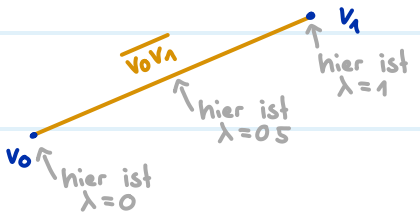
Konvexe Hülle

Definition Liniensegment: $\overline{v_0 v_1} := \{(1-\lambda)\, v_0 + \lambda\, v_1 \mid 0 \leq \lambda \leq 1\}$



Definition konvex: eine Menge C ist konvex $\overset{\text{def}}{\iff} \forall v_0, v_1 \in C : \overline{v_0 v_1} \subseteq C$

für beliebige zwei Punkte aus C muss das Liniensegment dazwischen auch in C sein

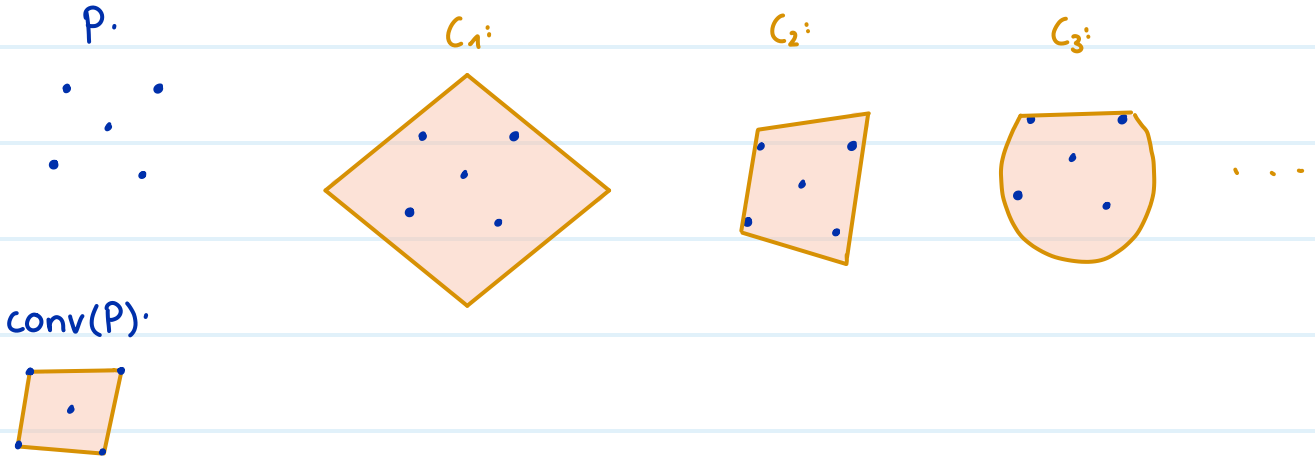
konvex? → Ja | konvex? → Ja | konvex? → Nein | konvex? → Nein | konvex? → Ja

Definition konvexe Hülle: Sei $P \subseteq \mathbb{R}^2$ eine endliche Punktemenge

$$\text{conv}(P) := \bigcap_{P \subseteq C \subseteq \mathbb{R}^2 \text{ mit } C \text{ konvex}} C$$

Schnittmenge aller konvexen Mengen, die P enthalten

Bsp:

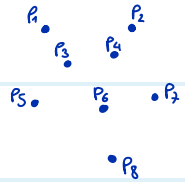


Wir können conv(P) durch ein Polygon $Q = (q_0, q_1, \ldots, q_{h-1})$ darstellen, dessen

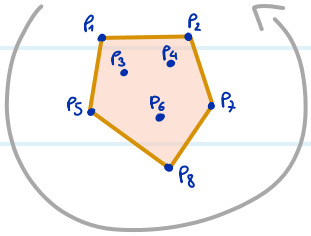
h Ecken Punkte aus P sind

Reihenfolge im Gegenuhrzeigersinn

P.



conv(P).

