewinn maximieren

Wie wollen wir dieses Ziel errei

- In jedem Auswahlschritt wählen wir Gewinn

Frage:

- Maximieren wir dadurch den Gesamtgewinn?

Offensichtlich ja!
Aber wir wollen dies trotzdem formal beweisen, um eine wichtige Beweistechnik kennenzulernen.

Gierige Algorithmen

Formales Problem:

- Eingabe: n unterschiedliche Zahlen C[1,…,n]
- Problem: Wähle k unterschiedliche Zahlen aus C aus, so dass die Summe der Zahlen maximiert wird
- Ausgabe: Feld J[1,…,k], dass die gewählten Zahlen enthält

GierigeAuswahl(C)

1. **for** i ←1 **to** k **do**
2.   J[i] ← die größte übrige Zahl aus C
3. **return** J

Gierige Algorithmen

Behauptung:

Algorithmus GierigeAuswahl berechnet eine optimale Menge von k Zahlen.

Behauptung:

Algorithmus GierigeAuswahl berechnet eine optimale Menge von k Zahlen.

$a=(a_1, a_2, a_3, a_4)$ Lösung von GierigeAuswahl

$b=(b_1, b_2, b_3, b_4)$ optimale Lösung $b \neq a$

Behauptung:

Algorithmus GierigeAuswahl berechnet eine optimale Menge von k Zahlen.

$a=(a_1,a_2,a_3,a_4)$ Lösung von GierigeAuswahl

$b=(b_1,b_2,b_3,b_4)$ optimale Lösung $b \neq a$

<u>ObdA.:</u>

$a_1 \geq \ldots \geq a_4$

$b_1 \geq \ldots \geq b_4$

Behauptung:

Algorithmus GierigeAuswahl berechnet eine optimale Menge von k Zahlen.

$a=(a_1, a_2, a_3, a_4)$ Lösung von GierigeAuswahl

$b=(b_1, b_2, b_3, b_4)$ optimale Lösung $b \neq a$

Sei j kleinster Index mit $a_j \neq b_j$

Behauptung:

Algorithmus GierigeAuswahl berechnet eine optimale Menge von k Zahlen.

$a=(a_1, a_2, a_3, a_4)$ Lösung von GierigeAuswahl

$b=(b_1, b_2, b_3, b_4)$ optimale Lösung $b \neq a$

Wegen unserer gierigen Strategie, gilt
$a_j > b_j$

Behauptung:

Algorithmus GierigeAuswahl berechnet eine optimale Menge von k Zahlen.

$a=(a_1, a_2, a_3, a_4)$ Lösung von GierigeAuswahl

$b=(b_1, b_2, b_3, b_4)$ optimale Lösung $b \neq a$

Wegen unserer gierigen Strategie, gilt
$a_j > b_j$

Weil die b_i absteigend sortiert sind, wird a_j nicht in Lösung b verwendet.

Behauptung:

Algorithmus GierigeAuswahl berechnet eine optimale Menge von k Zahlen.

$a=(a_1, a_2, a_3, a_4)$ Lösung von GierigeAuswahl

$b=(b_1, b_2, a_3, b_4)$ optimale Lösung $b \neq a$

Ersetze b_j durch a_j.
Dies verbessert die Lösung.
Widerspruch zur Optimalität von b.

Gierige Algorithmen

Gierige Algorithmen

- Konstruiere Lösung Schritt für Schritt
- In jedem Schritt: Optimiere ein einfaches, lokales Kriterium

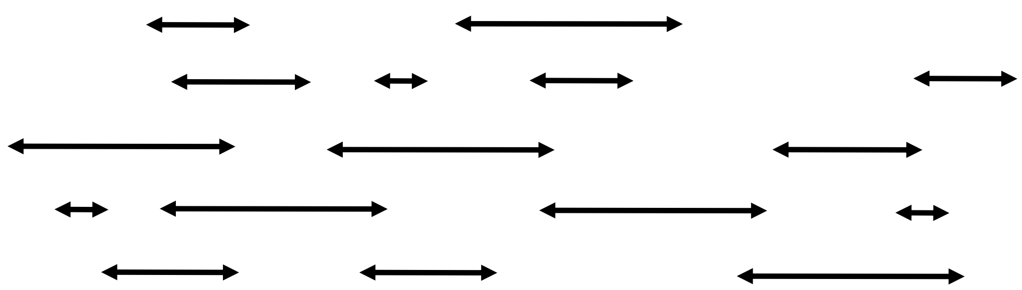
Beobachtung

- Man kann viele unterschiedliche gierige Algorithmen für ein Problem entwickeln
- Nicht jeder dieser Algorithmen löst das Problem korrekt

Gierige Algorithmen

Interval Scheduling

- Ressource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Kann ich die Ressource für den Zeitraum (t_1,t_2) nutzen?

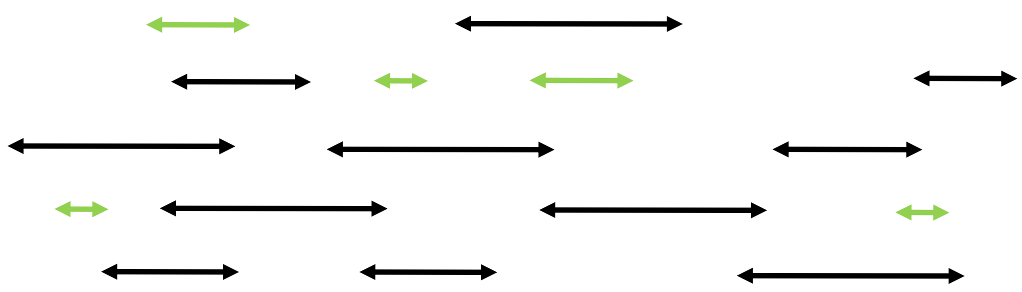


- Ziel: Möglichst viele Anfragen erfüllen

Gierige Algorithmen

Interval Scheduling

- Ressource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Kann ich die Ressource für den Zeitraum (t_1,t_2) nutzen?

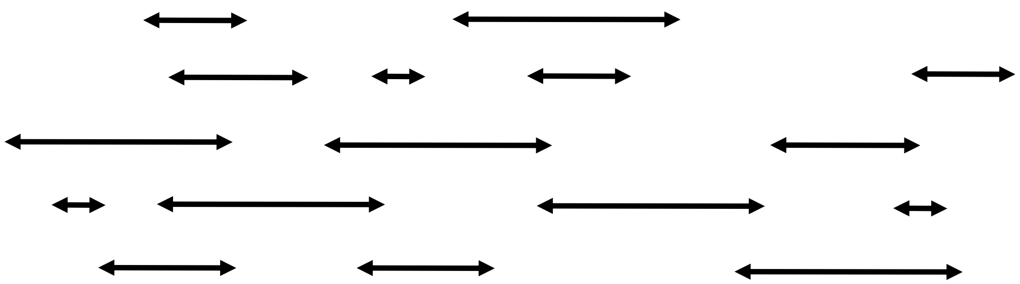


- Ziel: Möglichst viele Anfragen erfüllen

Gierige Algorithmen

Definition

- Zwei Anfragen heißen kompatibel, wenn sich die Intervalle nicht überschneiden.

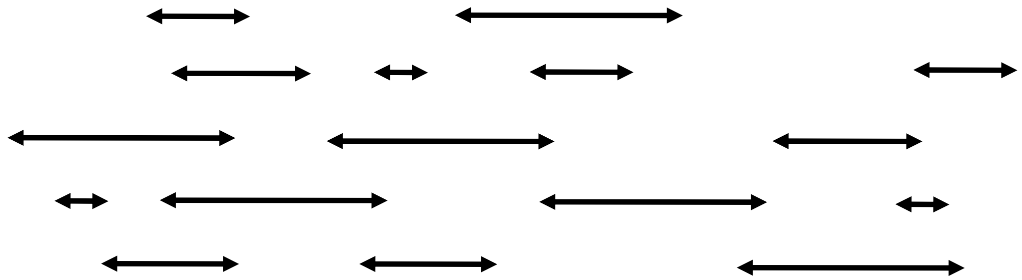


- Ziel: Möglichst viele Anfragen erfüllen

Gierige Algorithmen

Definition

- Zwei Anfragen heißen kompatibel, wenn sich die Intervalle nicht überschneiden.

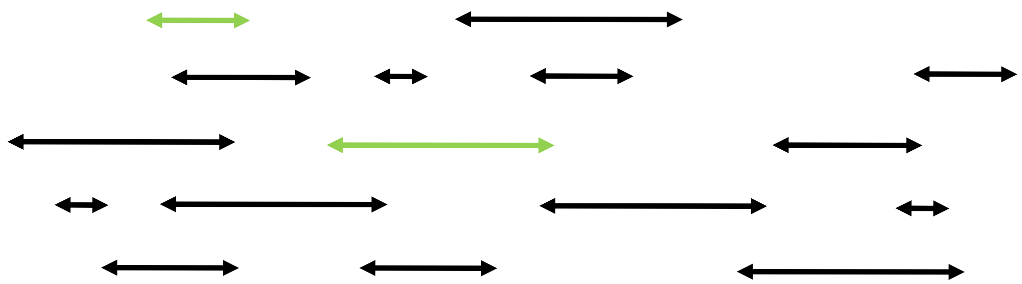


- Ziel: Möglichst viele Anfragen erfüllen

Gierige Algorithmen

Definition

- Zwei Anfragen heißen kompatibel, wenn sich die Intervalle nicht überschneiden.

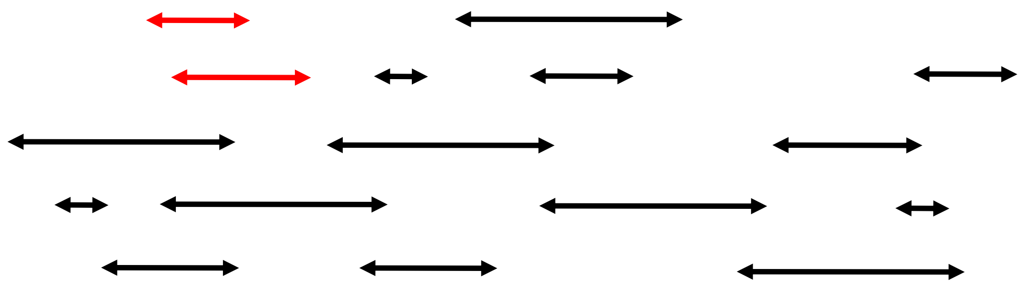


- Kompatibel

Gierige Algorithmen

Definition

- Zwei Anfragen heißen kompatibel, wenn sich die Intervalle nicht überschneiden.

