Design and Analysis of Algorithms
Lecture 5: Pseudo-code

童咏昕

北京航空航天大学
计算机学院

提纲

算法表示

案例分析

撰写工具

算法的表示

- 如何表示一个算法?
  - 自然语言

算法的设计者
依靠自然语言交流和表达

算法的表示

- 自然语言
  - 方法优势
    - 贴近人类思维，易于理解主旨
  - 不便之处
    - 语言描述繁琐，容易产生歧义
    - 使用了“…”等不严谨的描述

选择排序

- 第一次遍历找到最小元素
- 第二次在剩余数组中遍历找到次小元素
- …
- 第n次在剩余数组中遍历找到第n小元素

算法的表示

- 如何表示一个算法?
  - 自然语言
  - 编程语言

算法的表示

- 编程语言
  - 方法优势
    - 精准表达逻辑，规避表述歧义
  - 不便之处
    - 不同编程语言间语法存在差异
    - 过于关注算法实现的细枝末节

选择排序

```c
void select_sort(int *a, int n) {
    for (int i = 0; i < n - 1; i++) {
        int cur_min = a[i];
        int cur_min_pos = i;
        for (int j = i; j < n; j++) {
            if (cur_min > a[j]) {
                cur_min = a[j];
                cur_min_pos = j;
            }
        }
        int temp = a[i];
        a[i] = a[cur_min_pos];
        a[cur_min_pos] = temp;
    }
    return;
}
```

C语言

```python
def select_sort(a, n):
    for i in range(0, n - 1):
        cur_min = a[i]
        cur_min_pos = i
        for j in range(i, n):
            if cur_min > a[j]:
                cur_min = a[j]
                cur_min_pos = j
        temp = a[i]
        a[i] = a[cur_min_pos]
        a[cur_min_pos] = temp
```

Python语言

算法的表示

- 如何表示一个算法?
  - 自然语言：贴近人类思维，易于理解主旨
  - 编程语言：精准表达逻辑，规避表述歧义

问题：可否同时兼顾两类表示方法的优势?

算法的表示

- 伪代码
  - 非正式语言
    - 移植**编程语言**书写形式作为基础和框架
    - 按照接近**自然语言**的形式表达算法过程
  - 兼顾自然语言与编程语言优势
    - **简洁**表达算法本质，不拘泥于实现细节
    - **准确**反映算法过程，不产生矛盾和歧义

选择排序

```
输入: 数组A[a_1, a_2, ..., a_n]
输出: 升序数组A'[a'_1, a'_2, ..., a'_n]，满足 a'_1 ≤ a'_2 ≤ ... ≤ a'_n
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

算法的表示

- 伪代码
  - 书写约定

输入: 数组$A[a_1, a_2, ..., a_n]$
输出: 升序数组$A'[a_1', a_2', ..., a_n']$，满足 $a_1' \leq a_2' \leq ... \leq a_n'$

for $i \leftarrow 1$ *to* $n-1$ **do**
  $cur\_min \leftarrow A[i]$
  $cur\_min\_pos \leftarrow i$
  for $j \leftarrow i+1$ *to* n **do**
    if $A[j] < cur\_min\_pos$ **then**
      //记录当前最小值及其位置
      $cur\_min \leftarrow A[j]$
      $cur\_min\_pos \leftarrow j$
    end
  end
  交换 $A[i]$ 和 $A[cur\_min\_pos]$
end
return A

定义算法的输入和输出

选择排序

算法的表示

- 伪代码
  - 书写约定

循环
语句块缩进

输入: 数组$A[a_1, a_2, ..., a_n]$
输出: 升序数组$A'[a'_1, a'_2, ..., a'_n]$，满足 $a'_1 \leq a'_2 \leq ... \leq a'_n$

```
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

选择排序

算法的表示

- 伪代码
  - 书写约定

将$i+1$赋值给j

```
输入: 数组A[a_1, a_2, ..., a_n]
输出: 升序数组A'[a'_1, a'_2, ..., a'_n]，满足 a'_1 ≤ a'_2 ≤ ... ≤ a'_n
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

选择排序

算法的表示

- 伪代码
  - 书写约定

