**算法分析第5次作业**

Frog\_leap问题

问题描述：一只青蛙站在河岸的一边，河中有n块石头排成一条直线，青蛙需要跳到对岸。青蛙每次可以跳1块或2块石头。问有多少种不同的过河方式？

使用回溯算法，每个节点分别先对跳一步进行回溯，再对跳2步进行回溯。解空间树为二叉树，时间复杂度为O(2^n)

void frogLeap(int *position*,int *n*,vector<int>&*path*,vector<vector<int>>&*res*)

{

    if(*position*==*n*)

    {

*res*.push\_back(vector<int>(*path*));

        return;

    }

    if(*position*>*n*)

    {

        return;

    }

*path*.push\_back(*position*+1);

    frogLeap(*position*+1,*n*,*path*,*res*);

*path*.pop\_back();

*path*.push\_back(*position*+2);

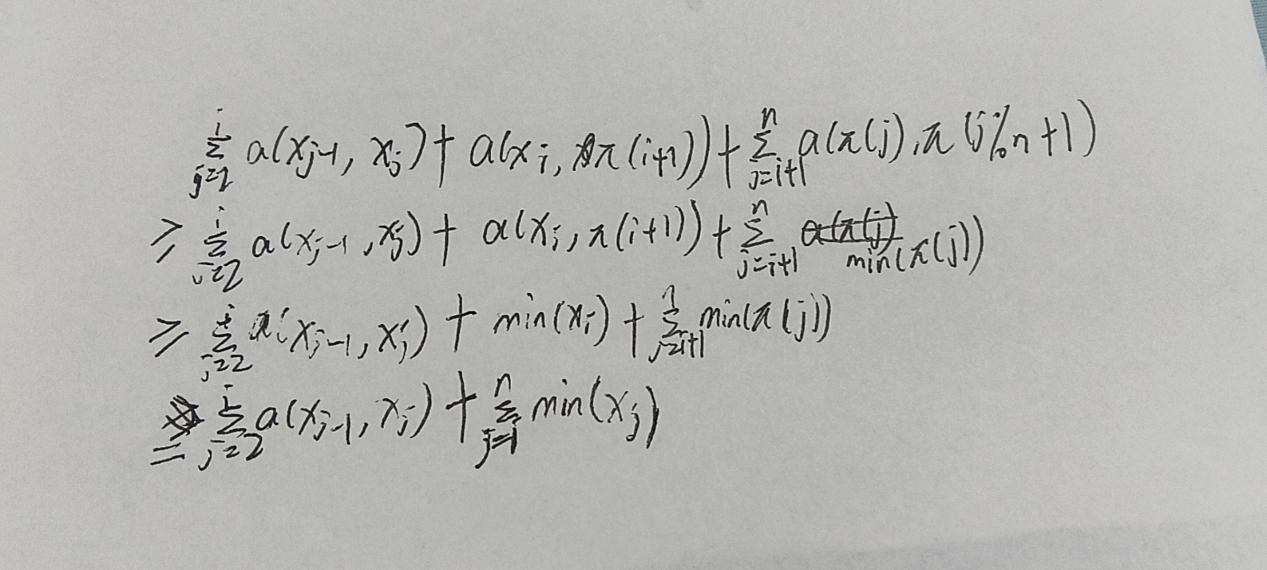
    frogLeap(*position*+2,*n*,*path*,*res*);

*path*.pop\_back();

}

算法分析题

5-6

1. 前缀为x[1:i]的旅行售货员回路可表示为n个顶点的一个排列：x[1],x[2],...,x[i],π(i+1),π(i+2),...,π(n)，费用为：，得证。
2. 实现计算上界的函数，计算当前最优解的上界，剪枝效率高于主教材中的（cc+a[x[i-1][x[j]]<bestx）

template<class Type>

class Traveling{

    friend Type TSP(int\*\*,int[],int,Type);

    private:

    void BackTrace(int *i*);

    Type Bound(int *i*);

    int n,\*x,\*bestx;

    Type \*\*a,cc,bestc,NoEdge;

};

template<class Type>

Type Traveling<Type>::Bound(int *i*)

{

    Type bound=cc;

    bool \*visited=new bool[n+1]();

    for(int j=1;j<*i*;j++)

    {

        visited[j]=true;

    }

    for(int j=1;j<=n;j++)

    {

        if(!visited[j])

        {

            Type minEdge=NoEdge;

            for(int k=1;k<=n;k++)

            {

                if(!visited[k]&&a[j][k]!=NoEdge)

                {

                    if(minEdge==NoEdge||a[j][k]<minEdge)

                    {

                        minEdge=a[j][k];