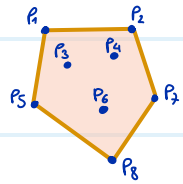


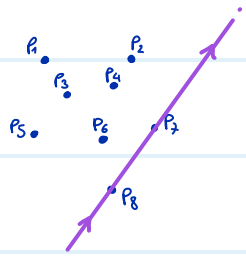
Polygon $Q = (p_1, p_5, p_8, p_7, p_2)$

allgemeine Lage

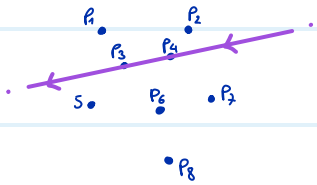
- keine 3 Punkte liegen auf einer Geraden
- keine 2 Punkte haben dieselbe x-Koordinate

Randkante: $(q, r) \in P^2$ mit $q \neq r$ ist eine Randkante, falls alle Punkte aus $P \setminus \{q, r\}$ links von der gerichteten Geraden durch q und r liegen





(p_7, p_8) ist eine Randkante



(p_4, p_3) ist keine Randkante

Lemma 3.34: $(q_0, q_1, \ldots, q_{h-1})$ ist die Eckenfolge des conv(P) umschliessenden Polygons gegen den Uhrzeigersinn

$\Leftrightarrow$ alle Paare (q_{i-1}, q_i) für $i = 1, 2, \ldots, h$ sind Randkanten (Indices mod h)

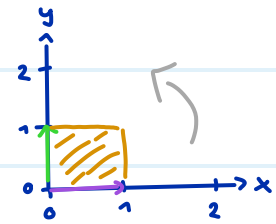
Lemma 3.35 Sei $p = (p_x, p_y)$, $q = (q_x, q_y)$, $r = (r_x, r_y)$ Punkte mit $q \neq r$

Dann liegt p links von der gerichteten Gerade durch q und r

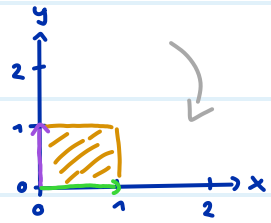
$\Leftrightarrow \det\left(\begin{bmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{bmatrix}\right) > 0$

Intuition: Determinante misst die Fläche des aufgespannten Parallelograms von a und b im Gegenuhrzeigersinn

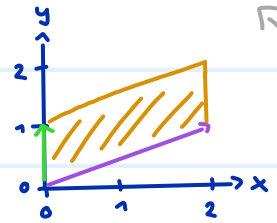
$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ → 

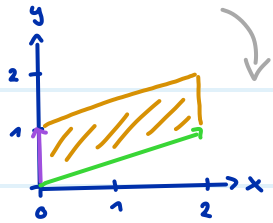
$\det = 1$

$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ → 

$\det = -1$

$\begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix}$ → 

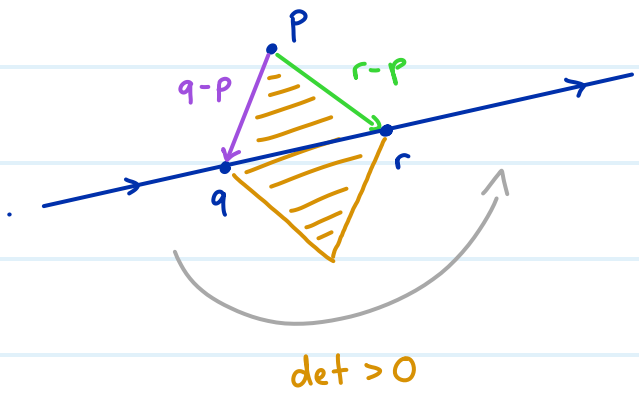
$\det = 2$

$\begin{bmatrix} 0 & 2 \\ 1 & 1 \end{bmatrix}$ → 

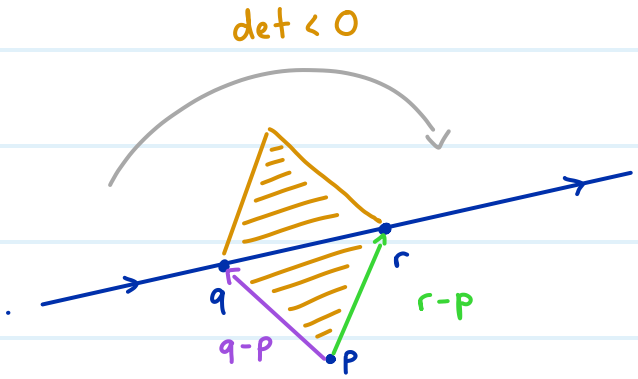
$\det = -2$

Vektor von Punkt A zu B = B-A

Fall links:

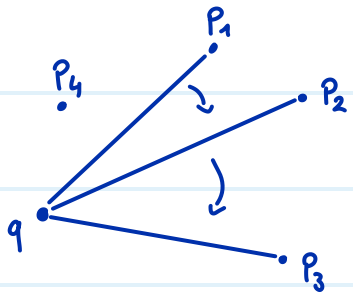


Fall rechts:



