**东北大学计算机科学与工程学院**

**数据结构课程设计报告**

题目 男女运动员最佳组合

课题组长 白千一

课题组成员 金忠信

专业名称 计算机科学与技术

班级 计1907

指导教师 杨雷

2020 年 1月

**课程设计任务书**

|  |
| --- |
| **题目：**  男女运动员最佳组合 |
| **问题描述：**  设有N个男羽毛球运动员和N个女羽毛球运动员，现组成N对男女混合最佳组合。每个男运动员对每个女运动员都有一个满意度排序，用矩阵mf[0:n-1][0:n-1]表示。mf[i][j]表示第i个男运动员对第j个女运动员的满意度，满意度值越高，满意程度越高。同理，每个女运动员对每个男运动员也有一个满意度排序，用矩阵fm[0:n-1][0:n-1]表示。男女运动员之间的一个完全匹配称为一个组合。 |
| **设计要求：**  设计对于给定的满意度，求最佳组合的程序，使得满意度总和达到最大。  （1）采用STL的一维向量类构造构造二维向量矩阵。  （2）应用基本运算，设计算法求解。 |
| **指导教师签字：**  **年　　月　　日** |

1. 课题概述

## 1.1课题任务

在给定男女运动员相互的满意度的情况下，求得一种匹配情况，使得匹配度的总和达到最大。

## 1.2 课题原理

1. 简单来说，如果无向图G=(V,E)中点可以被分为两组，并且使得所有边都跨越组的边界，则这就是一个二分图。准确地说：把一个图的顶点划分为两个不相交集U和V，使得每一条边都分别连接U、V中的顶点。如果存在这样的划分，则此图为一个二分图。如果G为加权二分图,则权值和最大的完备匹配称为最佳匹配。
2. KM算法，功能是求完备匹配下的最大权匹配。在一个二分图内，左顶点为X，右顶点为Y，现对于每组左右连接XiYj有权wij，求一种匹配使得所有wij的和最大。
3. 回溯算法实际上一个类似枚举的搜索尝试过程，主要是在搜索尝试过程中寻找问题的解，当发现已不满足求解条件时，就“回溯”返回，尝试别的路径。回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法。

