# 迭代加深和双向搜索

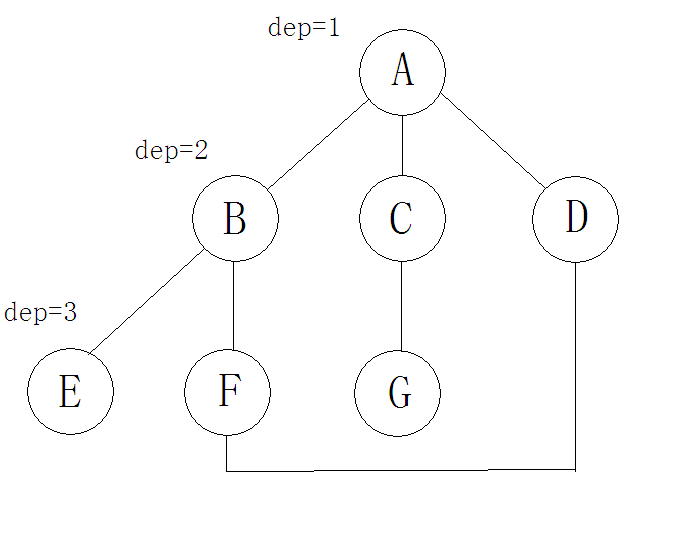
## 

## **迭代加深**

当答案的层数较低，并且搜索的分支较多时，如果直接搜索会消耗很多时间。这时候可以进行多次搜索，每次搜索可以限制一个深度，如果我们在当前深度下搜索不到答案，就增加深度限制，重新搜索一边答案，这样“迭代”且“加深”的过程称为迭代加深。但他的缺点也很明显，每次需要重新搜索一遍，所以在答案的层数比较深的时候不建议使用。

　　迭代加深搜索是在速度上接近广度优先搜索，空间上和深度优先搜索相当的搜索方式。由于在使用过程中引入了深度优先搜索，所以也可以当作深度优先搜索的优化方案。

　　迭代加深搜索适用于当搜索深度没有明确上限的情况。



　　例如上图的一棵搜索树，在进行深度优先搜索前先规定好这次搜索的最大深度dep，当搜索到达dep却还没搜索到结果时回溯。

　　之后不断加大搜索深度，重新搜索，直到找到结果为止。虽然这样搜索次数会累计很多次，但每一次搜索的范围和下一次搜索的范围相比微不足道，所以整体搜索速度不会受太大影响。

　　由于深度是从小到大逐渐增大的，所以当搜索到结果时可以保证搜索深度是最小的。这也是迭代加深搜索在一部分情况下可以代替广度优先搜索的原因（还比广搜省空间）。

【例题】Addition Chains (poj2248)

需要求一个长度（长度为m）最小的序列a满足

1、a[1]=1

2、a[m]=m

3、a[1]<a[2]<.....a[m−1]<a[m]

4、对于每个a[k](2≤k≤m)都存在a[k]=a[i]+a[j](1≤i,j≤k−1，i,j可以相等),其中n(n≤100)是给定的值

主要思路就是搜索，至于DFS还是BFS都可以，DFS需要加比较好的剪枝函数，不然会TLE。

宽搜BFS，因为要找到的是最短的加法链，宽搜是最快速的方法。算法的设计上，因为需要保存路径，所以用结构体node（int id,int val,int pre)保存每个节点，根据pre和id的映射关系使得路径成为链式，再宽搜路径即可

核心代码代码：

void bfs()

{

q.push(node(0,1,-1));

path[cnt++] = node(0,1,-1);

node now,next;

while(!q.empty())

{

now = q.front();

q.pop();

//now.print();

for(int i = now.id; i!=-1; i = path[i].pre)

{

int tmp = now.val + path[i].val;

if(tmp == n)

{

cout<<countPath(now.id)<<endl;

cout<<n<<" ";

output(now.id);

cout<<endl;

return;

}

if(tmp < n)

{

next = node(cnt, tmp, now.id);

path[cnt++] = next;

q.push(next);

}

}

}

}

 但是，由于每一层需要将该层的所有路径全部保存，而这些路径不能被清除，所以空间复杂度极其大，当n等于200，即200的加法链需要开辟10000的数组以存储路径，算法并不可行。

