marc-oliver pahl

Informatik II – Kapitel 11

„Such-Algorithmen“

Zusammenfassung des Kapitel 11
Küchlin, Weber, Einführung in die Informatik, 2.Auflage

10.6.2004

Wir wollen eine Funktion

`suchePosition( F, a )`

realisieren, die dem folgenden Algorithmenschema genügt:

suchePosition(F, a):

// Vorbedingung: F ist eine Folge, a ein Element
// Nachbed.: Rückgabe = (Position von a in F) | (-1 wenn a ∉ F)

i. Initialisiere:

res = -1; S = Mg. Aller Suchpos. In F;

ii. Trivialfall:

if (S == ∅) return res;

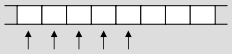
iii. Reduktion:

Wähle nächste Suchposition p; Entferne p aus S;

iv. Rekursion:

if (F[p] == a) return res=p; else weiter bei ii.;

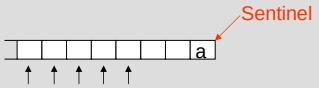

```
public static int linearSearch( Object[] f, Object a){
   for ( int i=0; i<f.length; i++)
        if (f[i].equals(a) )
              return i;
   return -1;
}
```



```
public static int linearSearch( Object[ ] f, Object a){
       int i=0;
       while ( i<f.length && !(f[i].equals(a)) ) i++;
       if (f[i].equals(a)) return i;
       else   return -1;
}
```



```
public static int linearSearch( Object[ ] f, Object a){
       int i=0; „f=f + a;“
       while ( !(f[i].equals(a)) ) i++;
       if (i =< f.length) return i;
       else   return -1;
}
```




```
public static int linearSearch( Object[ ] f, Object a){
       int i=0;
       while ( i<f.length && !(f[i].equals(a)) ) i++;
       if (f[i].equals(a)) return i;
       else   return -1;
}
```

Laufzeit in O(n) (linear)

Divide & Conquer binaere Suche

marc-oliver pahl

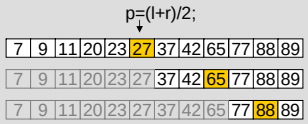
- **Binäre Suche** nach dem Divide & Conquer Prinzip macht nur Sinn, wenn nach dem Teile-Schritt ein Teil ausgeschlossen werden kann.

```
public static int binarySearch( Comparable[ ] f, Comparable a, int l, int r){
Initialisierung{ int p=(l+r)/2; // Teilungsposition
                 int c=f[p].compareTo(a); // Vergleich: 0: gleich, <0: größer, >0: kl

Trivialfall{     if (c == 0) return p; // gleich => gefunden
                 if (l==r) return -1; // nicht gefunden (letztes verbleibendes Element ist ungleich)

Red./ Rek.{      if (c<0) return binarySearch(f, a, l, p-1); // links weiter
                 else return binarySearch(f, a, p+1, r); // rechts weiter
}
```

binarySearch(*f*, „*88*“, 0, 11):



Laufzeit in O(log(n)) (logarithmisch)

Kombinationssuche

marc-oliver pahl

- Wie wir in den Übungen gesehen haben, ist eine Kombination beider Suchverfahren am effizientesten.