Obersemminar Softwareentwicklung

Thema: Algorithmusdesignverfahren

Vu Hoang Lam
IMN04
HTWK-Leipzig

Das Problem

Gegeben: ein Eingabevektor *x* mit *n* Elementen.
Gesucht: die maximale Summe, die in einem zusammenhängenden Subvektor der Eingabe gefunden werden kann.

Beispiel

31	-41	59	26	-53	58	97	-93	-23	84
		↑				↑			
		2				6			

Rückgabe der Summe von x[2..6] , oder 187

Das Problem

Wenn alle Zahlen positiv
→Der größtmögliche Untervektor= der ganze Eingabevektor

Wenn alle Zahlen negativ
Der größtmögliche Untervektor= leerer Vektor (Summe=0)

Entstehung aus Flächenmustererkennung,
Stellung durch Ulf Grenander an der Brown University

Ursprünglich.
Gegeben: ein zweidimensionales Array $n \times n$ mit $n \in \mathbb{R}$.
Gesucht: die maximale Summe, die ein rechteckiges Subarray enthält.

Keine Lösung des Problems in vernünftiger Zeit
Abstraktion einer Dimension →Einblick in die Struktur des Problems

Lösungsversuch

- **Einfacher Algorithmus mit kubischer Laufzeit**
- **Quadratischer Algorithmus mit quadratischer Laufzeit**
- **Teilen und Besieg Algorithmus mit n log n Laufzeit**
- **Scanning Algorithmus mit linearer Laufzeit**

Simple Algorithmus

- **Iteration über alle Integerpaare i und j, wobei $0 \leq i \leq j < n$,**

  n die Länge des Eingabevektors *x*

- **Berechnung der Summe von $x[i..j]$ für jedes Paar von i und j**

- **Überprüfung der errechneten und berechneten Summe**

Simple Algorithmus

```
maxsofar = 0
for i= [0,n)
   for j= [i,n)
      sum = 0
      for k= [i,j] sum += x[k]
         /* sum is sum of x[i..j] */
         maxsofar = max (maxsofar, sum)
```

Der Algorithmus hat kubische Laufzeit. O (n^3)
Messungen auf PC Pentium II 400Mhz, 256 Mb RAM:
Bei n= 10.000 beträgt die Laufzeit ca. 22 Minuten.
Bei n= 100.000 beträgt die Laufzeit schon ca. 15 Tage.

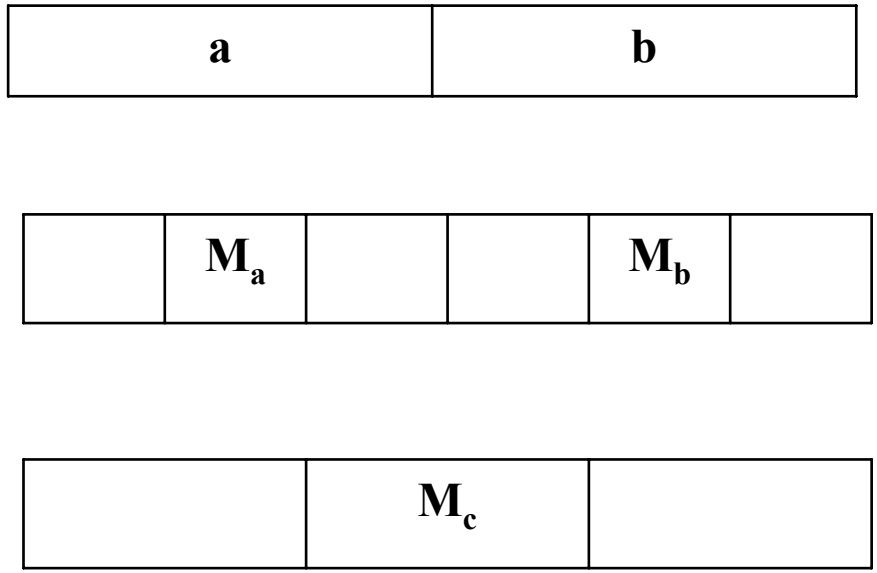
Zwei Quadratische Algorithmen

```
maxsofar = 0
for i= [0,n)
   sum = 0
   for j= [i,n)
      sum += x[j]
      /* sum is sum of x[i..j] */
      maxsofar = max (maxsofar, sum)
```

Der Algorithmus hat quadratische Laufzeit $O(n^2)$.
Bei n = 10.000 läuft der Algorithmus 797 ms.
Bei n = 100.000 hat der Algorithmus eine Laufzeit von ca. 1,3 Minuten.