**迭代加深**

搜索思路大概是枚举每个k进行搜索，把i，j作为分支填写到a[k]上进行搜索。

套路剪枝一下，倒序枚举，a[i]+a[j]判重。

由于n比较小，且要满足第四条条件，所以长度m的长度不会太大，所以深度不会太大，由于所有小于k的位置都可以作为i，j所以搜索的分支比较多。那么我们就可以使用迭代加深来优化速度。

#include<cstdio>

#include<iostream>

#include<cstring>

#include<algorithm>

#include<cmath>

#include<cstdlib>

using namespace std;

int n;int a[110];

bool v[110][110];

int dep;

bool flag;

void dfs(int k){

if(flag) return;

if(k==dep+1) return;

if(a[k-1]>n) return;

if(a[k-1]==n){

flag=true;

for(int i=1;i<k-1;i++) printf("%d ",a[i]);

printf("%d\n",a[k-1]);

return;

}

memset(v[k],false,sizeof(v[k]));

for(int i=k-1;i>=1;i--)

for(int j=i;j>=1;j--){

if(a[i]+a[j]<=a[k-1]) break;

if(v[k][a[i]+a[j]]==false){

v[k][a[i]+a[j]]=true;

a[k]=a[i]+a[j];

dfs(k+1);

a[k]=0;

}

}

}

int main()

{

while(scanf("%d",&n)&&n)

{

dep=1; flag=false;

a[1]=1;

while(1)

{

dfs(2); //2<=k<=m

if(flag) break;

dep++;

}

}

}

## **双向搜索**

双向搜索又名折半搜索。当搜索的复杂度在指数级的时候，我们可以通过将指数折半的方法降低搜索复杂度。

具体做法是从初态和终态出发各搜索一半状态，产生两颗深度减半的搜索树，两颗树交汇在一起形成最终答案，将O(nk)降低到O(nk/2+nk/2+1)的复杂度。

其实对于这样的指数级复杂度，如果指数除以2后可以接受的话，可以考虑双向搜索。

【例题】灯

[传送门](https://www.luogu.org/problemnew/show/P2962" \t "https://blog.csdn.net/CABI_ZGX/article/details/_blank)

# P2962 [USACO09NOV]灯Lights

## **题目描述**

Bessie and the cows were playing games in the barn, but the power was reset and the lights were all turned off. Help the cows get all the lights back on so they can resume their games.

The N (1 <= N <= 35) lights conveniently numbered 1..N and their switches are arranged in a complex network with M (1 <= M <= 595) clever connection between pairs of lights (see below).

Each light has a switch that, when toggled, causes that light -- and all of the lights that are connected to it -- to change their states (from on to off, or off to on).

Find the minimum number of switches that need to be toggled in order to turn all the lights back on.

It's guaranteed that there is at least one way to toggle the switches so all lights are back on.

贝希和她的闺密们在她们的牛棚中玩游戏。但是天不从人愿，突然，牛棚的电源跳闸了，所有的灯都被关闭了。贝希是一个很胆小的女生，在伸手不见拇指的无尽的黑暗中，她感到惊恐，痛苦与绝望。她希望您能够帮帮她，把所有的灯都给重新开起来！她才能继续快乐地跟她的闺密们继续玩游戏！ 牛棚中一共有N（1 <= N <= 35）盏灯，编号为1到N。这些灯被置于一个非常複杂的网络之中。有M（1 <= M <= 595）条很神奇的无向边，每条边连接两盏灯。 每盏灯上面都带有一个开关。当按下某一盏灯的开关的时候，这盏灯本身，还有所有有边连向这盏灯的灯的状态都会被改变。状态改变指的是：当一盏灯是开著的时候，这盏灯被关掉；当一盏灯是关著的时候，这盏灯被打开。 问最少要按下多少个开关，才能把所有的灯都给重新打开。 数据保证至少有一种按开关的方案，使得所有的灯都被重新打开。

## **输入输出格式**

****输入格式：****

\* Line 1: Two space-separated integers: N and M.

\* Lines 2..M+1: Each line contains two space-separated integers representing two lights that are connected. No pair will be repeated.

****输出格式：****

\* Line 1: A single integer representing the minimum number of switches that need to be flipped in order to turn on all the lights.

## **输入输出样例**

****输入样例#1：****

5 6

1 2

1 3

4 2

3 4

2 5

5 3

****输出样例#1：****

3

## **说明**

There are 5 lights. Lights 1, 4, and 5 are each connected to both lights 2 and 3.