## 1.3相关知识

1. 回溯算法及其“剪枝”
2. KM算法及其优化
3. 文件流相关知识的熟悉掌握。
4. 需求分析

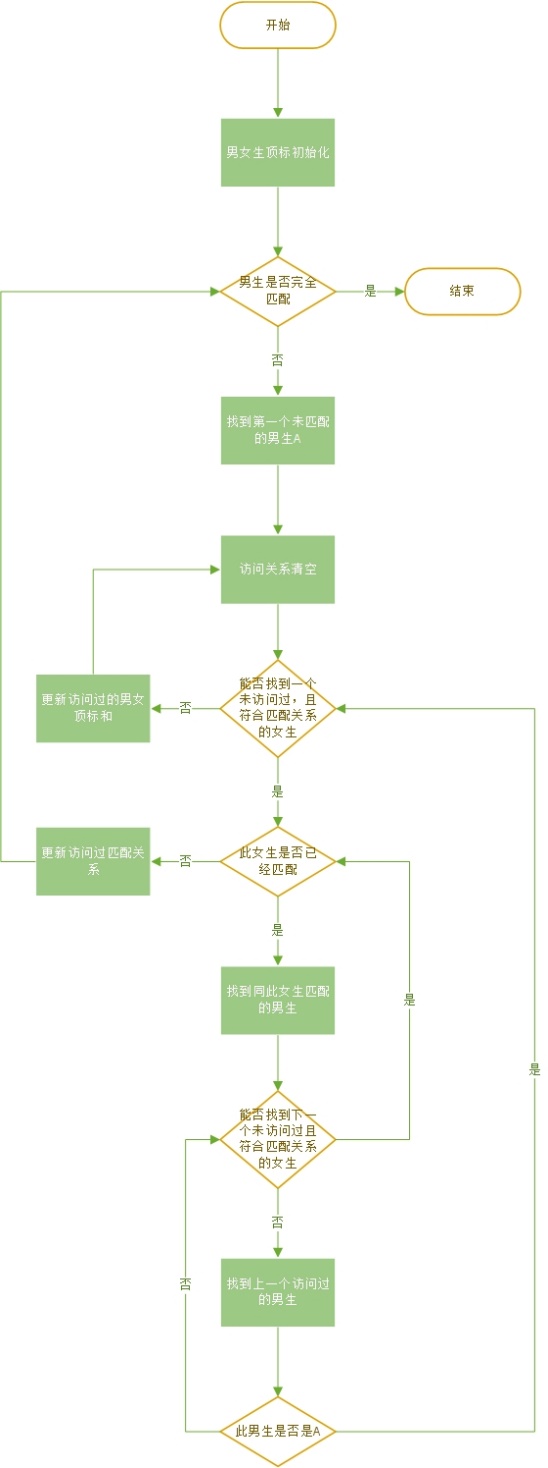
## 2.1 课题调研

在经济管理生活中，经常面临双边匹配的问题，本课题是其中双方满意度最大匹配的问题，抽象出来的数学模型就是二分图最大匹配问题，本课题的解决可以为同样二分图匹配的其他问题提供解决思路。

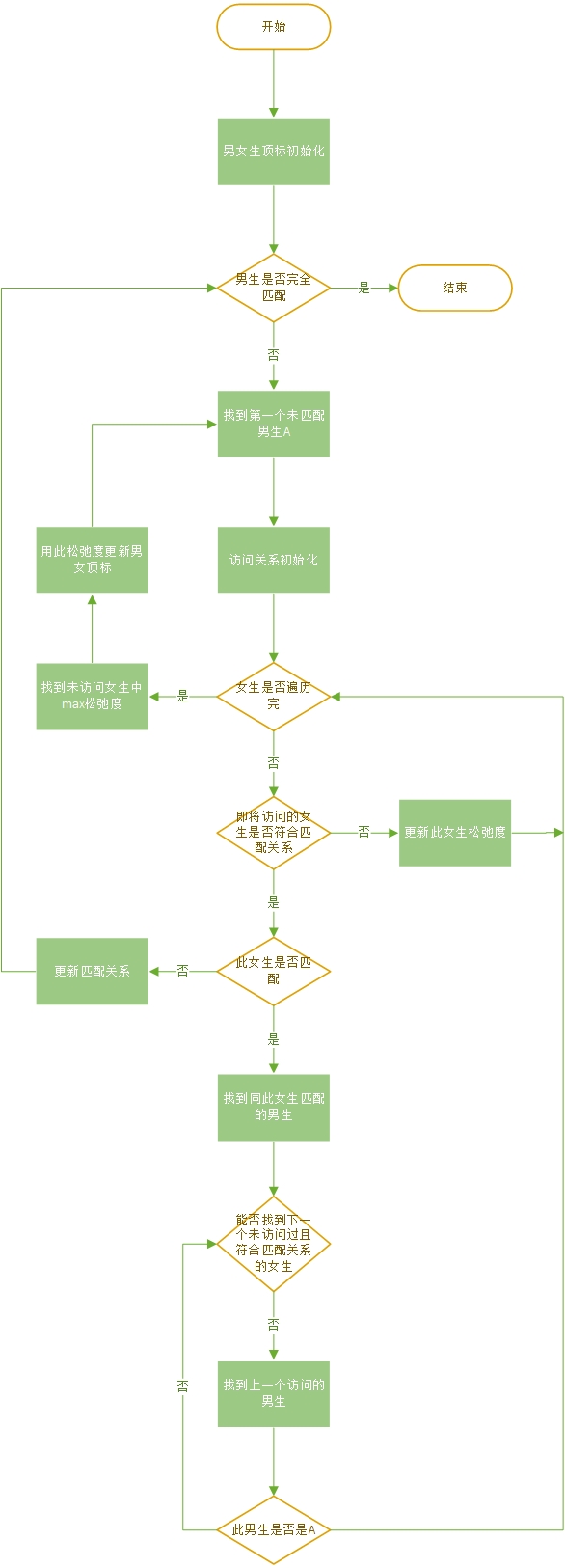
## 用户需求分析

在经济管理生活中，经常面临双边匹配的问题，比如出行场景中乘客与司机的匹配、物流领域中货物与车辆的匹配、教学领域学生与教师的匹配、营销领域奖励与用户的匹配等。在现实世界稀缺资源约束下(比如人力、物力、财力等)，我们希望最终做出的决策达到某种效率的最优，这里的效率可以是时间最少、行驶路程最短、双方满意度等，可以是多种单一指标的综合。

