

DFS

POJ 1979: Red and Black 简单的DFS找四联通块即可。代码如下

```
/*

*/

#define method_1
#ifdef method_1
/*

*/

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=20+5;
const int INF=0x3f3f3f3f;
const int dx[]={0,0,-1,1};
const int dy[]={1,-1,0,0};
int w,h,sx,sy,ans,vis[maxn][maxn]; //h为行数 w为列数
char mp[maxn][maxn];
bool check(int x,int y){
    if(x<1||x>h||y<1||y>w||mp[x][y]=='#') return false;
    return true;
}
void init(){
    ans=0;
    memset(vis,0,sizeof(vis));
}
void dfs(int x,int y){
    vis[x][y]=1,ans++;
    for(int i=0;i<3;i++){
        int newx=x+dx[i],newy=y+dy[i];
```

```

        if(check(newx,newy)&&!vis[newx][newy]){
            dfs(newx,newy);
        }
    }
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Red and Black.in", "r", stdin);
    while(cin>>w>>h){
        init();
        if(w==0&&h==0) break;
        for(int i=1;i<=h;i++) for(int j=1;j<=w;j++) {
            cin>>mp[i][j];
            if(mp[i][j]=='@'){
                sx=i,sy=j;
            }
        }
        dfs(sx,sy);
        cout<<ans<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

POJ 3009: Curling 2.0

冰球的移动规则较为复杂，但本质上是一个最短路问题。由于冰球会带来砖块破碎，所以需要回溯法求解。因此BFS求解最短路尽管效率高，但回溯困难，所以采用了DFS求解。代码如下

```

/*

*/
#define method_1
#ifdef method_1
/*

```

```

*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
using namespace std;
typedef long long ll;
const int maxn=20+5;
const int INF=0x3f3f3f3f;

int w,h;

int a[maxn][maxn];

int ans=INF;
void dfs(int deep,int sx,int sy,int tx,int ty) {
    if(deep>10) {
        return;
    }
    if(sx==tx&&sy==ty) {
        /*
        if(deep>=4){
            cout<<"....."<<endl;
        }
        */
        ans=min(ans,deep);
        return;
    }
    bool f1=false,f2=false,f3=false,f4=false;
    int j;
    //枚举四个方向 f1, f2, f3, f4记录是否可行
    //j为最后停留位置
    for(int i=sx; i<=w; i++) {
        if(a[sx+1][sy]==1) { //一步也动不了
            break;
        }
        if(a[i][sy]==1) {
            j=i-1;
            f1=true;
            break;
        }
    }
}

```

```

        if(a[i][sy]==3) { //直接移动到终点
            ans=min(ans,deep+1);
            return;
        }
    }
    if(f1) { //回溯
        a[j+1][sy]=0;
        dfs(deep+1,j,sy,tx,ty);
        a[j+1][sy]=1;
    }

    for(int i=sx; i>=1; i--) {
        if(a[sx-1][sy]==1) {
            break;
        }
        if(a[i][sy]==1) {
            j=i+1;
            f2=true;
            break;
        }
        if(a[i][sy]==3) {
            ans=min(ans,deep+1);
            return;
        }
    }
    if(f2) {
        a[j-1][sy]=0;
        dfs(deep+1,j,sy,tx,ty);
        a[j-1][sy]=1;
    }

    for(int i=sy; i<=h; i++) {
        if(a[sx][sy+1]==1) {
            break;
        }
        if(a[sx][i]==1) {
            j=i-1;
            f3=true;
            break;
        }
        if(a[sx][i]==3) {
            ans=min(ans,deep+1);
            return;
        }
    }
    if(f3) {
        a[sx][j+1]=0;
        dfs(deep+1,sx,j,tx,ty);
        a[sx][j+1]=1;
    }

```

```

    }

    for(int i=sy; i>=1; i--) {
        if(a[sx][sy-1]==1) {
            break;
        }
        if(a[sx][i]==1) {
            j=i+1;
            f4=true;
            break;
        }
        if(a[sx][i]==3) {
            ans=min(ans,deep+1);
            return;
        }
    }
    if(f4) {
        a[sx][j-1]=0;
        dfs(deep+1,sx,j,tx,ty);
        a[sx][j-1]=1;
    }
    return;
}

int main() {
    ios::sync_with_stdio(false);
    //freopen("Curling 2.0.in","r",stdin);
    while(cin>>h>>w&&w!=0&&h!=0) {
        ans=INF;
        memset(a,0,sizeof(a));
        int sx,sy,tx,ty;
        for(int i=1; i<=w; i++) {
            for(int j=1; j<=h; j++) {
                cin>>a[i][j];
                if(a[i][j]==2) {
                    sx=i;
                    sy=j;
                }
                if(a[i][j]==3) {
                    tx=i;
                    ty=j;
                }
            }
        }
        /*
        for(int i=1; i<=w; i++) {
            for(int j=1; j<=h; j++) {
                cout<<a[i][j]<<" ";
            }
            cout<<endl;
        */
    }
}

```

