Modul Su2 – Suche in ungeordneter linearer Struktur

(Suche in ungeordnetem Array,
Suche in ungeordneter verketteter Liste)

Zeitrahmen
40 Minuten

Zielgruppe
- Volksschule;
- Sekundarstufe I

Inhaltliche Voraussetzung
Keine

Lehrziel
Erkennen des Wertes von Struktur im Problemraum und Nutzen dieser Struktur.
Je nach Verbindung mit anderen Suchverfahren:
- in Verbindung mit ASu1 – blinder Suche: Erkennen, dass durch die leichte Vorstrukturierung des Problemfeldes (Experimentalgruppe ist in einer linearen Anordnung aufgestellt), das Suchproblem von einem komplexen zu wiederholenden paarweisem Vergleich auf eine lineare Suche reduziert werden konnte.
- in Verbindung mit ASu3 – Binärsuche: Erkennen, dass das Fehlen einer über die lineare Anordnung hinausgehenden Strukturierung im Problemraum nur systematisches Durchprobieren erlaubt. Dieses aber doch bereits effizienter ist als blinde Suche.

Motivation
Entwickeln algorithmischen Problemlösens in einfachen Problemdomänen.

Requisiten
bei Geburtstagssuche: keine
Varianten: Messwert, Glücksklee, …: gefaltete Blätter mit eingetragenem Messwert oder gesuchtem Symbol, Spielkarten mit „Schwarzer Peter“, … .

Partizipanden

Experimentalgruppe:	n 5 Jugendliche
Algoritm. Unterstützung:	1 Laufvariable (suchende Person) 1 Index oder Zeiger auf aktuell besten Zwischenwert, 1 Speicher für bisher besten Zwischenwert *(mit Bleistift und Papier ausgestattet, um aktuelle Werte zu notieren)*
Beobachtungs(gruppe):	
einfachste Variante:	Zähler, der die von der Laufvariablen ausgeführten Operationsschritte zählt.
verfeinerte Variante:	je ein Zähler für sämtliche Operationsarten der Akteure und ein weiterer Zähler für Abfrageoperationen. *(Bei Vorbereitung auf div. Sortierverfahren ist die verfeinerte Variante sinnvoll. Sonst reicht wohl die einfache Variante).*

Vorgehensweise

1. Experimentalgruppe stellt sich wie im Turnunterricht in einer nach Größe geordneten Riege auf.
2. Index-Person wird vor 1. Person in Experimentalgruppe positioniert.
3. Speicherperson wird Geburtstag der 1. Person mitgeteilt.
4. Laufvariable stellt sich vor 2. Person.
5. Laufvariable erhebt von der vor ihr stehenden Person den Geburtstag.
6. Speicherperson prüft ob dieser näher zum heutigen Tag ist, als der eben gespeicherte (wenn keine Speicherperson bestimmt wurde, dann unmittelbarer Vergleich mit dem Geburtstag der Person, vor der die Index-Person steht).
7. WENN Prüfung 6. erfolgreich ist:
   Index-Person wandert vor jene Person, vor der die Laufvariable steht;
   Speicherperson merkt sich den neuen „nächsten Geburtstag"
   SONST: ---
8. SOLANGE noch eine Person vorhanden:
   Laufvariable schreitet zur nächsten Person und setzt Verfahren bei Schritt 6. fort.
9. Elementarverfahren ist beendet. Dies bedeutet:
   Die Index-Person steht vor Person mit dem aktuellstem Geburtstag!

Fortsetzung des Algorithmus (des Verfahrens), wenn nach den *k* aktuellsten Geburtstagen gesucht werden soll:

10. Person mit aktuellstem Geburtstag tritt zurück, sodass eine sichtbare Lücke entsteht.
11. Index-Person tritt vor die nunmehr 1. Person der reduzierten Experimentalgruppe, Elementarverfahren wird bei Schritt 2 wieder aufgenommen. Lücke durch zurückgetretene Person(en) wird übersprungen.
12. Fortsetzung so lange, bis *k* Personen gefunden, also zurückgetreten sind.