1. 方案设计
2. KM算法设计



1. KM优化算法设计



1. 回溯算法设计

通过对男女匹配度的计算和转化，可以得到每个男女生匹配度组成的二维向量矩阵。这样这个问题就转化成了在矩阵的每一行中选取一个值，使得这些值不同行不同列，同时加和最大。利用回溯算法，在每一层中选取一个值，计算最后的总加和，搜索全部情况，选取最优解。

1. 回溯+剪枝算法设计

在解决过程中可以发现，如果当前层数累加和与下面全部层数每层匹配度的最大值的累加和相加仍小于历史最优解，就没有继续向下搜索的必要了。通过此种“剪枝”，可以降低计算量，获得更高的时间效率。

所以在上述剪枝算法中引入一个“当前层数最大值累加和”向量，在搜索过程中，如果剩余层数累加和（总层数累加和减去当前层数累加和）加上当前计算的和值小于历史最优解，则放弃搜索此条路径，回溯到上一层继续计算。

1. 方案实现

## 4.1开发环境与工具

硬件：个人PC

软件：Visual Studio 2019 + CLion 2020

开发环境：Windows 10 + VS2019 + CLion2020

## 4.2程序设计关键技术

1. KM算法求最大匹配问题
2. 加入“松弛度”的计算，对顶标减小量进行优化
3. 深度优先搜索（DFS）
4. 回溯算法及“剪枝”操作

## 4.3个人设计实现

### 4.3.1白千一个人设计实现

1. 利用KM算法解决问题

|  |
| --- |
| //男女运动员的数量  int N;  int minz;  //男女生匹配度，转化后为男女匹配的权重  vector<vector<int> > weight;  //男女生的顶标  vector<int>x\_ver;  vector<int>y\_ver;  //男女生匹配的对象下标  vector<int>x\_match;  vector<int>y\_match;  //女生是否已经匹配  vector<bool>visited;  //增广路径上存在的男女生标号  vector<int>pathx,pathy;    void init()  {  //对各变量初始化  weight = vector<vector<int> >(N,vector<int>(N,0));  x\_ver = y\_ver = vector<int>(N,0);  x\_match = y\_match = vector<int>(N,-1);  visited = vector<bool>(N,false);  pathx.clear();  pathy.clear();  minz = 1;  }  void input()  {  //输入男生对女生的匹配度  for(int i = 0; i < N; ++i)  for(int j = 0; j < N; ++j){  cin>>weight[i][j];  }  //输入女生对男生的匹配度，并进行计算获得对应男女生的权重  for(int i = 0; i < N; ++i)  for(int j = 0; j<N; ++j){  int m;  cin>>m;  weight[j][i] \*= m;  }  //利用stl求取男生顶标  for(int i = 0; i < N; ++i)  x\_ver[i] = \*(max\_element(weight[i].begin(),weight[i].end()));  }  bool dfs(int u)  {  //将此男生加入增广路径  pathx.emplace\_back(u);  //遍历女生，找到顶标和等于权重的女生  for(int v = 0; v < N; ++v){  if(weight[u][v] != x\_ver[u] + y\_ver[v])  continue;  if(visited[v]) continue;  visited[v] = true;  pathy.emplace\_back(v);  //找到条件合适的女生，检查是否已经匹配，或者能否经过调整匹配成功。  if(y\_match[v] == -1 || dfs(y\_match[v])){  y\_match[v] = u;  x\_match[u] = v;  return true;  }  }  return false;  }    void km()  {  for(int i = 0; i < N; ++i){  while(true){  pathx.clear();  pathy.clear();  visited = vector<bool>(N,false);  //对男生i进行匹配  if(dfs(i)) break;  //匹配不成功，男女生减少顶标  for(const auto&it:pathx){  x\_ver[it] -= minz;  }  for(const auto&it:pathy){  y\_ver[it] += minz;  }  }  }  } |