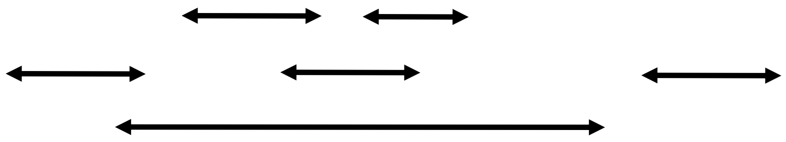


- Nicht kompatibel

Gierige Algorithmen

Generelle Überlegung

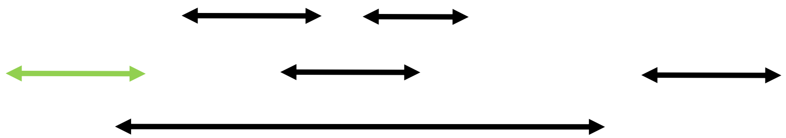
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Gierige Algorithmen

Generelle Überlegung

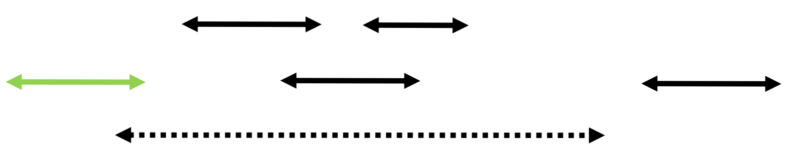
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Gierige Algorithmen

Generelle Überlegung

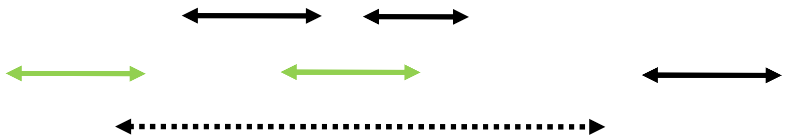
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Gierige Algorithmen

Generelle Überlegung

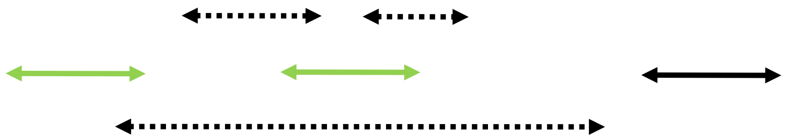
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Gierige Algorithmen

Generelle Überlegung

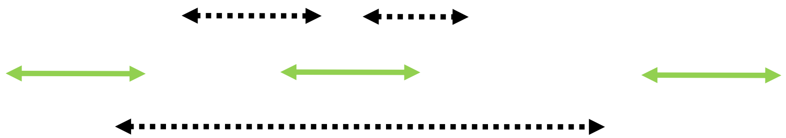
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Gierige Algorithmen

Generelle Überlegung

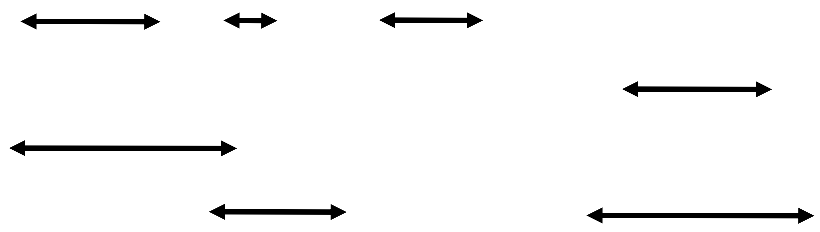
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Gierige Algorithmen

Strategie 1

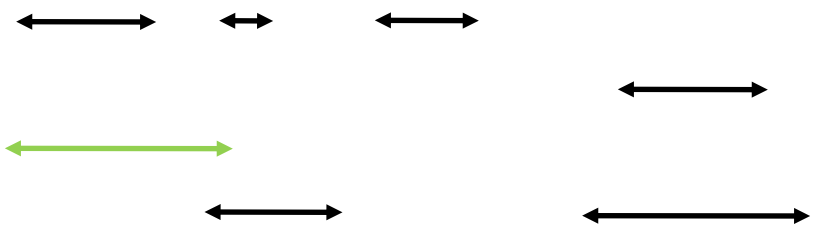
- Wähle immer die Anfrage, die am frühesten beginnt



Gierige Algorithmen

Strategie 1

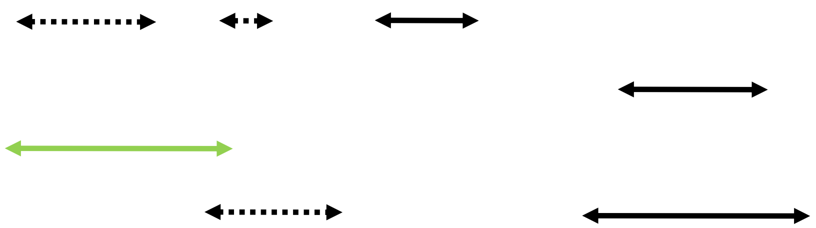
- Wähle immer die Anfrage, die am frühesten beginnt



Gierige Algorithmen

Strategie 1

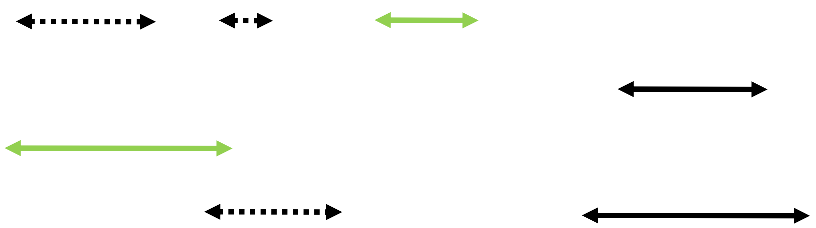
- Wähle immer die Anfrage, die am frühesten beginnt



Gierige Algorithmen

Strategie 1

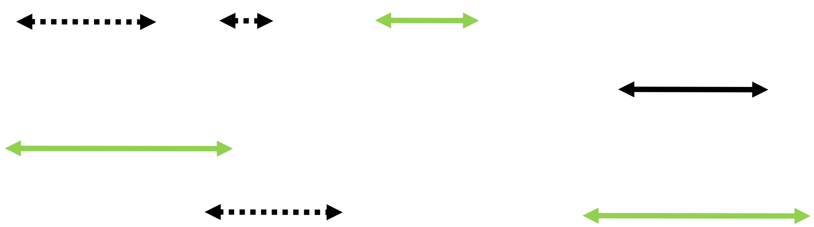
- Wähle immer die Anfrage, die am frühesten beginnt



Gierige Algorithmen

Strategie 1

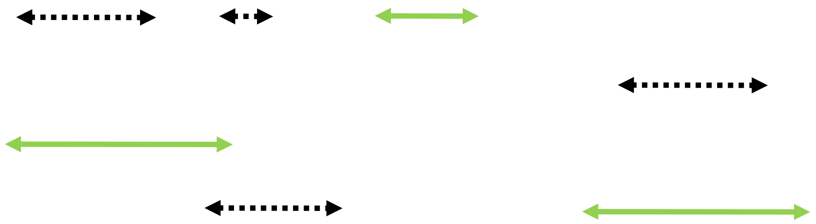
- Wähle immer die Anfrage, die am frühesten beginnt



Gierige Algorithmen

Strategie 1

- Wähle immer die Anfrage, die am frühesten beginnt

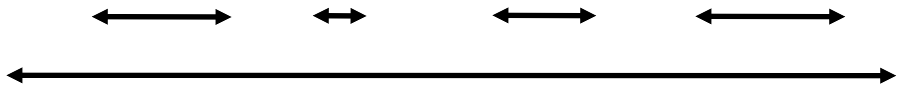


Gierige Algorithmen

Strategie 1

- Wähle immer die Anfrage, die am frühesten beginnt

Optimalität?

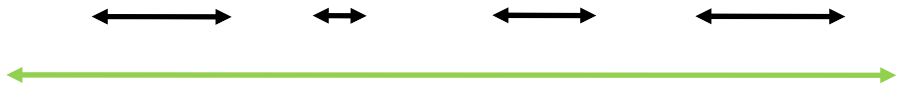


Gierige Algorithmen

Strategie 1

- Wähle immer die Anfrage, die am frühesten beginnt

Optimalität?



Gierige Algorithmen

Strategie 1

- Wähle immer die Anfrage, die am frühesten beginnt

Optimalität?

Gierige Algorithmen

Strategie 1

- Wähle immer die Anfrage, die am frühesten beginnt

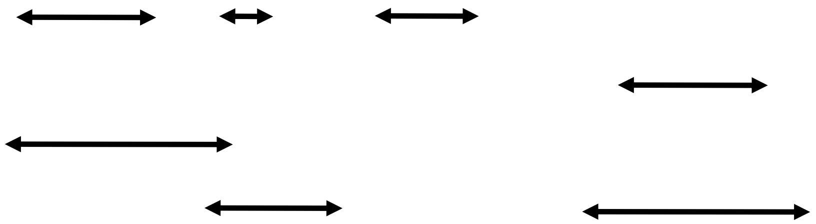
Optimalität?