```
输入: 数组A[a_1, a_2, ..., a_n]
输出: 升序数组A'[a'_1, a'_2, ..., a'_n]，满足 a'_1 ≤ a'_2 ≤ ... ≤ a'_n
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

注释使用“//”符号

选择排序

算法的表示

- 伪代码
  - 书写约定

```
输入: 数组A[a_1, a_2, ..., a_n]
输出: 升序数组A'[a'_1, a'_2, ..., a'_n]，满足 a'_1 ≤ a'_2 ≤ ... ≤ a'_n
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

条件
语句块缩进

选择排序

算法的表示

- 伪代码
  - 书写约定

```
输入: 数组A[a_1, a_2, ..., a_n]
输出: 升序数组A'[a'_1, a'_2, ..., a'_n]，满足 a'_1 ≤ a'_2 ≤ ... ≤ a'_n
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

不关注交换过程的实现细节

选择排序

算法的表示

- 伪代码
  - 书写约定

```
输入: 数组A[a_1, a_2, ..., a_n]
输出: 升序数组A'[a'_1, a'_2, ..., a'_n]，满足 a'_1 ≤ a'_2 ≤ ... ≤ a'_n
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

选择排序

之后出现的算法均使用伪代码描述

算法的表示

- 伪代码
  - 示例解读

4	8	13	14	17	18	21	22	24	28	32	37	40	40	47	48

输入: 数组$\boldsymbol{A[a_1, a_2, \ldots, a_n]}$
输出: 升序数组$\boldsymbol{A'[a'_1, a'_2, \ldots, a'_n]}$，满足 $\boldsymbol{a'_1 \le a'_2 \le \ldots \le a'_n}$

```
for i ← 1 to n − 1 do
    cur_min ← A[i]
    cur_min_pos ← i
    for j ← i + 1 to n do
        if A[j] < cur_min_pos then
            //记录当前最小值及其位置
            cur_min ← A[j]
            cur_min_pos ← j
        end
    end
    交换 A[i] 和 A[cur_min_pos]
end
return A
```

选择排序

算法的表示

- 伪代码
  - 示例解读

13	17	24	28	40	14	22	32	40	21	48	4	47	8	37	18

j

输入: **数组** $\boldsymbol{A[a_1, a_2, \ldots, a_n]}$

输出: **升序数组** $\boldsymbol{A'[a'_1, a'_2, \ldots, a'_n]}$**，满足** $\boldsymbol{a'_1 \leq a'_2 \leq \ldots \leq a'_n}$

for $j \leftarrow 2$ *to* n **do**

  $key \leftarrow A[j]$

  //将 $A[j]$ 插入到已排序的数组$A[1..j-1]$中

  $i \leftarrow j-1$

  while $i >$ *0 and* $A[i] > key$ **do**

    $A[i+1] \leftarrow A[i]$

    $i \leftarrow i-1$

  end

  $\mathrm{A}[i+1] \leftarrow key$

end

return A

插入排序

算法的表示方式比较

表示方式	语言特点
自然语言	贴近人类思维，易于理解主旨 表述不够精准，存在模糊歧义
编程语言	精准表达逻辑，规避表述歧义 受限语法细节，增大理解难度
伪代码	关注算法本质，便于书写阅读

提纲

算法表示

案例分析

撰写工具

优质案例

2 k 重有序问题

2.1 Main Idea

记每段长度为 $l = n/k$。对长度为 tl 的问题而言，首先将其调整成 2 重有序数组，方法如下：

1. 寻找中位数：找到当前长度为 tl 的数组的第 $tl/2$ 小数，记其值为 m。
2. Partition：扫描数组，找到任意一个 m。以它为 pivot，进行 QuickSort 中的 Partition 操作。操作后，左段小于等于 m，右段大于 m。但该值为 m 的 pivot 并不一定在第 $tl/2$ 位置。
3. 调整中位数：扫描左段，将所有值等于 m 的数调整到左段的末尾。由于右段大于 m，因此右段没有等于 m 的数。

此时，左段的末尾全部为值为 m 的数，且左段小于等于 m，右段大于 m。则必有值为 m 的数在第 $tl/2$ 位置，且左段小于等于它，右段大于它。则该数组 2 重有序。之后对每个无序数组先调整成 2 重有序数组，再递归划分长度均等的子数组，对每个子数组重复上述步骤直到当前数组长度为 l，则不进行操作，返回。

2.2 Pseudo Code

Algorithm 1: *DoubleOrder*$(A[1...L], l)$

