****

算法设计结课论文

矩阵乘法链问题的求解性能比较

子集和数的求解性能比较

学生：潘祎哲 | 指导老师：王征 | 日期：2020.12.25日期

实验目的

掌握利用算法进行问题求解的实验流程

强化对课堂所学算法原理的理解

提升利用算法原理进行编码实践的能力

了解算法研究的前沿动态，拓展算法改进的思路

实验课题

2.3算法求解类问题4.2对于子集和数利用回溯算法和4.3矩阵乘法链利用动态规划算法进行求解并分析不同算法解决同一问题的复杂度，归纳不同求解算法的优缺点。

研究问题描述

1. 子集和数

给定n个正数w(i)和另一个正数M，找出{w(i)，i=1,2,3..,n}中所有使得和数等于M的子集。

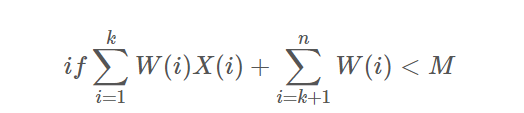
1. 矩阵乘法链

给定一个n个矩阵的序列（矩阵链）<A1,A2,...,An>，我们希望计算它们的乘积 A1A2...An使得按照某种最优组合可以使得所使用的乘法次数最少

回溯法子集和数设计流程

# 设计思路

这里我们可以思考每次展开这个子节点前先思考对于这个节点K是否有前面所有已选择的数的和+k节点后面所有数的总和值是否能够>=M，如果不能，则Kill掉次子节点不再展开，因为已知最大和值都已经不可能等于M了，则就没有必要展开了，限界函数表达如下:其中X(i)表示取1表示选择，取0表示不选择



后面一项没有乘X(i)是因为默认就是全部选择所以全部乘1，就没必要写了，然后表达式设置为小于是因为只有满足次条件是才应该返还true值从而Kill子节点。符合限界函数的定义要求。

回溯法可以理解成用深度优先算法遍历状态空间树，所以可以使用深度优先递归算法得到解；

定义MAXN为30，即暂时考虑最多有29个元素的集合；

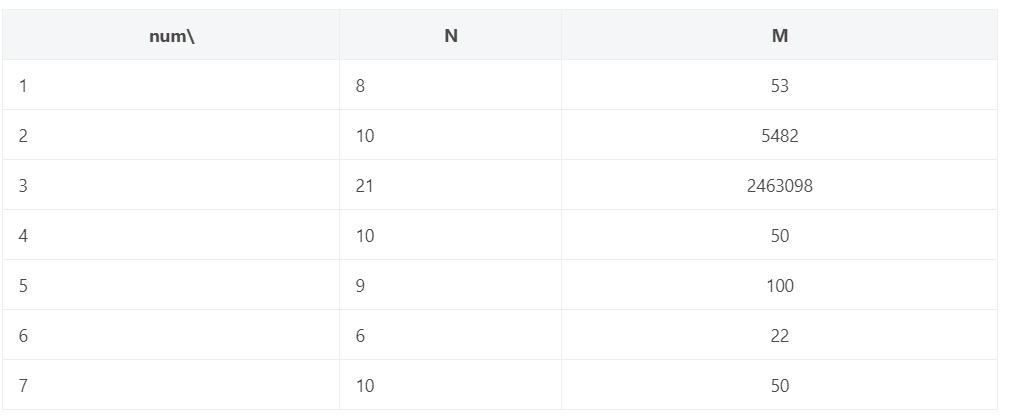
在DFS函数中传入需要的参数：

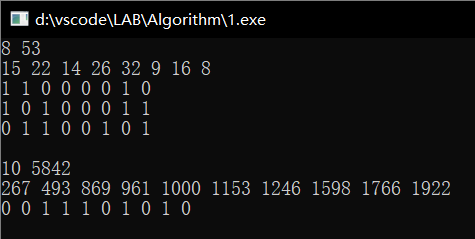
* TotalWeight：当前子集中所选的元素之和；
* RestWeight：除选出的元素外剩余元素之和；
* x[ ]：问题的一个解，由0和1组成，0代表xi不取，1代表取，1≤i≤n；
* i：当前元素位置；
* n，m，w[ ]：元素个数，和数，和所给集合；
* 在DFS函数中：
* 当TotalWeight与m相等时，输出一个解；
* 加上当前元素时还小于m，考虑下一位置元素，继续递归；
* 若加上剩余元素时大于m，则不选择当前位置元素，继续递归；
* 递归结束时，即可得到所有解。

