IIF
INSTITUT FÜR
INFORMATIK
FREIBURG

Vorlesung Informatik 2
Algorithmen und Datenstrukturen

(03 – Verschiedene Algorithmen für dasselbe Problem)

Prof. Dr. Susanne Albers

Das Maximum-Subarray Problem

Das Maximum-Subarray Problem:

Gegeben: Folge X von n ganzen Zahlen im Array

Gesucht: Maximale Summe einer zusammenhängenden Teilfolge von X

Beispiel:

$$31\ -41\ \underbrace{59\ 26\ -53\ 58\ 97}_{\sum = 187}\ -93\ -23\ 84$$

Das ist hier maximal

Frage: Wie findet man die maximale Teilfolge — möglichst effizient?

Das Maximum-Subarray Problem

Die naive Methode:

für jede untere Grenze u∈{0, …, n-1} für jede obere Grenze o∈{u, …,n-1} berechne X[u] + X[u+1] + … + X[o] und bestimme dabei das Maximum aller dieser Werte	

```
public static Integer3 maxSubArrayBF (int[] a) {
    Integer3 result = new Integer3 ();
    for (int u=0; u < a.length; u++)
        for (int o=u; o < a.length; o++) {
            int summe=0;
            for (int i=u; i <= o; i++) summe += a[i];
            if (summe > result.sum) result.set (u, o, summe);
        }
        return result;
} //maxSubArrayBF
```

Das Maximum-Subarray Problem

Die halb-naive Methode

für jede untere Grenze u $\in$ {0, …, n-1}

für jede obere Grenze o $\in$ {u, …, n-1}

berechne X[u] + X[u+1] + … + X[o]

(Dies kann inkrementell aus dem vorhergehenden Wert bestimmt werden)

```
public static Integer3 maxSubArraySBF (int[] a) {
      Integer3 result = new Integer3 ();
      for (int u=0; u < a.length; u++) {

        int summe=0;

        for (int o=u; o < a.length; o++) {
              summe += a[i];
              if (summe > result.sum) result.set (u, o, summe);
        }

       }

      return result;
} //maxSubArraySBF
```

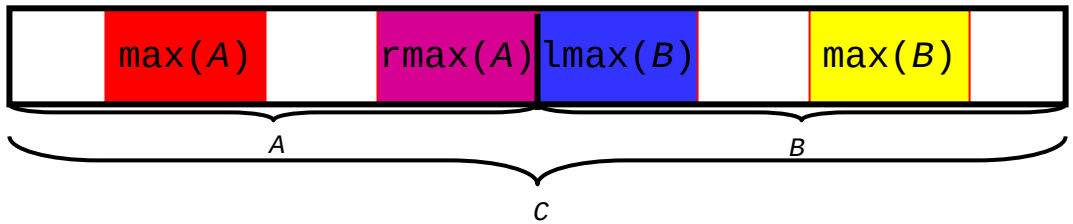
Divide and Conquer Lösung

Das Maximum-Subarray Problem

Der Divide-and-Conquer Ansatz:

1. Kleine Probleme werden direkt gelöst:
   falls $n = 0$: Max = 0
   falls $n = 1$ und $X[1] < 0$: Max = 0
   sonst: Max = X[1]

2. (Divide) Größere Probleme werden aufgeteilt (hier in 2 etwa gleich große Teile):



3. (Conquer) Für die Teile kann dasselbe Problem mit demselben Verfahren rekursiv gelöst werden: Das maximale Subarray steckt
   (a) ganz in A oder
   (b) ganz in B oder
   (c) es besteht aus Randmaxima in A und B

4. (Merge) Die Gesamt-Lösung ergibt sich aus den Teillösungen: Der maximale Wert ergibt sich hier als `max` aus (a), (b), (c)

Das Maximum-Subarray Problem

Der Divide-and-Conquer Ansatz:

```
public static Integer3 maxSubArrayDC (int[] a, int l, int u) {
    if (l == u) {                                  // kleines Problem
        if (a[u] <= 0) return new Integer3 (u+1, l-1, 0);
        else return new Integer3 (l, u, a[u]);
    }
    int m = (l+u) >> 1;                            // Divide
    Integer3 A = maxSubArrayDC (a, l, m);          // Conquer
    Integer3 B = maxSubArrayDC (a, m+1, u);
    Integer3 C = join (rmax (a, l, m), lmax (a, m+1, u));
    if (A.sum >= B.sum)                            // Merge
        if (A.sum >= C.sum)   return A;
        else                  return C;
    else if (C.sum >= B.sum)  return C;
         else                 return B;
}
```

Berechnung der Randmaxima

```
// berechnet maximale Summe am rechten Rand
public static Integer3 rmax (int[] a, int l, int u) {
    Integer3 ergebnis = new Integer3 (u+1, u, 0);
    for (int summe=0, i = u; i >= l; i--) {
        summe += a [i];
        if (summe > ergebnis.sum) ergebnis.set (i, u, summe);
    }
    return ergebnis;


// verbindet rechtes und linkes Randmaximum aus A und B
public static Integer3 join (Integer3 l, Integer3 r) {
    return new Integer3 (l.lower, r.upper, l.sum + r.sum);
}
```

Analyse des Divide-and-Conquer-Ansatzes

T(n) = # Schritte zur Lösung des Maximum-Subarry Problems für ein Array der Länge n.

$T(1) \leq a$

$T(n) \leq 2\ T(n/2) + b\ n$

Dieses Rekursionsgleichungssystem hat eine geschlossene Lösung:

$T(n) \leq a\ n + b\ n \log_2 n \in O(n \log_2 n)$

Lösen von Rekursionsgleichungen

Wie findet man Lösungen für die bei der Analyse rekursiver Programme auftretenden Rekursionsgleichungen?

1. Iterative Substitutionsmethode:

Beispiel: $T(1) = 1$, $T(n) = 2T(n/2) + n$ hat die Lösung $T(n) = n+n \log n$

Lösen von Rekursionsgleichungen

2. Lösung raten und per Induktion beweisen:

Beispiel: $T(1) = 1$, $T(n) = 2T(n/2) + n$ hat die Lösung $T(n) = n+n \log n$

Lösen von Rekursionsgleichungen

3. Verwenden von Computeralgebra-Programmen:

Beispiel (Maple):

```
rsolve ({T(1)=1, T(2*n)=2*T(n)+(2*n)},T);
```

Lösen von Rekursionsgleichungen

4. Anwenden des Master-Theorems:

Seien a ≥ 1 und b > 1 konstant und sei $f(n)$ eine reellwertige Funktion mit positiven Werten für $n \geq d$. Sei $T(n)$ definiert durch die Rekursionsformel

$T(n) = c$; falls $n < d$

$T(n) = aT(n/b) + f(n)$; sonst

Dann gilt:

Das Maximum-Subarray Problem

Scan-Line Ansatz

Das Maximum-Subarray Problem

Der Scan-Line Ansatz:

```
scanMax = 0;
bisMax = 0;
für i von 0 bis n-1:
  scanMax += X[i]
  falls (scanMax < 0) scanMax = 0;
  bisMax = max(scanMax, bisMax);
max = bisMax;
```

$$T(n) \leq a\,n + b \in O(n)$$

Die Lösung des Maximum-Subarray Problems erfordert einen Aufwand in $\Theta(n)$. (Jedes Element muss mindestens einmal betrachtet werden)