Nicht optimal, da eine optimale Lösung 4 Anfragen erfüllen kann

Gierige Algorithmen

Strategie 2

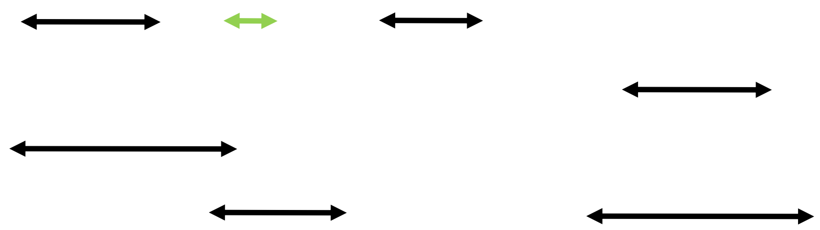
- Wähle immer das kürzeste Interval



Gierige Algorithmen

Strategie 2

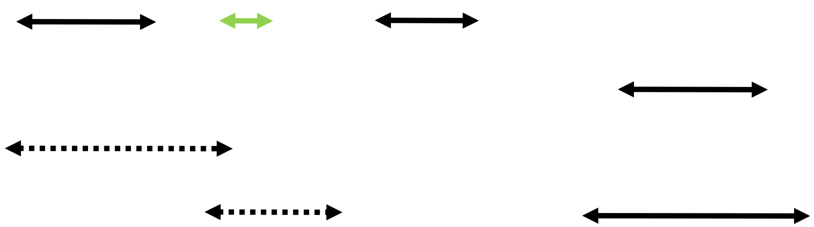
- Wähle immer das kürzeste Interval



Gierige Algorithmen

Strategie 2

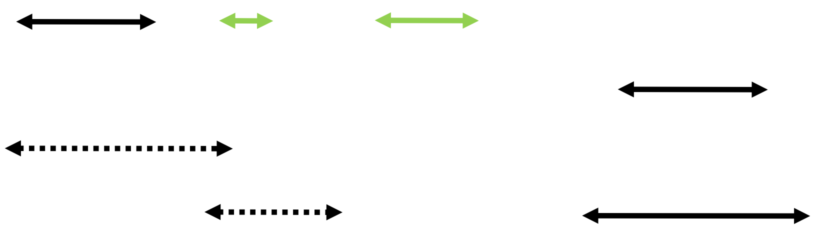
- Wähle immer das kürzeste Interval



Gierige Algorithmen

Strategie 2

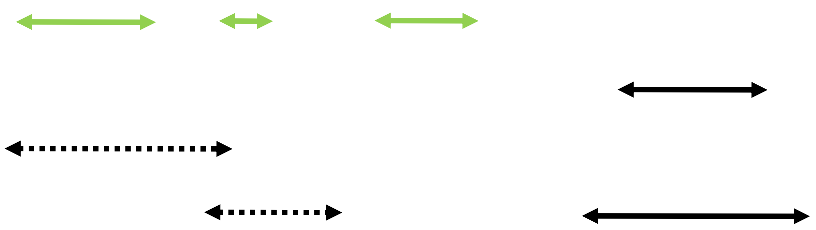
- Wähle immer das kürzeste Interval



Gierige Algorithmen

Strategie 2

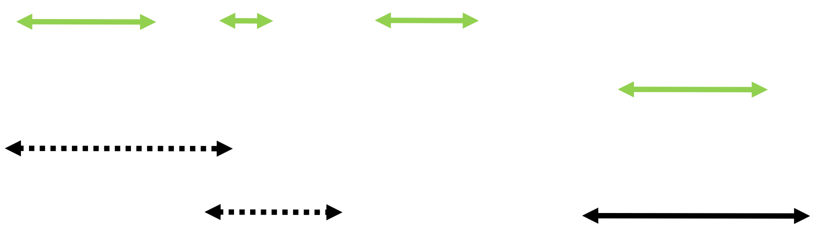
- Wähle immer das kürzeste Interval



Gierige Algorithmen

Strategie 2

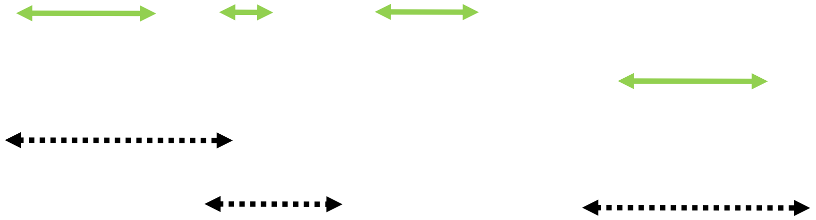
- Wähle immer das kürzeste Interval



Gierige Algorithmen

Strategie 2

- Wähle immer das kürzeste Interval

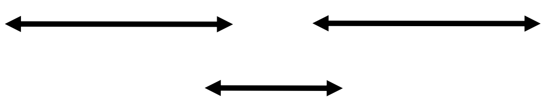


Gierige Algorithmen

Strategie 2

- Wähle immer das kürzeste Interval

Optimalität?

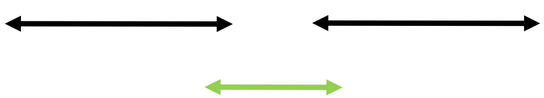


Gierige Algorithmen

Strategie 2

- Wähle immer das kürzeste Interval

Optimalität?



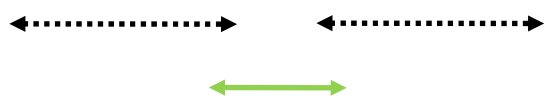
Gierige Algorithmen

Strategie 2

- Wähle immer das kürzeste Interval

Optimalität?

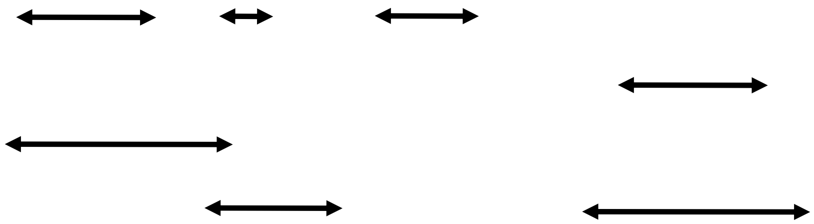
Ebenfalls nicht optimal, da man 2 Anfragen erfüllen kann!



Gierige Algorithmen

Strategie 3

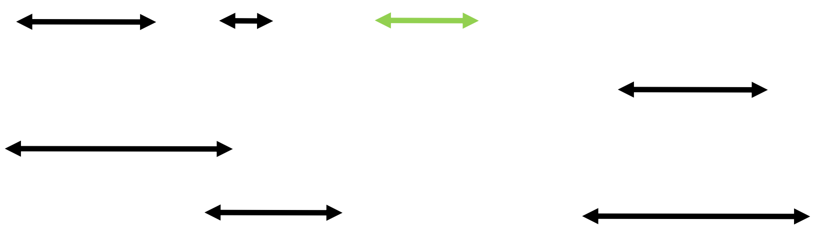
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

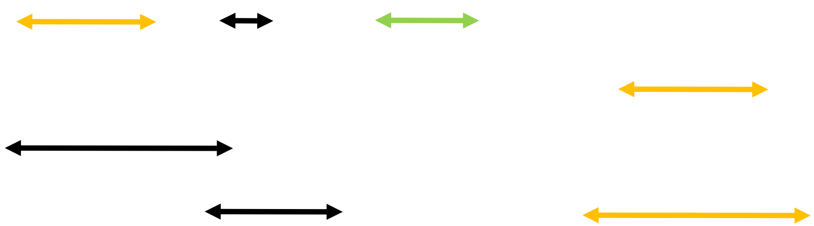
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

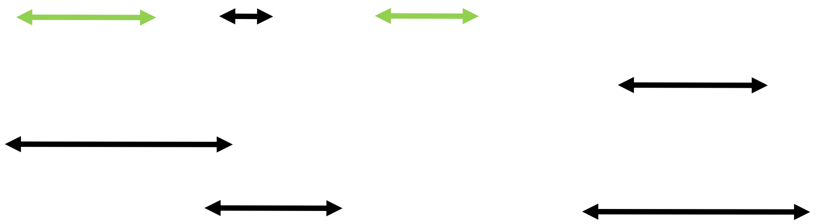
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

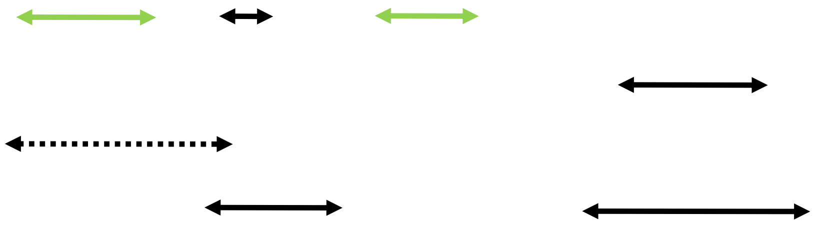
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

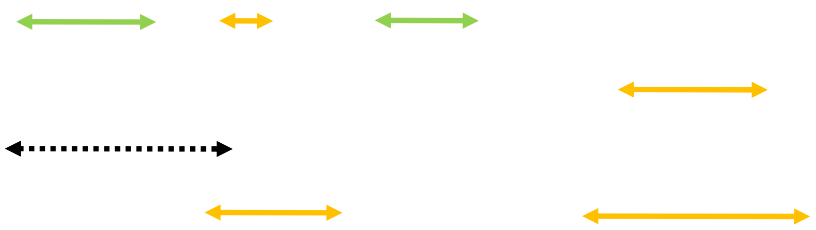
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

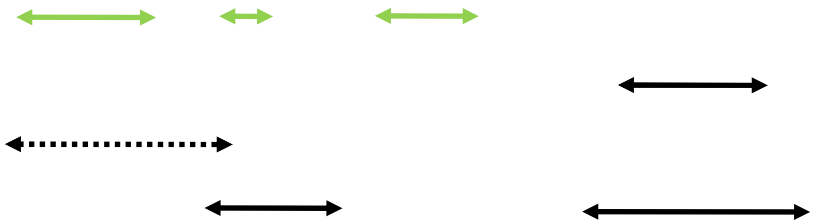
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

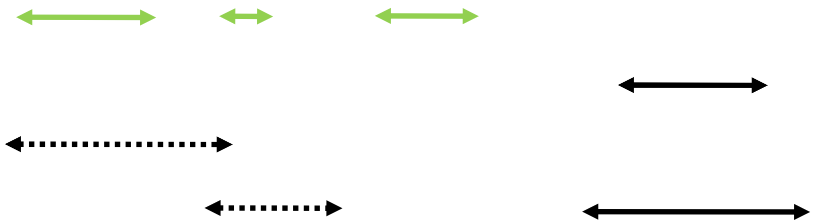
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

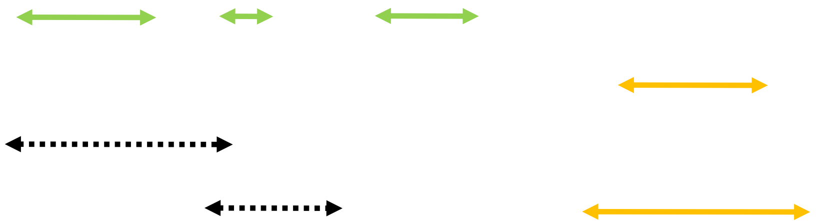
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

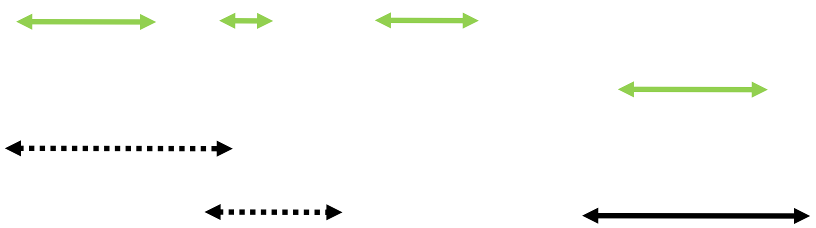
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

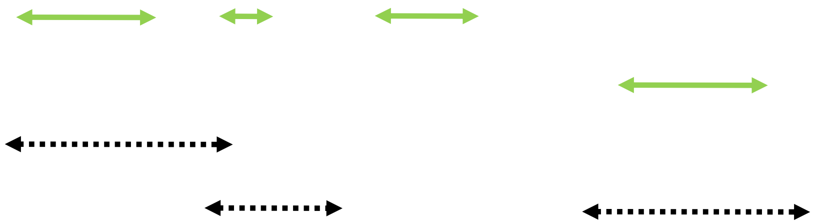
- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval



Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

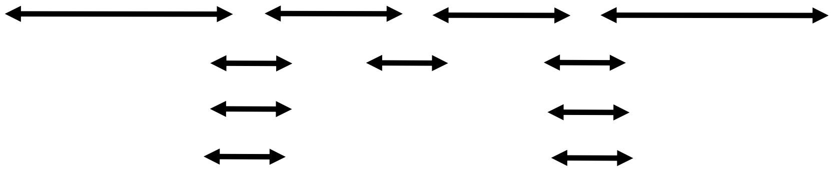


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?