FindNext(q) in $O(n)$

1: Wähle $p_0 \in P \setminus \{q\}$ beliebig
2: $q_{next} \leftarrow p_0$
3: **for all** $p \in P \setminus \{q, p_0\}$ **do** → $O(n)$ Iterationen
4: **if** p rechts von $q q_{next}$ **then** $q_{next} \leftarrow p$ → in $O(1)$ mit Determinante
5: **return** q_{next}



Ordnungsrelation $p_1 \prec_q p_2$ def. $\Leftrightarrow$ p_1 liegt rechts von $\underbrace{qp_2}$
= gerichtete Gerade durch q und p_2

Lemma 3.36: Sei q eine Ecke der konvexen Hülle von P

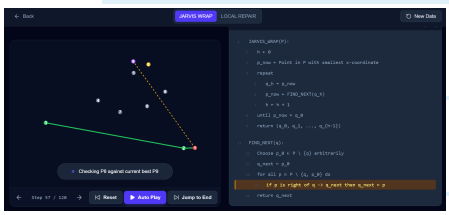
Dann ist $\prec_q$ eine totale Ordnungsrelation auf $P \setminus \{q\}$

total: $\forall a,b \in P \setminus \{q\}: a \prec_q b \vee b \prec_q a$

Und (q, p_{min}) ist eine Randkante, wobei p_{min} das Minimum von $\prec_q$ ist.

JarvisWrap(P) in $O(n\,h)$

1: $h \leftarrow 0$
2: $p_{now} \leftarrow$ Punkt in P mit kleinster x-Koordinate — ist immer garantiert ein Eckpunkt
3: **repeat** → h Iterationen
4: $q_h \leftarrow p_{now}$
5: $p_{now} \leftarrow$ FindNext(q_h) → in $O(n)$
6: $h \leftarrow h + 1$
7: **until** $p_{now} = q_0$
8: **return** $(q_0, q_1, \ldots, q_{h-1})$

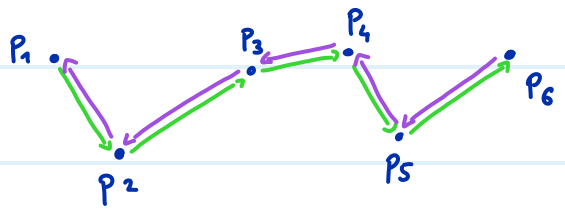


Wie gross ist h?

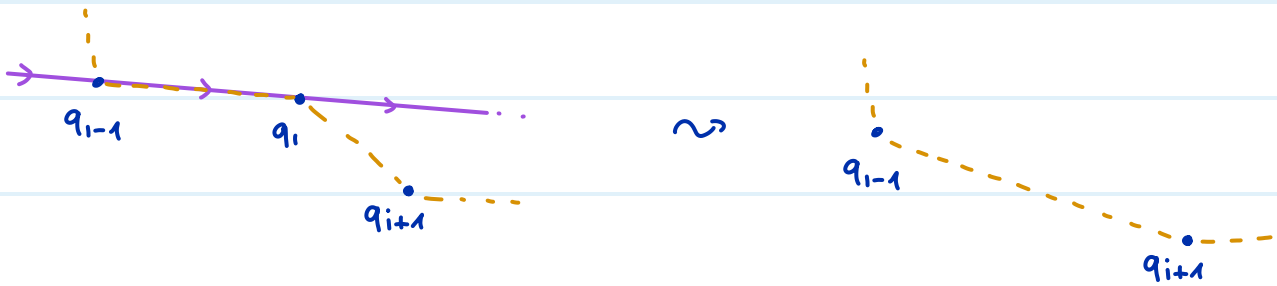
- $h \leq n$
- Für zufällig gleichverteilte Punkte aus einem Quadrat: $\mathbb{E}[h] \leq O(\log n)$

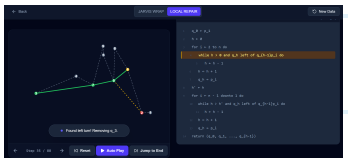
Local Repair:

- Zuerst sortieren wir P nach x-Koordinate aufsteigend: $P = (p_1, p_2, \dots, p_n)$
- Wir beginnen mit dem Polygon $(p_1, p_2, \dots, p_{n-1}, p_n, p_{n-1}, \dots, p_3, p_2)$
- p_1 und p_n sind garantiert Eckpunkte der konvexen Hülle von P
- Wir können in zwei Teilpolygonzüge aufteilen:
  - unterer Teilpolygonzug $(p_1, p_2, \dots, p_n)$, der x-monoton wachsend ist und keine Punkte aus P unter sich hat
  - oberer Teilpolygonzug $(p_n, p_{n-1}, \dots, p_1)$, der x-monoton fallend ist und keine Punkte aus P über sich hat



lokaler Verbesserungsschritt Wir betrachten drei Punkte q_{i-1}, q_i, q_{i+1} und entfernen q_i aus der Folge, falls q_{i+1} rechts von $q_{i-1}q_i$ liegt





LocalRepair$(p_1, p_2, \ldots, p_n)$ in O(n) BEI SORTIERTER EINGABE

▷ setzt $(p_1, p_2, \ldots, p_n)$, $n \geq 3$, nach x-Koordinate sortiert, voraus

```
1:  q_0 ← p_1
2:  h ← 0
3:  for i ← 2 to n do                    ▷ untere konvexe Hülle, links nach rechts
4:      while h > 0 und q_h links von q_{h-1}p_i do
5:          h ← h − 1                    → Verbesserung
6:      h ← h + 1
7:      q_h ← p_i                        ▷ (q_0, ..., q_h) untere konvexe Hülle von {p_1, ..., p_i}
8:  h' ← h
9:  for i ← n − 1 downto 1 do            ▷ obere konvexe Hülle, rechts nach links
10:     while h > h' und q_h links von q_{h-1}p_i do
11:         h ← h − 1                    → Verbesserung
12:     h ← h + 1
13:     q_h ← p_i
14: return (q_0, q_1, ..., q_{h-1})      ▷ Ecken der konvexen Hülle, gg. Uhrzeigersinn
```

Am Anfang: 2 (n-1) Ecken

Am Ende: h Ecken

} $\Rightarrow$ $2(n-1) - h \leq O(n)$ lokale Verbesserungen

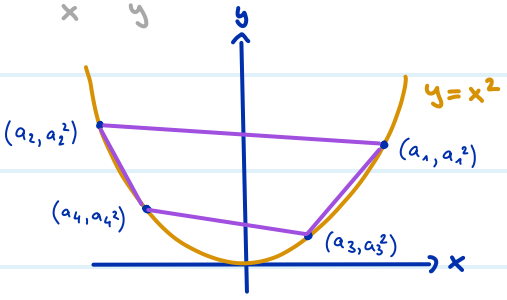
Und für jeden Punkt p_i zwei (einer pro Teilzug) erfolglose Tests $\leq O(n)$

$\Rightarrow$ Laufzeit $O(n)$

Untere Schranke $O(n \log n)$ für convex hull

Wir wollen das Array $(a_1, a_2, \ldots, a_n)$ sortieren

Sei $P = \{(a_i, a_i^2) \mid 1 \leq i \leq n\}$ (a_i: x, a_i^2: y)



Ecken von conv(P) im Gegenuhrzeigersinn

= Sortierung des Arrays

Aufgaben

Sei S eine (möglicherweise unendliche) Teilmenge von $\mathbb{R}^2$.

1. Wenn S endlich ist, dann ist $\text{conv}(S)$ endlich.
   [ ] Wahr
   [x] Falsch

   conv(S) ist nur endlich falls $|S| \leq 1$

2. Der Rand von $\text{conv}(S)$ ist immer ein Polygon. (ein Polygon muss mindestens 3 Kanten haben)
   [ ] Wahr
   [x] Falsch

   counterexample: $|S| = 2$

3. Angenommen, dass S endlich ist und mindestens die Größe $|S| \geq 3$ hat, dann ist es möglich, dass $\text{conv}(S)$ ein Liniensegment ist.
   [x] Wahr
   [ ] Falsch

4. Falls Algorithmus A das ConvexHull-Problem in $O(\log n)$ löst, können wir damit in $O(\log n)$ sortieren.
   [ ] Wahr
   [x] Falsch

   P erstellen dauert $O(n)$

5. Falls Algorithmus A das ConvexHull-Problem in $O(n)$ löst, können wir damit in $O(n)$ sortieren.
   [x] Wahr
   [ ] Falsch

Seien P und Q zwei endliche Teilmengen des $\mathbb{R}^2$.