1. 利用KM优化算法解决问题

|  |
| --- |
| //加入了松弛度的计算，对顶标每次减少量进行了优化。  #define inf INT\_MAX  int N;  vector<vector<int> > weight;  vector<int>x\_ver;  vector<int>y\_ver;  vector<int>x\_match;  vector<int>y\_match;  vector<bool>visited;  vector<bool>pathx,pathy;  //加入松弛度  vector<int>slack;  void init()  {  weight = vector<vector<int> >(N,vector<int>(N,0));  x\_ver = y\_ver = vector<int>(N,0);  x\_match = y\_match = vector<int>(N,-1);  pathx = pathy = vector<bool>(N,false);  visited = vector<bool>(N,false);  slack = vector<int>(N,inf);  pathx.clear();  pathy.clear();  }  void input()  {    for(int i = 0; i < N; ++i)  for(int j = 0; j < N; ++j){  cin>>weight[i][j];  }  for(int i = 0; i < N; ++i)  for(int j = 0; j<N; ++j){  int m;  cin>>m;  weight[j][i] \*= m;  }  for(int i = 0; i < N; ++i)  x\_ver[i] = \*(max\_element(weight[i].begin(),weight[i].end()));  }  bool dfs(int u)  {  pathx[u] = true;  for(int v = 0; v < N; ++v){  if(visited[v]) continue;  int gap = x\_ver[u] + y\_ver[v] - weight[u][v];  if(gap == 0){  visited[v] = true;  pathy[v] = true;  if(y\_match[v] == -1 || dfs(y\_match[v])){  y\_match[v] = u;  x\_match[u] = v;  return true;  }  }  else slack[v] = min(slack[v], gap);  }  return false;  }    void km()  {  for(int i = 0; i < N; ++i){  slack = vector<int>(N,inf);  while(true){  pathx = pathy = visited = vector<bool>(N,false);  if(dfs(i)) break;  int minz = inf;  //选取最小的松弛度  for(int i = 0; i < N; ++i)  if(!pathy[i]) minz = min(minz, slack[i]);  //利用最小的松弛度，调整每次顶标减小的量  for(int i = 0; i < N; ++i)  if(pathx[i]) x\_ver[i] -= minz;  for(int i = 0; i< N; ++i)  if(pathy[i]) y\_ver[i] += minz;  }  }  } |

### 4.3.2金忠信个人设计实现

1. 利用回溯方法解决问题

|  |
| --- |
| int N;  int answer;  //利用stl向量类存储男女匹配度信息  vector<vector<int> >mf;  vector<vector<int> >fm;  vector<vector<int> >sum;  //女生是否被选过  vector<bool>chosed;  //最终匹配结果存放向量  vector<int>match;  void init()  {  //各变量初始化  match.clear();  chosed.clear();  mf.clear();  fm.clear();  answer = -1;  sum = fm = mf = vector<vector<int> >(N, vector<int>(N, 0));  chosed = vector<bool>(N, false);  }  void input()  {  //输入男生对女生的匹配度  for (int i = 0; i < N; ++i)  for (int j = 0; j < N; ++j) {  int m;  cin >> m;  mf[i][j] = m;  }  //输入女生对男生的匹配度  for (int i = 0; i < N; ++i)  for (int j = 0; j < N; ++j) {  int m;  cin >> m;  fm[i][j] = m;  }  //计算男女生之间的权重（路径长）  for (int i = 0; i < N; ++i)  for (int j = 0; j < N; ++j)  sum[i][j] = mf[i][j] \* fm[j][i];  }  //利用回溯遍历男女生权重组成的二维向量矩阵  void dp(vector<int>& ve, int level = 1, int num = 0)  {  if (level > N) {  if (answer < num) {  answer = num;  match = ve;  }  }  for (int i = 0; i < N; ++i) {  if (!chosed[i]) {  chosed[i] = true;  ve.emplace\_back(i + 1);  dp(ve, level + 1, num + sum[level - 1][i]);  ve.pop\_back();  chosed[i] = false;  }  }  } |

