第 5 章 回溯法

习题 5-1　装载问题改进回溯法(一)

用主教材 5.2 节中的改进策略(1)重写装载问题回溯法,使改进后算法计算时间复杂性为$O(2^n)$。

分析与解答:

先运行只计算最优值的算法 backtrack1,计算出最优装载量 bestw。由于该算法不记录最优解,故所需的计算时间为 $O(2^n)$。

```
private static void backtrack1(int i)
{
    if (i>n) {bestw=cw;return;}
    r-=w[i];
    if (cw+w[i]<=c) {cw+=w[i];backtrack1(i+1);cw-=w[i];}
    if (cw+r > bestw)backtrack1(i+1);
    r +=w[i];
}
```

然后运行改进后的算法 backtrack,在首次到达的叶结点处,即首次遇到 $i>n$ 时终止算法。由此返回的 bestx 即为最优解。

```
private static void backtrack(int i)
{
   if(found)return;
   if(i>n){
        for (int j=1;j<=n;j++)bestx[j]=x[j];
        found=true;
        return;
   }
   r-=w[i];
   if(cw+w[i]<=c){
        x[i]=1;cw+=w[i];
        backtrack(i+1);
        cw-=w[i];
   }
```

```
    if(cw+r>=bestw){x[i]=0;backtrack(i+1);}
    r+=w[i];
}
```


习题 5-2　装载问题改进回溯法(二)

用主教材 5.2 节中的改进策略(2)重写装载问题回溯法,使改进后算法计算时间复杂性为 $O(2^n)$。

分析与解答:

在算法中动态地更新 bestx。在第 i 层的当前结点处,当前最优解由 $x[j]$,$1\leqslant j<i$ 和 bestx[j],$i\leqslant j\leqslant n$ 组成。每当算法回溯一层,将 $x[i]$存入 bestx[i]。这样在每个结点处更新 bestx 只需 $O(1)$时间,从而整个算法中更新 bestx 所需的时间为 $O(2^n)$。

```
private static void backtrack(int i)
{
    if (i>n){index=n;bestw=cw;return;}
    r -=w[i];
    if (cw+w[i]<=c){
        x[i]=1;cw+=w[i];
      backtrack(i+1);
      if (index==i){bestx[index]=1; index--;}
      cw -=w[i];
    }
    if (cw+r>bestw){
        x[i]=0;
        backtrack(i+1);
        if (index==i){bestx[index]=0; index--;}
    }
    r +=w[i];
}

public static int maxLoading(int [] ww, int cc, int [] xx)
{
    n=ww.length-1; w=ww; c=cc; cw=0; bestw=0;
    x=new int [n+1];bestx=xx; index=0;
    for (int i=1; i<=n; i++) r+=w[i];
    backtrack(1);
    return bestw;
}
```

习题 5-3　0-1 背包问题的最优解

重写 0-1 背包问题的回溯法,使算法能输出最优解。

分析与解答:

为了构造最优解,必须在算法中记录与当前最优值相应的当前最优解。为此,在类 Knap 中增加两个私有数据成员 x 和 bestx。x 用于记录从根至当前结点的路径;bestx 用于记录当前最优解。算法搜索到达叶结点处,就修正 bestx 的值。

修改后的算法描述如下。

backtrack 在回溯过程中记录从根至当前结点的路径。

```
private static void backtrack(int i)
{
    if (i>n){index=n;bestp=cp;return;}
    if (cw+w[i]<=c){
        x[i]=1;cw +=w[i];cp +=p[i];
        backtrack(i+1);
        if (index==i){bestx[id[i]]=1; index--;}
        cw -=w[i];cp -=p[i];
    }
    if (bound(i+1)>bestp){
        x[i]=0;
        backtrack(i+1);
        if (index==i){bestx[id[i]]=0; index--;}
    }
}
```

knapsack 作初始化,并用回溯法求解。

```
public static double knapsack(double [] pp, double [] ww, double cc, int [] xx)
{
    c=cc; n=pp.length-1; cw=0.0; cp=0.0; bestp=0.0;
    Element [] q=new Element [n];
    for (int i=1; i<=n; i++) q[i-1]=new Element(i, pp[i]/ww[i]);
    MergeSort.mergeSort(q);
    p=new double [n+1];
    w=new double [n+1];
    id=new int [n+1];
    for (int i=1; i<=n; i++){
        p[i]=pp[q[n-i].id];
        w[i]=ww[q[n-i].id];
        id[i]=q[n-i].id;
    }
    x=new int [n+1];
    bestx=xx; index=0;
    backtrack(1);
    if (bestp==0.0)for (int i=1; i<=n; i++) xx[i]=0;
    return bestp;
}
```