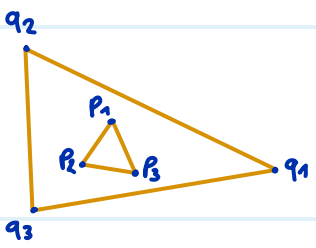
6. Für jede endliche Punktmenge P gilt, dass $x \in \text{conv}(P)$ für alle $x \in P$.
   [x] Wahr
   [ ] Falsch

   per Definition

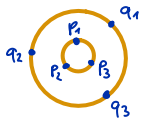
7. $P \subseteq Q \implies \text{conv}(P) \subseteq \text{conv}(Q)$
   [x] Wahr
   [ ] Falsch

8. $\text{conv}(P) \subseteq \text{conv}(Q) \implies P \subseteq Q$
   [ ] Wahr
   [x] Falsch

   counterexample:

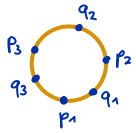


9. $C^\bullet(P) \subseteq C^\bullet(Q) \implies P \subseteq Q$
   [ ] Wahr
   [X] Falsch
   counterexample:

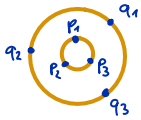


10. $P \subseteq Q \implies C^\bullet(P) \subseteq C^\bullet(Q)$
    [X] Wahr
    [ ] Falsch

11. $C(P) \subseteq C(Q) \implies P \subseteq Q$
    [ ] Wahr
    [X] Falsch
    counterexample:



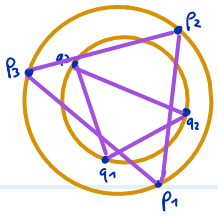
12. $P \subseteq Q \implies C(P) \subseteq C(Q)$
    [ ] Wahr
    [X] Falsch
    counterexample:



$P = \{p_1, p_2, p_3\}$
$Q = P \cup \{q_1, q_2, q_3\}$

13. $\text{conv}(P) \subseteq \text{conv}(Q) \implies C^\bullet(P) \subseteq C^\bullet(Q)$
    [X] Wahr
    [ ] Falsch

14. $C^\bullet(P) \subseteq C^\bullet(Q) \implies \text{conv}(P) \subseteq \text{conv}(Q)$
    [ ] Wahr
    [X] Falsch
    counterexample:



16. Alle Punkte aus P, die auf dem Rand $C(P)$ des kleinsten umschließenden Kreises liegen, sind auch Eckpunkte der konvexen Hülle $\text{conv}(P)$.
    [X] Wahr
    [ ] Falsch

17. Es gilt immer $\text{conv}(P) \subseteq C^\bullet(P)$.
    [X] Wahr
    [ ] Falsch

18. Wenn man die konvexe Hülle einer Punktmenge kennt, kann man ihren kleinsten umschließenden Kreis $C(P)$ berechnen, indem man ausschließlich die Eckpunkte von $\text{conv}(P)$ betrachtet.
    [X] Wahr
    [ ] Falsch

Maxflow Aufgabe (sehr schwierig)

Das Übertragungsnetz (Hochspannung) und Verteilnetz (Niederspannung)
Das gesamte Leitungsnetz ist in zwei verschiedene Netztypen unterteilt. Jeder Leitungsknotenpunkt gehört entweder zum Übertragungsnetz oder zum Verteilnetz.

- Jede Stromleitung **innerhalb** des Übertragungsnetzes hat eine maximale Übertragungskapazität von 30 Megawatt.
- Jede Stromleitung **innerhalb** des Verteilnetzes hat eine maximale Übertragungskapazität von 5 Megawatt.
- Um den Strom nutzbar zu machen, gibt es genau T Transformatoren. Ein Transformator verbindet immer exakt einen bestimmten Knoten des Übertragungsnetzes mit eventuell mehreren Knoten des Verteilnetzes. Strom kann hier nur in eine Richtung fliessen (von der Hochspannung zur Niederspannung). Jeder Transformator t kann maximal c_t Megawatt umwandeln.

Einspeisung

- Es gibt genau **zwei Atomkraftwerke**, A_1 und A_2. Diese erzeugen massive Mengen an Strom und sind daher ausschliesslich mit Knotenpunkten des Übertragungsnetzes verbunden. Das Atomkraftwerk A_1 kann höchstens 100 Megawatt in das Netz einspeisen, und A_2 maximal 150 Megawatt.
- Es gibt S lokale Solaranlagen an Knotenpunkten des Verteilnetzes. Jede speist zwingend exakt 1 Megawatt in ihren jeweiligen Knotenpunkt des Verteilnetzes ein.

