Backtracking

Klaus Kusche

Inhalt

- Anwendungsbereich
- Idee
- Programmgerüst
- Beispiele

Anwendungsbereich

Wofür?

Such- und Optimierungsprobleme,
deren Lösung aus *Einzelschritten* / Einzelentscheidungen zusammengesetzt ist.

Wie?

Systematisches *Durchprobieren aller Möglichkeiten,*
aber bei "sinnlosen" Teilschritten oder Sackgassen
gleich umkehren, restliche Schritte gar nicht probieren

==> *1 Schritt zurück,* deshalb "**Backtracking**".

==> Viele *sinnlose Kombinationen* werden
gar nicht berechnet!

Idee: Rekursion

Grundidee:

- *Jede Ebene der Rekursion / jeder Aufruf probiert alle Möglichkeiten für einen Teilschritt*
- … und macht *für jede sinnvolle Möglichkeit (und nur für diese!) einen rekursiven Aufruf* zur Lösung der *restlichen Teilschritte*.

Weiterentwicklungen:

u.a. *Alpha-Beta-Algorithmus* für 2-Personen-Spiele (Grundlage aller Schachprogramme)
= Backtracking + Bewertungsstrategie

Beispiele (1)

- 8-Damen-Problem
- Rucksack packen
- Geld herausgeben
- Weg im Labyrinth
- Landkarte färben
- Manche Rätsel (Sudoku, Wortsumme, ...)

Beispiele (2)

Was ist "**ein Teilschritt**" bzw. **der i-te Schritt**?

- 8-Damen-Problem:
  Setze die i-te Dame in Zeile i.
- Rucksack packen:
  Entscheide den i-ten zur Wahl stehenden Gegenstand.
- Geld herausgeben:
  Gib die i-te Münzsorte heraus.
- Weg im Labyrinth:
  Mach einen Schritt auf's nächste Feld.
- Landkarte färben:
  Färbe das i-te Land.

Beispiele (3)

Was sind die "**alle Möglichkeiten**" im i-ten Teilschritt?

- 8-Damen-Problem:
  i-te Dame in die 1., 2., ..., 8. Spalte setzen?
- Rucksack packen:
  i-ten Gegenstand einpacken oder zu Hause lassen?
- Geld herausgeben:
  0, 1, 2, ... Stück der i-ten Münzsorte herausgeben.
- Weg im Labyrinth:
  i-ter Schritt links, rechts, rauf oder runter.
- Landkarte färben:
  i-tes Land rot, grün, blau, ... anmalen.

Programmgerüst (1)

Funktion *"Mache den i-ten Schritt"*:

```
if Ziel erreicht / letzter schon Schritt gemacht
then Drucke Lösung
else
   for alle Möglichkeiten für den i-ten Schritt
      if Möglichkeit ist zulässig
         Speichere den aktuellen i-ten Schritt in der Lösung
         Mache rekursiv den (i+1)-ten Schritt
         Ev.: Lösche den i-ten Schritt wieder aus der Lösung
      else
         Ignoriere die Möglichkeit
```

Programmgerüst (2)

Programm wie auf der vorigen Seite
==> Findet automatisch ***alle*** möglichen Lösungen!

Nur ***eine*** (die *erste*) Lösung gesucht:

==> Nachdem die erste Lösung gefunden ist,
müssen *alle* rekursiven Aufrufe *sofort zurückkehren*!

Am besten:

- **`bool`**-Returnwert für *"Lösung gefunden"*
  ==> Nach Finden/Speichern/Drucken einer Lösung
  kehrt der *innerste* Aufruf sofort mit **`true`** zurück

- Jeden *rekursiven Aufruf* mit **`if`** *prüfen*
  ==> Kehrt der rekursive Aufruf mit **`true`** zurück,
  auch sofort mit **`true`** *zurückkehren*

Programmgerüst (3)

Die ***beste*** *Lösung* gesucht:

Verwendet am besten eine *globale Variable*
zum Speichern der bisher besten gefundenen Lösung!

Statt "*Drucke Lösung*":

==> Neue Lösung mit der bisher besten gefundenen *vergleichen*.

==> Wenn besser: Neue Lösung als beste *speichern*.

==> Ganz am Schluss (nach Rückkehr aus dem Backtracking)
im **`main`** die gespeicherte Lösung ausgeben (ist die beste).

Code 8 Damen

```
void probier(int brett[ANZAHL], int zeile)
{
  if (zeile == ANZAHL) {
    drucke(brett);
  } else {
    // setze die Dame in Zeile Nummer zeile
    int spalte;
    for (spalte = 0; spalte < ANZAHL; ++spalte) {
      if (ok(brett, zeile, spalte)) {
        brett[zeile] = spalte;
        probier(brett, zeile + 1);  // Rekursion!
      }
    }
  }
}
```

Code Labyrinth

```
void probier(int i, int j)
{ // feld[][] sei global
  if (feld[i][j] == ZIEL)
    loesung();
  else if (feld[i][j] == FREI) {
    feld[i][j] = WEG;
    probier(i, j + 1);
    probier(i + 1, j);
    probier(i, j - 1);
    probier(i - 1, j);
    feld[i][j] = FREI;
  }
}
```

Code Geldwechseln

```
void probier(int sorte, int betrag)
{ // vorrat[], wert[] und anzahl[] global
  if (betrag == 0) {
    loesung();
    return;
  }
  if (sorte == SORTEN) return;  // Keine Lösung!
  for (int anz = 0; (anz <= vorrat[sorte]) &&
            (anz * wert[sorte] <= betrag); ++anz) {
    anzahl[sorte] = anz;
    probier(sorte + 1, betrag - anz * wert[sorte]);
  }
  anzahl[sorte] = 0;
}
```