                    }

                }

            }

            if(minEdge==NoEdge)

            {

                for(int k=1;k<=n;k++)

                {

                    if(a[j][k]!=NoEdge&&(minEdge==NoEdge||a[j][k]<minEdge))

                    {

                        minEdge=a[j][k];

                    }

                }

            }

            if(minEdge!=NoEdge)

                bound+=minEdge;

        }

    }

    delete[] visited;

    return bound;

}

template<class Type>

void Traveling<Type>::BackTrace(int *i*)

{

    if(*i*==n)

    {

        if(a[x[n-1]][x[n]]<NoEdge&&a[x[n]][1]!=NoEdge&&(cc+a[x[n-1]][0]+a[x[n]][1]<bestc||bestc==NoEdge))

        {

            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]]!=NoEdge&&(Bound(*i*+1)<bestc||bestc==NoEdge))

            {

                swap(x[*i*],x[j]);

                cc+=a[x[*i*-1]][x[*i*]];

                BackTrace(*i*+1);

                cc-=a[x[*i*-1]][x[*i*]];

                swap(x[*i*],x[j]);

            }

        }

    }

}

template<class Type>

Type TSP(Type\*\* *a*,int *v*[],int *n*,Type *NoEdge*)

{

    Traveling<Type> Y;

    Y.x=new int[*n*+1];

    for(int i=1;i<=*n*;i++)

    {

        Y.x[i]=i;

    }

    Y.a=*a*;

    T.n=*n*;

    Y.bestc=*NoEdge*;

    Y.bestx=*v*;

    Y.cc=0;;

    T.NoEdge=*NoEdge*;

    Y.BackTrace(2);

    delete[] Y.x;

    return Y.bestc;

}

算法实现题

5-1 对子集中是否加入该数进行回溯，解空间树为子集树，时间复杂度为O(2^n)。

bool findSum(int *arr*[],int *n*,int *sum*,int *i*,vector<int>& *res*)

{

    if(*sum*<0)

    {

        return false;

    }

    if(*sum*==0)

    {

        for(int j=0;j<*res*.size();j++)

        {

            cout<<*res*[j]<<" ";

        }

        return true;

    }

    for(int j=*i*+1;j<*n*;j++)

    {

*res*.push\_back(*arr*[j]);

        if(findSum(*arr*,*n*,*sum*-*arr*[j],j,*res*))

        {

            return true;

        }

*res*.pop\_back();

    }

    return false;

}

5-3 对某部件选择的供应商进行回溯，上界是总价值小于d和总重量小于当前最好重量。解空间树为子集树，时间复杂度为O(2^n)。

int bestw=INT\_MAX;

int bestm[100];

void getMinWeight(int \*\**c*,int \*\**w*,int *n*,int *m*,int *d*,int *i*,int *weight*,int *value*,vector<int>& *res*)

{

    if(*i*==*n*)

    {

        bestw=*weight*;

        for(int i=0;i<*n*;i++)

        {

            bestm[i]=*res*[i];

        }

    }

    for(int j=0;j<*m*;j++)

    {

        if(*value*+*c*[*i*][j]<*d*&&bestw>*w*[*i*][j]+*weight*)

        {

*res*.push\_back(j);

            getMinWeight(*c*,*w*,*n*,*m*,*d*,*i*+1,*weight*+*w*[*i*][j],*value*+*c*[*i*][j],*res*);

*res*.pop\_back();

        }

    }

}

5-6 遍历子集，递增ans，对ans是否插入当前子集进行回溯，上界是如果插入，则会造成和集，解空间为子集树，时间复杂度为O(2^n)

int bestw1;

int \*\*bestm1;

*//h[i][j]表示第i个子集中是否含有j，a[i][0]表示第i个子集的元素个数，a[i][j]表示第i个集合的第j个元素*

void noSumSet(int *level*,int *n*,bool\*\**h*,int \*\**a*,int &*ans*)

{

    if(*level*==*n*)

    {

        if(*ans*<=bestw1+1)

        {

            return;

        }

        bestw1=*ans*-1;

        for(int i=0;i<*n*;i++)

        {

            for(int j=0;j<*a*[i][0];j++)

            {

                bestm1[i][j]=*a*[i][j];

            }

        }

        return;

    }

    else

    {

        bool flag=true;

        for(int i=1;i<=*a*[*level*][0];i++)

        {

*//如果存在和集，不能插入该子集，尝试插入下一个子集*

            if(*h*[*level*][*ans*-*a*[*level*][i]]&&*ans*-*a*[*level*][i]!=*a*[*level*][i])

            {

                flag=false;

                break;