# 代码展示

1. #include <iostream>
2. #define MAXN 30
3. **using** **namespace** std;
4. **void** Solution\_DFS(**int** TotalWeight, **int** RestWeight, **int** x[], **int** i, **int** n, **int** m, **int** w[]);
5. //回溯法解决子集和数问题
6. **int** main()
7. {
8. **int** n, m;
9. **while** (cin >> n >> m)
10. {
11. **int** w[MAXN] = {0};
12. **int** RestWeight = 0;
13. **for** (**int** i = 1; i <= n; i++)
14. {
15. **int** temp;
16. cin >> temp;
17. w[i] = temp;
18. RestWeight += temp;
19. }
20. **int** x[MAXN] = {0};
21. Solution\_DFS(0, RestWeight, x, 1, n, m, w);
22. }
23. system("pause");
24. **return** 0;
25. }
26. **void** Solution\_DFS(**int** TotalWeight, **int** RestWeight, **int** x[], **int** i, **int** n, **int** m, **int** w[])
27. {
28. **if** (i > n)
29. {
30. **if** (TotalWeight == m)
31. {
32. **for** (**int** i = 1; i <= n; i++)
33. cout << x[i] << " ";
34. cout << endl;
35. }
36. }
37. **else**
38. {
39. **if** (TotalWeight + w[i] <= m)
40. {
41. x[i] = 1;
42. Solution\_DFS(TotalWeight + w[i], RestWeight - w[i], x, i + 1, n, m, w);
43. }
44. **if** (TotalWeight + RestWeight > m)
45. {
46. x[i] = 0;
47. Solution\_DFS(TotalWeight, RestWeight - w[i], x, i + 1, n, m, w);
48. }
49. }
50. }

# 测试结果

我们采用的数据集均来自Florida State University: 7个数据集：（具体输入样例请查看文件夹内子集和数.txt）



对于不同的输入样例所得到的的结果数值表如下：

# 算法分析

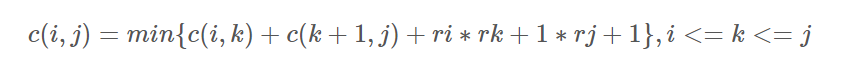
在加上限界条件进行剪枝策略后避免了许多一直不可能得到最终结果的情况，相较于深搜加快了搜索效率，最终时间复杂度可控制在O（2^n）。

动态规划法设计流程

# 一．设计思路

学过线性代数的同学都知道，不同的矩阵相乘方法，所用的乘法次数也是不同的，比如假定A为100×1矩阵,B为1×100矩阵,C为100×1矩阵,(A\*B)\*C需乘法数为100×1×100＋100×100×1＝20000而 A\*(B\*C) 所需乘法数为1×100×1＋100×1×1＝200,(温馨提示：nm×n矩阵A与n×p矩阵B相乘需要做m\*n\*p个元素乘法,原因想一想就懂了)。

矩阵乘法链问题就是求得所用乘法次数最少的乘法顺序并求得最小乘法次数，我们考虑，每次Mi×…,×Mj (i<=j)，每次将这部分分成两部分，即去一个i<=k<=j，将矩阵乘法链分为M(i, k)×M(k+1,j)。那么计算M(i,j)的优化乘法顺序在计算子链M(i,k)和M(k+1,j)时的也是优化的。因此我们可以得到以下递归式：



用c(i,j)表示为计算M(i，j)的优化乘法数，并且kay(i,j)为达到最小值得k，并且c(i,i+1)表示Mi×Mi+1,即两个矩阵相乘无法在分隔可以求得具体数值，c(i,i)则表示一个矩阵即默认最后在乘所以次数为0，所以可以解题。