```
Input: Array A[1...L] whose length is L
Output: The Rearranged 2-Ordered Array A[1...L]
if L = l then return;
median ← FindKth(A[1...L], L/2);
pivot ← 1;
for i ← 1 to L do
    if A[i] = median then  pivot ← i ;
end
swap A[pivot] and A[L];
pivotPosition =PartitionWithRightPivot(A[1...L]);
lowerMedian ← pivotPosition;
for i ← pivotPosition − 1 to 1 do
    if A[i] = median then
        lowerMedian ← lowerMedian − 1;
        swap A[lowerMedian] and A[i];
    end
end
DoubleOrder(A[1...L/2], l);
DoubleOrder(A[L/2 + 1...L], l);
```

已知中位数的基于 Partition 的重整 2 重有序算法：
已知中位数后，进行以中位数为 pivot 的 Partition 操作，算法的期望复杂度和最坏复杂度均为 $O(n)$。
之后进行扫描调整中位数到左段的末尾，复杂度也为 $O(n)$。
因此该阶段的期望复杂度和最坏复杂度均为 $O(n)$。

因此，对于每个长为 L 的数组，将其重排成 2 重有序数组的期望时间复杂度为 $O(L)$。

总时间复杂度：
设每段长度为常数 $l = n/k$。则某个段数为 t 的问题的总运行时间 $T(t, l)$ 有：

$$T(t,l) = \begin{cases} 1 & t = 1 \\ 2T(t/2, l) + O(tl) & t = 2^b, b = 1, 2, ... \end{cases}$$

解得 $T(t,l) = O(tl\log t)$。现在将初始问题的记号 $l = n/k, t = k$ 代回该式，得到：

$$T(n,k) = O(n\log k)$$

1. 如果$k=1$，那么$A[1..n]$即是所求
2. 否则先对序列进行 KTH-ELEMENT$(A[1..n], n/2)$，然后分别进行 ROUGHLY-SORT$(A[1..n/2], k/2)$，以及 ROUGHLY-SORT$(A[n/2+1..n], k/2)$

伪代码

算法 1 将序列排成k重有序

输入: A数组，k有序数组的重数

输出: 排序后的数组A

```
function KthElement(A, k)
    randomly choose a number p ∈ [1, n]
    call Partition(A, p), and set the new location of A[p] to p′
    if p′ ≥ k then
        KthElement(A[1..p′], k)
    else
        KthElement(A[p′ + 1..n], k − p′)
    end if
end function
function RoughlySort(A[1..n], k)
    if k = 1 then
        return
    else
        call KthElement(A[1..n], n/2)
        call RoughlySort(A[1..n/2], k/2)
        call RoughlySort(A[n/2 + 1..1], k/2)
    end if
end function
```

复杂度分析

其次分析ROUGHLY-SORT的时间复杂度$T(n,k)$，可以发现递归式：

$$T(n,k) = \begin{cases} 1 & ,k=1 \\ 2T(n/2, k/2) + f(n) & ,\text{else} \end{cases}$$

其中$f(n) = O(n)$。所以，可以得出，$T(n,k) = O(n\log k)$。

优质案例

name: **getK**

input

array A: 数组 A

begin: 研究范围的起点

end: 研究范围的终点

output： k

伪代码：

