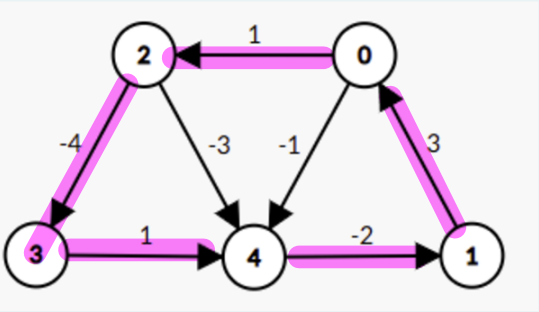
QUIZ-NACHBESPRECHUNG

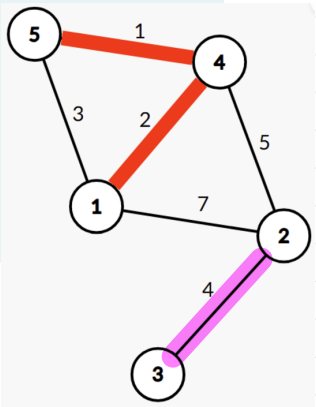


Wahr oder Falsch: Der oben dargestellte gewichtete, gerichtete Graph hat einen negativen geschlossenen Weg.

Bitte wählen Sie eine Antwort:

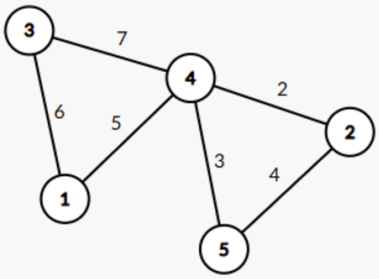
- ○ Wahr
- ○ Falsch

Im oben dargestellten gewichteten Graphen sind die Kanten eines (unvollständigen) minimalen Spannbaums, der während der Ausführung des Algorithmus von Kruskal konstruiert wurde, *dick und rot* markiert.
Welche Kante wird als nächstes hinzugefügt?



Wählen Sie eine Antwort:

- ○ a. $\{1, 2\}$
- ○ b. $\{1, 5\}$
- ✗ c. $\{2, 3\}$
- ○ d. $\{2, 4\}$



Betrachte den oben dargestellten gewichteten, ungerichteten Graphen G.
Wahr oder Falsch: Es existiert ein minimaler Spannbaum (MST) von G, der die Kante $\{2, 5\}$ enthält.

Bitte wählen Sie eine Antwort:

- ○ Wahr
- ✗ Falsch

Sei $G = (V, E)$ ein ungerichteter, gewichteter Graph mit positiven Kantegewichten $w_e > 0$. Sei $e \in E$ eine Kante mit $w_e \leq w_f$ für alle $f \in E$.
Wahr oder falsch: Angenommen, G ist zusammenhängend. Dann muss es einen minimalen Spannbaum (MST) von G geben, der e enthält.

Bitte wählen Sie eine Antwort:

- ✗ Wahr
- ○ Falsch

Sei $G = (V, E)$ ein gewichteter, ungerichteter Graph mit positiven Kantengewichten. Angenommen, G ist zusammenhängend und hat mindestens 2 Kanten.
Die Algorithmen von Kruskal und Prim konstruieren beide einen minimalen Spannbaum, indem sie Kanten nacheinander hinzufügen. Angenommen, beide Algorithmen fügen die gleiche Kante im ersten Schritt hinzu und seien e_{Kruskal} und e_{Prim} die jeweils *zweite* Kante, die von diesen Algorithmen hinzugefügt wird.

Wahr oder falsch: Es muss gelten $w_{e_{\text{Kruskal}}} \leq w_{e_{\text{Prim}}}$.

Bitte wählen Sie eine Antwort:

- ✗ Wahr
- ○ Falsch

KRUSKAL

Idee: · sichere Kanten sortiert nach Gewicht
· nimm günstigste Kante – führt Hinzufügen zu einem Zyklus?
→ja: lösche und wiederhole
→nein: füge MST hinzu, lösche und wiederhole

Kruskal(G)

$F \leftarrow \emptyset$
for $\{u,v\} \in E$, E aufsteigend sortiert
if u und v in verschiedenen ZHKs von (V,F)
$F \leftarrow F \cup \{u,v\}$

Laufzeit:
pro Iteration: $O(n)$
#Iterationen: $O(m)$
sortieren: $O(m \log m)$
insgesamt: $O(nm + m \log m)$

geht es besser?