```

    }
    */
    dfs(0,sx,sy,tx,ty);
    if(ans==INF||ans>10) {
        cout<<"-1"<<endl;
        continue;
    }
    cout<<ans<<endl;
}

return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

BFS

P0J 3669: Meteor Shower 将所有陨石拓展一格，在所有格子上标注最早毁灭时间。

用三元组 (x,y,t) 表示状态，即当前位于 (x,y) 时间为 t ，bfs时保证 t 小于所在格子的最早毁灭时间。如果某个各自的最早毁灭时间为无穷，则当前状态三元组的 t 即为所求。

本题还有很多实现细节，详见代码注释。代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*
将所有陨石拓展一格，在所有格子上标注最早毁灭时间
用三元组 $(x,y,t)$ 表示状态，即当前位于 $(x,y)$ 时间为 $t$ ，bfs时保证 $t$ 小于所在格子的最早毁灭时间
如果某个各自的最早毁灭时间为无穷，则当前状态三元组的 $t$ 即为所求。
*/
#include<iostream>

```

```

#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=300+5;
const int n=300;
const int INF=0x3f3f3f3f;
const int dx[]={0,0,-1,1};
const int dy[]={1,-1,0,0};
int m,die[maxn][maxn],vis[maxn][maxn];
struct node{
    int x,y,t;
};
queue<node>q;
bool check(int x,int y){
    if(x<0||x>n+2||y<0||y>n+2) return false; //注意 不要写成x>n
    因为可以移动到x>300的坐标
    return true;
}
void init(){
    memset(die,INF,sizeof(die));
    cin>>m;
    int x,y,t;
    for(int i=1;i<=m;i++){
        cin>>x>>y>>t;
        die[x][y]=min(die[x][y],t);
        for(int j=0;j<=3;j++){
            int newx=x+dx[j],newy=y+dy[j];
            if(check(newx,newy)) die[newx][newy]=min(die[newx][newy],t);
        }
    }
}
int bfs(){
    node st;st.x=0,st.y=0,st.t=0;
    q.push(st);

```

```

// q.push((node){0,0,0}); //用这种写法 poj会CE
while(q.size()){
    node now=q.front();q.pop();
    if(die[now.x][now.y]==INF) return now.t;
    for(int i=0;i<=3;i++){
        int newx=now.x+dx[i],newy=now.y+dy[i];
        if(check(newx,newy)&&!vis[newx][newy]&&now.t+1<die[newx][newy]){
            vis[newx][newy]=1;
            node nxt;nxt.x=newx,nxt.y=newy,nxt.t=now.t+1;
            q.push(nxt);
//            q.push((node){newx,newy,now.t+1});
        }
    }
}
return -1;
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Meteor Shower.in","r",stdin);
    init();
    cout<<bfs();
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

穷竭搜索

[POJ 2718: Smallest Difference](#) 首先，最优解的组合必然是两个位数最接近的数。

即假设有 n 个数，那么一个是 $n/2$ 位数，一个是 $n-n/2$ 位数。

因此暴力dfs枚举第一个集合的选则方式即 $C(n,n/2)$ 。

然后对于每种情况，先将第一个集合里头的数字全部dfs出来，排序。

再dfs第二个集合里头的情况，再在第一个集合里头二分查找比较即可。

对于这个算法，题目有些地方需要特判，详见代码注释。代码如下

```
/*
```

```
*/
```

```
#define method_1
```

```
#ifdef method_1
```

```
/*
```