使用动态规划方法解决矩阵乘法链问题，需在不同大小的子问题的优化值之间建立递归关系，得到最优解；

同时满足优化原理，即优化解包含的子问题的解也是优化解。使用枚举法建立不同长度子问题的优化值之间的递归关系；

求此问题的优化解，即求c(1,n)，c(1,n)=min{c(1,k)+c(k+1,n)+r1×r(k+1)×rn}，利用此式得到问题的解，再根据每次得到的k值回溯找到优化的乘法顺序；

在代码中，可以使用二维数组存放每次递归关系式中大括号中的每个值，最后找到最小值，即可得到问题的优化解。

# 二、代码展示

1. #include <iostream>
2. **using** **namespace** std;
3. //动态规划解决矩阵乘法链问题
4. **const** **int** INT\_MAX=2147483647;
5. **const** **int** MAXN=10;
6. **void** Solution(**int** \*r,**int** Length,**int** m[][MAXN],**int** s[][MAXN]);
7. **void** POMCWP(**int** s[][MAXN],**int** i,**int** j);//Acronyms Print Optimal Matrix Chain With Parentheses
8. **int** main()
9. {
10. **int** n;
11. **while**(cin>>n)
12. {
13. **int** r[MAXN]={0};
14. **for**(**int** i=0;i<n+1;i++)
15. {
16. **int** temp;
17. cin>>temp;
18. r[i]=temp;
19. }
20. **int** m[MAXN][MAXN],s[MAXN][MAXN];
21. Solution(r,n+1,m,s);
22. cout<<"Minimum: ";
23. cout<<m[1][n]<<endl;
24. cout<<"Order: ";
25. POMCWP(s,1,n);
26. }
27. system("pause");
28. **return** 0;
29. }
30. **void** Solution(**int** \*r,**int** Length,**int** m[][MAXN],**int** s[][MAXN])
31. {
32. **int** q,n=Length-1;
33. **for**(**int** i=1;i<=n;i++) m[i][i]=0;
34. **for**(**int** l=2;l<=n;l++)
35. {
36. **for**(**int** i=1;i<=n-l+1;i++)
37. {
38. **int** j=i+l-1;
39. m[i][j]=INT\_MAX;
40. **for**(**int** k=i;k<=j-1;k++)
41. {
42. q=m[i][k]+m[k+1][j]+r[i-1]\*r[k]\*r[j];
43. **if**(q<m[i][j])
44. {
45. m[i][j]=q;
46. s[i][j]=k;
47. }
48. }
49. }
50. }
51. }
52. **void** POMCWP(**int** s[][MAXN],**int** i,**int** j)
53. {
54. **if**(i == j) cout<<"M"<<i;
55. **else**
56. {
57. cout<<"(";
58. POMCWP(s,i,s[i][j]);
59. POMCWP(s,s[i][j]+1,j);
60. cout<<")";
61. }
62. }

# 三、测试结果

采用的是github: 9个数据集。部分测试结果如下：（具体输入样例请查看文件夹内矩阵乘法链.txt）

Minimum: 15125  
Order: ((M1(M2M3))((M4M5)M6))

Minimum: 2200  
Order: ((M1(M2M3))M4)

# 四、算法分析

使用动态规划建立递归式每次都选取子问题中最小的乘法次数进行求解，相较于枚举法减少了许多不必要的计算次数，每次只需要求解最优情况进行计算，并且得到不同最优解所对应的key值和最终的最少乘法次数和乘法链组合，计算开销大幅减小，时间复杂度控制在O(N^3),由于使用二维数组进行存储中途计算数值，空间复杂度为 O(N^2)。

文献参考

1. <https://blog.csdn.net/bendanban/article/details/73603884>
2. <https://blog.csdn.net/gl620321/article/details/108801724>
3. 《算法导论》回溯法和动态规划章节以及子集和数与矩阵乘法连案例分析