```
//递归情况
if end – begin >1:
    mid = (begin+end)/2
    //左右分治，
    if a[mid] > a[begin]:
        return getK(A, mid, end)
    else:
        return getK(A, begin, mid)
//基本情况
else:
    if A[begin]>A[end]:
        return end
    else:
        return 0
```

设计算法，考虑分治，对于一个区间，折半去判断是否要均分为左右两段按规则计算。先需要准备一个堡垒中士兵总数前缀和 Pre。复杂度为 $T(2^l) = 2T(2^{l-1}) + O(1)$，即 $T(2^l) = O(2^l)$ 预处理的复杂度也是 $O(2^l)$ 故总复杂度为 $O(2^l)$

Algorithm 1 calculate the supply value for [L,R]

function $\text{CALC}(L, R)$

 $mid = \frac{L+R}{2}$

 $num \leftarrow Pre[R] - Pre[L-1]$

 if $num = 0$ **then**

  $now = A$

 else

  $now = num \times (R - L + 1) \times B$

 end if

 if $L == R$ **then**

  $result = now$

 else

  $result = \min(\ \text{CALC}(L, mid) + \text{CALC}(mid+1, R)\ ,\ now)$

 end if

 return $result$

end function

低分案例

3. 将战线平均分为两段，计算每段采取哪种方案花费最少，然后再将这段平均分为两段。

$$T(n)=2T(n/2)+n\times\frac{2}{2^k}\times B$$

$k_{max}=\log_2 n$，$k_{min}=1$

$$\therefore\ T(n)=2T(n/2)+n*2^{l}\times B$$

时间复杂度为 $O(n^2)$

文字描述非常笼统，没有说明时间复杂度是怎么得出的，而且计算的结果也是错误的。

3.战线补给问题

已知士兵有 n 个，堡垒有 2^l 个。设

情况 1：直接为整条战线[1,2^l]提供补给

cost = n * 2^l * B

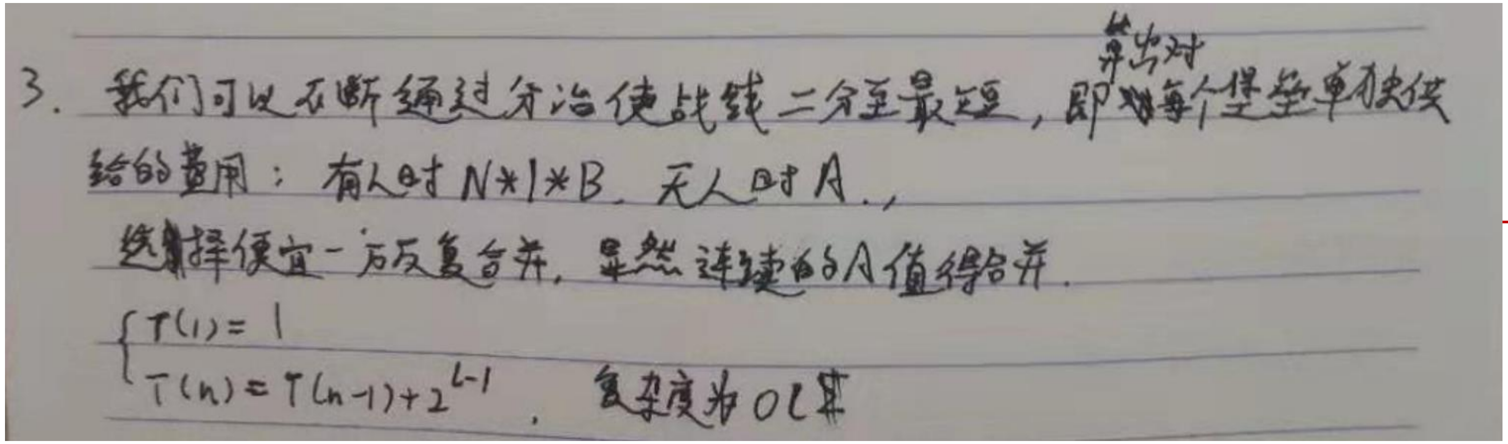
情况 2：均分补给

伪代码如下：

```
MinCost(n,l,left,right)

begin
while l != 0 do
if(isnull([left,right ])) then
cost ← cost + A;
end
else begin
if soilder_num([left,right ]) = 1then
cost ← cost + B ;
end
else begin
l ← l - 1;
end
end
end
End