1. 利用回溯+剪枝解决问题

|  |
| --- |
| int N;  int answer;  vector<vector<int> >sum;  //加入和向量：表示前n个男生全部选取最大匹配度的女生的匹配度和  vector<int>sigma;  vector<bool>chosed;  vector<int>match;  void init()  {  match.clear();  chosed.clear();  sum.clear();  answer = -1;  sum = vector<vector<int> >(N,vector<int>(N,0));  chosed = vector<bool>(N,false);  sigma = vector<int>(N,0);  }  void input()  {  for(int i = 0; i < N; ++i)  for(int j = 0; j < N; ++j){  cin>> sum[i][j];  }  for(int i = 0; i < N; ++i){  for(int j = 0; j < N; ++j){  int m;  cin >> m;  sum[j][i] \*= m;  }  }  //计算前n项和  for(int i = 0; i < N; ++i){  sigma[i] = \*max\_element(sum[i].begin(),sum[i].end()) + ((i-1>=0)?sigma[i-1]:0);  }  }  void dp(vector<int> &ve,int level = 1,int num = 0)  {  if(level > N){  if(answer < num){  answer = num;  match = ve;  }  }  for(int i = 0; i < N ;++i){  if(!chosed[i]){  //进行剪枝，当剩余男生全部选最大都比当前答案小的话，就不继续向下考虑了  if(num+sum[level-1][i]+sigma[N-1]-sigma[level-1] < answer)  continue;  chosed[i] = true;  ve.emplace\_back(i+1);  dp(ve, level+1, num+sum[level-1][i]);  ve.pop\_back();  chosed[i] = false;  }  }  } |

1. 测试和调试

## 5.1测试数据

3

10 2 3

2 3 4

3 4 5

2 2 2

3 5 3

4 5 1

3

2 3 9

9 4 2

5 9 4

4 9 8

3 2 9

9 1 6

4

1 2 3 4

4 3 2 1

7 6 5 3

1 2 4 7

8 6 4 7

3 4 5 8

4 5 7 3

4 5 6 7

5

1 3 4 5 6

4 2 3 5 6

4 5 2 5 5

3 5 5 6 3

4 6 7 23 54

25 45 345 657 34

42 45 644 23 46

243 35 56 2 4

342 534 35 243

342 23 3 3 1

预计结果为52、243、115、14047四个最大匹配值

## 5.2个人测试

### 5.2.1白千一测试

1. KM算法测试

|  |
| --- |
| 1. 编写测试程序   int main()  {  freopen(stdin,"testdata.txt", "r");  while (cin >> N) {  init();  input();  vector<int>ve;  dp(ve);  //输出匹配结果  for (int i = 0; i < N; ++i) {  cout << "man" << i + 1 << " mtach:" << match[i] << endl;  }  //输出最大匹配度  printf("最大匹配度为：%d\n", answer);  }  fclose(stdin);  return 0;  }  同时引入一个全局变量，记录计算的总迭代次数   1. 运行程序观察运行结果     从运行结果看，每次匹配都对当前情况进行判断，遇到不满足条件的情况，就对匹配值进行调整，直到结果正确。从最后一行可以得知，递归次数为6410次 |

1. KM优化算法测试

|  |
| --- |
| 1. 编写测试程序   int main()  {  freopen(stdin,"testdata.txt", "r");  while (cin >> N) {  init();  input();  vector<int>ve;  dp(ve);  //输出匹配结果  for (int i = 0; i < N; ++i) {  cout << "man" << i + 1 << " mtach:" << match[i] << endl;  }  //输出最大匹配度  printf("最大匹配度为：%d\n", answer);  }  fclose(stdin);  return 0;  }  同时引入一个全局变量，记录计算的总迭代次数   1. 运行程序观察结果     根据结果来看，通过测试数据得到的计算数值正确。最后递归次数都远小于非优化算法递归次数，大大提升了算法效率 |

### 5.2.2金忠信测试

1. 回溯算法测试

|  |
| --- |
| 1. 编写测试程序   int main()  {  freopen(stdin,"testdata.txt", "r");  while (cin >> N) {  init();  input();  vector<int>ve;  dp(ve);  //输出匹配结果  for (int i = 0; i < N; ++i) {  cout << "man" << i + 1 << " mtach:" << match[i] << endl;  }  //输出最大匹配度  printf("最大匹配度为：%d\n", answer);  }  fclose(stdin);  return 0;  }  同时引入一个全局变量，记录计算的总迭代次数   1. 运行程序观察结果     可见测试结果正确 |

1. 回溯+剪枝算法测试

|  |
| --- |
| 1. 编写测试程序   int main()  {  freopen(stdin,"testdata.txt", "r");  while (cin >> N) {  init();  input();  vector<int>ve;  dp(ve);  //输出匹配结果  for (int i = 0; i < N; ++i) {  cout << "man" << i + 1 << " mtach:" << match[i] << endl;  }  //输出最大匹配度  printf("最大匹配度为：%d\n", answer);  }  fclose(stdin);  return 0;  }  同时引入一个全局变量，记录计算的总迭代次数   1. 运行程序观察结果     结果同上面的方法，都为正确。不同的是，经过剪枝后，回溯次数都有了下降，在性能上有较大提高 |