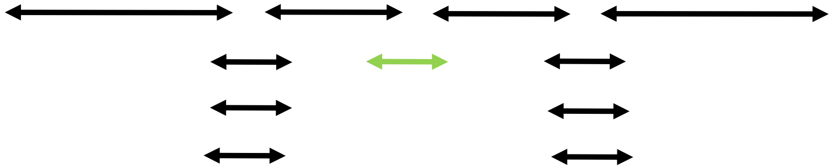


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?

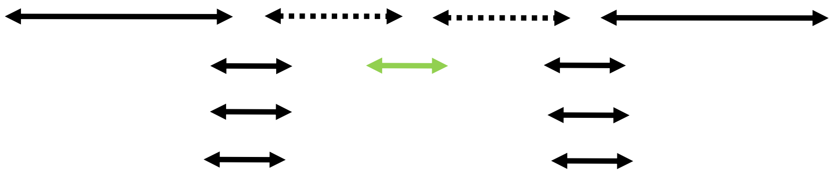


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?

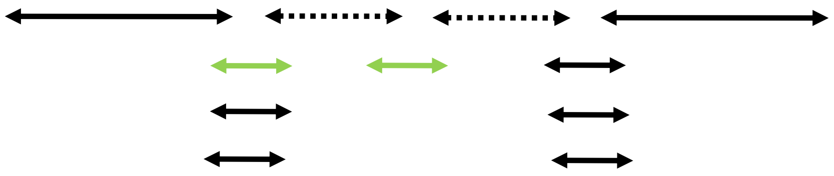


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?

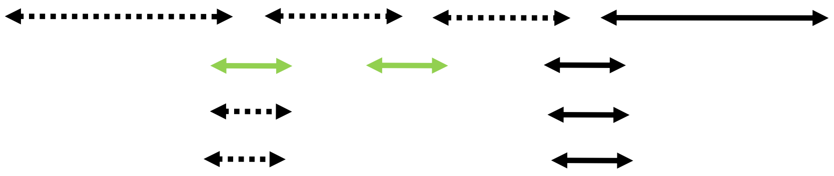


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?

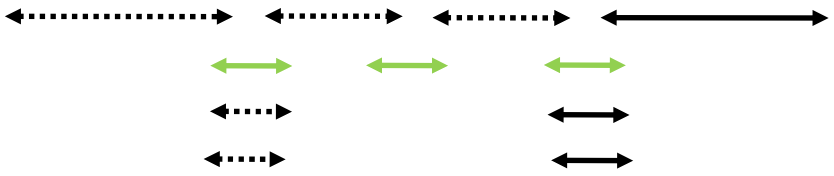


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?

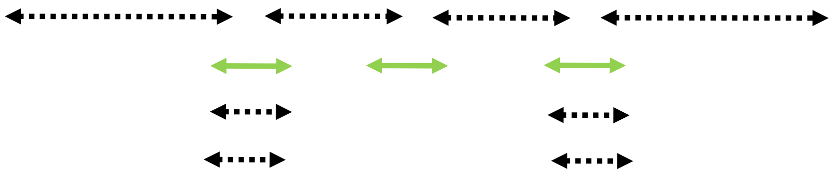


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?

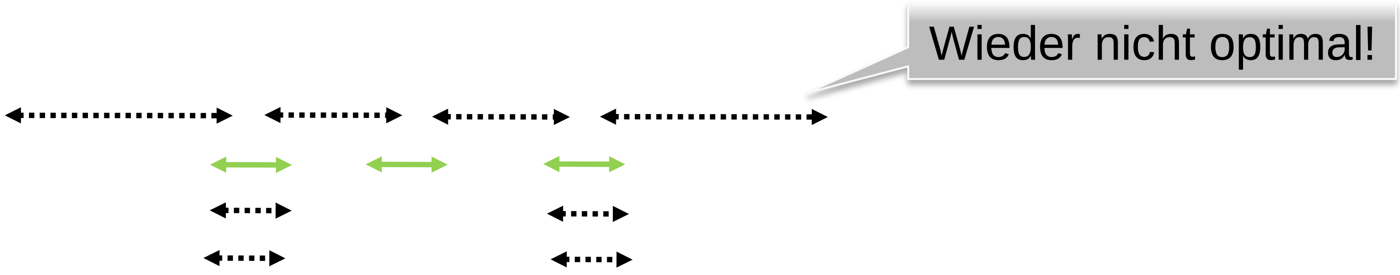


Gierige Algorithmen

Strategie 3

- Wähle immer das Interval mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Interval

Optimalität?



Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

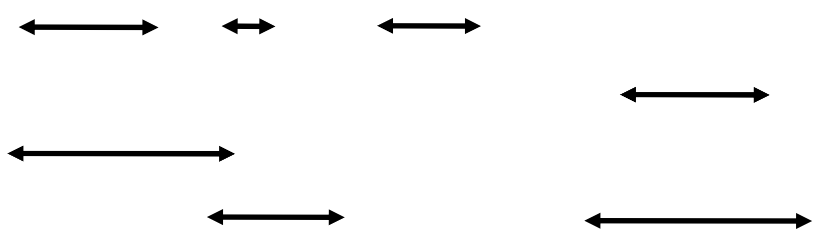
Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.



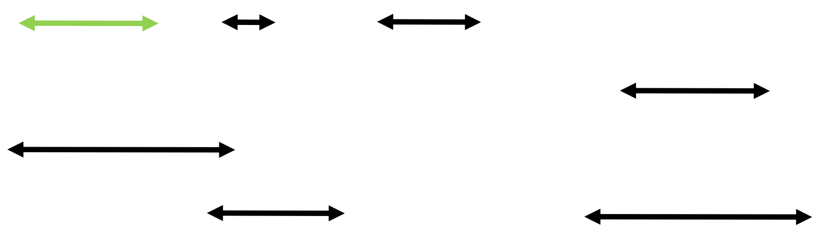
Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.



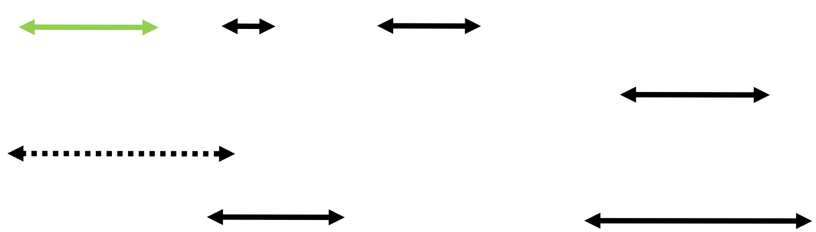
Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.



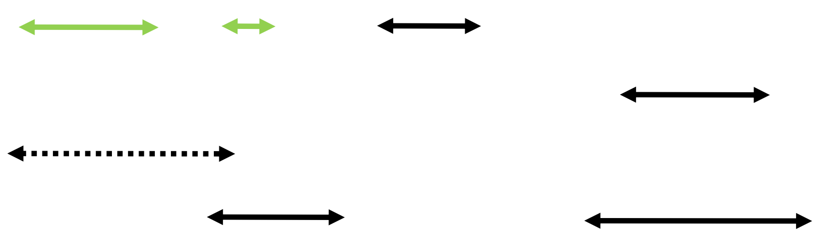
Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.



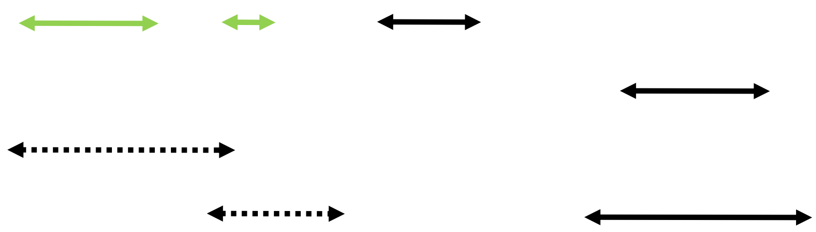
Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.



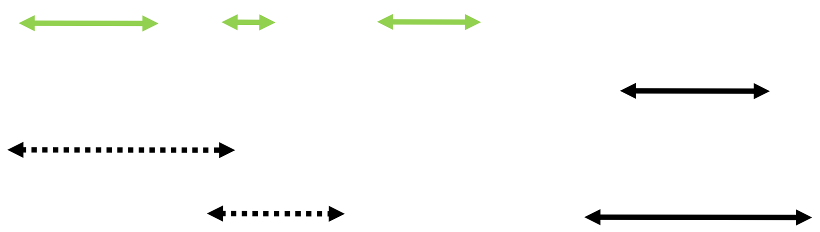
Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.



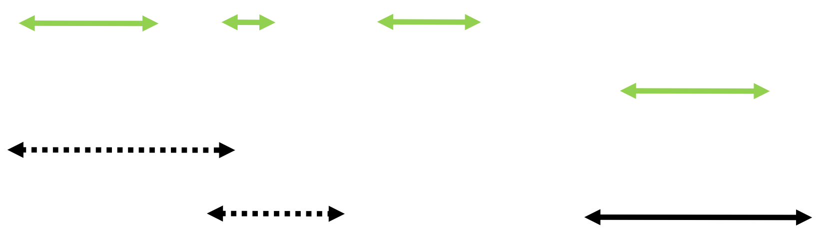
Gierige Algorithmen

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.