习题 5-4　最大团问题的迭代回溯法

试设计一个解最大团问题的迭代回溯法。

分析与解答:

与主教材中装载问题的迭代回溯法类似,最大团问题的迭代回溯法描述如下。

```
static void iterClique()
{
    for(int i=0;i<=n;i++)x[i]=0;
    int i=1;
    while(true){
        while(i<=n && ok(i)){x[i++]=1;cn++;}
        if(i>=n){
            for (int j=1;j<=n;j++)bestx[j]=x[j];
            bestn=cn;
        }
        else x[i++]=0;
        while(cn+n-i<=bestn){
            i--;
            while(i>0 && x[i]==0)i--;
            if(i==0) return;
            x[i++]=0;cn--;
        }
    }
}
```

ok 用于判断当前顶点是否可加入当前团。

```
static boolean ok(int i)
{
    for(int j=1;j<i;j++)if(x[j]>0 && a[i][j]==0) return false;
    return true;
}
```

IterClique 作初始化,并调用迭代回溯法求解。

```
public static int IterClique()
{
    cn=0;bestn=0;
    iterClique();
    return bestn;
}
```

习题 5-5　旅行售货员问题的费用上界

设 G 是有 n 个顶点的有向图,从顶点 i 发出的边的最大费用记为 $\max(i)$。

(1) 证明旅行售货员回路的费用不超过 $\sum_{i=1}^{n}\max(i)+1$。

(2) 在旅行售货员问题的回溯法中,用上面的界作为 bestc 的初始值,重写该算法,并尽可能地简化代码。

分析与解答:

(1) 任一旅行售货员回路可表示为 n 个顶点的一个排列$(\pi(1),\pi(2),\cdots,\pi(n))$,这个回路的费用为 $h(\pi)=\sum_{i=1}^{n}a(\pi(i),\pi(i \bmod n+1))$。由此可知

$$h(\pi)=\sum_{i=1}^{n}a(\pi(i),\pi(i \bmod n+1)) \leqslant \sum_{i=1}^{n}\max(\pi(i))=\sum_{i=1}^{n}\max(i)<\sum_{i=1}^{n}\max(i)+1$$

(2) 对图 G 的简单遍历即可计算出 $\sum_{i=1}^{n}\max(i)+1$ 的值。

```
static float tsp()
{
    bestc=1;
    for(int i=1;i<=n;i++){
        float MaxCost=0;
        for(int j=1;j<=n;j++)
            if(a[i][j]<Float.MAX_VALUE && a[i][j]>MaxCost)
            MaxCost=a[i][j];
        if(MaxCost==Float.MAX_VALUE) return Float.MAX_VALUE;
        bestc+=MaxCost;
    }
    x=new int[n+1];
    for(int i=1;i<=n;i++)x[i]=i;
    cc=0;
    backtrack(2);
    return bestc;
}
```

在主教材的 TSP 回溯法中，语句 bestc==Float.MAX_VALUE 可以删去，修改如下：

```
static void backtrack(int i)
{
    if(i==n){
        if(a[x[n-1]][x[n]]<Float.MAX_VALUE  &&
            a[x[n]][1]<Float.MAX_VALUE  &&
            (cc+a[x[n-1]][x[n]]+a[x[n]][1]<bestc)){
              for(int j=1;j<=n;j++)bestx[j]=x[j];
              bestc=cc+a[x[n-1]][x[n]]+a[x[n]][1];
        }
    }
    else{
        for(int j=i;j<=n;j++)
            if(a[x[i-1]][x[j]]<Float.MAX_VALUE && (cc+a[x[i-1]][x[j]]<bestc)){
                MyMath.swap(x,i,j);
                cc+=a[x[i-1]][x[i]];
                backtrack(i+1);
                cc-=a[x[i-1]][x[i]];
                MyMath.swap(x,i,j);
            }
    }
}
```

习题 5-6　旅行售货员问题的上界函数

设 G 是有 n 个顶点的有向图,从顶点 i 发出的边的最小费用记为 $\min(i)$。

(1) 证明图 G 的所有前缀为 $x[1:i]$的旅行售货员回路的费用至少为 $\sum_{j=2}^{i}a(x_{j-1},x_j)+\sum_{j=i}^{n}\min(x_j)$,其中 $a(u,v)$是边(u,v)的费用。

(2) 利用上述结论设计一个高效的上界函数,重写旅行售货员问题的回溯法,并与主教材中的算法进行比较。

分析与解答:

(1) 前缀为 $x[1:i]$的旅行售货员回路任一旅行售货员回路可表示为 n 个顶点的一个排列 $(x[1],x[2],\cdots,x[i],\pi(i+1),\pi(i+2),\cdots,\pi(n))$。

这个回路的费用为

$$h(\pi)=\sum_{j=2}^{i}a(x_{j-1},x_j)+a(x_i,\pi(i+1))+\sum_{j=i+1}^{n}a(\pi(j),\pi(j \bmod n+1))$$

由此可知,

$$\begin{aligned}h(\pi)&\geqslant\sum_{j=2}^{i}a(x_{j-1},x_j)+\min(x_i)+\sum_{j=i+1}^{n}\min(\pi(j))\\&=\sum_{j=2}^{i}a(x_{j-1},x_j)+\sum_{j=i}^{n}\min(x_j)\end{aligned}$$

(2) 先对图 G 简单遍历,计算出 $\sum_{i=1}^{n}\min(i)$的值。

算法实现题 5-1　子集和问题

★ 问题描述

子集和问题的一个实例为$\langle S,t\rangle$。其中,$S=\{x_1,x_2,\cdots,x_n\}$是一个正整数的集合,c 是一个正整数。子集和问题判定是否存在 S 的一个子集 S_1,使得 $\sum_{x\in S_1}x=c$。

试设计一个解子集和问题的回溯法。

★ 算法设计

对于给定的正整数的集合 $S=\{x_1,x_2,\cdots,x_n\}$和正整数 c,计算 S 的一个子集 S_1,使得 $\sum_{x\in S_1}x=c$。

★ 数据输入

由文件 input.txt 提供输入数据。文件第 1 行有 2 个正整数 n 和 c,n 表示 S 的大小,c 是子集和的目标值。第 2 行中有 n 个正整数,表示集合 S 中的元素。

★ 结果输出

将子集和问题的解输出到文件 output.txt 中。当问题无解时,输出“No solution!”。

输入文件示例	输出文件示例
input.txt	output.txt
5 10	2 2 6
2 2 6 5 4	

分析与解答：

与装载问题类似，可设计解子集和问题的回溯法如下：

```
static boolean backtrack(int i)
{
    if (i>n) {
        for (int j=1; j<=n; j++) bestx[j]=x[j];
        bestw=cw;
        if(bestw==c)return true;
        else return false;
    }
    r-=w[i];
    if (cw+w[i]<=c) {
        x[i]=1;
        cw+=w[i];
        if (backtrack(i+1)) return true;
        cw -=w[i];}
    if (cw+r>bestw) {
        x[i]=0;
        if(backtrack(i+1))return true;}
    r +=w[i];
    return false;
}
```

算法实现题 5-2　最小长度电路板排列问题

★ 问题描述

最小长度电路板排列问题是大规模集成电路系统设计中提出的实际问题。该问题的提法是，将 n 块电路板以最佳排列方案插入带有 n 个插槽的机箱中。n 块电路板的不同的排列方式对应于不同的电路板插入方案。

设 $B=\{1,2,\cdots,n\}$ 是 n 块电路板的集合。集合 $L=\{N_1, N_2,\cdots,N_m\}$ 是 n 块电路板的 m 个连接块。其中每个连接块 N_i 是 B 的一个子集，且 N_i 中的电路板用同一根导线连接在一起。

例如，设 $n=8$，$m=5$。给定 n 块电路板及其 m 个连接块如下：

$B=\{1,2,3,4,5,6,7,8\}$；$L=\{N_1, N_2, N_3, N_4, N_5\}$；

$N_1=\{4,5,6\}$；$N_2=\{2,3\}$；$N_3=\{1,3\}$；$N_4=\{3,6\}$；$N_5=\{7,8\}$。

这 8 块电路板的一个可能的排列如图 5-1 所示。

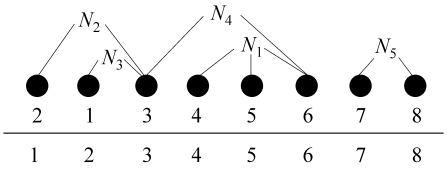

图 5-1　8 块电路板的排列

在最小长度电路板排列问题中，连接块的长度是指该连接块中第 1 块电路板到最后 1 块电路板之间的距离。例如，在图 5-1 所示的电路板排列中，连接块 N_4 的第 1 块电路板在插槽 3 中，它的最后 1 块电路板在插槽 6 中，因此 N_4 的长度为 3。同理 N_2 的长度为 2。图中连接块最大长度为 3。试设计一个回溯法找出所给 n 个电路板的最佳排列，使得 m 个连接块中最大长

