算法

第七章 宽度优先搜索算法

bfs

广搜算法

□广度优先搜索算法如下：（用QUEUE）

(1) 把初始节点S0放入Open表中；

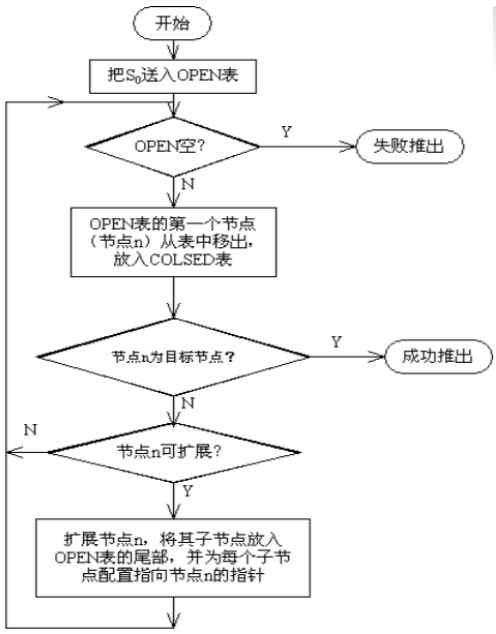
(2) 如果Open表为空，则问题无解，失败退出；

(3) 把Open表的第一个节点取出放入Closed表，并记该节点为n；

(4) 考察节点n是否为目标节点。若是，则得到问题的解，成功退出；

(5) 若节点n不可扩展，则转第(2)步；

(6) 扩展节点n，将其不在Closed表和Open表中的子节点(判重）放入Open表的尾部，并为每一个子节点设置指向父节点的指针(或记录节点的层次），然后转第(2)步。



15

奶牛

百炼4001

题解

抓住那头牛(百练习4001)

农夫知道一头牛的位置，想要抓住它。农夫和牛都位于数轴上，农夫起始位于点N(0<=N<=100000)，牛位于点K(0<=K<=100000)。农夫有两种移动方式：

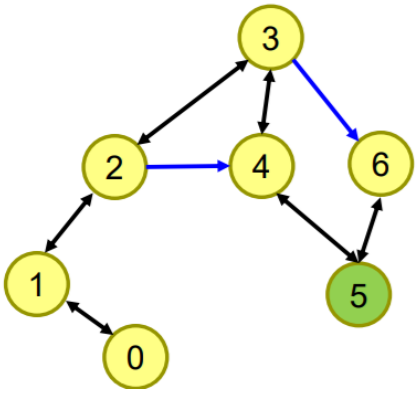
1、从X移动到X-1或X+1，每次移动花费一分钟

2、从X移动到2*X，每次移动花费一分钟

假设牛没有意识到农夫的行动，站在原地不动。农夫最少要花多少时间才能抓住牛？

5

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?