Gierige Algorithmen

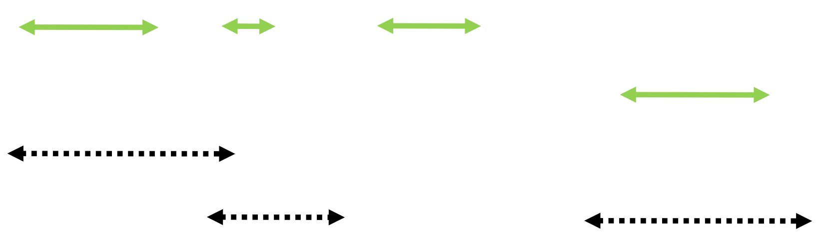
Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie

- Nimm die Anfrage, die am frühesten fertig ist.

Diese Strategie ist optimal!
Aber wie beweist man das?



Gierige Algorithmen

Formale Problemformulierung:

- Problem: Interval Scheduling
- Eingabe: Felder s und f, die die Intervalle (s[i], f[i]) beschreiben
- Ausgabe: Indizes der ausgewählten Intervalle

Wichtige Annahme:

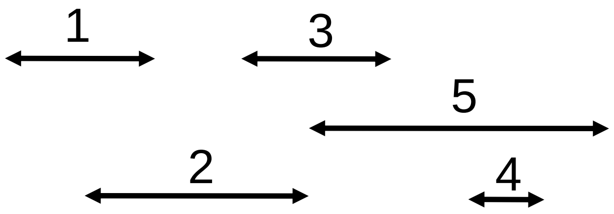
- Eingabe nach Intervallendpunkten sortiert, d.h.
- $f[1] \leq f[2] \leq \ldots \leq f[n]$

Gierige Algorithmen

IntervalScheduling(s,f)

1. n $\leftarrow$ length[s]
2. A $\leftarrow$ {1}
3. j $\leftarrow$ 1
4. **for** i $\leftarrow$ 2 **to** n **do**
5.   **if** s[i] $\geq$ f[j] **then**
6.   A $\leftarrow$ A $\cup$ {i}
7.   j $\leftarrow$ i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

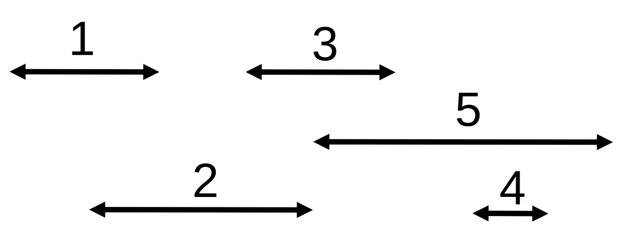


Gierige Algorithmen

IntervalScheduling(s,f)

1. $\boxed{n \leftarrow \text{length}[s]}$
2. $A \leftarrow \{1\}$
3. $j \leftarrow 1$
4. **for** $i \leftarrow 2$ **to** n **do**
5. **if** $s[i] \geq f[j]$ **then**
6. $A \leftarrow A \cup \{i\}$
7. $j \leftarrow i$
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

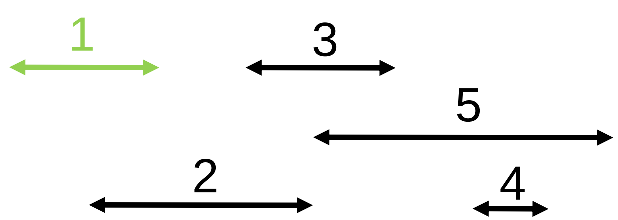


Gierige Algorithmen

IntervalScheduling(s,f)

1. $n \leftarrow$ length[s]
2. $\boxed{A \leftarrow \{1\}}$
3. $j \leftarrow 1$
4. **for** $i \leftarrow 2$ **to** n **do**
5.   **if** $s[i] \geq f[j]$ **then**
6. $A \leftarrow A \cup \{i\}$
7. $j \leftarrow i$
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

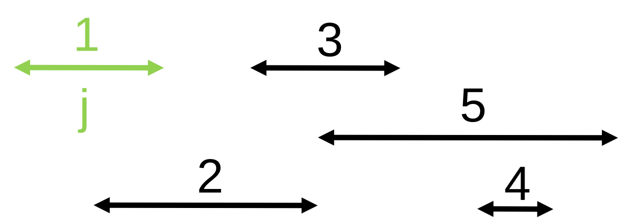


Gierige Algorithmen

IntervalScheduling(s,f)

```
n ← length[s]
A ← {1}
j ← 1
for i ← 2 to n do
    if s[i] ≥ f[j] then
        A ← A ∪ {i}
        j ← i
return A
```

s	1	2	4	7	5
f	3	5	6	8	9

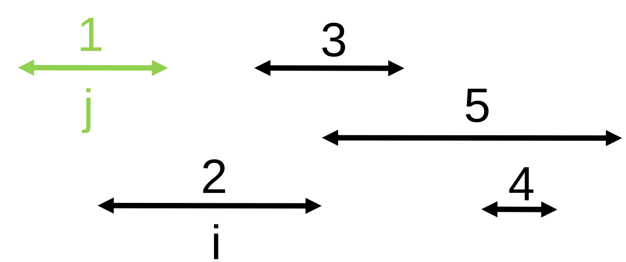


Gierige Algorithmen

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5.   **if** s[i] ≥ f[j] **then**
6.   A ← A ∪ {i}
7.   j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

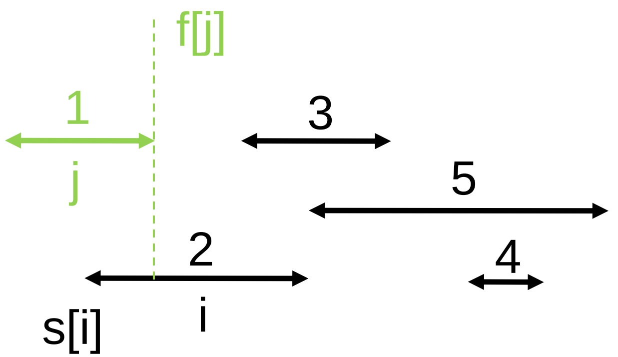


Gierige Algorithmen

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

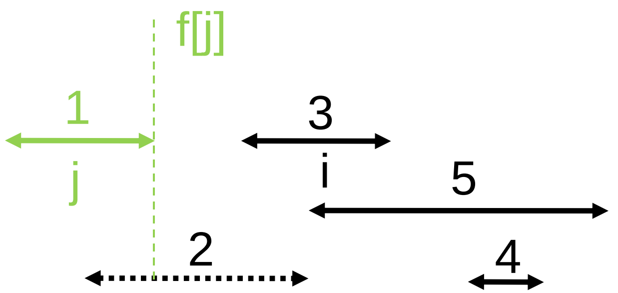


Gierige Algorithmen

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. $\boxed{\textbf{for } i \leftarrow 2 \textbf{ to } n \textbf{ do}}$
5.   **if** s[i] ≥ f[j] **then**
6.   A ← A ∪ {i}
7.   j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

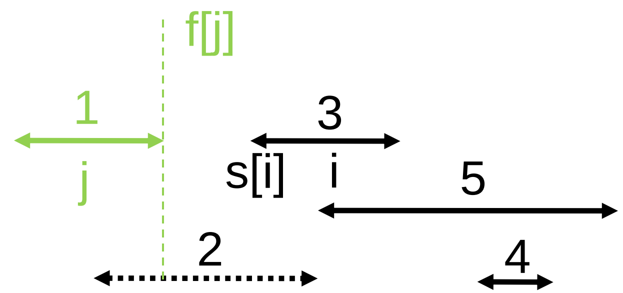


Gierige Algorithmen

IntervalScheduling(s,f)

```
n ← length[s]
A ← {1}
j ← 1
for i ← 2 to n do
    if s[i] ≥ f[j] then
        A ← A ∪ {i}
        j ← i
return A
```

s	1	2	4	7	5
f	3	5	6	8	9

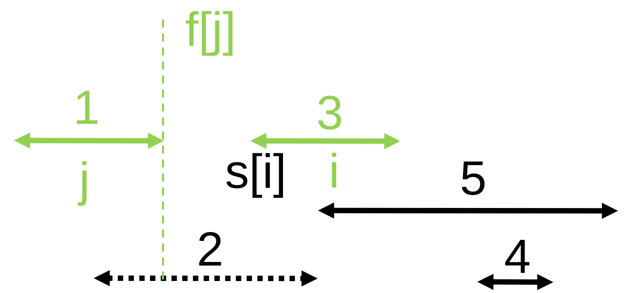


Gierige Algorithmen

IntervalScheduling(s,f)

1. $n \leftarrow$ length[s]
2. $A \leftarrow \{1\}$
3. $j \leftarrow 1$
4. **for** $i \leftarrow 2$ **to** n **do**
5.   **if** $s[i] \geq f[j]$ **then**
6. $\boxed{A \leftarrow A \cup \{i\}}$
7. $j \leftarrow i$
8. **return** A

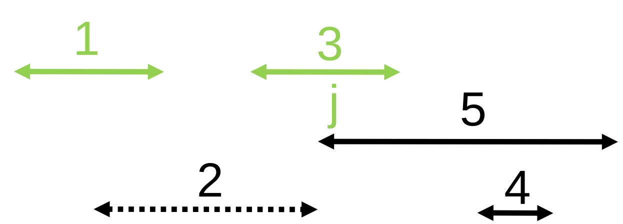
s	1	2	4	7	5
f	3	5	6	8	9



Gierige Algorithmen

```
IntervalScheduling(s,f)
n ← length[s]
A ← {1}
j ← 1
for i ← 2 to n do
    if s[i] ≥ f[j] then
        A ← A ∪ {i}
        j ← i
return A
```

s	1	2	4	7	5
f	3	5	6	8	9

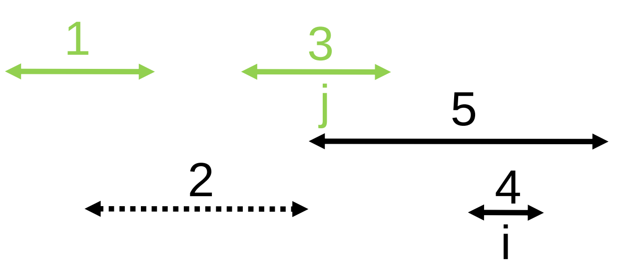


Gierige Algorithmen

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

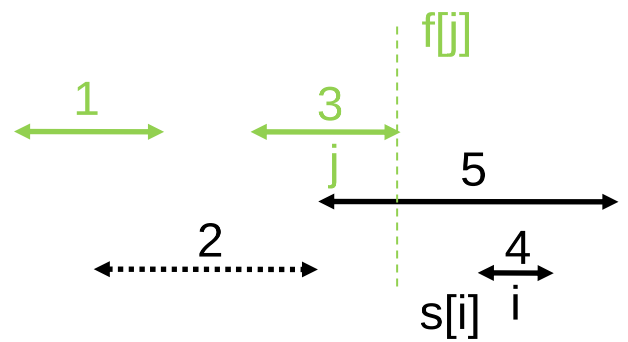


Gierige Algorithmen

IntervalScheduling(s,f)

```
n ← length[s]
A ← {1}
j ← 1
for i ← 2 to n do
    if s[i] ≥ f[j] then
        A ← A ∪ {i}
        j ← i
return A
```

