**算法分析第6次作业**

算法实现题

6-1 使用队列式分支限界法，每个分支为是否对两个连接块进行交换，解的空间为排列树。使用low和high两个数组存储每个连接块的第一块电路板和最后一块电路板的位置，二者插值只增不减，如果当前分支前几个数的排列的最小长度已经大于已求得的最小长度，则进行剪枝。

struct Node

{

    int \*x,\*low,\*high;

    int level;

};

int len(Node &*node*,int *n*,int *m*)*//计算当前排列的长度*

{

    int temp=0;

    for(int k=1;k<=*m*;k++)

    {

        if(*node*.high[k]<=*n*&&*node*.low[k]>0&&temp<*node*.high[k]-*node*.low[k])

        {

            temp=*node*.high[k]-*node*.low[k];

        }

    }

    return temp;

}

int bestBoard(int \*\**B*,int *n*,int *m*,int \*&*bestx*)

{

    queue<Node> q;

    Node node;

    node.x=new int[*n*+1];

    for(int i=1;i<=*n*;i++)

    {

        node.x[i]=i;

    }

    node.low=new int[*m*+1];

    node.high=new int[*m*+1];

    for(int i=1;i<=*m*;i++)

    {

        node.low[i]=*n*+1;

        node.high[i]=0;

    }

    node.level=0;

    int bestd=*n*+1;

    do

    {

        if(node.level==*n*-1)*//如果到达队尾*

        {

            for(int j=1;j<=*m*;j++)

            {

                if(*B*[node.x[*n*]][j]&&*n*>node.high[j])*//将右值更新为最大值*

                {

                    node.high[j]=*n*;

                }

            }

            int ld=len(node,*n*,*m*);

            if(ld<bestd)*//更新最小长度，并更新最优排列*

            {

                bestd=ld;

                delete [] *bestx*;

*bestx*=new int[*n*+1];

                for(int i=1;i<=*n*;i++)

                {

*bestx*[i]=node.x[i];

                }

            }

            else

            {

                delete[] node.x;

            }

            delete[] node.low;

            delete[] node.high;

        }

        else

        {

            int curlevel=node.level+1;

            for(int i=node.level+1;i<=*n*;i++)

            {

                Node newnode;

                newnode.low=new int[*m*+1];

                newnode.high=new int[*m*+1];

                for(int j=1;j<=*m*;j++)

                {

                    newnode.low[j]=node.low[j];

                    newnode.high[j]=node.high[j];

                    if(*B*[node.x[i]][j])

                    {

                        if(newnode.low[j]>curlevel)

                        {

                            newnode.low[j]=curlevel;

                        }

                        if(newnode.high[j]<curlevel)

                        {

                            newnode.high[j]=curlevel;

                        }

                    }

                }

                int ld=len(newnode,*n*,*m*);

                if(ld<bestd)

                {

                    newnode.x=new int[*n*+1];

                    newnode.level=curlevel;

                    for(int j=1;j<=*n*;j++)

                    {

                        newnode.x[j]=node.x[j];

                    }

                    newnode.x[curlevel]=node.x[i];

                    newnode.x[i]=node.x[curlevel];*//交换两个位置上的元素*

                    q.push(newnode);

                }

                else*//当前排列已经不满足，剪枝*

                {

                    delete[] newnode.low;

                    delete[] newnode.high;

                }

            }

            delete[] node.x;

        }

        try{

            q.pop();

        }catch(out\_of\_range)

        {

            return bestd;

        }

    } while (!q.empty());

    return bestd;

}

6-2 使用优先队列式分支限界法，优先权值和小的结点，解空间树是子集树，根节点没有结点被选入集合，左子树表示选择结点进入集合，右子树表示不选节点进入集合。当节点不满足全覆盖时，才将左子树加入队列，右节点一定进入队列，当首次到达叶节点时，即最优解。

struct Node

{

    bool operator <(const Node &*node*) const

    {

        return weight>*node*.weight;*//定义优先级*

    }

    int num;*//节点序号*

    int weight;

    int \*x;

    bool \*c;*//覆盖的节点*

};

bool check(Node &*node*,int *n*)