UNION FIND

Methoden
make(V): erstelle Datenstruktur für $F = \emptyset$
same(u,v): teste ob u und v in selber ZHK von F
union(u,v): vereinige ZHKs von u und v

```
int[] par;
int[] sizeComp;

                                  make(V)
public UnionFind(int n){ ←  implemented in constructor
  //initializes UnionFind with every node as its own parent
  //and each component size = 1
  this.par = new int[n];
  this.sizeComp = new int[n]; // sizeComp[v] = size of ZHK(v)
  for(int i=0; i<n; i++) {
    par[i] = i;
    sizeComp[i]= 1;
  }
}
```

$O(n)$

```
//returns the representant of ZHK(x)
public int find(int x) {  helper method
  if(par[x] == x) return x;
  return find(par[x]);
}
```

$O(\log(n))$

```
                                  same(u,v)
//tests if ZHK(x)==ZHK(y)  ←
public boolean isSameSet(int x, int y){
  return find(x) == find(y);
}
```

$O(\log(n))$

```
//unites ZHK(x) and ZHK(y)   union(u,v)
public void unite(int x, int y){
   int parX = find(x);
   int parY = find(y);
   if(sizeComp[parX] > sizeComp[parY]){
     par[parY] = parX;
   } else if (sizeComp[parX] > sizeComp[parY]) {
     par[parX] = parY;
   } else {
     par[parY] = parX;
     sizeComp[parX]++;
   }
}
```

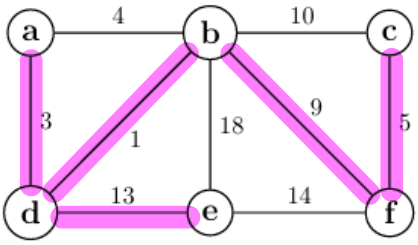
warum testen wir hier immer welche Komponente grösser ist?
→ $O(\log(n))$

Laufzeit Kruskal: $O(n \log n + m \log m)$
Union Find · Sortieren der Kanten

Aufgabe: finde MST mit Kruskal

/ 2 P

e) *Minimum Spanning Tree:* Consider the following graph:



i) Highlight the edges that are part of the minimum spanning tree. (Either in the picture above, or you can recreate the graph below).

TIPPS für die Lernphase

- überlegt euch vorher, was ihr bis wann geschafft haben möchtet
- macht euch Themenübersichten für die Fächer und überlegt, was ihr priorisieren möchtet
- plant Lernpuffer ein
- alte Prüfungen sind key
- fangt früh an, nicht erst eine Woche vor der Prüfung
- es wird gute und schlechte Tage geben → nicht aufgeben!
- sucht euch einen Ausgleich, z.B. Sport
- persönliche Empfehlung: - in jedem Fach zuerst eine Prüfung lösen, um ein Gefühl zu bekommen, was erwartet wird
  - ca. 1 Woche Theorierecap
  - dann alte Prüfungen und Übungen
- PVWs nur, wenn ihr wirklich denkt es ist wichtig (nicht aus FOMO)
- persönlich profitiere ich viel vom Prüfungen alleine lösen, diese dann aber zu peergraden
- Schlaf-gerade vor den Prüfungen- ist in der Prüfung wichtiger als noch 1h mehr zu lernen
- nehmt eine analoge Uhr mit zu den Prüfungen

A&D spezifisch

- Leetcode ist nicht alles → nicht zu stark priorisieren (meine Meinung)
- DP-Aufgaben sind oft nur Abwandlungen von Problemen, die ihr schon gesehen habt
  → schaut euch die gut an
- übt CodeExpert Graphenaufgaben, man braucht normalerweise nur simple Algorithmen wie BFS und DFS
  → jedes Jahr sehr ähnlich (fast gleiches Pattern)
- kein Cheatsheet bedeutet auswendig lernen bringt oft Punkte
  → lernt Such- und Sortieralgorithmen, Rechenregeln für O-Notation, Laufzeiten für SP- oder MST-Algorithmen, etc. auswendig

in der Prüfung

- achtet bei der Theorieprüfung auf die Zeit (meist knapp) und hängt euch nicht an einzelnen Aufgaben auf
- in der Programmierung würde ich mir kurz die DP anschauen und dann mit der Graphenaufgabe anfangen
  → gibt Teilpunkte, DP ist oft 16 oder 0 Punkte
  → man gewinnt etwas Sicherheit und vermeidet Panik in der Prüfung, falls man nach 1,5h die DP-Aufgabe noch nicht gelöst hat
- passt auf bei MC-Aufgaben → oft Abzug für falsche Antworten

Liste der wichtigsten Themen (kein Anspruch auf Vollständigkeit)

- [ ] Beweis durch vollständige Induktion/Invariantenbeweise
- [ ] asymptotische Notation
- [ ] Code Snippets für Laufzeitbestimmung
- [ ] Maximum Subarray Sum
- [ ] Suchalgorithmen: lineare Suche, binäre Suche, untere Schranke fürs Suchen
- [ ] Sortieralgorithmen
  - [ ] Invarianten
  - [ ] Bubble sort
  - [ ] Selection sort
  - [ ] Insertion sort
  - [ ] Merge sort
  - [ ] Heap sort
  - [ ] Quick sort
  - [ ] untere Schranke fürs Sortieren
- [ ] Heap
- [ ] ADT: Liste
- [ ] Binärbaum
- [ ] AVL-Baum
- [ ] DP
  - [ ] Fibonacci
  - [ ] Memoization
  - [ ] bottom-up
  - [ ] Jump Game
  - [ ] Longest Common Subsequence
  - [ ] Edit Distance
  - [ ] Subset Sum
  - [ ] Knapsack
  - [ ] Longest Increasing Subsequence
  - [ ] Approximationsalgorithmus
- [ ] NP=P
- [ ] Definitionen für Graphentheorie
- [ ] Eulerzyklus und andere Graphentheoreme
- [ ] Topologische Sortierung
- [ ] Adjazenzliste vs. Adjazenzmatrix
- [ ] DFS/Tiefensuchbaum/Kantenklassifizierung
- [ ] BFS
- [ ] SP
  - [ ] SP-Tree
  - [ ] Dijkstra
  - [ ] Bellman-Ford
  - [ ] Floyd-Warshall
  - [ ] Johnson
- [ ] MST
  - [ ] Prim
  - [ ] Kruskal
  - [ ] Boruvka