1. 课题总结

## 6.1课题评价

本次课题的B题为“求解运动员最大匹配度问题”。小组共采用四种不同的方法对此问题进行求解，分别是：KM算法、回溯算法，以及分别对这两种算法在时间复杂度方面上的优化。经过多组数据的测试和调试，我们成功地采用了多种方案解决了此问题。同时也存在不足之处，没有对本课题进行可视化界面的编写，使得本课题的解决思路过于的抽象难懂。

## 6.2团队协作

整个工程分工细致，任务明确，在组长的带领下，小组全体组员出色的完成了自己的任务。经过各个成员的反复修改，小组程序的错误越来越少，并在验收中一次通过。在任务完成过程中，团队成员的团队协作能力得到明显的提高。

## 6.3下一步工作

首先计划对代码进行优化，剔除冗余代码，增加注释提升代码可读性和可维护性。其次，学习更加优秀的压缩代码对程序性能进行优化，在保证找出最大匹配度的同时，降低时间复杂度，使算法获得更优良的时间性能。其次，尝试将本次解决过程可视化，编写界面，直观的展现整个匹配过程。

## 6.4个人设计小结

### 6.4.1白千一设计小结

本次课题完成过程中，最提升能力的地方是对新算法的学习和代码的编写。拿到课题题目后，查阅了大量资料，进行了大量思考，对新的算法彻底理解后才进行代码的编写，而且编写过程中出现了各种错误，解决的过程中加深了对代码编写的理解。不足之处在于，认为本题重点在于算法，没有可视化匹配的具体过程。

### 6.4.2金忠信设计小结

课题完成过程中，我想到的方法是“暴力回溯法”，但是这种方法的无论在时间复杂度还是空间复杂度上都欠佳，因此我查找了大量相关资料，并于组长多次讨论，了解到了通过“剪枝”来减少查找次数的方法。由于该题是二分图的问题，我们想到了“匈牙利算法”。在查找资料后，我们发现还有更加适合这个题目的“KM算法”，并在学习算法的过程中了解到“增广路径”等知识，开阔了我的视野，使我惊叹于这种方法的巧妙，也激发了我对算法学习的兴趣。

1. 附录

## A课题任务分工

### A-1课题程序设计分工

**课题程序设计分工**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 程序设计函数原型、类 | 功能说明 |
| 20194701 | 白千一 | 1. Km算法 2. Km算法优化 | 1. Km算法及其优化解题 |
| 20194744 | 金忠信 | 1. 回溯算法 2. 回溯算法优化 | 1. 回溯算法及其优化解题 |

### A-2课题报告分工

**课题报告分工**

|  |  |  |
| --- | --- | --- |
| 章节 | 内容 | 完成人 |
| 1 课题概述 | 1.1 课题任务  1.2 课题原理  1.3 相关知识 | 白千一 |
| 2 需求分析 | 2.1 课题调研  2.2 用户需求分析 | 金忠信 |
| 3 方案设计 | 3.1 总体功能设计  3.2 数据结构设计  3.3 函数原型设计  3.4 输入输出设计  3.5 主算法设计  3.6 用户界面设计 | 白千一  金忠信 |
| 4 方案实现 | 4.1 开发环境与工具  4.2 程序设计关键技术  4.3 个人设计实现（按组员分工）  4.3.1白千一个人设计实现  4.3.2金忠信个人设计实现 | 白千一  金忠信 |
| 5 测试与调试 | 5.1 个人测试（按组员分工）  5.1.1白千一测试  5.1.2金忠信测试  5.2 组装与系统测试 | 白千一  金忠信 |
| 6 课题总结 | 6.1 课题评价  6.2 团队协作  6.3 下一步工作  6.4 个人设计心得（按组员分工）  6.4.1白千一设计小结  6.4.2金忠信设计小结 | 白千一  金忠信 |

## B课题设计文档

### B-1源程序代码（见附录项目文件夹）

### B-2工程与可执行文件（见附录项目文件夹）