Dynamische Programmierung

Enes Yilmaz, Demir Dolovac

Inhaltsverzeichnis

Ursprung

- Wurde in den 1950ern von Richard Bellmann entwickelt
- Nutzung auf dem Gebiet der Regelungstheorie
- Optimalitätsprinzip von Bellman: Um die optimale Lösung des Gesamtproblems zu erhalten, müssen die Teillösungen der Teilprobleme auch optimal sein.

Wofür und wie wird dyn. Programmierung eingesetzt?

- Ist eine Methode, die für das Lösen von Optimierungsproblemen verwendet wird
- Problem wird in kleine Teilaufgaben zerlegt
- Teillösungen werden abgespeichert
- Teillösungen werden für die Gesamtlösung verwendet
- Eignet sich zum Lösen einiger NP-Schweren Problemen

Anwendungsbereiche

- Routenplanung: Bellman-Ford-Algorithmus, berechnet iterativ den kürzesten Knoten zu allen anderen Knoten und speichert diese ab.

- Spieltheorie: optimiert Entscheidungen, indem mögliche Spielverläufe in kleinere Teilprobleme unterteilt und die besten Strategien speichert.

- KI/Machine Learning: Wird besonders beim Reinforcement Learning verwendet, um die optimalen Aktionen für einen Agenten zu ermitteln.

- Textverarbeitung: angewandt bei Autokorrektur oder Suchvorschläge wie z.B in Google. Dies wird z.B mit der Levenshtein-Distanz realisiert

Bottom-Up-Ansatz (Tabulation)

- Teilprobleme werden iterativ der Größe nach (von klein nach groß) gelöst
- Jede Lösung wird in einer Tabelle gespeichert, um später darauf zugreifen zu können
- Die Lösung des Gesamtproblems ist die Kombination der zuvor berechneten Lösungen der Teilprobleme
- Vorteil:
  - Schneller als Memoisierung
- Nachteil
  - oft komplizierter Tabulation-Lösungen für komplexe Probleme zu entwerfen

Top-Down-Ansatz (Memoisierung)

- Problem wird in kleinere Teilprobleme zerlegt, die rekursiv gelöst werden
  - Rekursion bietet eine Möglichkeit, Probleme schrittweise aufzuteilen und die Lösung in einer strukturierten Weise zu erreichen.

- Jede Lösung eines Teilproblems wird berechnet und gespeichert

- Bereits berechnete Lösungen werden bei späteren Bedarf wiederverwendet

- Vorteil:
  - Wenn das Problem eine rekursive Struktur bietet, lässt es sich simpler mit Memoisierung lösen
  - Effizienter wenn nicht alle Teilprobleme Benötigt werden
- Nachteil:
  - Kann mehr Speicher und Zeit kosten, wenn viele Rekursionsaufrufe gemacht werden
  - Schlecht für größere Eingabe Daten, zu Tiefe Rekursion

Programmbeispiele

- Fibonacci
- Rucksackproblem
- Subset-Sum Problem
- Levenshtein-Distanz von Strings

- Link zum GitHub für die Programmierbeispiele: https://github.com/ebossy/DynamischeProgrammierung

Vergleich zu anderen Programmierungsmethoden

Greedy:

- + schnell
- - befriedigende Lösung

Divide & Conquer:

- + Gut, wenn die Teilprobleme unabhängig voneinander sind
- - Schlecht, wenn es eine Überlappung der Teilprobleme gibt

Dyn. Programmierung:

- + optimale Lösung
- + Effizienter für überlappende Teilprobleme
- - längere Laufzeit

Fragen?