{

    for(int i=1;i<=*n*;i++)

    {

        if(*node*.x[i]==0&&!*node*.c[i])

        {

            return false;

        }

    }

    return true;

}

int minCover(int \*\**a*,int \**weight*,int \*&*bestx*,int *n*)

{

    priority\_queue<Node,vector<Node>,Node> q;

    int bestd=0;

    Node node;

    node.x=new int[*n*+1];

    node.c=new bool[*n*+1];

    for(int i=1;i<=*n*;i++)

    {

        node.x[i]=0;

        node.x[i]=false;

    }

    int num=1,w=0;

    while(true)

    {

        if(num>*n*)*//到达叶节点*

        {

            if(check(node,*n*))

            {   for(int i=1;i<=*n*;i++)

                {

*bestx*[i]=node.x[i];

                }

                bestd=w;

                break;*//优先队列，直接是最优解*

            }

        }

        else

        {

            if(!check(node,*n*))*//当前没有完全覆盖，左节点加入到集合中*

            {

                Node newNode;

                newNode.x=new int[*n*+1];

                newNode.c=new bool[*n*+1];

                for(int i=1;i<=*n*;i++)

                {

                    newNode.x[i]=node.x[i];

                    newNode.c[i]=node.c[i];

                }

                newNode.x[num]=1;

                newNode.weight=w+*weight*[num];

                for(int i=1;i<=*n*;i++)

                {

                    if(*a*[num][i])

                    {

                        newNode.c[i]=true;*//如果有一点属于集合且与i相连，表明这个点被覆盖了*

                    }

                }

                newNode.num=num;

                q.push(newNode);

            }

*//右节点加入到集合中*

            Node newNode;

            newNode.x=new int[*n*+1];

            newNode.c=new bool[*n*+1];

            for(int i=1;i<=*n*;i++)

            {

                newNode.x[i]=node.x[i];

                newNode.c[i]=node.c[i];

            }

            newNode.x[num]=0;

            newNode.weight=w;

            newNode.num=num;

            q.push(newNode);

        }

        if(q.empty())

            break;

        node=q.top();

        q.pop();

        w=node.weight;

        num=node.num+1;

    }

    return bestd;

}

6-4 使用优先队列式分支限界法，优先重量小的节点，解空间树是排列树，每个节点都要从m个供应商中选择一个。当最小重量大于已求得的最小重量或当前总价格大于d时剪枝。

struct Node

{

    int weight,cost,i,j;

    Node(int *weight*,int *cost*,int *i*,int *j*):weight(*weight*),cost(*cost*),i(*i*),j(*j*){}

    bool operator <(const Node &*node*) const

    {

        if(weight!=*node*.weight)

            return weight>*node*.weight;

        else if(i!=*node*.i)

            return i>*node*.i;

        else

            return j>*node*.j;

    }

    int \*x;

};

int minWeightMachine(int \*\**c*,int \*\**w*,int *n*,int *m*,int *d*,int \*&*bestx*)

{

    priority\_queue<Node> q;

    int bestd=INT\_MAX;

    for(int i=0;i<*m*;i++)

    {

        Node temp=Node(*w*[0][i],*c*[0][i],0,i);

        if(temp.cost>*d*)

        {

            continue;

        }

        else

        {

            temp.x[0]=i+1;

            q.push(temp);

        }

    }

    while(!q.empty())

    {

        Node node=q.top();

        q.pop();

        if(node.i<*n*-1)

        {

            for(int i=0;i<*m*;i++)

            {

                Node temp=Node(node.weight+*w*[node.i+1][i],node.cost+*c*[node.i+1][i],node.i+1,i);

                if(temp.i==*n*-1)*//到达叶节点*

                {

                    if(temp.weight>bestd||temp.cost>*d*)

                    {

                        continue;

                    }

                    else if(temp.weight<bestd)

                    {

                        bestd=temp.weight;