Input:n,l

Output:min

Find cost(n,l)
begin
MinCost(n,l,1,2^(l-1));
MinCost(n,l,2^(l-1),2^l);
return minimum of the two cost;
end
```

伪码格式不规范，未分析算法时间复杂度。

低分案例


3. 我们可以不断通过分治使战线二分至最短，即算出对每个堡垒单独供给的费用：有人时 N*l*B，无人时 A。，
选择便宜一方反复合并，显然连续的A值得合并。

$$\begin{cases} T(1)=1 \\ T(n)=T(n-1)+2^{l-1} \end{cases}$$

，复杂度为O(l

文字描述非常笼统，
没有说明时间复杂度是怎么得出的。

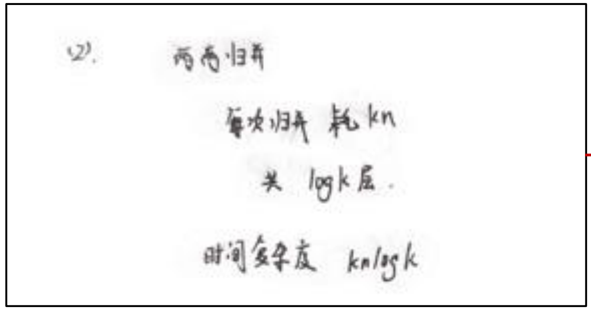
三、战线补给问题

当不分段运输战线补给的时候，最小费用为$\frac{(1+2^{l})\ *2\wedge l}{2}*2^{l}*B$,此时的时间复杂度为

O(n^2)。

只用一句话笼统地描述算法，未给出算法流程，
时间复杂度没有分析流程。

低分案例



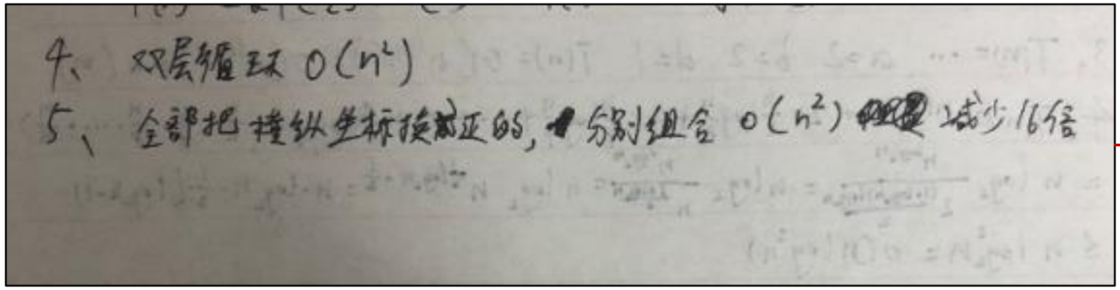
(2). 两两归并

每次归并 耗kn

共 logk层.

时间复杂度 knlogk

只提供了没有体现算法思路的潦草的文字描述。



4、双层循环 $O(n^2)$

5、全部把横纵坐标换成正的，分别组合 $O(n^2)$ 减少16倍

只有简单的一行算法描述且描述不清楚。

低分案例

五.

显然可对所有向量取绝对值即 $v_i=(|x_i|,|y_i|)$

有最短模长为 $(|x_i|+|x_i|)^2+(|y_i|+|y_i|)^2$ 对 x 进行平均分，取一直线 l: x = t;

令 t 的值为 x 的中位数，对于 l 两侧分别求最小值，有两侧最小值分别得 d1,d2

令 d=(d1,d2)再设两直线 l1,l2 分别距 l 为 d 有 l1,l2 内点 a,b 若可能为最小点对，则必有 d(a,b)<d,则可知 ab 两点定在一 d*2d 的空间中。又可知任何一侧点的距离都不小于 d，可推出矩形中至多有 6 个平面中的点，即检查 6*n/2 对点距离，y 轴同理分析可得

有 T(n) = 2T(n/2) + n

O(nlogn)

未给出时间复杂度分析流程。

五. 向量的最小和问题

遍历 i 从 1 到 n. j 从 $i+1$ 到 n

比较 $\begin{cases}(x_i+x_j)^2 \text{ 与 } (x_i-x_j)^2. \\ (y_i+y_j)^2 \text{ 与 } (y_i-y_j)^2\end{cases}$ 用两者小的相加

然后再在每组之间进行比较.

时间复杂度 为 $\frac{n(n-1)}{2} = O(n^2)$

算法流程描述不详细，

5

本题与书例 2.5 解法类似，可将两向量相加模长最小转换为求两点间最短距离，算法时间复杂度为 $O(n\log^2 n)$

算法描述过于笼统

提纲

算法表示

案例分析

撰写工具

Overleaf在线编辑器

- 在线多人协作编辑器
  - https://www.overleaf.com/
  - https://www.overleaf.com/latex/templates/tagged/algorithm