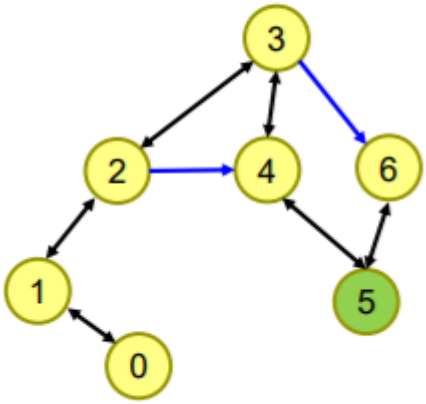


策略2）广度优先搜索：

给节点分层。起点是第0层。从起点最少需n步就能到达的点属于第n层。

第1层：2,4,6
第2层：1,5
第3层：0

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



策略2）广度优先搜索：

搜索过程（节点扩展过程）：

3
2 4 6
1 5

问题解决。
扩展时，不能扩展出已经走过的节点(要判重) 。

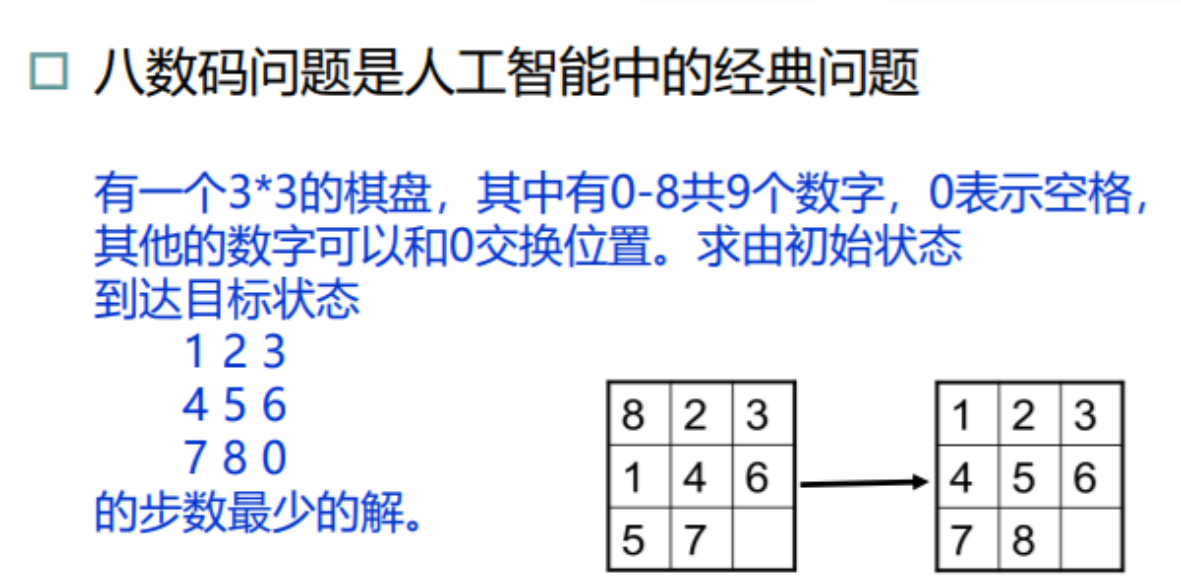
code c++

```
#include<iostream>
#include<queue>
using namespace std;
int N, K;
const int MAXN = 100000;
int visited[MAXN + 10];
struct Step{
    int x;
    int steps;
    Step(int xx, int s) :x(xx), steps(s){}
};
queue<Step> q;
int main(){
    cin >> N >> K;
    memset(visited, 0, sizeof(visited));
    q.push(Step(N, 0));
    visited[N] = 1;
    while (!q.empty()){
        Step s = q.front();
```

```
        if (s.x == K){ cout << s.steps; system("pause"); return 0; }
        else
        {
            if (s.x - 1 >= 0 && !visited[s.x - 1]){
                q.push(Step(s.x - 1, s.steps + 1));
                visited[s.x - 1] = 1;
            }
            if (s.x +1 <=MAXN && !visited[s.x + 1]){
                q.push(Step(s.x + 1, s.steps + 1));
                visited[s.x + 1] = 1;
            }
            if (s.x*2 <= MAXN && !visited[s.x*2]){
                q.push(Step(s.x*2, s.steps + 1));
                visited[s.x *2] = 1;
            }
        }
        q.pop();
    }
    system("pause");
    return 0;
}
```

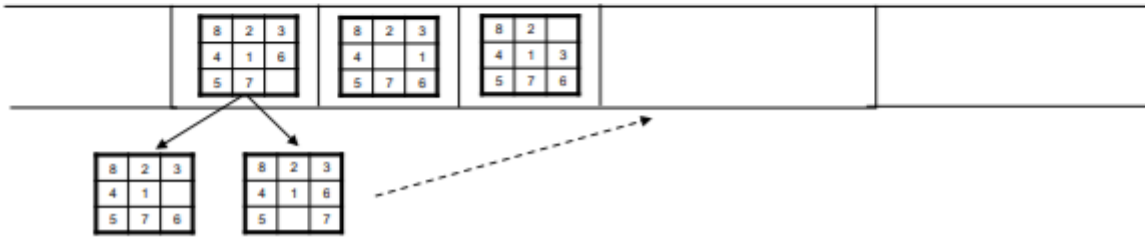
八数码问题

题解



广度优先搜索

- 用队列保存待扩展的节点
- 从队首队取出节点，扩展出的新节点放入队尾，直到队首出现目标节点（问题的解）



广度优先搜索的代码框架

```
BFS()
{
        初始化队列
        while(队列不为空且未找到目标节点)
        {
         取队首节点扩展，并将扩展出的非重复节点放入队尾；
                必要时要记住每个节点的父节点；
        }
}
```