We say that an integer $n \in \mathbb{N}$ is a *k-sum* if it can be written as a sum $n = a_1^k + \cdots + a_p^k$ where $a_1, \ldots, a_p$ are distinct natural numbers, for some arbitrary $p \in \mathbb{N}$.

For example, 36 is a *3-sum*, since it can be written as $36 = 1^3 + 2^3 + 3^3$.

Describe a DP algorithm that, given two integers n and k, returns True if and only if n is a k-sum. Your algorithm should have asymptotic runtime complexity at most $O(n^{1+\frac{2}{k}})$.

Hint: *The intended solution has complexity* $O(n^{1+\frac{1}{k}})$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Definition of the DP table*: What is the meaning of each entry?
3. *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

1) $m = \lfloor \sqrt[k]{n} \rfloor = \lfloor n^{\frac{1}{k}} \rfloor$
DP[0..m][0..n]

2) DP[i,j] = mit ersten i Zahlen können wir eine k-Summe mit $\{a_1, \ldots a_p\} \subseteq \{1, \ldots, i\} \cdot j = a_1^k + \ldots + a_p^k$
wert j erreichen

3) DP[i,j] = DP[i-1,j] ∨ DP[i-1, j-i^k]
false, false $j < i^k$

DP[0,j] = 0 $\quad 0 < j \leq n$

DP[i,0] = 1 $\quad 0 \leq i \leq m$

4) for i=0..m
for j=0..n
compute DP[i,j]

5) DP[m,n]

6) $O(mn) = O(n^{\frac{1}{k}} n) = O(n^{1+\frac{1}{k}})$

A number n of animal species have been recently discovered in Africa. The zoo of Zürich is interested in acquiring as many animals from the new species as possible before a special exhibition that is taking place on December 1st, and you were put in charge of this task. Because of the time constraint, you can only organize one shipping of animals. The shipment can hold a maximum total weight of W. Furthermore, due to logistical constraints, you cannot isolate the animals during the shipment. Therefore, you cannot simultaneously bring two animals where one of them is a predator of the other.

Let $A_1, \ldots, A_n$ be the $n > 4$ discovered species. You know that the species A_1, A_2 and A_3 are not predators, but for $4 \leq i \leq n$, the species A_i is a predator of only the species A_{i-1}, A_{i-2} and A_{i-3} (this means that, for example, A_i it is not a predator of species A_{i-4} or A_{i+1}).

For every $1 \leq i \leq n$, an animal from the species A_i has weight $w_i > 0$, and provides a value $v_i > 0$ to the zoo. You would like to figure out the collection of animals that you can bring to the zoo, and which provides the maximum total value to the zoo. We assume that $(w_i)_{1 \leq i \leq n}$ and W are all positive integers. If you bring one animal from a species, then bringing another animal from the same species does not provide any additional value to the zoo. Therefore, there is no point in bringing two or more animals from the same species.

Provide a *dynamic programming* algorithm that solves this problem. The input to your algorithm are the weights $(w_i)_{1 \leq i \leq n}$ and values $(v_i)_{1 \leq i \leq n}$ of the animal species, and the maximum total weight W that is allowed in one shipping. In order to get full points, the runtime of your algorithm should be $O(nW)$.

Address the following aspects in your solution:

1. *Dimensions of the DP table*: What are the dimensions of the table $DP[\ldots]$?
2. *Definition of the DP table*: What is the meaning of each entry?
3. *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the final solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

1) $(n+1) \times (W+1)$ $\quad DP[0 \ldots n][0 \ldots W]$

2) $DP[i,j]$ = max. Profit erreichbar mit Spezien $A_1, \ldots, A_i$ und Gewichtsbeschränkung j

3) $n \geq i \geq 4;\ 0 \leq j \leq W$

$DP[i,j] = \max\{ DP[i-1,j],\ \underbrace{DP[i-4, j-w_i] + v_i}_{=0 \text{ falls } j < w_i} \}$

$1 \leq i \leq 3$ (Base Cases)

$DP[i,j] = \max\{DP[i-1,j],\ \underbrace{DP[i-1, j-w_i] + v_i}_{=0 \text{ falls } j < w_i}\}$

$DP[0,j] = 0 \quad$ für $0 \leq j \leq W$

4)
```
for i = 0 ... n
    for j = 0 ... W
        compute DP[i,j]
```

5) $DP[n,W]$

6) $O(nW)$ + justification