            }

        }

        if(!flag)

        {

            noSumSet(*level*+1,*n*,*h*,*a*,*ans*);

        }

        else*//可以插入该子集，尝试插入或不插入该子集*

        {

*a*[*level*][++*a*[*level*][0]]=*ans*;

*h*[*level*][*ans*++]=true;

            noSumSet(0,*n*,*h*,*a*,*ans*);

            --*a*[*level*][0];

*h*[*level*][--*ans*]=false;

            noSumSet(*level*+1,*n*,*h*,*a*,*ans*);

        }

    }

}

5-13 对每个工作分配给某个人进行回溯，解空间为排列树，时间复杂度为O(n!)。

*//算法实现5-13*

int bestw2;

int \*bestm2;

*//cost[i][j]表示工作i分配给第j个人的费用，r[i]表示工作i分配给第r[i]个人*

void getMinCost(int *i*,int *n*,int \*\**cost*,int \**r*)

{

    if(*i*>*n*)

    {

        int temp=0;

        for(int i=1;i<=*n*;i++)

        {

            temp+=*cost*[i][*r*[i]];

        }

        if(temp<bestw2)

        {

            bestw2=temp;

            for(int i=1;i<=*n*;i++)

            {

                bestm2[i]=*r*[i];

            }

        }

    }

    else

    {

        for(int j=*i*;j<=*n*;j++)

        {

            swap(*r*[*i*],*r*[j]);

            getMinCost(*i*+1,*n*,*cost*,*r*);

            swap(*r*[*i*],*r*[j]);

        }

    }

}

5-16 首先遍历可能选择的位数，也就决定了回溯的层级，对选择的整数进行回溯，解空间为排列树，时间复杂度为O(k!)，对运算符进行回溯，时间解空间为四叉树，时间复杂度为O(4^k)。

bool cal(int *k*,int \**num*,int \**ope*,int *target*)

{

    int temp=*num*[0];

    for(int i=0;i<*k*;i++)

    {

        switch (*ope*[i])

        {

        case 0:temp+=*num*[i+1];break;

        case 1:temp-=*num*[i+1];break;

        case 2:temp\*=*num*[i+1];break;

        case 3:temp/=*num*[i+1];break;

        }

    }

    return temp==*target*;

}

bool trackback(int *i*,int *k*,int *n*,int \**num*,int \**ope*,bool \**flag*,int *target*)

{

    if(*i*>*k*)

    {

        return cal(*k*,*num*,*ope*,*target*);

    }

    for(int j=0;j<*n*;j++)

    {

        if(!*flag*[j])

        {

*num*[*i*]=j;

*flag*[j]=true;

            for(int t=0;t<4;t++)

            {

*ope*[*i*]=t;

                if(trackback(*i*+1,*k*,*n*,*num*,*ope*,*flag*,*target*))

                {

                    return true;

                }

            }

*flag*[*i*]=false;

        }

    }

    return false;

}

bool isCanGetNum(int &*k*,int *n*,int *target*,int \**num*,int \**ope*)

{

    bool \*flag=new bool[*n*];

    for(*k*=0;*k*<*n*;*k*++)

    {

        if(trackback(0,*k*,*n*,*num*,*ope*,flag,*target*))

        {

            return true;

        }

    }

    return false;

}

总时间复杂度O(n!\*4^n\*n)。

5-20 对是否选择该居民进行回溯，经过遍历，当前选择节点中没有仇敌，则有选或不选的分支，否则只能不选，如果后面的节点即使全选也没有原来的最好的结果大，则直接剪枝，时间解空间为子集树，时间复杂度为O(2^n)

int bestw3;

bool \*bestm3;

*//a表示图，如果是仇敌则不为0，selecet[i]表示是否选择i*

void selectTribe(int \*\**a*,int *i*,int *n*,bool \**select*,int &*ans*)

{

    if(*i*>*n*)

    {

        for(int j=1;j<=*n*;j++)

        {

            bestm3[j]=*select*[j];

        }

        bestw3=*ans*;

        return;

    }

    if(*ans*+*n*-*i*+1<=bestw3)

    {

        return;

    }

    bool flag=true;*//判断能否选择该居民*

    for(int j=1;j<=*i*;j++)

    {

        if(*select*[j]&&*a*[*i*][j]!=0)

        {

            flag=false;

            break;

        }

    }

    if(flag)

    {

*select*[*i*]=true;

*ans*++;

        selectTribe(*a*,*i*+1,*n*,*select*,*ans*);

*select*[*i*]=false;

*ans*--;

    }

    selectTribe(*a*,*i*+1,*n*,*select*,*ans*);

}