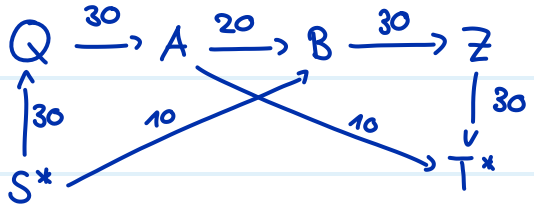
Die Endverbraucher

- Es gibt n Endverbraucher, wobei der i-te Endverbraucher einen Stromverbrauch von e_i Megawatt hat. Jeder Endverbraucher ist über mindestens eine gerichtete Leitung ohne Kapazitätsgrenze mit einem Knotenpunkt des Verteilnetzes verbunden.

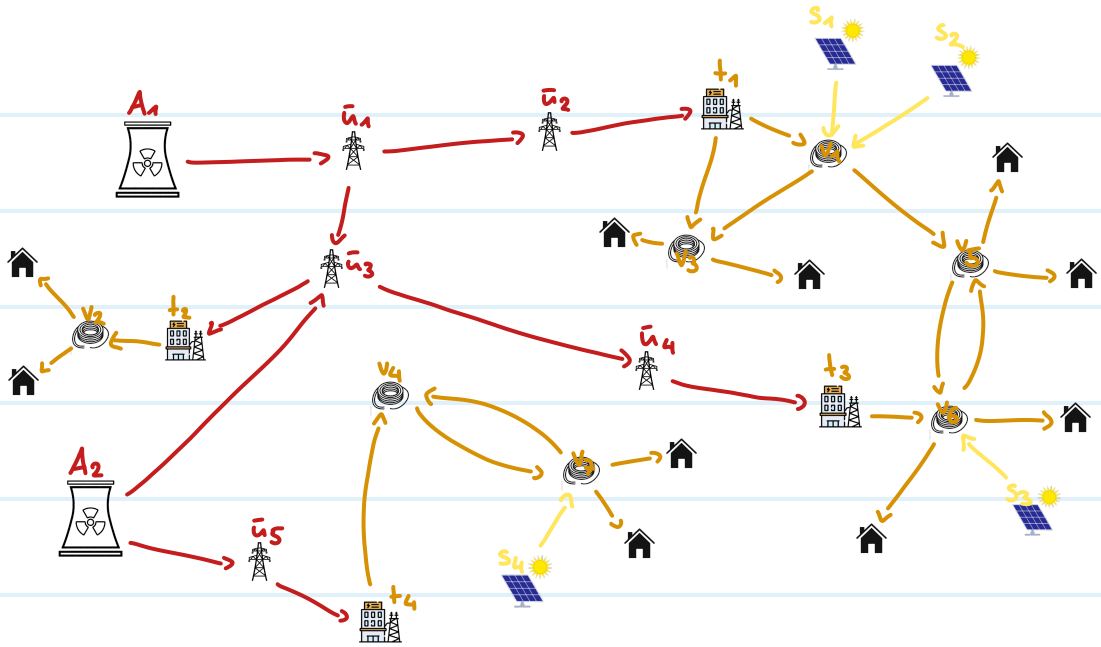
Sicherheitsauflagen

- **Das veraltete Atomkraftwerk A_1:** Dieses Kraftwerk muss zwingend mindestens 20 Megawatt in das Netz einspeisen, sonst explodiert es.
- **Die veraltete Hochspannungsleitung:** Die Hochspannungsleitung von Knoten $\ddot{u}_1$ zu $\ddot{u}_2$ im Übertragungsnetzwerk muss mindestens 10 Megawatt leiten, sonst geht sie kaputt.

Skizziere ein Netzwerk, indem man mit dem maxflow Algorithmus herausfinden kann, ob es eine solche Zuteilung gibt.

