Aha! Algorithmen

Sebastian Vetter
20.April 2005

Gliederung

1. Einstieg

2. Drei Probleme
   - 2.1. Integer- Problem (Binäre Suche)
   - 2.2. Vektor- Rotation
   - 2.3. Anagramme

3. Prinzipien
   - 3.1. Sortierung
   - 3.2. Binäre Suche
   - 3.3. Signaturen
   - 3.4. Problemdefinition
   - 3.5. Perspektive des Problemlösenden

- viele, vor allem grundlegende Algorithmen werden gelehrt
- Aber!

  Algorithmen haben wichtigere Bedeutung

- Martin Gardner:

  „Ein Problem, dass schwer scheint, kann eine einfache Lösung haben.“

è dies bedarf Überlegungen

vor, **während** und **nach** der Programmierung

Integer- Problem

- gegeben: Datei, mit max. 4 Mrd. Integer (32 Bit) sequentiell aber zufällig gesichert
- Aufgabe: Zahl finden, die nicht in der geg. Datei existiert!

- Wie würde man dieses Problem mit den gewöhnlichen Mengen von RAM lösen?

- Wie würde man dieses Problem lösen, wenn man nur einige 100 Bytes und „Notiz“- Dateien zur Verfügung hat?

- Verweis auf binäre Suche

  è in sortiertem Array wird nach einem Element gesucht

- Laufzeiten (n Elemente):

  sequentielle Suche = n/2 Vergleiche

  binäre Suche = ld n Vergleiche

Binäre Suche:

- oftmals beginnen Programmierer mit sequentieller Suche, da diese anfangs meist ausreichend schnell sind

ABER:

- wenn das dann doch zu langsam wird, wird Tabelle sortiert, binäre Suche genutzt

  è Flaschenhals verschwindet

- Datei mit max. 4 Mrd. Integer
  - à eine muss mindestens fehlen, da Integer 2^{32} è 4.294.967.296

a) genügend RAM vorhanden:
  - è Bitmaptechnik nutzen und 536.870.912 Bytes für Bitmap reservieren, welches Integers repräsentieren
  - è 4.294.967.296 Bits vorhanden
  - è alle mit 0 initialisiert
  - è wenn Zahl vorhanden è Bit auf 1 setzen

b) nur wenige 100 Bytes vom RAM, aber mehrer sequentielle Dateien vorhanden:

Abbildung auf Binäre Suche Definition von:

- Intervall
- Repräsentation der Elemente im Intervall
- Prüfmethode um Intervallhälfte zu bestimmen, die die fehlende Zahl enthält

- Intervall = Sequenz von bekannten Ganzzahlen, die mind. ein fehlendes Element enthalten
- Repräsentation des Intervalls = Datei, die all die Integer enthält

- **Aha- Effekt:**
  - è Intervallprüfung durch Zählen der Elemente über und unter dessen Mitte
  - è entweder hat das obere oder das untere mindestens die Hälfte des ganzen Intervalls
  - è weil ganzes Intervall ein fehlendes Element hat, fehlt es auch im kleineren Teilintervall

- Algorithmus durch Ed Reingold

- beim Einlesen der Integer aus dem Eingabedatei
  è prüfen des führenden Bits (Bitmaske)

- Zahlen in zwei Dateien sammeln
  a) mit 1 an der Position
  b) mit 0 an der Position

- Bitmaske ein Bit „weiter“ schieben
- kleinere Datei (mit weniger Zahlen) als neue Eingabe verwenden
- Nach einigen Durchläufen
  à Sortieren & Scannen der Datei à Lösung

Vektor- Rotation

- Vektor um i Positionen nach links rotieren
- mehrere verschiedene Lösungen

a) temporäres Array

- Kopie der ersten i Elemente
- „vorkopieren“ der übrigen Elemente
- anhängen der i Elemente aus temp. Array

b) Schiebefunktion (um eine Position nach links)

- wird i mal aufgerufen
- zeitaufwändig

c) andere Sicht auf Vektor

- Vektor „ab“ muss in Form „ba“ überführt werden
- a repräsentiert ersten i Elemente; b die übrigen
- Annahme: a kürzer als b
  - à b in b_l und b_r geteilt (Länge b_r und a sind gleich)
  - à a b_l b_r in Form b_r b_l a überführen (durch Tausch)
  - à a ist (gesamt) an richtiger Position
  - à rekursives Fortsetzen

d) Ken Thompson, 1971 (ähnlich Lösung c)

- aber es existiert eine Reverse- Funktion, die die Elemente einer gegebenen Menge eines Array umdreht
- von Vektor „ab“ wird a überführt in a^r
- b in b^r überführt
- $(a^r\ b^r)$ in $(a^r\ b^r)^r$ überführt

  è $(a^r\ b^r)^r$ entspricht „ba“

Beispiel: Vektor „abcdefgh“ (n = 8; i = 3)

```
reverse(0, i-1);  /* cbadefgh */
reverse(i, n-1);  /* cbahgfed */
reverse(0, n-1);  /* defghabc */
```

- ist effizient bezüglich Zeit und Platz
- kurz und einfach à schwer, Fehler zu machen
- Anwendung:

  „Software Tools in Pascal“ von Brian Kernighan & P.J. Plauger um Zeilen in einem Texteditor zu bewegen

  à Lösungsversuche mittels verketteten Listen waren fehlerhaft

  à K. Thompson‘s Lösung sofort fehlerfrei

Anagramme

- gegeben: Wörterbuch mit 230 000 Wörtern

- Aufgabe: Anagrammklassen finden

- **Anagrammklasse:**
  enthält Wörter die aus gleichen Buchstaben (jeweils gleicher Anzahl) bestehen

  Bsp: „post“, „stop“, „tops“, „pots“

- einige Denkansätze ineffektiv und zum Scheitern verurteilt (z.B. wenn alle Permutationen beachtet werden)
- Bsp: Wörter mit 22 Buchstaben
  „cholecystoduodenostomy“ &
  „duodenocholecystostomy“

  duodenum = Zwölffingerdarm

  1. = chirurgische Verbindung der Gallenblase und Zwölffingerdarm
  2. = ? Verweis auf erstes Wort ?

- 22! Permutationen ~ 1.124×10^{21} Permutationen
- Annahme: 1 ps / Permutation à 1.1×10^{9} s
  à > 11 500 Tage à > 31 Jahre

- Methoden zum Vergleichen aller Paare von Wörtern benötigen ~ 14,7 Stunden

230 000 Wörter x 230 000 Vergleiche / Wort x 1 s / Vergleich
= $52\,900 \times 10^{6}$ s
= 52 900 s
~ 14,7 Stunden

- bessere Lösung erforderlich
- **Aha! Effekt:** è Signatur an jedes Wort

- gleiche Signatur bedeutet gleiche Angrammklasse
- verschiedene Signatur bedeutet verschiedene Klasse

è Reduzierung auf 2 Teilprobleme
  - è Auswählen einer Signatur
  - è Sammeln aller Wörter mit gleicher Signatur

1. Teilproblem
   - signaturbasiertes Sortieren à ordnen der Buchstaben in alphabetischer Reihenfolge
   - „post“ hat Signatur „opst“, genau wie „stop“

2. Teilproblem
   - Sortierung der Wörter nach Signatur
   - treffende Beschreibung von Tom Cargill (Handwellen)

Horizontale Handwellen à Sortierung innerhalb eines Wortes

Vertikale Handwelle à Sortierung innerhalb d. Wörterbuches

- 3 stufige Pipeline

- Eingabedatei
  - à Signierung
  - à Sortierung
  - à Quetschen (squash/ zeilenweise Ausgabe)

- Ausgabedatei

Signierung (sign)

```
int charcomp(char *x, char *y){return *x- *y ;}

#define WORDMAX 100

int main(void)
{
    char word[WORDMAX], sig[WORDMAX];
    while( scanf( “%s”, word) != EOF )
    {
          strcpy(sig, word);
          qsort(sig, strlen(sig), sizeof(char),
                                          charcomp);
          printf(“%s %s\n”, sig, word);
    }
    return 0;
}
```

Quetchen (squash)

```
int main(void)
{
    char word[WORDMAX], sig[WORDMAX], oldsig[WORDMAX];
    int linenum = 0;
    strcpy(oldsig, „“);
    while( scanf( „%s %s“, sig, word) != EOF )
    {
        if( strcmp(oldsig, sig) != 0 && linenum > 0)
            printf(“\n”);
        strcpy(oldsig, sig);
        linenum++;
        printf(“%s ”, word);
    }
    printf(“\n”);
    return 0;
}
```

Ergebnis: `sign <dictionary | sort | squash >gramlist`

Prinzipien

Sortierung

nahe liegende Verwendung

- à um sortierte Aufgabe zu erhalten, entweder als Bestandteil oder als Vorbereitung
  (z.B. für anschliessende Binäre Suche)

Anagramm- Problem

- à Reihenfolge der Klassen nicht von Interesse
- à Sortierung um gleiche Elemente zusammenzuführen

Binäre Suche

- sehr schneller und effizienter Algorithmus für Suche in einer sortierten Tabelle / Array

Nachteil:

- Tabelle / Array muss komplett bekannt und sortiert sein

Signatur

- nützlich bei Definitionen von Aquivalenzklassen
- jedes Element der Klasse hat die gleiche Signatur, und ein anderes Element eben nicht

- Buchstabensortierung ist nur eine mögliche Signatur
- andere Varianten repräsentieren Duplikate durch Zahlen
- Bsp: i4m1p2s4 ist Signatur von mississippi

Problemdefinition:

- essentieller Teil der Programmierung

- Welche primitiven / grundlegenden Algorithmen nützen um das Problem trivial zu machen ?

  è Aha- Effekt

Perspektive des Problemlösenden

- gute Programmierer sind faul
- sitzen lange Zeit „nur“ da und überlegen bis die richtige Idee kommt, statt die erste zu verfolgen
- gewisses Augenmaß; erlangbar durch Erfahrung

- Jon Bentley - „Programming Pearls“ (Second Edition) 1999 – Addison Wesley

- URL:

  http://www-2.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15750-s04/www/HW4_sol.txt

  http://www.merckmedicus.com