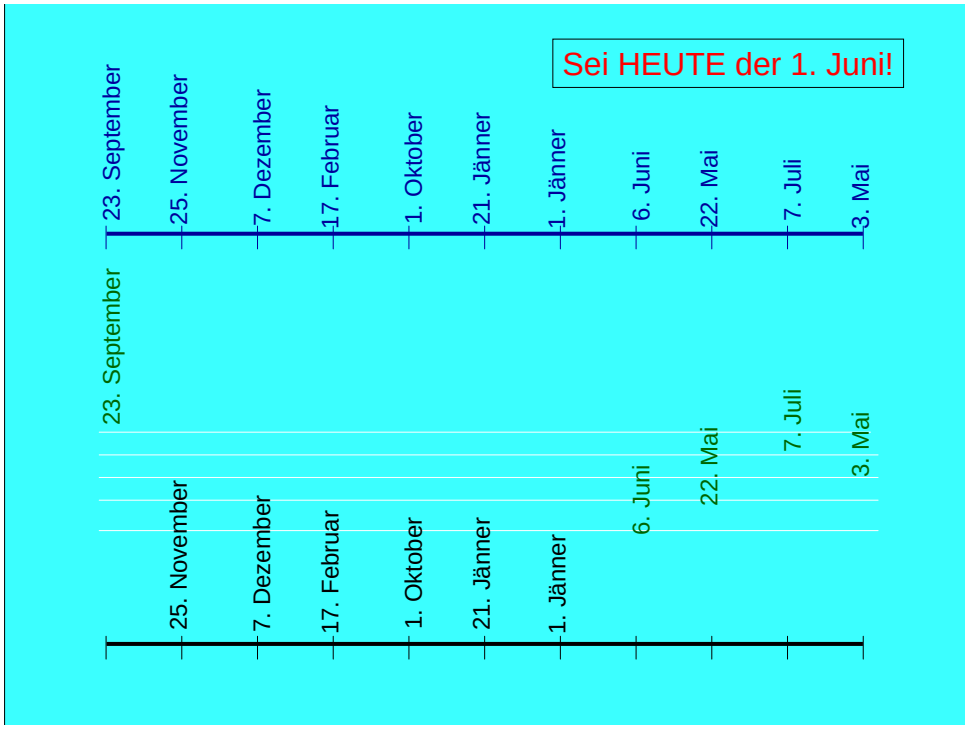
Hierbei ist wesentlich, dass das gesamte Verfahren jeweils neu durchgeführt wird, also dass der „Speicher" jedes Mal mit einem neuen, blanken Blatt Papier seine Arbeit beginnt.

Analyse des Algorithmus

- Welche Beobachtungen wurden von der Beobachtungsgruppe gemacht:
  - Wie oft musste der Suchende (Laufvariable) fragen?
  - Wie viel Einzelschritte wurden durchgeführt?
  - Wie lassen sich diese Beobachtungen auf die Zahl der Personen in der Experimentalgruppe umlegen? Wie würden die Zahlen aussehen, wenn in der Experimentalgruppe 10, 100, 1000 Personen gestanden wären?
  - Wie viel Schritte wären nötig, wenn man alle Geburtstage gesucht hätte?

- Welche Struktur wäre entstanden, wenn die zurücktretenden Personen nicht jeweils einen Schritt zurückgetreten wären, sondern jene Person, die im *k*-ten Versuch gefunden wurde, wäre *k* Schritte zurückgetreten?

(ggf. durch Tafelskizze unterstützen)



- Feststellen, dass der Aufwand dieses Verfahrens direkt von der Zahl der vorhandenen Personen abhängt. Wir müssen also, wenn *n* Personen vorhanden sind, alle *n* nach ihrem Geburtstag fragen. Wir sagen daher, dass der Aufwand des Elementarverfahrens, der Suche nach der Person mit dem nächsten Geburtstag, von der Größenordnung *n* ist bzw. O(n) Schritte benötigt. Die Suche nach *k* Personen erfordert, dass wir die Riege *k* mal durchfragen. Dies bedeutet, dass der Gesamtaufwand von der Ordnung k*O(n) ist. Wollen wir Vollsortierung erzielen, müssen wir das Verfahren für alle *n* Teilnehmer wiederholen, also beträgt er Aufwand n*O(n), also $O(n^2)$ Schritte.

An dieser Stelle mag eine Person richtiger Weise feststellen, dass bei jedem dieser Schritte die Riege ja um eine Person kürzer wird. Wir suchen daher zuerst unter n, dann unter (n-1), unter (n-2), …, schließlich nur mehr unter 2 Personen und im n-ten Verfahrensschritt bleibt nur eine Person mehr übrig, die als letzte selbstverständlich den am weitesten entfernten Geburtstag hat, also gar nicht mehr gefragt werden müsste.