首先，最优解的组合必然是两个位数最接近的数

即假设有 n 个数 那么一个是 $n/2$ 位数 一个是 $n-n/2$ 位数

因此暴力dfs枚举第一个集合的选则方式即 $C(n, n/2)$

然后对于每种情况，先将第一个集合里头的数字全部dfs出来，排序。

再dfs第二个集合里头的情况，再在第一个集合里头二分查找比较即可。

```
*/
```

```
#include<iostream>
```

```
#include<cstdio>
```

```
#include<algorithm>
```

```
#include<cmath>
```

```
#include<set>
```

```
#include<map>
```

```
#include<queue>
```

```
#include<stack>
```

```
#include<vector>
```

```
#include<cstring>
```

```
#include<cstdlib>
```

```
#include<iomanip>
```

```
#include<ctime>
```

```
#define D(x) cout<<#x<<" = "<<#x<<" "
```

```
#define E cout<<endl
```

```
using namespace std;
```

```
typedef long long ll;
```

```
typedef pair<int,int>pii;
```

```
const int maxn=10+5;
```

```
const int INF=0x3f3f3f3f;
```

```
int T,ans,n,a[maxn],vis[maxn];
```

```
vector<int>choose[2],result;
```

```
void init(){
```

```
    ans=INF,n=0;
```

```
    memset(vis,0,sizeof(vis));
```

```
    choose[0].clear();choose[1].clear();
```

```
    result.clear();
```

```
}
```

```
void gen(){ //暴力枚举第一个集合能够组成的所有数字
```

```
    result.clear();
```

```
    do{
```

```
        if(choose[0][0]==0) continue;
```

```
        int sum=0;
```

```
        for(int i=0;i<choose[0].size();i++){
```

```
            sum=sum*10+choose[0][i];
```

```

        }
        result.push_back(sum);
    }while(next_permutation(choose[0].begin(),choose[0].begin()+choose[0].size())
);
    //这种方式产生的result必然已经升序排序
    // for(int i=0;i<result.size();i++) cout<<result[i]<<" ";
    // cout<<endl;
}
void change(int sum){ //暴力枚举第二个集合能够组成的所有数字 并和第一个比较
    int pos=lower_bound(result.begin(),result.end(),sum)-result.begin();
    // cout<<ans<<endl;
    if(pos+1<result.size()) ans=min(ans,abs(result[pos+1]-sum));
    //注意这里必须用if判断是否越界
    if(pos<result.size()&&pos>=0)ans=min(ans,abs(result[pos]-sum));
    //注意这里必须用if判断是否越界
    if(pos-1<result.size()&&pos-1>=0)ans=min(ans,abs(result[pos-1]-sum));
    /*
    if(ans==0){
        cout<<sum<<endl;
        cout<<pos<<endl;
        cout<<result[pos]<<endl;
        for(int i=0;i<result.size();i++) cout<<result[i]<<" ";
        cout<<endl;
    }
    */
}
void update(){ //暴力枚举第二个集合能够组成的所有数字
    do{
        if(choose[1][0]==0) continue;
        int sum=0;
        for(int i=0;i<choose[1].size();i++){
            sum=sum*10+choose[1][i];
        }
        change(sum);
        // cout<<sum<<" ";
    }while(next_permutation(choose[1].begin(),choose[1].begin()+choose[1].size())
);
    // cout<<endl;
}
void dfs1(int p,int m){ //枚举到第p个数字 已经选择了m个数字
    if(m==n/2){
        choose[1].clear();
        for(int i=1;i<=n;i++) if(!vis[i]) choose[1].push_back(a[i]);
        /*
        for(int i=0;i<choose[0].size();i++) cout<<choose[0][i]<<" ";
        */
    }
}

```

```

        cout<<endl;
        for(int i=0;i<choose[1].size();i++) cout<<choose[1][i]<<" ";
        cout<<endl<<endl;
        */
        gen();
        update();
        return;
    }
    if(p>n) return;
    if(m+n-p+1<n/2) return; //把剩下的全部选上也不够 剪枝
    choose[0].push_back(a[p]);
    vis[p]=1;
    dfs1(p+1,m+1);
    vis[p]=0;
    choose[0].pop_back();
    dfs1(p+1,m);
}
int main() {
    // ios::sync_with_stdio(false); //用了getchar 就不能关闭流同步
    //freopen("Smallest Difference.in","r",stdin);
    // freopen("Smallest Difference.out","w",stdout);
    cin>>T;
    getchar(); //忽略第一个换行
    while(T--){
        init();
        char ch;
        while((ch=getchar())!='\n'){
            if(ch=='\n') break;
            else if(isdigit(ch)) a[++n]=ch-'0';
        }
        // for(int i=1;i<=n;i++) cout<<a[i]<<" ";
        // cout<<endl;
        if(n==2){ //这里要特判 因为原来的算法没有考虑只有一位, 而且这一位是0的情况
            cout<<abs(a[1]-a[2])<<endl;
            continue;
        }
        dfs1(1,0);
        cout<<ans<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif

```

```

#ifdef method_3
/*

```

```

*/

```

```

#endif

```

[POJ 3187: Backward Digit Sums](#) next_permutation这么快的吗。

看到 $n \leq 10$ 就放弃了使用全排列。想从终态开始搜索然后剪枝。

结果去看题解发现就是枚举初态全排列然后判断即可。理论复杂度是 $O(n!10^{10})$

没想到竟然能AC。代码如下 (PS:method_1结果为TLE, method_2注意特判1 1这组数据

```

/*
next_permutation这么快的吗
看到 $n \leq 10$ 就放弃了使用全排列
想从终态开始搜索然后剪枝
结果去看题解发现就是枚举初态全排列然后判断即可
理论复杂度是 $O(n! * 10^{10})$ 没想到竟然能AC
*/
#define method_2
#ifdef method_1
/*
从终态开始搜索然后剪枝
TLE
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=10+5;
const int INF=0x3f3f3f3f;