Teilen- und Besieg Algorithmus

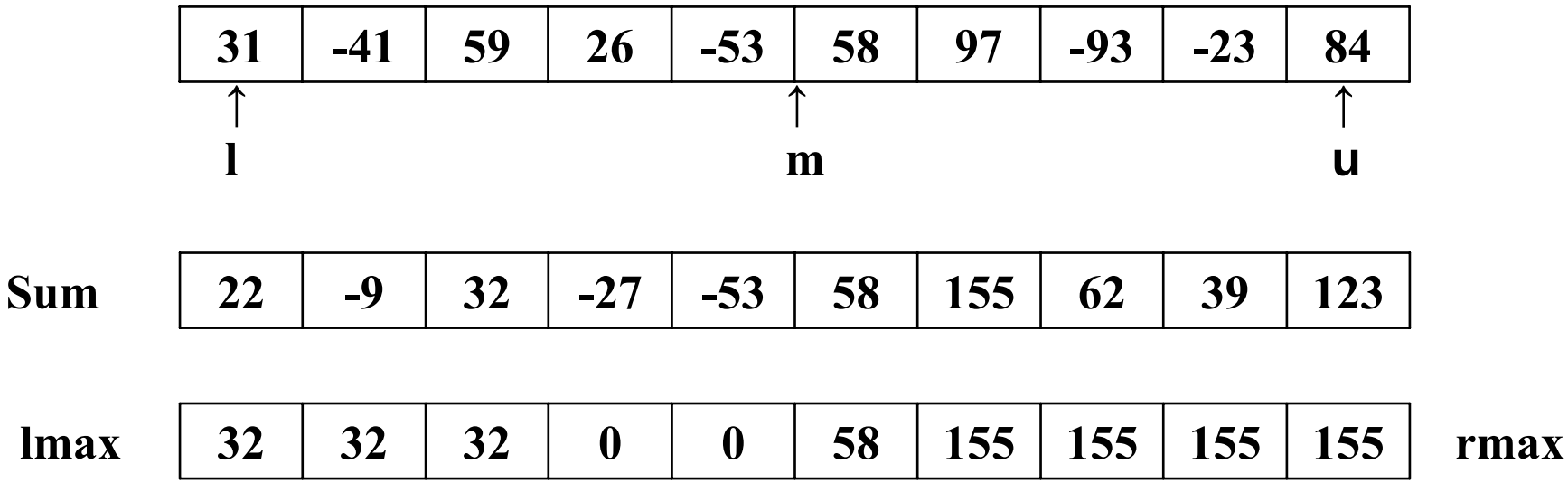
- **Rekursive Lösung zweier Teilprobleme von Größe n/2**
- **Lösung des vollständigen Problems durch ihre Kombination**



Teilen und Besieg Algorithmus

```
float maxsum3(1, u)
 if (1 > u)                          /* zero elements */
return 0
   if (1 == u)                       /* one element */
return max(0, x[1])
m = (1 + u) / 2
lmax = sum = 0                       /* find max crossing to left */
for (i = m; i >= 1; i--)
sum += x[i]
lmax = max(lmax, sum)                /* find max crossing to right */
rmax = sum = 0
for i = (m, u]
sum += x[i]
rmax = max(rmax, sum)
return max(lmax+rmax, maxsum3(1, m), maxsum3(m+1, u))
/* Algorithm 3 is originally invoked by the call */
answer = maxsum3(0, n-1)
```

Teilen und Besieg Algorithmus



32+155 =187

Scanning Algorithmus

Angenommen das Problem ist für x[0..i-1] gelöst:

Das maximale Subarray in den ersten i Elementen befindet sich entweder in den ersten i-1 Elementen (gespeichert in maxsofar), oder es endet auf Position i (gespeichert in maxendinghere)

Scanning Algorithmus

```
maxsofar = 0;
maxendinghere = 0;
for i= [0,n)
maxendinghere = max (maxendinghere+x[i], 0);
maxsofar = max (maxsofar, maxendinghere);
```

- **Laufzeit O(n)**
- **Die Laufzeit bei n= 100.000 Elementen beträgt 5 ms!**

Prinzipien

- **Vermeidung der Wiederberechnung durch Abspeichern des Zustandes**
- **Vorverarbeitung der Information in Datenstrukturen**
- **Teilung und Besiege Algorithmen**
- **Scanning Algorithmen**
- **Sammeln**
- **Geringere Schranke**

Laufzeit Tabelle

ALGORITHMUS		1	2	3	4
Laufzeit in Nanosekunden		$1.3\ n^3$	$10\ n^2$	$47n \log_2 n$	$48n$
Zeit zu lösen ein Problem von Größe	10^3 10^4 10^5 10^6 10^7	1.3 Sek 22 Min 15 Tage 41 J 41 TDJ	10 Msek 1 Sek 1,7 Min 2.8 Std 1.7 W	.4 Msek 6 Msek 78 Msek .94 Msek 11 Sek	.05 Msek .5 Msek 5 MSek 48 Msek .48 Msek
In einem gelöstes Max-Größen- problem	Sec Min Stunde Tage	920 3600 14.000 41.000	10.000 77.000 6.0×10^5 2.9×10^6	1.0×10^6 4.9×10^7 2.4×10^9 5.0×10^{10}	2.1×10^7 1.3×10^9 7.6×10^{10} 1.8×10^{12}
Wenn n sich um 10 vermehrt, vermehrt Zeit von		1000	100	10+	10
Wenn Zeit sich um 10 vermehrt, vermehrt n von		2.15	3.16	10-	10