**算法分析与设计实验报告**

**第 4 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 杨杰 | 学号 | 201908010705 | | 班级 | 计科1907 |
| 时间 | 6.2 | 地点 | 软件大楼 | | | |
| 实验名称 | 用回溯法实现工作分配问题 | | | | | |
| 实验目的 | 通过上机实验，要求掌握回溯法实现工作分配问题的问题描述、算法设计思想、程序设计。 | | | | | |
| 实验原理 | 解空间为排列树，用回溯法(dfs)搜索所有方案找到最优解。 | | | | | |
| 实验步骤 | 1. 判断第j个工作是否已经完成，若未完成，将其完成，向下一层搜索 2. 当不满足约束函数或限界函数时，剪枝 3. 回溯 | | | | | |
| 关键代码 | void Backstrack(int i, int c)*// i表示从第i个人开始 ，c表示当前需要的总费用*  {      if (c > scost)      {  *// cout << "剪枝\n";*          return;      }      if (i == n + 1)*// 当最后一个人也分配好工作后判断总费用*      {          if (c < scost)              scost = c;          return;      }      for (int j = 1; j <= n; j++)      {          if (isC[j] == 0)*// 判断第j个工作是否已经完成，类似剪枝函数*          {              isC[j] = 1;  *// cout << "第" << i << "个人完成第" << j << "个工作\n";*              Backstrack(i + 1, c + cost[i][j]);              isC[j] = 0;*// 回溯法*  *// cout << "第" << j << "个工作未完成\n";*          }      }  } | | | | | |
| 算法复杂度分析 | 时间复杂度：  当所给问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树。这类排列树通常有n!个叶结点。遍历排列树的算法需O(n!)计算时间。  空间复杂度：  需要用到二维数组cost[N][N]来表示第i个人完成第j件工作需要的费用，所以空间复杂度为O(n^2). | | | | | |
| 测试结果  （含运行时间） | **小规模数据**    **中规模数据**    **大规模数据**    从这里也可以看出O(n!)的时间复杂度是非常恐怖的，数据规模仅仅大了一点，程序运行时间却是天壤之别。 | | | | | |
| 实验心得 | 通过这个实验，我复习了回溯法。  回溯法思路的简单描述是：把问题的解空间转化成图或者树的结构表示，然后使用深度优先搜索策略进行遍历，遍历的过程中记录和寻找所有可行解或者最优解。  基本思想类同于：   * 图的深度优先搜索 * 二叉树的后序遍历   回溯法按深度优先策略搜索问题的解空间树。首先从根节点出发搜索解空间树，当算法搜索至解空间树的某一节点时，先利用剪枝函数判断该节点是否可行（即能得到问题的解）。如果不可行，则跳过对该节点为根的子树的搜索，逐层向其祖先节点回溯；否则，进入该子树，继续按深度优先策略搜索。  回溯法的基本行为是搜索，搜索过程中使用剪枝函数来避免无效的搜索。剪枝函数包括两类：1. 使用约束函数，剪去不满足约束条件的路径；2.使用限界函数，剪去不能得到最优解的路径。  经过这次实验，我对于用回溯法实现工作分配问题的相关代码已基本熟悉，算法知识得到了复习与巩固。在写代码与调试的过程中，在解决问题过程中，丰富了个人编程的经历和经验，提高了个人解决问题的能力。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

*//回溯法解工作分配*

#include <iostream>

#include <fstream>

#include <windows.h>

using namespace std;

#define N 1000

int cost[N][N];*// 表示第i个人完成第j件工作需要的费用*

int isC[N] = {0};*// 用于记录第n个工作是否完成，0表示未完成*

int n;

int scost;*// 表示总费用*

void Backstrack(int i, int c)*// i表示从第i个人开始 ，c表示当前需要的总费用*

{

    if (c > scost)

    {

*// cout << "剪枝\n";*

        return;

    }

    if (i == n + 1)*// 当最后一个人也分配好工作后判断总费用*

    {

        if (c < scost)

            scost = c;

        return;

    }

    for (int j = 1; j <= n; j++)

    {

        if (isC[j] == 0)*// 判断第j个工作是否已经完成，类似剪枝函数*

        {

            isC[j] = 1;

*// cout << "第" << i << "个人完成第" << j << "个工作\n";*

            Backstrack(i + 1, c + cost[i][j]);

            isC[j] = 0;*// 回溯法*

*// cout << "第" << j << "个工作未完成\n";*

        }

    }

}

int main()

{

    ifstream infile("gongzuofenpei\_in.txt", ios::in);

    ofstream outfile("gongzuofenpei\_out.txt", ios::out);

    double time = 0;

    LARGE\_INTEGER nFreq, nBeginTime, nEndTime;

    infile >> n;

    outfile << n << endl;

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= n; j++)

        {

            infile >> cost[i][j];

            outfile << cost[i][j] << " ";

        }

        outfile << endl;

    }

    infile.close();

    scost = N;*// 给总费用设置一个很大的值*

    QueryPerformanceFrequency(&nFreq);

    QueryPerformanceCounter(&nBeginTime);*//开始计时*

    Backstrack(1, 0);

    QueryPerformanceCounter(&nEndTime);*//停止计时*

    time = (double)(nEndTime.QuadPart - nBeginTime.QuadPart) / (double)nFreq.QuadPart;*//计算程序执行时间单位为s*

    outfile << "程序耗时" << time \* 1000 << "ms" << endl;

    outfile << "最小总费用为" << scost;

    outfile.close();

    return 0;

}

*//工作分配随机数据生成器*

#include <iostream>

#include <fstream>

#include <time.h>

#define random(a, b) (rand() % (b - a + 1) + a)

using namespace std;

int main()

{

    ofstream outfile("gongzuofenpei\_in.txt", ios::out);

    srand((int)time(NULL));

    int n = 0;

    cin >> n;

    outfile << n << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

            outfile << random(1, 20) << " ";

        outfile << endl;

    }

    return 0;

}