                        for(int j=0;j<*n*-1;j++)

                        {

*bestx*[j]=node.x[j];

                        }

*bestx*[*n*-1]=i+1;

                    }

                }

                else

                {

                    if(temp.cost>*d*)

                    {

                        continue;

                    }

                    else

                    {

                        for(int j=0;j<temp.i;j++)

                        {

                            temp.x[j]=node.x[j];

                        }

                        temp.x[temp.i]=i+1;

                        q.push(temp);

                    }

                }

            }

        }

    }

    return bestd;

}

6-5 使用优先队列式分支限界法，优先当前和与剩余节点的上界之和大的节点，解空间树为排列树，通过由男选女，每个节点的分支表示是否交换女的排列顺序。使用优先队列，第一次到达叶节点即为最优解，可以通过判断当前层级之和与之前层级的最大值进行比较来剪枝，但只能减小队列的长度，对效率无影响。

struct Node

{

    int sum,r,i;

    int \*x;

    bool operator <(const Node &*node*) const

    {

        return sum+r<*node*.sum+r;

    }

};

int maxAdvance(int \*\**P*,int \*\**Q*,int *n*)

{

    vector<int> maxsum(*n*,0);

    for(int i=0;i<*n*;i++)

    {

        int temp=0;

        for(int j=0;j<*n*;j++)

        {

            temp=max(temp,*P*[i][j]\**Q*[j][0]);

        }

        maxsum[i]=temp;

    }

    priority\_queue<Node> q;

    Node node;

    node.sum=0;

    node.r=0;

    node.x=new int[*n*];

    node.i=0;

    for(int i=0;i<*n*;i++)

    {

        node.r+=maxsum[i];

        node.x[i]=i;

    }

    while(node.i!=*n*)

    {

        for(int i=node.i;i<*n*;i++)

        {

            Node newNode;

            newNode.i=node.i+1;

            newNode.x=new int[*n*];

            for(int j=0;j<*n*;j++)

            {

                newNode.x[j]=node.x[j];

            }

            newNode.x[node.i]=node.x[i];

            newNode.x[i]=node.x[node.i];

            newNode.sum=node.sum+*P*[node.i][newNode.x[node.i]]\**Q*[newNode.x[node.i]][node.i];

            newNode.r=node.r-maxsum[node.i];

            q.push(newNode);

        }

        if(!q.empty())

        {

            node=q.top();

            q.pop();

        }

        else

        {

            return 0;

        }

    }

    return node.sum;

}

6-10 使用优先队列式分支限界法，警卫数少的节点优先，解空间树为子集树，每个节点有放节点和不放警卫的分支，根节点没有放任何警卫。如果当前节点已经被监控，则只进入右子树，即不放警卫。

struct Node

{

    int i,j,numg,numd;*//警卫数、展柜数*

    vector<vector<int>> posg;*//警卫位置*

    vector<vector<int>> posd;*//展柜位置*

    bool operator <(const Node &*node*) const

    {

        return numg>*node*.numg||(numg==*node*.numg&&numd>*node*.numd);

    }

};

int dir[5][2]={{0,0},{0,1},{0,-1},{-1,0},{1,0}};*//方向：自己本身加上下左右*

int minGuard(int *n*,int *m*,int \*\**bestx*)

{

    priority\_queue<Node> q;

    int bestd=*n*\**m*;

    Node node;

    node.i=1;

    node.j=1;

    node.numg=0;

    node.numd=0;

    node.posg.resize(*n*+1,vector<int>(*m*+1,0));

    node.posd.resize(*n*+1,vector<int>(*m*+1,0));

    q.push(node);

    while(!q.empty())

    {

        Node node=q.top();

        q.pop();

        if(node.numd>=*n*\**m*)*//满足条件*

        {

            if(node.numg<bestd)

            {

                bestd=node.numg;

                for(int i=1;i<=*n*;i++)

                {

                    for(int j=1;j<=*m*;j++)

                    {

*bestx*[i][j]=node.posd[i][j];