```

```

int n,sum,a[maxn];
set<int>s;
void dfs(int dep,int b[]){ //目前搜索深度为dep 分解情况为数组b
    if(dep==0){
        bool f=true;s.clear();
        for(int i=1;i<=n;i++){
            if(s.count(b[i])==0&&b[i]>=1&&b[i]<=n) s.insert(b[i]);
            else f=false;
        }
        if(f){
            for(int i=1;i<=n;i++) cout<<b[i]<<" ";
            exit(0);
        }
        return;
    }
    int c[maxn];
    for(int i=1;i<b[1];i++){ //这样枚举能够实现字典序
        c[1]=i;
        bool flag=true;
        for(int j=2;j<=n-dep+1;j++){
            c[j]=b[j-1]-c[j-1];
            if(c[j]<=0){ //剪枝 过程中不可能存在非正数
                flag=false;
                break;
            }
        }
        if(flag) dfs(dep-1,c);
    }
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Backward Digit Sums.in","r",stdin);
    cin>>n>>sum;
    a[1]=sum;
    dfs(n-1,a);
    return 0;
}
#endif
#ifdef method_2
/*
188ms
AC
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>

```

```

#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=10+5;
const int INF=0x3f3f3f3f;
int n,sum,a[maxn],b[maxn],c[maxn];
bool check(){
    memcpy(c,a,sizeof(a)); //备份a数组
    for(int i=2;i<=n;i++){
        for(int j=1;j<=n-i+1;j++){
            b[j]=a[j]+a[j+1];
//            cout<<b[j]<<" ";
        }
//        cout<<endl;
        memcpy(a,b,sizeof(b));
    }
    memcpy(a,c,sizeof(c));
    return b[1]==sum;
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Backward Digit Sums.in","r",stdin);
    cin>>n>>sum;
    if(n==1){ //需要特判
        cout<<1;
        return 0;
    }
    for(int i=1;i<=n;i++) a[i]=i;
    do{
//        for(int i=1;i<=n;i++) cout<<a[i]<<" ";
//        cout<<endl;
        if(check()){
            for(int i=1;i<=n;i++) cout<<a[i]<<" ";
            return 0;
        }
    }while(next_permutation(a+1,a+n+1));
    return 0;
}
#endif

```

```
#ifdef method_3
/*
```

```
*/
```

```
#endif
```

[POJ 3050: Hopscotch](#) 爆搜即可，map判重，注意poj上map套string要加头文件 #include。

代码如下

```
/*
```

```
*/
```

```
#define method_1
```

```
#ifdef method_1
```

```
/*
```

爆搜即可

map判重

```
*/
```

```
#include<iostream>
```

```
#include<cstdio>
```

```
#include<algorithm>
```

```
#include<cmath>
```

```
#include<set>
```

```
#include<map>
```

```
#include<queue>
```

```
#include<stack>
```

```
#include<vector>
```

```
#include<cstring>
```

```
#include<cstdlib>
```

```
#include<iomanip>
```

```
#include<ctime>
```

```
#include<string>
```

```
#define D(x) cout<<#x<<" = "<<x<<" "
```

```
#define E cout<<endl
```

```
using namespace std;
```

```
typedef long long ll;
```

```
typedef pair<int,int>pii;
```

```
const int maxn=5+5;
```

```
const int n=5;
```

```
const int INF=0x3f3f3f3f;
```

```
const int dx[]={0,0,-1,1};
```

```
const int dy[]={1,-1,0,0};
```

```
int a[maxn][maxn],ans=0;
```

```
map<string,int>mp; //poj上map套string要加头文件 #include<string>
```

```
bool check(int x,int y){
```

```
    if(x<1||x>n||y<1||y>n) return false;
```

```

        return true;
    }
    void dfs(int x,int y,string s){
        s+=char(a[x][y]+'0');
        if(s.length()==6){
            if(mp[s]==0) mp[s]=++ans;
            return;
        }
        for(int i=0;i<=3;i++){
            int newx=x+dx[i],newy=y+dy[i];
            if(check(newx,newy)) dfs(newx,newy,s);
        }
    }
    int main() {
        ios::sync_with_stdio(false);
        //freopen("Hopscotch.in","r",stdin);
        for(int i=1;i<=5;i++) for(int j=1;j<=5;j++) cin>>a[i][j];
        for(int i=1;i<=5;i++) for(int j=1;j<=5;j++) dfs(i,j,"");
        cout<<ans;
        return 0;
    }
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```