计算机算法设计与分析

分支限界法

易凯

2017 年 5 月 24 日

班　　级 软件 53 班

学　　号 2151601053

邮　　箱 williamyi96@gmail.com

联系电话 13772103675

个人网站 https://williamyi96.github.io

williamyi.tech

实验日期 2017 年 5 月 24 日

提交日期 2017 年 6 月 6 日

目录

插图

1 分支限界法与回溯法基本区别

1.1 求解目标不同

回溯法的求解目标是找出解空间中满足约束条件的所有解，而分支限界法的求解目标则是找出满足约束条件的一个解或者是某种意义上的最优解。

1.2 搜索方式不同

回溯法采用深度优先的方式对解空间进行搜索，而分支限界法采用广度优先或者是最小耗费优先的方式进行搜索。

2 分支限界法实现方式

分支限界法根据从活结点表中选择下一个扩展结点的不同方式从而派生出了分支限界法的两种不同实现方式。其分别为队列式分支限界法和优先队列式分支限界法。

2.1 队列式分支限界法

队列式分支限界法就是将活结点表排成一个队列，按照队列的先进先出原则选择下一个结点为当前的扩展结点。

2.2 优先队列式分支限界法

优先队列式的分支限界法将活结点表组织成一个优先队列，并按照优先队列中规定的结点优先级选取优先级最高的下一个结点成为当前扩展结点。

3 栈式分支限界法和回溯法的区别

3.1 题目描述

栈式分支限界法将活结点表以 LIFO(后进先出) 的方式存储在一个栈中。试设计一个解 0-1 背包问题的栈式分支限界法，并说明栈式分支限界法和回溯法之间的区别。

3.2 题目解答

该算法的基本思想较为清晰，具体的代码如下：

```
template<class Typew, class Typep>
Typep Knap<Typew, Typep>::StackKnapsack() {
    //栈式分支限界法，返回最大的价值，同时定义栈的容量初始化为10000
    S = new Stack<HeapNode<Typep, Typew>>(10000);
    int i = 1, e = 0, cw = cp = 0;
    //e为当前的扩展结点，cw为该结点对应的重量，cp表示的是相应的价值，up是价值上界。
    Typep bestp = 0; //bestp 为当前最优值；
    Typep up = Bound(1); //up为价值上界
    //进行子集空间树搜索
    while(1) {
        //检查当前扩展结点的左儿子结点
        Typew wt = cw + w[i]
        if(wt <= c) {
            //左儿子结点为可行结点
            if(cp + p[i] > bestp) bestp = cp + p[i];
            AddLiveNode(up, cp + p[i], cw + w[i], true, i+1);
        }
        up = Bound(i+1);
        //检查当前扩展结点的右儿子结点
        if(up >= bestp) //右子树可能含有最优解
            AddLiveNode(up, cp, cw, false, i+1);
        //取下一扩展结点
        if(S->empty()) return bestp;
        HeapNode<Typep, Typew> H;
        S -> pop(H);
        e = H.ptr;
        cw = H.weight;
        cp = H.profit;
        up = H.upprofit;
        i = H.level;
    }
}
```

然后通过上述方式进行结点的使用即可，其中后续的部分与有限队列式分支限界法大致相同。在此不进行赘述。

3.3 栈式分支限界法与回溯法的区别

根据其名称上来看，我们就可以得到栈式分支限界法与回溯法的最大区别就是分支限界法和回溯法的区别，而其主要是对于当前扩展结点所采用的扩展方式不同。在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中将不可行的解进行舍弃，然后再将可行的儿子结点加入表中。那些在或结点表中的结点然后依据栈的特点逐个取出相应结点，使其成为扩展结点，重复上述过程知道找到最优解。而由于回溯法使用的是深度优先策略，则其每个结点可能多次成为活结点。

4 修改 MaxLoading

4.1 题目描述

修改解装载问题的分支限界算法 MaxLoading，使得算法在结束前释放所有已由 EnQueue 产生的结点

4.2 题目求解

```

    以下仅说明主要更改的部分：
    while(1) {
        Type wt = ew - w[i];
        if(wt <= c) {//结点可选
            if(wt > bestw) bestw = wt;
            EnQueue(Q, wt, i, n, bestw, e, beste, bestx, true);
        }
    }
```

5 旅行售货员分支限界法的修改

5.1 题目描述

试修改解旅行售货员问题的分支限界法，使得算法保存已产生的排列树。

5.2 问题求解

remain to be seen.