度达到最小。

★ 算法设计

对于给定的电路板连接块,设计一个算法,找出所给 n 个电路板的最佳排列,使得 m 个连接块中最大长度达到最小。

★ 数据输入

由文件 input.txt 给出输入数据。第1行有2个正整数 n 和 m(其中 $1\leqslant m,n\leqslant 20$)。接下来的 n 行中,每行有 m 个数。第 k 行的第 j 个数为0表示电路板 k 不在连接块 j 中,1表示电路板 k 在连接块 j 中。

★ 结果输出

将计算出的电路板排列最小长度及其最佳排列输出到文件 output.txt。文件的第1行是最小长度;第2行是最佳排列。

输入文件示例	输出文件示例
input.txt	output.txt
8 5	4
1 1 1 1 1	5 4 3 1 6 2 8 7
0 1 0 1 0	
0 1 1 1 0	
1 0 1 1 0	
1 0 1 0 0	
1 1 0 1 0	
0 0 0 0 1	
0 1 0 0 1	

分析与解答:

与主教材中电路板排列问题类似,可设计解最小长度电路板排列问题的回溯法如下。主要区别是计算连接块的长度,由算法 len 完成。

```
static int len(int ii)
{
    for (int i=1; i<=m; i++) {high[i]=0;low[i]=n+1;}
    for (int i=1; i<=ii; i++)
        for (int k=1; k<=m; k++)
            if(B[x[i]][k]>0){
                if(i<low[k]) low[k]=i;
                if(i>high[k]) high[k]=i;
            }
    int tmp=0;
    for (int k=1; k<=m; k++)
        if(low[k]<=n && high[k]>0 && tmp<high[k]-low[k])tmp=high[k]-low[k];
    return tmp;
}
```

回溯法实体是 backtrack。

```
static void backtrack(int i)
{
    if (i==n) {
        int tmp=len(i);
        if(tmp<bestd){bestd=tmp;for (int j=1;j<=n;j++) bestx[j]=x[j];}
    }
    else
        for (int j=i; j<=n; j++) {
            MyMath.swap(x,i,j);
            int ld =len(i);
            if (ld<bestd) backtrack(i+1);
            MyMath.swap(x,i,j);
        }
}
```

最后由 arrange 完成计算。

```
public static int arrange(int [][]BB, int nn, int mm, int []bestxx)
{
    n=nn;m=mm;
    x=new int[n+1];
    low=new int[m+1];
    high=new int[m+1];
    B=BB;bestx=bestxx; bestd=n+1;
    for (int i=1; i<=n; i++) x[i]=i;
    backtrack(1);
    return bestd;
}
```

实现算法的主函数如下：

```
public static void main(String [] args)
{
    ReadStream keyboard=new ReadStream();
    int n=keyboard.readInt();
    int m=keyboard.readInt();
    int []p=new int[n+1];
    int [][]B=new int[n+1][m+1];
    for (int i =1; i<=n; i++)
        for (int j=1; j<=m; j++) B[i][j]=keyboard.readInt();
    System.out.println(arrange(B, n, m, p));
    for (int i=1; i<=n; i++) System.out.print(p[i]+" ");
    System.out.println();
}
```

算法实现题 5-3　最小重量机器设计问题

★ 问题描述

设某一机器由 n 个部件组成,每一种部件都可以从 m 个不同的供应商处购得。设 w_{ij} 是从供应商 j 处购得的部件 i 的重量,c_{ij} 是相应的价格。

试设计一个算法,给出总价格不超过 c 的最小重量机器设计。

★ 算法设计

对于给定的机器部件重量和机器部件价格,计算总价格不超过 d 的最小重量机器设计。

★ 数据输入

由文件 input.txt 给出输入数据。第 1 行有 3 个正整数 n,m 和 d。接下来的 $2n$ 行,每行 n 个数。前 n 行是 c,后 n 行是 w。

★ 结果输出

将计算出的最小重量,以及每个部件的供应商输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
3 3 4	4
1 2 3	1 3 1
3 2 1	
2 2 2	
1 2 3	
3 2 1	
2 2 2	

分析与解答:

与背包问题类似,可设计解最小重量机器设计问题的回溯法如下:

```
static boolean backtrack(int i)
{
    if (i>n) {
        bestw=cw;
        for(int j=1;j<=n;j++)bestx[j]=x[j];
        return true;
    }
    boolean found=false;
    if(bestw<=cc)found=true;
    for(int j=1;j<=m;j++){
        x[i]=j;cw+=w[i][j];cp+=c[i][j];
        if (cp<=cc && cw<bestw) if(backtrack(i+1)) found=true;
        cw -=w[i][j];cp -=c[i][j];
    }
    return found;
}
```