广度优先搜索

(跳转自宽度优先搜索)

介绍

广度优先搜索法（BFS）

在深度优先搜索算法中，是深度越大的结点越先得到扩展。如果在搜索中把算法改为按结点的层次进行搜索，本层的结点　没有搜索处理完时，不能对下层结点进行处理，即深度越小的结点越先得到扩展，也就是说先产生的结点先得以扩展处，　这种搜索算法称为广度优先搜索法。英语中用Breadth-First-Search表示，所以我们也把广度优先搜索法简称为BFS。

广度优先搜索基本算法:

1）从某个顶点出发开始访问，被访问的顶点作相应的标记，并输出访问顶点号;

2）从被访问的顶点出发，依次搜索与该顶点有边的关联的所有未被访问的邻接点，并作相应的标记。

3）再依次根据2）中所有被访问的邻接点，访问与这些邻接点相关的所有未被访问的邻接点，直到所有顶点被访问为止。

【算法过程】

```
program bfs;
var
  head,tail,i:longint;
  a:array[0..max] of 结点;
begin
  赋值初始结点a[0]
  head:=-1;
  tail:=1;
  repeat
    inc(head); //提取一个没有访问过的结点。
     if (当前结点有子结点) then
    for i:=1 to 最大可扩展结点数 do//扩展当前结点的每个子结点
       if (该子结点并没有出现过) then
      begin
        inc(tail);
        a[tail]:=新结点
       end;
  until head=tail; //所有结点已生成。
end.
```

(ABOVE FROM http://noip.tanghu.net/aosai/shuanfa/024.htm)

算法实例:

问题描述： 有A，B，C三个桶，容量分别为3L，7L，10L。现C桶有10L水。要求在水只能在桶间转移的前提下，使得C桶与B桶平分10L水。求最简洁操作。

```
program BFS;
const
        v:array[1..3] of integer= (3,7,10); //三种桶的容量。
type
        node=record
                l:array[1..3] of longint;//三个水桶。
                p:longint;//每个结点的父结点。
             end;
var
        i,j,head,tail:longint;
        t:boolean; //找到目标的标志。
        a:array[0..100] of node;

procedure init;
var
        i,j:longint;
begin
        fillchar(a,sizeof(a),0);
        t:=false;
        a[0].l[3]:=10;
        head:=-1;
        tail:=0;
end;

procedure pour(x,y:longint);
var
        i,j:longint;
begin
        if (a[head].l[x]=0) or t then exit;

        inc(tail);
        a[tail]:=a[head];
        a[tail].p:=head;

        if a[tail].l[x]>v[y]-a[tail].l[y] then
        begin
                dec(a[tail].l[x],v[y]-a[tail].l[y]);
                a[tail].l[y]:=v[y];
        end else
        begin
                inc(a[tail].l[y],a[tail].l[x]);
                a[tail].l[x]:=0;
        end;
        for i:=0 to tail-1 do //检查该状态是否出现过，是的话删除。
        begin
                if a[i]=a[tail] then
                begin
                        dec(tail);
                        exit;
                end;
        end;
       if a[tail].l[3]=5 then t:=true;
end;

procedure main;
var
        i,j:longint;
begin
        repeat
                inc(head);
                pour(1,2); //pour函数的作用是尝试把x桶里的水倒入y桶，看能不能产生新的状态。
                pour(2,1);
                pour(1,3);
                pour(3,1);
                pour(2,3);
                pour(3,2);
        until (a[tail].l[3]=5) or (tail=100); //当找到目标或者已经超出预定的搜索范围的时候退出。
end;

procedure print;
var
        c:array[1..100] of longint;
        i,j:longint;
begin
        i:=0;
        while a[tail].p<>0 do
        begin
                inc(i);
                c[i]:=tail;
                tail:=a[tail].p;
        end;
        for j:=i downto 1 do writeln(a[c[j]].l[1],' ',a[c[j]].l[2],' ',a[c[j]].l[3]);
end;

begin
        init;
        main;
```

```
        print;
end.
```

附：双向广度优先搜索

双向广度优先搜索（BIBFS）是指搜索沿两个方向同时进行：

```
正向搜索：从初始结点向目标结点方向的搜索；
逆向搜索：从目标结点向初始结点方向搜索；
```

双向广度优先搜索的数据结构要比单向的广度优先搜索复杂一些。由于双向广度优先搜索在搜索的过程中形成两棵方向相反的解答树，因此必

须设置四张表：

```
OPEN0表，CLOSE0表——储存正向搜索中产生的待扩展以及已扩展的结点
OPEN1表，CLOSE1表——储存逆向搜索中产生的待扩展以及已扩展的结点
```

其中设置CL[0，0]，OP[0，0]，CL[1，0]，OP[1，0]为指针。

算法介绍

```
procedure print(st,k:integer);
begin
        if k<>1 then
        begin
                if st=1 then 输出list[st,k].state //逆向搜索,按 f 指针输出 list[1,k]...list[1,1]路
径
                print(st,list[st,k].f);
                if st=0 then 输出list[st,k].state //正向搜索,输出 list[0,k]...list[0,1]
        end;
end;

procedure check(st:0..1);
var
        i:integer;
begin
        for i:=1 to op[1.st]-1 do //检查 st 相反方向扩展的每一个结点。
                if list[st,op[1.st]-1 then
                        if list[st,op[st]].state相交于list[1-st,i].state then
                        begin
                                if st=0 then
                                begin
                                        print(o,op[st]); //当前为正方向,则先输出
list[0..1]..list[0,op[st]],然后输出list[1,i]..list[1,1]
                                        print(1,i);
                                end else
                                begin
                                        print[0,i];//当前为逆方向。
                                        print[1,op[st]);
                                end;
                                halt;
                        end;
end;

procedure expand(st:0..1);
begin
        q:=list(st,cl[st]);
        while (q结点可以扩展) and (op[st]<maxn) do
        begin
                沿 st 方向扩展出 q 的子结点 qt.state;
                list[st,op[st]].state:=qt.state;
                list[st,op[st]].father:=cl[st];
                check(st);//两个方向搜索相交于qt则输出。
                op[st]:=sp[st]+1;
        end;
        cl[st]:=cl[st]+1
end;

begin
        list[0,1]:=起始状态;
        list[0,1].father:=0;
        op[0]=2;
```

```
        cl[0]:=1;
        list[1,1]:=目标状态;
        list[1,1].father:=0;
        op[1]=2;
        cl[1]:=1;

        while ((op[0]<=maxn) and (cl[0]<op[0])) or ((op[1]<=maxn) and (cl[1]<op[1])) do if
op[0]<op[1] then expand(0)

else expand(1);
end.
```

图论及图论算法 [编辑]

图 - 有向图 - 无向图 - 连通图 - 强连通图 - 完全图 - 稀疏图 - 零图 - 树 - 网络

基本遍历算法：宽度优先搜索 - 深度优先搜索 - A* - 并查集求连通分支 - Flood Fill

最短路：Dijkstra - Bellman-Ford（SPFA） - Floyd-Warshall - Johnson算法

最小生成树：Prim - Kruskal

强连通分支：Kosaraju - Gabow - Tarjan

网络流：增广路法（Ford-Fulkerson，Edmonds-Karp，Dinic） - 预流推进 - Relabel-to-front

图匹配 - 二分图匹配：匈牙利算法 - Kuhn-Munkres - Edmonds' Blossom-Contraction

1个分类：图论