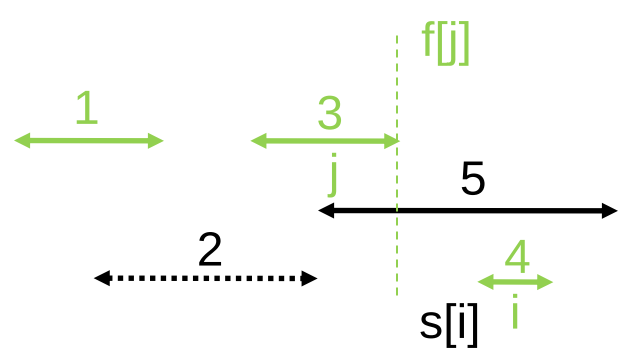
s	1	2	4	7	5
f	3	5	6	8	9



Gierige Algorithmen

```
IntervalScheduling(s,f)
n ← length[s]
A ← {1}
j ← 1
for i ← 2 to n do
    if s[i] ≥ f[j] then
        A ← A ∪ {i}
        j ← i
return A
```

s	1	2	4	7	5
f	3	5	6	8	9

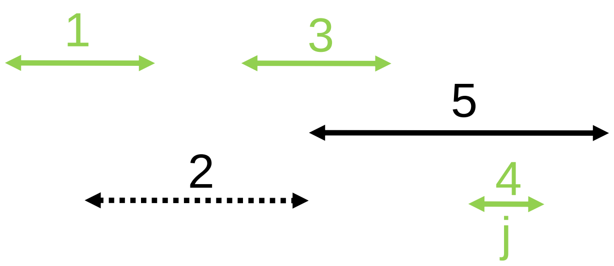


Gierige Algorithmen

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. $\boxed{j \leftarrow i}$
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

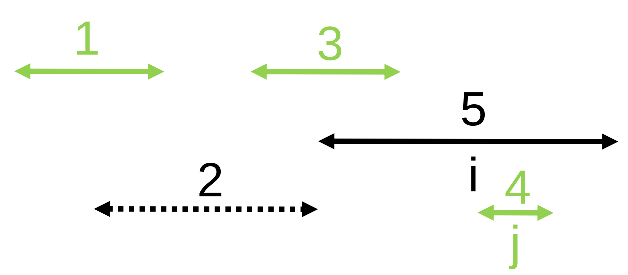


Gierige Algorithmen

IntervalScheduling(s,f)

n ← length[s]
A ← {1}
j ← 1
for i ← 2 **to** n **do**
    if s[i] ≥ f[j] **then**
        A ← A ∪ {i}
        j ← i
return A

s	1	2	4	7	5
f	3	5	6	8	9

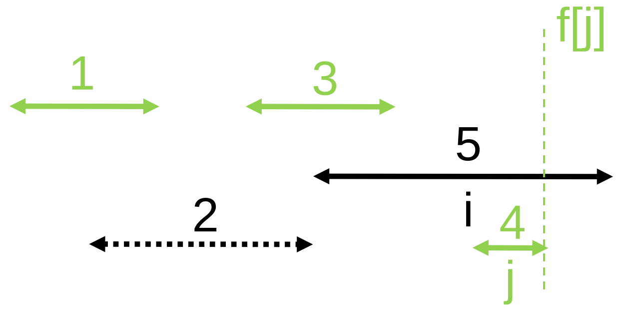


Gierige Algorithmen

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

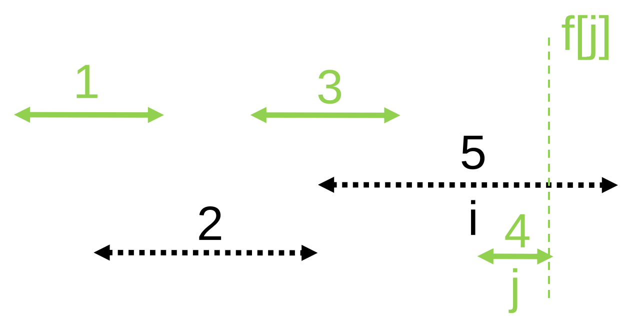


Gierige Algorithmen

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

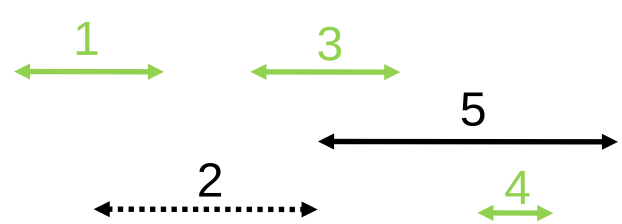


Gierige Algorithmen

IntervalScheduling(s,f)

1. n $\leftarrow$ length[s]
2. A $\leftarrow$ {1}
3. j $\leftarrow$ 1
4. **for** i $\leftarrow$ 2 **to** n **do**
5.   **if** s[i] $\geq$ f[j] **then**
6.   A $\leftarrow$ A $\cup$ {i}
7.   j $\leftarrow$ i
8. $\boxed{\textbf{return } A}$

s	1	2	4	7	5
f	3	5	6	8	9



Gierige Algorithmen

Beweisidee: Der gierige Algorithmus „liegt vorn“

- Wir vergleichen eine optimale Lösung mit der Lösung des gierigen Algorithmen zu verschiedenen Zeitpunkten
- Wir zeigen: Die Lösung des gierigen Algorithmus ist bzgl. eines bestimmten Kriteriums mindestens genauso gut wie die optimale Lösung

Vergleichzeitpunkte

- Nach jedem Hinzufügen eines Intervals zur aktuellen Lösung

Vergleichskriterium

- Maximaler Endzeitpunkt der bisher ausgewählten Anfragen

Gierige Algorithmen

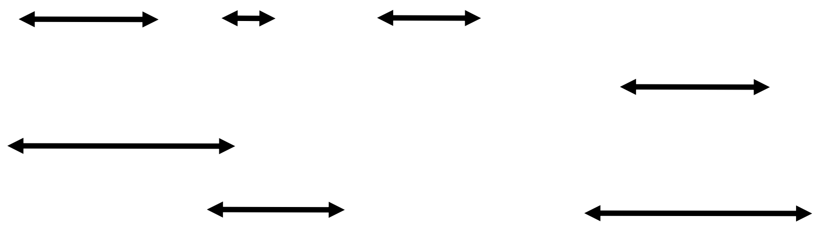
Erste Beobachtung

- A ist eine Menge von kompatiblen Anfrage.

Gierige Algorithmen

Wie können wir Optimalität zeigen?

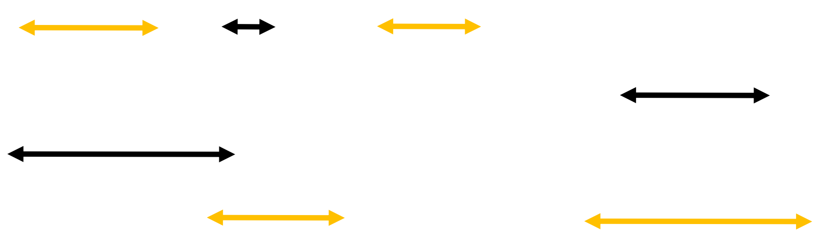
- Sei eine O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Gierige Algorithmen

Wie können wir Optimalität zeigen?

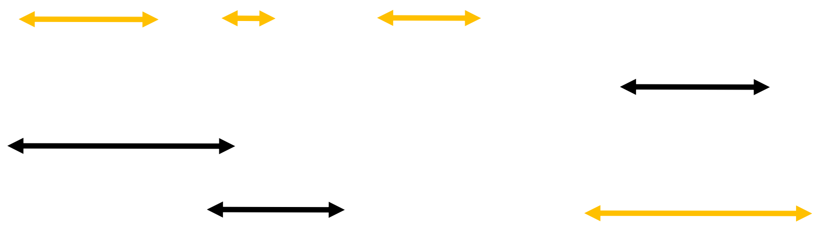
- Sei eine O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Gierige Algorithmen

Wie können wir Optimalität zeigen?

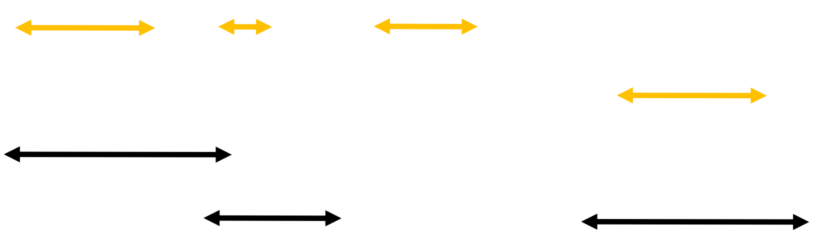
- Sei eine O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Gierige Algorithmen

Wie können wir Optimalität zeigen?

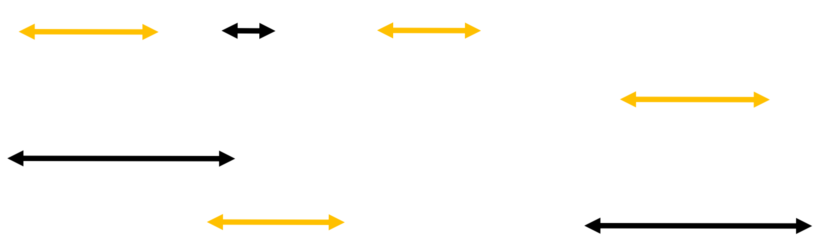
- Sei eine O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Gierige Algorithmen

Wie können wir Optimalität zeigen?