Toggle the switches on lights 1, 4, and 5.

双向搜索的例题，首先状压灯的开关状态。我们知道起点状态时全灭，终点状态是全亮，那么就可以分成两半搜索，搜到相反数即可。假设有四个灯，我们把灯分成左右两部分，低两位和高两位按钮，那加起来就是答案，可以更新最小步数。

那么对于一个点相连的所有点，我们可以对一个点x记录一个二进制f[x]，将他相连的点（包括他自己）记为1，搜索的时候记录一个当前状态now，每次开关x的时候就可以让now异或f[x]，就可以得到开关这个点可以达到的状态啦。

#include<cstdio>

#include<iostream>

#include<cstring>

#include<algorithm>

#include<map>

using namespace std;

typedef long long ll;

const int INF=2147483647;

ll bin[40];

ll f[40],ed;

bool half=false;

int cnt,minn;

map<ll,int>s;//到now这个状态需最少要多少步，折半后第二次加上第一次的值

void dfs(int x,ll now,int step)

{//当前在x这个点，当前状态为now，当前走了step步

if(x==cnt+1)

{

if(now==ed)

minn=min(minn,step);

else

{

if(!half)//没过半

{

int t=s[now];

if(t==0 || step<t) s[now]=step;//维护最少需要多少步

}

else//过半,看看now和其他合并能否到达最终状态

{

int t=s[ed-now];

if(t!=0) minn=min(minn,t+step);

}

}

return;

}

dfs(x+1,now^f[x],step+1);

dfs(x+1,now,step);

}

int main()

{

int n,m;scanf("%d%d",&n,&m);

f[1]=bin[1]=1; for(int i=2;i<=n+1;i++) f[i]=bin[i]=bin[i-1]\*2;

ed=bin[n+1]-1;

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

{

int x,y;

scanf("%d%d",&x,&y);

f[x]+=bin[y],f[y]+=bin[x];

}

minn=INF;

half=false;

cnt=n/2;

dfs(1,0,0);//搜前半部分

half=true;

cnt=n;

dfs(n/2+1,0,0);//搜后半部分

printf("%d\n",minn);

return 0;

}

【例题】送礼物(tyvj1340)

在N个数中选若干个数，使得他们加起来小于等于W且最接近W，输出这个最接近W的值（ N<=45 W<=2^31-1）。

可以看到W是很大的，所以不用背包来做。

搜索的话，对于一个物品，朴素的有选和不选两种选择，那么这样的复杂度是O(2N)，但如我们上面所说，指数级复杂度如果折半可以接受的话，可以考虑双向搜索。我们发现O(222+223)是可以接受的，所以我们可以考虑折半搜索。

首先，我们第一遍搜索出前一半数中选出若干个可以达到的0~W之间的所有重量值，把他们存放在一个有序数组A中。

然后，第二遍在后一半选出若干个数，记录他们的重量和now，对于一个重量和now在A中二分查找<=W-now中最大的一个t，然后用t+now更新答案。

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

#include<climits>

#include<set>

using namespace std;

typedef long long ll;

const int N=50;

const int INF=INT\_MAX;

int W;int n,a[N];

int A[1<<25],tmp=0;

int cnt;int maxx;

bool half;

int cmp(int a,int b){return a>b;}

int erfen(int val)

{

int l=1,r=tmp,ans;

while(l<=r)

{

int mid=(l+r)/2;

if(A[mid]<=val)

{

ans=A[mid];

l=mid+1;

}

else

r=mid-1;

}

return ans;

}

void dfs(int x,int now)

{

if(x==cnt+1)

{

if(!half)

A[++tmp]=now;

else

{

int t=erfen(W-now);

maxx=max(maxx,t+now);

}

return;

}

if((ll)now+a[x]<=W) dfs(x+1,now+a[x]);

dfs(x+1,now);

}

int main()

{

scanf("%d%d",&W,&n);

for(int i=1;i<=n;i++) scanf("%d",&a[i]);

tmp=0; maxx=0;

sort(a+1,a+n+1,cmp);

half=false; cnt=n/2; dfs(1,0);

sort(A+1,A+tmp+1);

half=true; cnt=n; dfs(n/2+1,0);

printf("%d\n",maxx);

return 0;

}