**安徽大学集成电路学院**

**《算法设计与分析》第六次报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 算法设计与分析 | | | 日期 |  |
| 专业 |  |  |  |  |  |
| 【实验目的】  了解矩阵链乘法问题，并使用matlab语言对其进行实现 | | | | | |
| 【实验原理（预习）】  **矩阵链乘法问题介绍**  在矩阵链乘法问题中，我们有一系列矩阵需要相乘，并且我们需要确定一种最优的乘法顺序，使得计算这些矩阵的总乘法次数最少。这里的乘法次数指的是标量乘法的次数，而不是矩阵乘法的次数。  假设我们有 𝑛 个矩阵 𝐴1,𝐴2,…,𝐴𝑛，其中矩阵 𝐴𝑖的维度为 𝑝𝑖−1×𝑝𝑖。问题的目标是找到一个乘法次序，使得计算矩阵链 𝐴1,𝐴2,…,𝐴𝑛 所需的标量乘法次数最少。  解决该问题的伪代码如下   |  | | --- | | 伪代码 | | 算法1 RecurMatrixChain(P, i, j)  1. m[i, j] ← ∞  2. s[i, j] ← i  3. for k←i to j-1 do  4. q ← RecurMatrixChain(P, i, k)  + RecurMatrixChain(P, k+1, j)  + p\_{i-1} \* p\_k \* p\_j  5. if q < m[i, j]  6. then m[i, j] ← q  7. s[i, j] ← k  8. Return m[i, j] |   **伪代码解释**  这段伪代码实现了一个递归的矩阵链乘法算法，具体步骤如下：  1. 初始化：将最小乘法次数 . 初始化为正无穷大（表示尚未计算过），并将分割点初始化为起始值 。  2. 遍历可能的分割点：从i到j逐一尝试将矩阵链 Ai到Aj分成两个子链。  3. 递归计算：递归计算子链Ai到Ak 以及Ak+1到Aj的最小乘法次数，并加上将这两个子链结果相乘的乘法次数。  4. 更新最小乘法次数：如果当前计算的总乘法次数小于已知的最小值，则更新和对应的分割点。  5. 返回结果：返回子链Ai到Aj的最小乘法次数。  **具体步骤**  步骤1：初始化，设定最小乘法次数为无穷大。  步骤2：初始化分割点为起始点i。  步骤3：尝试所有可能的分割点k，将链Ai到Aj分成两个子链Ai到Ak和Ak+1到Aj。  步骤4：递归计算两个子链的最小乘法次数，并加上合并两个子链所需的乘法次数。  步骤5：如果当前分割点的总乘法次数小于已知的最小乘法次数，则更新和。  步骤6：最终返回作为子链Ai到Aj的最小乘法次数。  ### 算法复杂度  递归版本的矩阵链乘法问题求解方法具有指数级时间复杂度，因此通常采用动态规划来优化此算法，使其时间复杂度降低到。 | | | | | |
| 【实验内容与记录（题号、操作步骤、数据记录与处理、附图编号、代码等）】  matlab代码   |  | | --- | | RecurMatrixChain.m | | function [result, s] = RecurMatrixChain(P, i, j)      % 初始化 m 和 s 矩阵      n = length(P) - 1;      m = inf(n, n);      s = zeros(n, n);      % 调用递归函数并返回 m(i, j)      [result, m, s] = RecurMatrixChainRecursive(m, s, P, i, j);  end  function [result, m, s] = RecurMatrixChainRecursive(m, s, P, i, j)      if m(i, j) < inf          result = m(i, j);          return;      end      if i == j          m(i, j) = 0;      else          for k = i:j - 1              [m(i,k),m,s] = RecurMatrixChainRecursive(m,s,P,i,k);              [m(k+1,j),m,s] = RecurMatrixChainRecursive(m,s,P,k+1,j);              q = m(i, k) + m(k + 1, j) + P(i) \* P(k + 1) \* P(j + 1);              if q < m(i, j)                  m(i, j) = q;                  s(i, j) = k;              end          end      end      result = m(i, j);  end |  |  | | --- | | Input.m | | P = [30, 35, 15, 5, 10, 20, 25];  [i, j] = size(P);  [result, s] = RecurMatrixChain(P, 1, j - 1);  fprintf('最优解为: %d\n', result);  fprintf('括号化方案为：\n');  print\_optimal\_parens(s, 1, j - 1);  function print\_optimal\_parens(s, i, j)      if i > size(s, 1) || j > size(s, 2)          error('Index out of bounds');      end      if i == j          fprintf('A%d', i);      else          fprintf('(');          print\_optimal\_parens(s, i, s(i, j));          print\_optimal\_parens(s, s(i, j) + 1, j);          fprintf(')');      end  end |     得到了正确的结果，并且获得了最优括号化方案((A1(A2A3))((A4A5)A6)) | | | | | |

|  |
| --- |
| 【小结与讨论】  在本次实验中，我使用MATLAB语言成功实现了矩阵链乘法的动态规划算法。通过这次实践，我对算法设计与分析课程中学习的动态规划思想有了更深入的理解。  矩阵链乘法问题是一个经典的动态规划问题，其目标是找到一个最优的矩阵相乘顺序，使得总的乘法操作次数最少。通过将问题分解为子问题，并利用最优子结构性质，我们可以设计出一个高效的算法来解决这个问题。  在实验过程中，我首先分析了问题的特点，确定了动态规划的状态转移方程。然后，我使用MATLAB语言实现了算法，并进行了测试。通过观察输出结果，我验证了算法的正确性。  实验结果表明，动态规划算法能够有效地解决矩阵链乘法问题。通过填充m和s矩阵，算法能够找到最优的矩阵相乘顺序，并且时间复杂度为O(n^3)，其中n是矩阵的数量。这比暴力枚举所有可能的相乘顺序要高效得多。  在实现过程中，我遇到了一些问题，例如数组索引无效等。通过仔细检查代码并进行调试，我最终解决了这些问题，使得程序能够正常运行。  总的来说，这次实验让我对动态规划算法有了更深刻的认识。我意识到，在设计算法时，需要仔细分析问题的特点，找出最优子结构和重叠子问题，并根据状态转移方程设计算法。同时，我也提高了使用MATLAB语言实现算法的能力，学会了如何调试和优化代码。  在未来的学习中，我将继续深入研究动态规划和其他算法设计技术，并将它们应用到更多的实际问题中。同时，我也将继续提高编程能力，使用不同的编程语言实现算法，以便更好地解决实际问题。  通过这次实验，我对算法设计与分析这门课程有了更大的兴趣。我期待着学习更多的算法，并将它们应用到实际问题中，以提高自己的问题解决能力。 |