                    }

                }

            }

        }

        else*//分三种情况*

        {

            if(node.i<*n*)*//在下面放守卫*

            {

                Node newNode;

                newNode.j=node.j;

                newNode.i=node.i+1;

                newNode.numg=node.numg+1;

                newNode.numd=node.numd;

                newNode.posg.resize(*n*+1,vector<int>(*m*+1,0));

                newNode.posd.resize(*n*+1,vector<int>(*m*+1,0));

                for(int i=1;i<=*m*;i++)

                {

                    for(int j=1;j<=*m*;j++)

                    {

                        newNode.posg[i][j]=node.posg[i][j];

                        newNode.posd[i][j]=node.posd[i][j];

                    }

                }

                newNode.posg[node.i+1][node.j]=1;

                for(int j=0;j<5;j++)

                {

                    int xx=node.i+1+dir[j][0],yy=node.j+dir[j][1];

                    newNode.posd[xx][yy]++;

                    if(newNode.posd[xx][yy]==1)

                    {

                        newNode.numd++;

                    }

                }

                while(newNode.i<=*n*&&newNode.posd[newNode.i][newNode.j]!=0)

                {

                    newNode.j++;

                    if(newNode.j>*m*)

                    {

                        newNode.j=1;

                        newNode.i++;

                    }

                }

                q.push(newNode);

            }

            if((node.i==*n*&&node.j==*m*)||node.posd[node.i][node.j+1]==0)*//在当前位置放守卫*

            {

                Node newNode;

                newNode.j=node.j;

                newNode.i=node.i;

                newNode.numg=node.numg+1;

                newNode.numd=node.numd;

                newNode.posg.resize(*n*+1,vector<int>(*m*+1,0));

                newNode.posd.resize(*n*+1,vector<int>(*m*+1,0));

                for(int i=1;i<=*m*;i++)

                {

                    for(int j=1;j<=*m*;j++)

                    {

                        newNode.posg[i][j]=node.posg[i][j];

                        newNode.posd[i][j]=node.posd[i][j];

                    }

                }

                newNode.posg[node.i][node.j]=1;

                for(int j=0;j<5;j++)

                {

                    int xx=node.i+dir[j][0],yy=node.j+dir[j][1];

                    newNode.posd[xx][yy]++;

                    if(newNode.posd[xx][yy]==1)

                    {

                        newNode.numd++;

                    }

                }

                while(newNode.i<=*n*&&newNode.posd[newNode.i][newNode.j]!=0)

                {

                    newNode.j++;

                    if(newNode.j>*m*)

                    {

                        newNode.j=1;

                        newNode.i++;

                    }

                }

                q.push(newNode);

            }

            if(node.j<*m*&&(node.posd[node.i][node.j+1]==0||node.posd[node.i][node.j+2]==0))*//当右边放守卫*

            {

                Node newNode;

                newNode.j=node.j+1;

                newNode.i=node.i;

                newNode.numg=node.numg+1;

                newNode.numd=node.numd;

                newNode.posg.resize(*n*+1,vector<int>(*m*+1,0));

                newNode.posd.resize(*n*+1,vector<int>(*m*+1,0));

                for(int i=1;i<=*m*;i++)

                {

                    for(int j=1;j<=*m*;j++)

                    {

                        newNode.posg[i][j]=node.posg[i][j];

                        newNode.posd[i][j]=node.posd[i][j];

                    }

                }

                newNode.posg[node.i][node.j+1]=1;

                for(int j=0;j<5;j++)

                {

                    int xx=node.i+dir[j][0],yy=node.j+1+dir[j][1];

                    newNode.posd[xx][yy]++;

                    if(newNode.posd[xx][yy]==1)

                    {

                        newNode.numd++;

                    }

                }

                while(newNode.i<=*n*&&newNode.posd[newNode.i][newNode.j]!=0)

                {

                    newNode.j++;

                    if(newNode.j>*m*)

                    {

                        newNode.j=1;

                        newNode.i++;

                    }

                }

                q.push(newNode);

            }

        }

    }

    return bestd;

}