Dynamic Programming

=> Problem: Rekursion berechnet das gleiche Resultat wieder und wieder => sehr ineffizient

=> DP-Lösung: Berechnungen für jedes Teilproblem speichern und merken => wiederverwenden, wenn dieses Teilproblem zur Lösung eines Anderen beiträgt.

→ Vorgehen:

0. Definiere Teilprobleme
1. Rekursion:
   verbinde die Teilprobleme durch Aufzählen lokaler Eigenschaften
2. Bestimme Datenstruktur und die Abhängigkeiten der Teilprobleme
3. Lösen des Problems

1. **DP-Tabelle** mit Lösung zu den Teilproblemen
   Dimension der Tabelle? Bedeutung der Einträge?
2. Berechnung der **Randfälle**
   Welche Einträge hangen nicht von anderen ab (triviale Lösung)?
3. **Berechnungsreihenfolge** bestimmen
   In welcher Reihenfolge können Einträge berechnet werden, so dass benötigte Einträge jeweils vorhanden sind?
4. Auslesen der **Lösung**
   Wie kann sich Lösung aus der DP-Tabelle konstruieren lassen?

Laufzeit: (typisch) Anzahl Einträge der Tabelle mal Aufwand pro Eintrag.

Zuerst:

i) Teilprobleme erkennen / definieren → e.g. some von ... bis stelle i, max, min etc

ii) Teilprobleme mit einander verbinden → Rekursion
e.g. $x_{n+1} = f(x_n)$

iii) Datenstruktur und Abhängigkeit der Teilprobleme untereinander bestimmen. → meist eine Tabelle d.h. Liste oder Array

Dann:

1) DP Tabelle definieren → Tabelle T → oft 1 x n oder n x n
   ↳ was sind dimensionen der Tabelle?
   ↳ Was ist die Bedeutung eines Eintrags? (→ siehe (i): es hilft immer wenn klar ist was man tatsächlich berechnen muss und was nicht)

2) Berechnungsregel/vorschrift und Randfälle
   (welche Einträge hängen nicht von den anderen ab)

3) Berechnungsreihenfolge* und Startpunkt d. Berechnung → meist/immer klein → gross bzgl. Problemgrösse
   ↳ In welche Richtung muss berechnet werden damit immer alle benötigten (vorhergehenden) Werte vorhanden sind?

4) Auslesen der Lösung
   ↳ wie kann die gesuchte Lösung aus der Tabelle rekonstruiert werden, oder steht sie schon irgendwo? → ist die Lösung einfach ein Wert (e.g. letzter eintrag) oder müssen wir sie aus der Tabelle rekonstruieren
   ↳ also: was ist tatsächlich die gesuchte Lösung?

* Richtung mit der ihr durch die DP Tabelle und Richtung mit der ihr durch eine allfälligen Datenstruktur der Problemstellung iteriert muss nicht unbedingt gleich sein. → hängt auch davon ab wie ihr T definiert.

02.05.2023

Tabulation vs. Memoization

⇒ Gleiche situation: Viele teure Fkt calls / Berechnungen.
Berechnungen merken und wiederverwenden.

→ Tabulation und Memoization sind dabei zwei verschiedene Ansätze.

Tabulation and memoization are two techniques used in dynamic programming to optimize the execution of a function that has repeated and expensive computations.

→ Memoization:

Memoization is a top-down approach where we cache the results of function calls and return the cached result if the function is called again with the same inputs. It is used when we can divide the problem into subproblems and the subproblems have overlapping subproblems. Memoization is typically implemented using recursion and is well-suited for problems that have a relatively small set of inputs.

→ top-down, oft rekursiv

→ Tabulation:

Tabulation is a bottom-up approach where we store the results of the subproblems in a table and use these results to solve larger subproblems until we solve the entire problem. It is used when we can define the problem as a sequence of subproblems and the subproblems do not overlap. Tabulation is typically implemented using iteration and is well-suited for problems that have a large set of inputs.

→ bottom-up, oft iterativ

Bsp. https://www.geeksforgeeks.org/tabulation-vs-memoization/