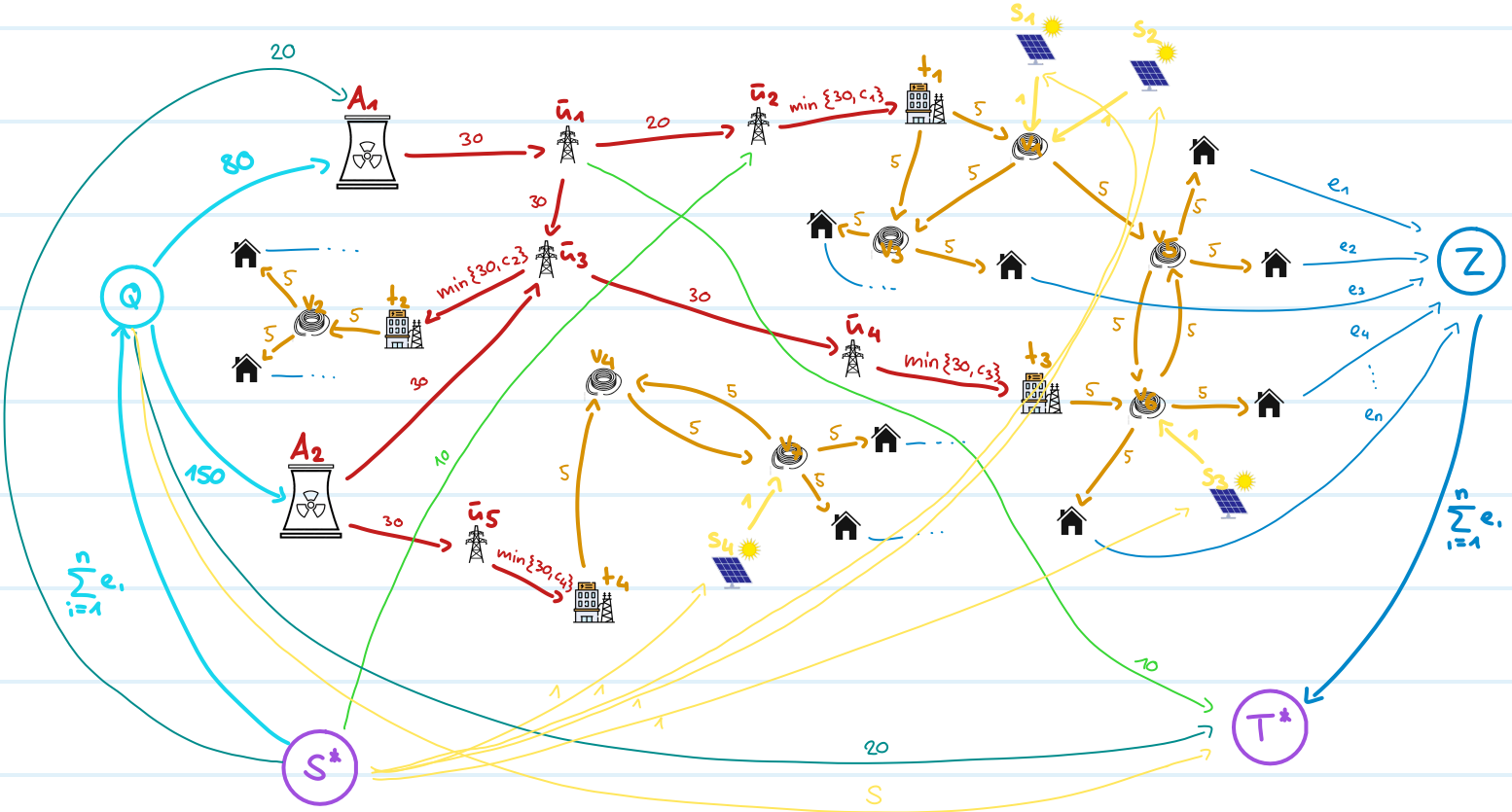


Vorlage:



Verteilnetz, Übertragungsnetz, Solarstrom,

Stromproduktion, Stromverbrauch, A_1 mindestens 20, Leitung $\bar{u}_1 \to \bar{u}_2$ mindestens 10



Zuteilung möglich $\Leftrightarrow$ maxflow von S^* nach $T^* = 10 + 20 + S + \sum_{i=1}^{n} e_i$

Wahrscheinlichkeits - DP

You are finally sitting for the notoriously unforgiving "Advanced Algorithms" exam. The invigilator is pacing the room, and you have exactly T minutes before your paper is forcefully collected.

The exam consists of N sequential tasks. The rules of the exam dictate that you must attempt the tasks in order, and you cannot skip back and forth. For every task i (where $1 \leq i \leq N$), you must decide on your approach before moving to the next one. You have two options for each task:

1. **The Quick Guess:** You spend **1** minute sketching out a brief answer. You have a probability p_i of getting the task entirely correct (earning **1** point).
2. **The Deep Dive:** You spend **3** minutes meticulously deriving the answer. Because you are taking your time, you have a higher probability q_i (where $q_i \geq p_i$) of getting the task correct (earning **1** point).

If you fail to get the task correct under either approach, you earn **0** points for that task. If you run out of time, you automatically score **0** on all remaining tasks.