*Dies ist richtig. Bei exakterer Berechnung müssten wir feststellen, dass für die Vollsortierung n*n/2 Schritte nötig sind. Da jedoch bei Aufgaben großen Umfangs das „n" sich viel stärker im Ergebnis niederschlägt als jeder beliebige konstante Faktor, sieht die O-Notation über konstante Glieder großzügig hinweg (siehe auch Glossar).*

Eine andere Frage könnte lauten: Ist es nicht selbstverständlich, dass wir jede Person fragen mussten? *Die Antwort darauf lautet: Ja, in diesem Fall schon, weil wir nicht mehr Information zur Verfügung hatten. Allerdings werden wir in Modul Su3 – Binärsuche eine Situation kennenlernen, in der wir zum Ergebnis kommen, ohne alle gefragt zu haben. – Wie könnte das gelingen?*

- Skizzierung des Algorithmus in Pseudocode mit expliziter Schleifenstruktur und Erkennen der O(n) Schritte für die Suche einer Person bzw. O(k*n) für *k* Personen aus der Schleifenstruktur. Würde man der Reihe nach aus den jeweils verbliebenen Personen jene mit dem nächstliegenden Geburtstag suchen wären mithin O(n*n), also $O(n^2)$ Schritte erforderlich.

  Unter Pseudocode versteht man eine programmiersprachenähnliche Notation. In Modul P2 wurden die Strukturierungselemente von Pseudocode eingeführt. Dort haben wir innerhalb dieser Konsturkte (geschweifte Klammern und Schlüsselworte wie WENN, DANN, SOLANGE und MACHE noch freisprachlichen Text geschrieben. Der unten angegebene Pseudocode hat bereits Prozedurköpfe und logische Operatoren (z.B. UND). Dies ist für Kinder durchaus bereits lesbar. Problematisch ist vielleicht der Begriff INDEXTYP. Er wäre, gegebenenfalls durch den Begriff „Position in der Riege" zu ersetzen. Ebenso ist „Feld" bereits ein Fachbegriff der ggf. durch „Riege" ersetzt werden könnte.

  Wir entschieden uns jedoch für diese Begrifflichkeit, damit Klassen, die dieses Beispiel durch Entwicklung eines Programms fortsetzen wollen, den Weg dorthin erkennen, ohne sich sofort mit der Detailsyntax ihrer Programmiersprache auseinandersetzen zu müssen.

  Es mag sinnvoll sein, vor Skizzierung des Pseudocodes für das gestellte Problem (nächster Geburtstag) den Pseudocode für die Suche nach einem konkreten Geburtstag anzugeben. Auf der Grundlage dieses Pseudocodes und des durchgespielten Algorithmus sollte es kein Problem sein, den Pseudocode für das durchgespielte Problem zu entwickeln. Dies könnte man gleich mit der folgenden Frage einleiten:

  Ändert sich der Aufwand des Verfahrens, wenn wir nicht nach der Person mit dem nächstliegenden Geburtstag suchen, sondern nach jener, die einen konkreten Geburtstag hat?

Pseudocode für lineare Suche nach konkretem Wert in einem Array:

```
SucheLW (gesuchterWert, Feld, anfang, ende) : suchergebnis {
VAR:  sucher: INDEXTYP {

      suchergebnis := „nicht gefunden";
      sucher := anfang;
      SOLANGE sucher ≤ ende UND Feld[sucher] ≠ gesuchterWert
         sucher := sucher + 1;
      WENN Feld[sucher] = gesuchterWert DANN
         suchergebnis := sucher
}}.
```

Pseudocode für lineare Suche nach Wert mit minimaler Abweichung von einem Zielwert in einem Array:

```
SucheLMin (zielWert, Feld, anfang, ende) : suchergebnis {
VAR:  sucher: INDEXTYP;
      bisherbesterEintrag: INDEXTYP;
      abweichung: FELDELEMENTTYP {

      abweichung := abs(Feld[anfang] - zielWert);
      bisherbesterEintrag := anfang;
      sucher := anfang+1;              % Annahme: ende - anfang > 1
      SOLANGE sucher ≤ ende {
         WENN abs(Feld[sucher] - zielWert) < abweichung DANN {
            abweichung := abs(Feld[sucher] - zielWert);
            bisherbesterEintrag := sucher
         }
         sucher := sucher +1
      }
      suchergebnis := bisherbesterEintrag
}}.
```

- Diskussion der Unterschiede beider Algorithmen:
  - Warum unterscheidet sich die Schleifenendbedingung bei den beiden Algorithmen?
  - Könnte man in SucheLMin die Schleife nicht in einigen Fällen auch abbrechen bevor das letzte Element im Feld erreicht wird?
  - Würde sich durch diese „Optimierung“ möglicherweise etwas am Ergebnis ändern?

    Ja, wenn es zwei oder mehrere unterschiedliche Personen gibt, die den Zielwert exakt erfüllen. In obigem Verfahren erhält man jene, die dem Ende des Feldes am nächsten ist, im „optimierten“ Verfahren, jene, die am nächsten beim Anfang steht.
  - Warum wird es in der Regel nicht sinnvoll sein, diese „Optimierung“ durchzuführen? Wovon hängt es ab, ob eine solche „Optimierung“ sinnvoll ist?