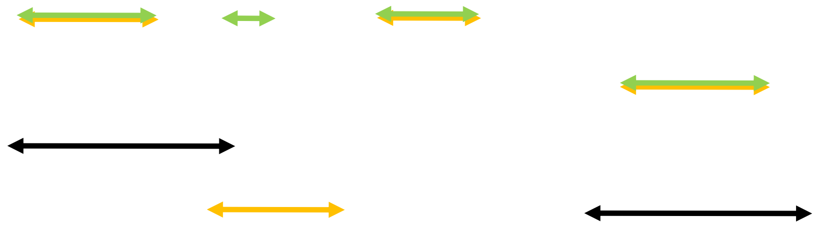
- Sei eine O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Gierige Algorithmen

Wie können wir Optimalität zeigen?

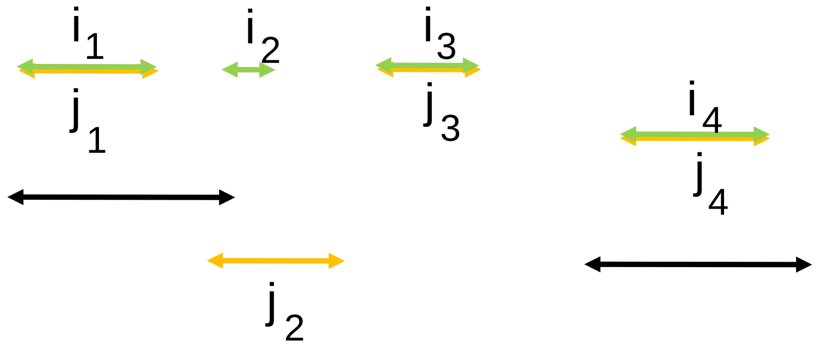
- Sei eine O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Gierige Algorithmen

Notation

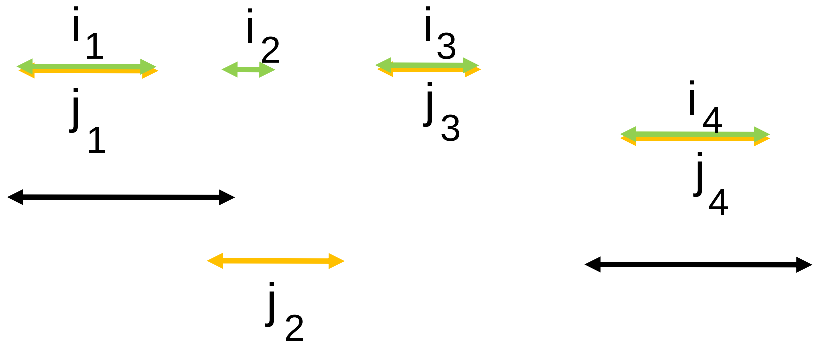
- $i_1, \ldots, i_k$ Intervalle von A in Ordnung des Hinzufügen
- $j_1, \ldots, j_m$ Intervalle von O sortiert nach Endpunkt
- Zu zeigen: k = m



Gierige Algorithmen

Der gierige Algorithmus liegt vorn

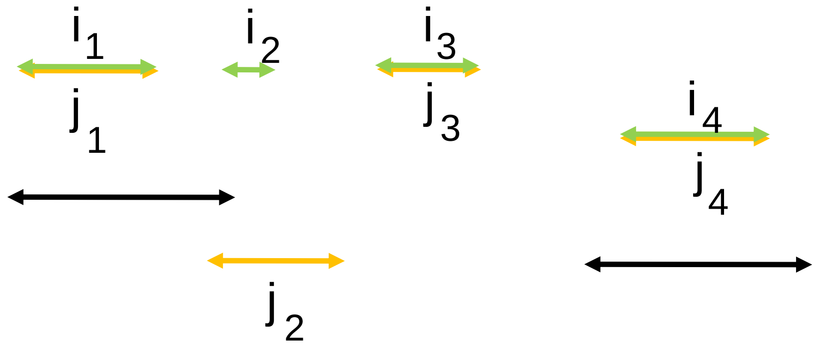
- Idee des Algorithmus: Die Resource soll so früh wie möglich wieder frei werden
- Dies ist war für das erste Interval: $f[i_1] \leq f[j_1]$
- Zu zeigen: Gilt für alle Intervalle



Gierige Algorithmen

Der gierige Algorithmus liegt vorn

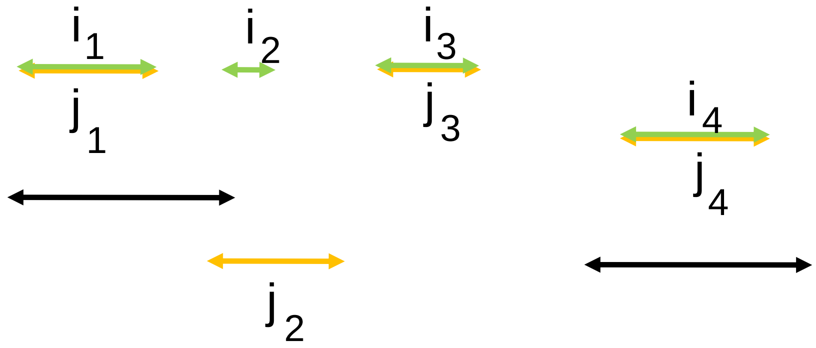
- Idee des Algorithmus: Die Resource soll so früh wie möglich wieder frei werden
- Dies ist war für das erste Interval: $f[i_1] \le f[j_1]$
- Zu zeigen: Gilt für alle Intervalle



Gierige Algorithmen

Lemma 10

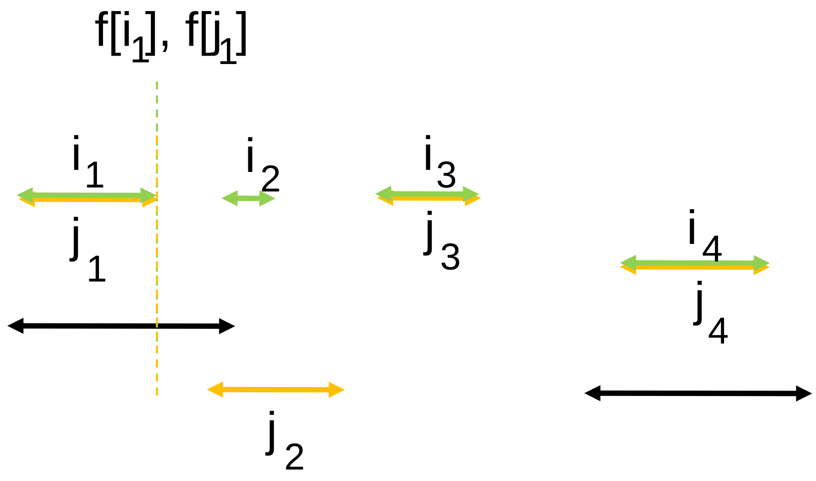
Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Gierige Algorithmen

Lemma 10

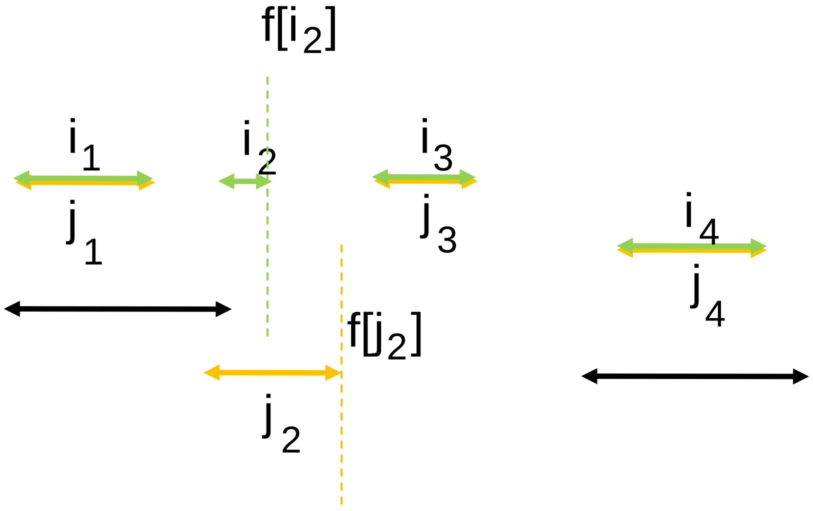
Für alle r≤k gilt $f[i_r] \leq f[j_r]$.



Gierige Algorithmen

Lemma 10

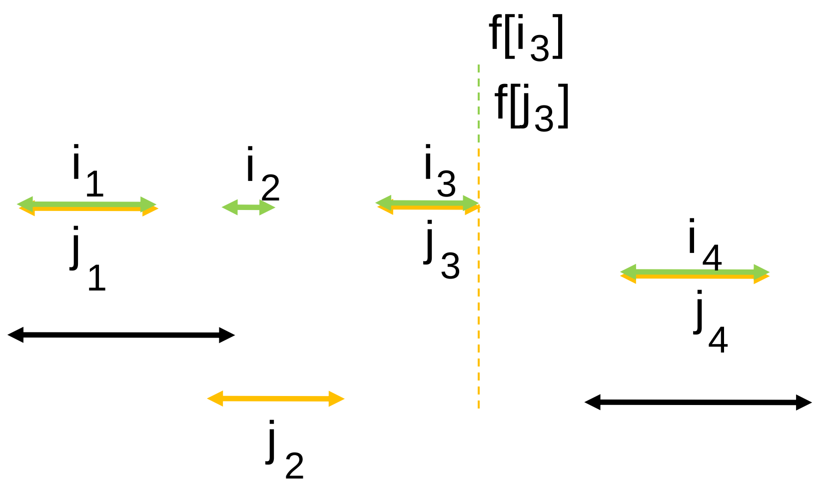
Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Gierige Algorithmen

Lemma 10

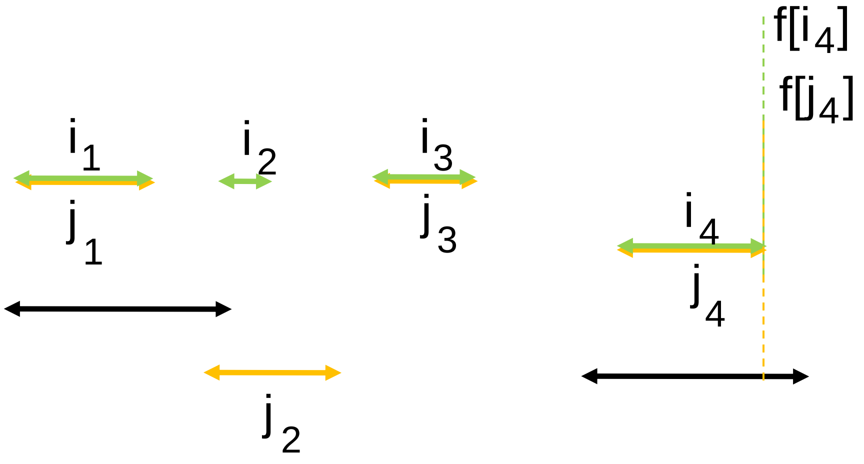
Für alle r≤k gilt $f[i_r] \leq f[j_r]$.