```latex
%最大子数组暴力算法
    \begin{algorithm}
        %\caption{暴力算法}
%       \begin{spacing}{0.8}
            \KwIn{一个大小为$n$的数组$A[1..n]$}
            \KwOut{最大子数组之和$Vmax$}
            $Vmax$ $\leftarrow$ $A[1]$\;
            \For{$i$ $\leftarrow$ $1$ $to$ $n$}
            {

                \For{$j$ $\leftarrow$ $i$ $to$ $n$}{
                    $V$ $\leftarrow$ $0$;{\color{blue}{// 计算 $V(i,j)$
                    \For{$x$ $\leftarrow$ $i$ $to$ $j$}{
                        $V$ $\leftarrow$ $V$ + $A[x]$\;
                    }
                    \If{$V$ $>$ $Vmax$ }{
                    $Vmax$ $\leftarrow$ $V$\;
                    }
                }
            }
            \KwRet $Vmax$\;
%       \end{spacing}
    \end{algorithm}

%最大子数组 递推算法
\begin{algorithm}
    %\caption{data-reuse algorithms}

    \KwIn{一个大小为$n$的数组$A[1..n]$}
    \KwOut{最大子数组之和$Vmax$}
    $Vmax$ $\leftarrow$ $A[1]$\;
    \For{$i$ $\leftarrow$ $1$ $to$ $n$}{
        $V$ $\leftarrow$ $0$;{\color{blue}{// 计算 $V(i,j)$}\\}
        \For{$j$ $\leftarrow$ $i$ $to$ $n$}{

            $V$ $\leftarrow$ $V$ + $A[j]$\;
        }
        \If{$V$ $>$ $Vmax$ }{
         $Vmax$ $\leftarrow$ $V$\;
        }
    }
    \KwRet $Vmax$\;
\end{algorithm}
```

输入: 一个大小为 n 的数组 $A[1..n]$
输出: 最大子数组之和 $Vmax$
$Vmax \leftarrow A[1]$;
for $i \leftarrow 1$ *to* n **do**
  for $j \leftarrow i$ *to* n **do**
    $V \leftarrow 0$;// 计算 $V(i,j)$
    for $x \leftarrow i$ *to* j **do**
      $V \leftarrow V + A[x]$;
    end
    if $V > Vmax$ **then**
      $Vmax \leftarrow V$;
    end
  end
end
return $Vmax$;

输入: 一个大小为 n 的数组 $A[1..n]$
输出: 最大子数组之和 $Vmax$
$Vmax \leftarrow A[1]$;
for $i \leftarrow 1$ *to* n **do**
  $V \leftarrow 0$;// 计算 $V(i,j)$
  for $j \leftarrow i$ *to* n **do**
    $V \leftarrow V + A[j]$;
  end
  if $V > Vmax$ **then**
    $Vmax \leftarrow V$;
  end
end
return $Vmax$;

Latex相关

- Latex官网
  - https://www.latex-project.org/
- Latex下载
  - https://www.latex-project.org/get/
- 使用教程
  - https://www.latex-tutorial.com/tutorials/
  - https://www.tug.org/twg/mactex/tutorials/ltxprimer-1.0.pdf

谢谢

BDA

Beihang University