Algorithmen

**Vollständigkeit**: Vollst. Beschreibung eines Lösungsverfahren darf keinen Schritt auslassen und muss alle relevanten Vor- und Rahmenbedingungen.

**Eindeutigkeit**: Jede Aktion muss eindeutig interpretierbar sein. Es darf keinen Interpretationsspielraum geben.

**Ausführbarkeit**: Jede Aktion muss ausführbar sein. Die Art und Weise der Ausführung muss klar sein.

**Statische** Endlichkeit: Der Umfang der Beschreibung eines Algorithmus muss endlich sein, unabhängig von der Notifikation

**Dynamische** Endlichkeit: Ein Algorithmus heisst abbrechend oder terminierend, wenn seine Ausführung nach endlich vielen Schritten anhält, sonst heisst der Algorithmus nicht abbrechend oder nicht terminierend.

**Korrektheit**: Ein Algorithmus ist korrekt, wenn er für jede Eingabeinstanz nach endlich vielen Schritten hält. Ein inkorrekter Algorithmus terminiert nicht oder terminiert mit einer falschen Ausgabe.

**Effizienz**: Ein Algorithmus soll seinen Zweck unter bestmöglicher Ausnutzung aller benötigten Ressourcen erfüllen.

|  |  |  |
| --- | --- | --- |
| Ganze Zahlen | integer | Int |
| Reelle Zahlen | real | Real |
| Wahrheitswerte | boolean | Bool |
| Zeichen | charcter | Char |
| Zeichenkette | String | string |

Datenobjekt deklarieren: Art (Konst/Variable), Name, Datentyp, Wert(optional) Bsp: var v: real=3.12

Für Standart-Algo &-Datenstr. bew. Bibliotheken nehmen

Prog kann Darstellung v. Algo sein, aber auch sinnlose Folge von Operationen

Algo kann in versch. Darstellungsformen beschreibbar sein. Prog muss immer in d. Sprache d. Rechners formuliert sein.

Algo spricht mann nur im Zusammenhang mit einem Problem.

**Detail/Feinanalyse:**

|  |  |  |  |
| --- | --- | --- | --- |
| Anweisung | Anzahl | Einzelgewicht (versch. Operationen sep aufschreiben) | Gewicht(alle Einzel-gewichte zusammen) |
| row = 1 | 1 | 1.0 | 1.0 |
| while row <= n1 | n1 +1 | 1.5 |  |
| col = 1 | n1 | 1.0 | 1.0 |
| while col <= n2 | n1\*(n2+1) | 1.5 | 1.5 |
| matrix[row,col] | n1\*n2 | 4.2+1.0 | 5.2 |
| col ++ | n1\*n2 | 1.0+1.4 (zuw.+add) | 2.4 |
| end; |  |  |  |
| row ++ | n1 | 1.0+1.4 (zuw.+add) | 2.4 |
| end; |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Anweisung | Anzahl | Einzelgewicht | Gewicht |
| While i<= n | (n1\*n2)+1 | 1.5 | 1.5 |
| if col =n2 | n1\*n2 | 1.5 | 1.5 |
| col =1 | n1 | 1.0 | 1.0 |
| else |  |  |  |
| col ++ | (n1\*n2) –n1 | 1.0+1.4 | 2.4 |
| end |  |  |  |
| end |  |  |  |

t(a1) = 1+1.5\*(n1+1)+…=n1(8.1\*n2+6.4)+2.5=8.1\*n1\*n2+6.4\*n1+2.5

t(a2)=1+1+1+33+1.5\*(n1\*n2)+…=n1(12\*n2)+7.8=12\*n1\*n2+1.0\*n1+7.8

**Grobanalyse**

Günstigster Fall: 3… 🡪 Smin(n)=1

Ungünstiger Fall: …3 🡪 Smax(n)=n

Durchschnittlicher Fall:

|  |  |  |
| --- | --- | --- |
| Erstes Auftreten von 3 | Wahscheinlichkeit | Schleifendurchläufe |
| Pos 1 | 1/3 | 0 |
| Pos 2 |  | 1 |
| Pos n |  | n-1 |
| Gar nicht |  | n-1 |

Savg(n)=

Bsp:

Annahme: sowohl Klein-, wie auch Grossbuchstaben und Zahlen wurden verwendet (26 Klein-, 26 Grossbuchstaben und 10 Ziffern)

Günstigster Fall: Smin(n)=1 (Bsp: im 1. Anlauf wird mit nur einem „a“ versucht und der Durchlauf ist sogleich erfolgreich)

Ungünstiger Fall: Smax(n)=n (Erst die letzte mögliche Kombination ist erfolgreich; Bsp: „99999999“)

Durchschnitt: Savg(n)=

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 Stellig | 2 Stellig | 3 Stellig | 4 Stellig | 5 Stellig | 6 Stellig | 7 Stellig | 8 Stellig |
| 62/2=31 | 62 +622/2 | 62 +622+623/2 | usw |  |  |  |  |

Danach wird die 1-Stellige Lösung mit der durchschnittlichen Wahrscheinlichkeit (62:Worstcase) multipliziert + 2 Stellige Lösung\* 622/Worstcase usw. Die erhaltene Zahl wird dann durch

Guasssche Summenformel :

🡪 Laufzeit steigt mit dem Quadrant der Matrizengrösse

Anzahl Suchschritte:

S(n)=1+ Anzahl Schleifendurchläufe