用合理的编码表示“状态”，减小存储代价

- 方案五：

8	2	3
4	1	6
5	7	

- 还是把一个状态看作一个数的10进制表示形式
- 用set<int>进行判重。每入队一个状态，就将其加到set里面，判重时，查找该set，看能否找到状态

code

```
#include<iostream>
#include<bitset>
#include<cstring>
```

```
#include<cstdio>
#include<cstdlib>
#include<set>
using namespace std;
int goalStatus = 123456780;
const int MAXS = 400000;
char result[MAXS];
struct Node{
    int status;
    int father;
    char move;
    Node(int s, int f, char m) :status(s), father(f), move(m){}
    Node(){}
};
Node myQueue[MAXS];
int qHead = 0;//队头指针
int qTail = 1;//队尾指针
char moves[] = "udrl";

int NewStatus(int status, char move){
    //求status 经过move得到的新状态
    char tmp[20];
    int zeroPos;//字符‘0’的位置
    //int j = 8;
    //while (status ){
    //  int t = status % 10;
    //  tmp[j--] = t;
    //  status = status / 10;
    //}
    sprintf_s(tmp, "%09d", status);//保存前导0

    for (int i = 0; i < 9; ++i){
        if (tmp[i] == '0'){
            zeroPos = i;
            break;
        }//返回空格的位置

    }
    switch (move)
    {
    case 'u':
        if (zeroPos - 3 < 0)
            return -1;//空格在第一行
        else{
            tmp[zeroPos] = tmp[zeroPos - 3];
            tmp[zeroPos - 3] = '0';
        }
        break;
    case 'd':
        if (zeroPos + 3 >8)
            return -1;//空格在第san行
        else{
            tmp[zeroPos] = tmp[zeroPos + 3];
            tmp[zeroPos + 3] = '0';
        }
        break;
    case 'l':
        if (zeroPos % 3 == 0)
```

```
            return -1;//空格在第一lie
        else{
            tmp[zeroPos] = tmp[zeroPos - 1];
            tmp[zeroPos - 1] = '0';
        }
        break;
    case 'r':
        if (zeroPos % 3 == 2)
            return -1;//空格在第一行
        else{
            tmp[zeroPos] = tmp[zeroPos + 1];
            tmp[zeroPos + 1] = '0';
        }
        break;
    default:
        break;
    }
    return atoi(tmp);
}



bool Bfs(int status){
    int newStatus;
    set<int> expanded;
    myQueue[qHead] = Node(status, -1, 0);
    expanded.insert(status);
    while (qHead != qTail){
        //队列不为空
        status = myQueue[qHead].status;
        if (status == goalStatus)//找到目标状态
            return true;
        for (int i = 0; i < 4; ++i){
            //尝试四种移动
            newStatus = NewStatus(status, moves[i]);
            if (newStatus == -1)continue;//不可以移动下一种
            if (expanded.find(newStatus) != expanded.end())
                continue;//已经扩展过
            expanded.insert(newStatus);
            myQueue[qTail++] = Node(newStatus, qHead, moves[i]);
        }
        qHead++;
    }
    return false;
}



int main(){
    char line1[50]; char line2[20];
    while (cin.getline(line1, 48)){
        int i, j;
        for (i = 0, j = 0; line1[i]; i++){
            if (line1[i] != ' '){
                if(line1[i] == 'x')
                    line2[j++] = '0';
                else line2[j++] = line1[i];
            }
```

```
        }
        line2[j] = 0;
        for (auto c : line2)cout << c << "  ";
        cout << endl;
        if (Bfs(atoi(line2))){
            int t = 0;
            int pos = qHead;
            do{
                result[t++] = myQueue[pos].move;
                pos = myQueue[pos].father;
            } while (pos);
            for (int i = t - 1; i >= 0; --i){
                cout << result[i];
            }
        }
        else
        {
            cout << "unsolvable" << endl;
        }
    }
}
```