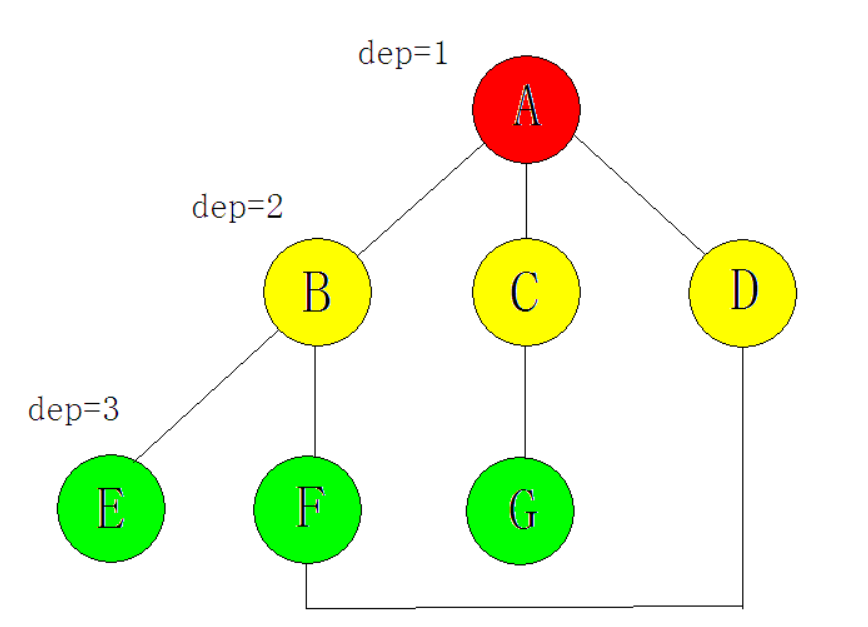
**迭代加深搜索（IDS）**

IDS搜索又成为迭代加深搜索，与深度优先搜索的区别在于它限制了搜索深度，就是在深度无上限的情况下，先预估一个深度（尽量小）进行搜索，如果没有找到解，再逐步放大深度搜索。总的来说，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。其适用于当搜索深度没有明确上限的情况。

下图可以看出IDS搜索和广搜的搜索形式相同，但是只要搜索到答案的深度就可以了，不像广搜会跑完整张图。



例题：在古埃及，人们使用单位分数的和(形如1/a的, a是自然数)表示一切有理数。 如：2/3=1/2+1/6,但不允许2/3=1/3+1/3,因为加数中有相同的。 对于一个分数a/b,表示方法有很多种，但是哪种最好呢？ 首先，加数少的比加数多的好，其次，加数个数相同的，最小的分数越大越好。 如： 19/45=1/3 + 1/12 + 1/180 19/45=1/3 + 1/15 + 1/45 19/45=1/3 + 1/18 + 1/30, 19/45=1/4 + 1/6 + 1/180 19/45=1/5 + 1/6 + 1/18. 最好的是最后一种，因为1/18比1/180,1/45,1/30,1/180都大。 给出a,b(0〈a〈b〈1000),编程计算最好的表达方式。

代码实现：

#include<iostream>

using namespace std;

double frac;

int depth = 1;

inline double abs(double d)

{

return d > 0 ? d : -d;

}

const double ERROR = 1e-7;

bool ids(int d, double now, int dino)

{

if (d == depth)

{

if (abs(frac - now) < ERROR)

{

return true;

}

return false;

}

int maxdino = (int)((depth - d) / (frac - now));

cout << "depth : " << d << ", checking from " << dino << " to " << maxdino << endl;

for (; dino <= maxdino; ++dino)

{

if (ids(d + 1, now + 1.0 / dino, dino + 1))

{

cout << dino << ' ';

return true;

}

}

return false;

}

int main()

{

int a, b;

cin >> a >> b;

frac = (double)a / b;

while (!ids(0, 0.0, (int)(1.0 / frac) + 1))

{

++depth;

}

cout << endl;

return 0;

}

运行结果：