6 无优先级运算问题

6.1 问题描述

给定 n 个正整数和 4 个运算符 +，-，*，/，且运算符无优先级，如 2+3*5=25。对于任意给定的整数 m，是设计一个算法，用以上给出的 n 个数和 4 个运算符，产生整数 m，且用的运算次数最少。给出的 n 个树中的每个数最多只能用 1 次，但每种运算符可以任意使用。

6.2 算法设计

对于给定的 n 个整数，采用回溯方法来设计一个算法，用最少的无优先级运算次数产生整数 m。

6.3 数据输入

输入的第一行有两个正整数 n 和 m，第二行是给定的用于运算的 n 个正整数。

6.4 结果输出

将计算的产生整数 m 的最少无优先级计算次数以及最优无优先级表达式输出到屏幕上。

6.5 程序代码

```
#include<iostream>
using namespace std;

int k;
class readin {
        friend int nreadin(int n,int m);
private:
```

```
8         bool found();                    //found判断是否找到解
9         bool search(int t);
10        int n,m,x;
11    int* a;                          //给定的用于运算的n个正整数的存放位置
12        int* num;                        //存放运算的产生整数m
13        int* operate;
14        int* flag;
15        char* ptr;                       //存储结果中的运算符
16};
17
18bool readin::search(int depth)  {
19        if(depth>k) {
20                if(found())
21                        return true;            //判断结点是否满足条件，即是否找到解
22                else
23                        return false;
24        }
25        else
26                for(int i=0;i<n;i++)
27                        if(flag[i]==0){
28                                num[depth]=a[i];
29                                flag[i]=1;
30                                for(int j=0;j<4;j++){
31                                        operate[depth]=j;
32                                        if(search(depth+1))
33                                                return true;
34                                }
35                                flag[i]=0;
36                        }
37                        return false;
38}
39
40bool readin::found(){
41        int x=num[0];
42        for(int i=0;i<k;i++){
43                switch (operate[i]){
44                case 0:x+=num[i+1];ptr[i]='+';break;
45                case 1:x-=num[i+1];ptr[i]='-';break;
46                case 2:x*=num[i+1];ptr[i]='*';break;
```

```
47                  case 3:x/=num[i+1];ptr[i]='/';break;
48                  }
49          }
50          return(x==m);
51  }
52
53  //读入初始数据
54  int nreadin(int n,int m){
55          readin X;
56          int* a=new int[n];
57      int* num=new int[n];
58      int* operate=new int[n];
59      int* flag=new int[n];
60          char* ptr=new char[n];
61          X.n=n;
62          X.m=m;
63          X.a=a;
64          X.operate=operate;
65          X.flag=flag;
66          X.num=num;
67          X.ptr=ptr;
68          for(int i=0;i<n;i++)
69  {
70                  cin>>a[i];
71                  flag[i]=0;
72          }
73          for(k=0;k<n;k++)
74                  if(X.search(0)){
75                  cout<<k<<endl;
76                  for(int i=0;i<=k;i++){
77                                  cout<<num[i]<<ptr[i];
78                          }
79                          cout<<endl;
80                          return 0;
81                  }
82                  return 0;
83  }
84
85  int main(){
```

```
86          int n;
87      int m;
88          cin>>n>>m;
89      nreadin(n,m);
90      return 0;
91  }
```

6.6 样例运行

```
5 25
5 2 3 6 7
2
2+3*5

--------------------------------
Process exited after 7.835 seconds with return value 0
```

图 1: 无优先级问题样例运行