实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | **实验四 回溯算法设计** | | |
| 实验日期 | 2023年4月18日 |  |  |
| 学 号 | 2021213193 | 姓 名 | 李田 |
| 专业班级 | 2021级软件工程 2班 | | |
| 指导教师 | 谷志新 | | |

东北林业大学

软件工程专业

|  |
| --- |
| 1. 实验目的   掌握用回溯法解题的算法框架；根据回溯法解决实际问题。 |
| 1. 实验环境   Windows 7 以上操作系统，PC机，codeblocks环境 |
| 1. 实验内容及结果   算法总体思想：回溯法的基本做法是搜索，或是一种组织得井井有条的，能避免不必要搜索的穷举式搜索法。这种方法适用于解一些组合数相当大的问题。回溯法在问题的解空间树中，按深度优先策略，从根结点出发搜索解空间树。算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解。如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先策略搜索。  （1）问题的解向量：回溯法希望一个问题的解能够表示成一个n元式(x1,x2,…,xn)的形式。  （2）显约束：对分量xi的取值限定。  （3）隐约束：为满足问题的解而对不同分量之间施加的约束。  （4）解空间：对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。  基本步骤：  （1）针对所给问题，定义问题的解空间，主要有子集树（如图1所示）和排列树（如图2所示）两种解空间形式。  （2）确定易于搜索的解空间结构；  （3）以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。  t51  图1 子集树  t53  图2 排列树  **1、0-1背包问题**  有N件物品和一个容量为V的背包。第i件物品的重量是w[i]，价值是v[i]。求解将哪些物品装入背包可使这些物品的重量总和不超过背包容量，且价值总和最大。要求每个物品要么放进背包，要么不放进背包。  #include <bits/stdc++.h>  using namespace std;  typedef long long LL;  const int N = 5050;  #define rep(i, a, b) for (int i = a; i <= b; i++)  #define dep(i, a, b) for (int i = a; i >= b; i--)  #define lowbit(x) (x & -x)  const double eps = 1e-8;  clock\_t startTime;  double getCurrentTime()  {      return (double)(clock() - startTime) / CLOCKS\_PER\_SEC;  }  int v[N], w[N];  int f[N][N];  int n,m;  void solve()  {      cin>>m>>n;      for(int i=1;i<=n;i++)cin>>v[i]>>w[i];      // 求解背包可以获得的最大值      for(int i=1;i<=n;i++)      {          for(int j=0;j<=m;j++)          {             if(j>=v[i]) f[i][j]=max(f[i-1][j],f[i-1][j-v[i]]+w[i]);             else f[i][j]=f[i-1][j];          }      }      cout<<"MaxValue=="<<f[n][m]<<endl;      vector<int >ans;      int i=n,j=m;      //自顶向下，用非递归的回溯求出装入了所有装入物品的下标。      while(i&&j)      {          // f[i][j]- f[i-1][j],f[i-1][j-v[i]]+w[i]            if(f[i][j]-w[i]==f[i-1][j-v[i]]) ans.push\_back(i),j-=v[i],i--;          else i--;      }      for(int i=ans.size()-1;i>=0;i--)      {          cout<<ans[i]<<" "<<endl;      }  }  // 90 5  // 3  12  // 56 28  // 77 49  // 21 9  // 87 35  int main()  {      int tt;      tt = 1;      startTime = clock();      while (tt--)      {          solve();          // printf("%.2lf",getCurrentTime());      }  }  **测试样例：**    **2、旅行售货员问题**  设有一个售货员从城市1出发，到城市2,3,..,n去推销货物，最后回到城市1。假定任意两个城市i，j间的距离dij(dij=dji)是已知的，问他应沿着什么样的路线走，才能使走过的路线最短。  #include <bits/stdc++.h>  using namespace std;  typedef long long LL;  #define PII pair<int,int>  const int N = 5050;  #define rep(i, a, b) for (int i = a; i <= b; i++)  #define dep(i, a, b) for (int i = a; i >= b; i--)  #define lowbit(x) (x & -x)  const double eps = 1e-8;  clock\_t startTime;  double getCurrentTime()  {      return (double)(clock() - startTime) / CLOCKS\_PER\_SEC;  }  int dist[N];  int f[N][N];  int st[N];  int d[N][N];  int pre[N][N];  int n,m;  vector<int>res;  void dfs(int x,int y)  {      if(pre[x][y]==x) return ;      int t=pre[x][y];      if(t) res.push\_back(t);      dfs(t,y);      return ;  }  void solve()  {      int n;      int fina;        cin>>n; cin>>fina;      rep(i,1,n)rep(j,1,n)cin>>d[i][j];      // 把pre数组初始化为正无穷。      memset(pre,0,sizeof pre);      rep(i,1,n) f[i][i]=0;      cout<<"start="<<1<<" "<<"endpoint="<<fina<<endl;      //采用floyd算法求出从1到n的最短路径（本质上利用动态规划算法）      rep(i,1,n)      rep(j,1,n)      rep(k,1,n)      {          if(d[i][j]>d[i][k]+d[k][j]) d[i][j]=d[i][k]+d[k][j],pre[i][j]=k;      }      cout<<"MinDistanceFrom-One-To-Final=="<<d[1][fina]<<endl;      res.push\_back(fina);      dfs(1,fina);      res.push\_back(1);      for(int i=res.size()-1;i>=0;i--)      {          cout<<res[i]<<"-\n"[i==0];      }  // 5 5  // 0 1 2 3 4  // 1 0 3 4 1  // 2 3 0 2 5  // 3 4 2 0 10  // 4 1 5 10 0  }  int main()  {      int tt;      tt = 1;      startTime = clock();      while (tt--)      {          solve();          // printf("%.2lf",getCurrentTime());      }  }  测试样例： |
|  |

|  |
| --- |
| 四、实验过程分析与讨论  在对背包问题状态转移的时候，出现了错误：  源代码：  for(int i=1;i<=n;i++)  for(int j=v[i];j<=m;j++)          {             f[i][j]=min(f[i-1][j],f[i-1][j-v[i]]+w[i]);          }  错因：当j<v[i]是状态没有转移，导致后续状态出错。  更正后代码  for(int i=1;i<=n;i++)  for(int j=0;j<=m;j++)          {             if(j>=v[i]) f[i][j]=max(f[i-1][j],f[i-1][j-v[i]]+w[i]);             else f[i][j]=f[i-1][j];          } |
| 五、指导教师意见  指导教师签字：  年 月 日 |
|  |