Gierige Algorithmen

Lemma

Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Gierige Algorithmen

Lemma

Für alle $r \le k$ gilt $f[i_r] \le f[j_r]$.

Beweis

Induktion über r.

(I.A.) Für r=1 ist die Aussage offensichtlich korrekt.

(I.V.) Die Aussage gelte für r-1.

Gierige Algorithmen

Lemma

Für alle $r \le k$ gilt $f[i_r] \le f[j_r]$.

Beweis

Induktion über r.

(I.A.) Für r=1 ist die Aussage offensichtlich korrekt.

(I.V.) Die Aussage gelte für r-1.

(I.S.) Nach (I.V.) gilt $f[i_{r-1}] \le f[j_{r-1}]$. Da die Intervalle in O kompatibel sind, gilt $f[j_{r-1}] \le s[j_r]$ und somit auch $f[i_{r-1}] \le s[j_r]$.

Gierige Algorithmen

Lemma

Für alle $r \le k$ gilt $f[i_r] \le f[j_r]$.

Beweis

Induktion über r.

(I.A.) Für r=1 ist die Aussage offensichtlich korrekt.

(I.V.) Die Aussage gelte für r-1.

(I.S.) Nach (I.V.) gilt $f[i_{r-1}] \le f[j_{r-1}]$. Da die Intervalle in O kompatibel sind, gilt $f[j_{r-1}] \le s[j_r]$ und somit auch $f[i_{r-1}] \le s[j_r]$.

Damit ist j_r in der Menge der Intervalle, die mit den ersten r-1 Intervallen kompatibel sind, die IntervalScheduling ausgewählt hat.

Gierige Algorithmen

Lemma

Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.

Beweis

Induktion über r.

(I.A.) Für r=1 ist die Aussage offensichtlich korrekt.

(I.V.) Die Aussage gelte für r-1.

(I.S.) Nach (I.V.) gilt $f[i_{r-1}] \leq f[j_{r-1}]$. Da die Intervalle in O kompatibel sind, gilt $f[j_{r-1}] \leq s[j_r]$ und somit auch $f[i_{r-1}] \leq s[j_r]$.

Damit ist j_r in der Menge der Intervalle, die mit den ersten r-1 Intervallen kompatibel sind, die IntervalScheduling ausgewählt hat.

Da der Algorithmus das Interval mit kleinstem f-Wert auswählt, gilt $f[i_r] \leq f[j_r]$.

Gierige Algorithmen

Lemma

Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.

Beweis

Induktion über r.

(I.A.) Für r=1 ist die Aussage offensichtlich korrekt.

(I.V.) Die Aussage gelte für r-1.

(I.S.) Nach (I.V.) gilt $f[i_{r-1}] \leq f[j_{r-1}]$. Da die Intervalle in O kompatibel sind, gilt $f[j_{r-1}] \leq s[j_r]$ und somit auch $f[i_{r-1}] \leq s[j_r]$.

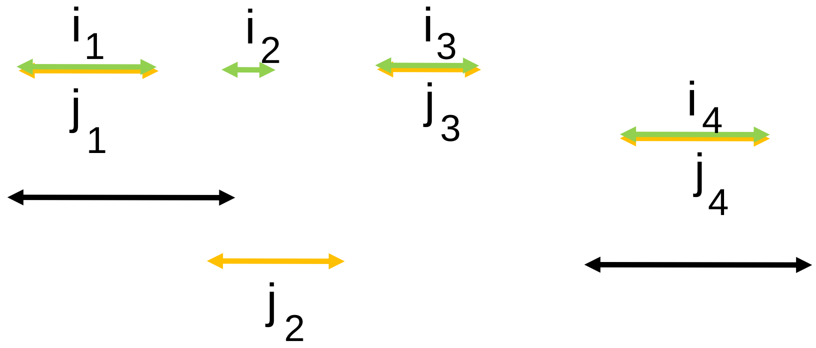
Damit ist j_r in der Menge der Intervalle, die mit den ersten r-1 Intervallen kompatibel sind, die IntervalScheduling ausgewählt hat.

Da der Algorithmus das Interval mit kleinstem f-Wert auswählt, gilt $f[i_r] \leq f[j_r]$.

Gierige Algorithmen

Satz

Die von Algorithmus IntervalSchedule berechnete Lösung A ist optimal.



Gierige Algorithmen

Satz

Die von Algorithmus IntervalSchedule berechnete Lösung A ist optimal.

Beweis (durch Widerspruch)

Ist A nicht optimal, so hat O mehr Anfragen, d.h. es gilt m>k. Nach unserem Lemma mit r=k gilt $f[i_k] \le f[j_k]$.

Gierige Algorithmen

Satz

Die von Algorithmus IntervalSchedule berechnete Lösung A ist optimal.

Beweis (durch Widerspruch)

Ist A nicht optimal, so hat O mehr Anfragen, d.h. es gilt m>k. Nach unserem Lemma mit r=k gilt $f[i_k]\le f[j_k]$.

Da m>k gibt es eine Anfrage j_{k+1} in O, die startet, nachdem j_k und somit auch i_k endet, d.h. $s[j_{k+1}] \ge f[i_k]$. Außerdem gilt natürlich $f[j_{k+1}] \ge s[j_{k+1}] \ge f[i_k]$.

Gierige Algorithmen

Satz

Die von Algorithmus IntervalSchedule berechnete Lösung A ist optimal.

Beweis (durch Widerspruch)

Ist A nicht optimal, so hat O mehr Anfragen, d.h. es gilt m>k. Nach unserem Lemma mit r=k gilt $f[i_k] \le f[j_k]$.

Da m>k gibt es eine Anfrage j_{k+1} in O, die startet, nachdem j_k und somit auch i_k endet, d.h. $s[j_{k+1}] \ge f[i_k]$. Außerdem gilt natürlich $f[j_{k+1}] \ge s[j_{k+1}] \ge f[i_k]$.

Betrachten wir nun den Zeitpunkt, zu dem IntervalScheduling Interval i_k in A aufnimmt. Da die Intervalle nach Endzeitpunkten sortiert sind, wurde j_{k+1} noch nicht betrachtet.

Gierige Algorithmen

Satz

Die von Algorithmus IntervalSchedule berechnete Lösung A ist optimal.

Beweis (durch Widerspruch)

Ist A nicht optimal, so hat O mehr Anfragen, d.h. es gilt m>k. Nach unserem Lemma mit r=k gilt $f[i_k] \le f[j_k]$.

Da m>k gibt es eine Anfrage j_{k+1} in O, die startet, nachdem j_k und somit auch i_k endet, d.h. $s[j_{k+1}] \ge f[i_k]$. Außerdem gilt natürlich $f[j_{k+1}] \ge s[j_{k+1}] \ge f[i_k]$.

Betrachten wir nun den Zeitpunkt, zu dem IntervalScheduling Interval i_k in A aufnimmt. Da die Intervalle nach Endzeitpunkten sortiert sind, wurde j_{k+1} noch nicht betrachtet.

Da kein weiteres Interval in A aufgenommen wird, muss für alle noch nicht betrachteten Intervalle der Startzeitpunkt vor $f[i_k]$ liegen.

Gierige Algorithmen

Satz

Die von Algorithmus IntervalSchedule berechnete Lösung A ist optimal.

Beweis (durch Widerspruch)

Ist A nicht optimal, so hat O mehr Anfragen, d.h. es gilt m>k. Nach unserem Lemma mit r=k gilt $f[i_k] \le f[j_k]$.

Da m>k gibt es eine Anfrage j_{k+1} in O, die startet, nachdem j_k und somit auch i_k endet, d.h. $s[j_{k+1}] \ge f[i_k]$. Außerdem gilt natürlich $f[j_{k+1}] \ge s[j_{k+1}] \ge f[i_k]$.

Betrachten wir nun den Zeitpunkt, zu dem IntervalScheduling Interval i_k in A aufnimmt. Da die Intervalle nach Endzeitpunkten sortiert sind, wurde j_{k+1} noch nicht betrachtet.

Da kein weiteres Interval in A aufgenommen wird, muss für alle noch nicht betrachteten Intervalle der Startzeitpunkt vor $f[i_k]$ liegen.

Widerspruch, denn wir haben bereits gezeigt, dass $s[j_{k+1}] \ge f[i_k]$ gilt.

Gierige Algorithmen

Satz

Die von Algorithmus IntervalSchedule berechnete Lösung A ist optimal.

Beweis (durch Widerspruch)

Ist A nicht optimal, so hat O mehr Anfragen, d.h. es gilt m>k. Nach unserem Lemma mit r=k gilt $f[i_k] \leq f[j_k]$.

Da m>k gibt es eine Anfrage j_{k+1} in O, die startet, nachdem j_k und somit auch i_k endet, d.h. $s[j_{k+1}] \geq f[i_k]$. Außerdem gilt natürlich $f[j_{k+1}] \geq s[j_{k+1}] \geq f[i_k]$.

Betrachten wir nun den Zeitpunkt, zu dem IntervalScheduling Interval i_k in A aufnimmt. Da die Intervalle nach Endzeitpunkten sortiert sind, wurde j_{k+1} noch nicht betrachtet.

Da kein weiteres Interval in A aufgenommen wird, muss für alle noch nicht betrachteten Intervalle der Startzeitpunkt vor $f[i_k]$ liegen.

Widerspruch, denn wir haben bereits gezeigt, dass $s[j_{k+1}] \geq f[i_k]$ gilt.

Gierige Algorithmen

IntervalScheduling(s,f)

```
n ← length[s]
A ← {1}                  Θ(1)
j ← 1
for i ← 2 to n do
    if s[i] ≥ f[j] then   Θ(n)
        A ← A ∪ {i}
        j ← i
return A                 Θ(1)
                         ----
                         Θ(n)
```

Gierige Algorithmen

Satz

Algorithmus IntervalSchedule berechnet in $\Theta(n)$ Zeit eine optimale Lösung, wenn die Eingabe nach Endzeit der Intervalle (rechter Endpunkt) sortiert ist. Die Sortierung kann in $\Theta(n \log n)$ Zeit berechnet werden.