To pass this dreaded class, you need to score at least K points in total. Because you hacked into the grading server last night (purely for theoretical study, of course), you already know the probabilities p_i and q_i for all N tasks in advance.

You start at task **1** with T minutes on the clock and **0** points. By choosing the optimal approach for each task, what is the maximum probability of you passing the exam (scoring at least K points)?

erster task hat Index 0

Dimension: $DP[0,\dots,N][0,\dots,T][0,\dots,K]$

Teilproblem: $DP[i,t,p]$ = maximale Wahrscheinlichkeit, mindestens p Punkte zu erreichen, startend bei task i (inklusive), gegeben dass wir noch t Minuten Zeit haben

Base Cases: $DP[i,t,p] = 0$ falls $(i = N \vee t = 0) \wedge p > 0$
(keine tasks oder keine Zeit mehr)

$DP[i,t,p] = 1$ falls $p \leq 0$

Rekursion:

$$DP[i,t,p] = \max\{\underbrace{p_i \; DP[i+1,t-1,p-1]}_{\text{Erfolg in kurzer Zeit}} + \underbrace{(1-p_i) \; DP[i+1,t-1,p]}_{\text{Fail in kurzer Zeit}},$$

$$\underbrace{q_i \; DP[i+1,t-3,p-1]}_{\text{Erfolg in langer Zeit}} + \underbrace{(1-q_i) \; DP[i+1,t-3,p]}_{\text{Fail in langer Zeit}}\}$$

$:= 0$ falls $t < 3$

Reihenfolge: TOP-DOWN

Lösung: DP[0, T, K]

Laufzeit: O(N T K)

```java
public static double maxPassingProbability(int N, int T, int K, double[] quick, double[] deep) {
  DP = new double[N + 1][T + 1][K + 1];
  for (int i = 0; i<DP.length; i++) {
    for (int j = 0; j<DP[0].length; j++) {
      for (int k = 0; k<DP[0][0].length; k++) DP[i][j][k] = -1;
    }
  }
  // Lösung:
  return compute(0, T, K);
}

public static double compute(int i, int t, int p) {
  // Base Case:
  if (p <= 0) return 1;
  if ((i == N || t == 0) && p > 0) return 0;

  // Memoization:
  if (DP[i][t][p] != -1) return DP[i][t][p];

  // Rekursion:
  double quickAttempt = quick[i] * compute(i+1, t-1, p-1)
                      + (1-quick[i]) * compute(i+1, t-1, p);

  double deepAttempt = 0;
  if (t >= 3) deepAttempt = deep[i] * compute(i+1, t-3, p-1)
                      + (1-deep[i]) * compute(i+1, t-3, p);

  // Store result in DP table:
  DP[i][t][p] = Math.max(quickAttempt, deepAttempt);
  return DP[i][t][p];
}
```

Prüfungs-Informationen

The exam takes place on **August 10, 2026, from 9:30 to 12:30** during the official exam session and consists of a single 3-hour exam. It will be partially on paper and partially on ETH exam computers. The exam covers all material from the lecture and exercises. Additional topics in the script that are not covered in the lecture or exercises will not be directly examined.

The exam consists of three parts:

- Part A (50% of total points): *Short questions similar to those in the bi-weekly minitests. Some questions may be slightly more challenging, as you will have more time to think about them during the exam.*
- Part B (25% of total points): *Two questions requiring you to write a formal proof on paper.*
- Part C (25% of total points): *Programming questions to be solved on ETH exam computers using Code Expert, similar in style to the weekly programming exercises.*

1) hier anfangen
3.)
2.)

A mock exam is available below and can be seen as a good indicator of how the exam will be structured and what types of questions to expect.

- Mitnehmen: Legi, analoge Uhr, Ohropax, genug Wasser
- Pro-Tipp: Falls ihr eine multiple choice Frage zu Wahrscheinlichkeit nicht wisst, dann simuliert das Zufallsexperiment in code expert (wenn am Ende noch genug Zeit ist)

Lern-Tipps:

- Lernziele für A&W auf josia-heger.com bei Woche 14
- Lieber zu viel lernen als zu wenig
- ~2 Wochen Ferien ist ok
- 6-8 Stunden lernen pro Tag als Richtwert
- Bei PProg und DDCA unbedingt viele alte Prüfungen lösen
- Für Analysis viele Aufgaben lösen (AI ist sehr hilfreich zum erstellen und korrigieren von Aufgaben)
- Macht genug Sport
- PVWs nur wenn ihr sehr Schwierigkeiten habt mit einem Fach