

Kabaca baktığımızda 8 puzzle da 9 sayı var, bütün olası tahta dizilimlerini ulaşılabilir kabul edersek maksimum 9! Durum oluyor. ($9!=362880$)

Problemi daha derin incelediğimizde (http://en.wikipedia.org/wiki/Fifteen_puzzle) belli bir başlangıç konumundan, her tahta dizilimine ulaşamadığımızı görüyoruz. Bu 9! Durumdan az olduğunu gösteriyor.

Her halukarda problemin 8 Puzzle olan versiyonu durum sayısı az olduğundan kolay bir şekilde çözülebilir.

Ben daha öğretici olması bakımından size herhangi bir N^2 Puzzle a uygulandığında kayda değer bir performans sunacak olan A* algoritması ile olan çözümünü anlatacağım.

Herhangi bir tahta durumundan maksimum 4 farklı tahta durumuna erişebiliriz. (Boşluğu sağ, sol, alt, üst e oynatmak). Ve çözümü ararken bir tahta durumunu birden fazla kontrol etmek gereksizdir. Bir tahta durumunu birden fazla kontrol etmemek için eriştiğiniz her durumu ya bir binary tree ye, ya da bir hash tablosuna atmanız gerektir. Ben çözümümde kolaylık olması açısından, C++ da yer alan map'ı kullandım.

C++ map hakkında daha detaylı bilgi için <http://www.cplusplus.com/reference/stl/map/> adresine bakmanızı öneririm. Arka planında red black tree kullanmaktadır. Ekleme yapmanın ve bir elemana erişmenin maliyeti $O(\log N)$ dir.

Şu ana kadar öğrendiklerimizi göze alırsak, sırayla 4 olası hamleden uygun olanlarını sırayla deneyip, elde ettiğimiz durumu maplersek (tekrar ulaşmamak için), kesinlikle çözüme ulaşacağız. Fakat çözüme ulaşmak için yaptığımız hamle sayısı ile ilgili herhangi bir iyileştirme yapmadık.

Olabildiğinde minimum hamlede çözüme ulaşmak için bir priority queue kullanacağız. Ben kolaylık olması açısından C++ priority_queue kullandım. Her durumu queue ya atacağız. Her duruma belli bir sayı atamız gerekiyor ki queue da öne çıkması açısından bir kriterimiz olsun. Bu atayacağımız sayı için kullanacağımız fonksiyon $F(D)$ olsun.

Eğer $F(D) = \text{Hamle_sayisi}$ dersek

Her aşamada minimum hamlede ulaştığımız durumu işlemiş oluruz.

Açıklamak gerekirse başta queue ya tahtanın ilk durumunu atıyoruz.

Sonra bu ilk durumu queue dan çıkarıp, olası hamleleri yapıp oluşacak 4(max) durumu tekrar Queue ya atıyoruz. Bu durumlar için $F(D)=1$ oluyor.

Sonra bu durumlardan birini Queue dan alıyoruz. Olası hamleleri yapıp maksimum 4 durumu tekrar queue ya atıyoruz. Bu durumlar için $F(D)=2$ oluyor.

Bu işlemi tamamladıktan sonra queue dan bir eleman daha alacağız, queue da $F(D)$ si 1 ve 2 olan elemanlar var, öncelikle 1 olanları işleyeceğiz.

Böylelikle önce en az hamle ile ulaşılan durumlara bakmış oluyoruz. Herhangi bir aşamada ulaştığımız durum, bize inputta verilen son durum ise, hamleleri yazdırıp program sonlandırıyoruz.

Eğer bizden en az hamle ile ulaşılan çözümü isteseydi yapmamız gereken bu olacaktı, ve 8 puzzle için en mantıklısı bu.

Fakat bir 15 puzzle I ele alırsak oluşabilecek kabaca 15! Durum var ki bu çok fazla, bu 15!durumun hepsini işlememiz mümkün değil ama yinede olabilecek en iyi çözümü bulmak istiyorsak A* algoritması mantıklı bir seçim olacaktır.

Queue ya elemanları eklerken bir F(D) değeri ile ekleyelim demiştim. A* uygulandığında bu F(D) ye bir heuristic fonksiyonu ekleniyor

$F(D) = \text{heuristic}()$ oluyor.

Ben kolaylık olması açısından bu heuristic fonksiyonunu şöyle tanımladım.

$\text{Heuristic}() = \text{hamle_sayisi} + \text{tahtadaki_her_elemanin_asil_olmasi_gerektigi_yere_olan_uzakligi}$

Bu heuristic akla gelen kullanılabilir en basit heuristic fonksiyonu, 100 Puzzle da bile iyi çalışıyor denebilir.

Kısa algoritması:

- 1) İlk durumu F(D) sini hesaplayıp queue ya at
- 2) Queue dan bir eleman al
- 3) Eğer son durumsa hamleleri yazdır, program sonlandır
- 4) Bu durumdan elde edilebilecek yeni tahta durumlarını üret
- 5) Her birini F(D) lerini hesaplayıp queue ya at
- 6) (2) ye git

Farkettiyseniz çözüm bulana kadar 2-6 arasında dolanıyor, (3) de çözüm bulunca duruyor.

Hamleleri işaretleme kısmını pseudo code a eklemedim, N^2 Puzzle ile uğraşacaksanız gereksiz bir özellik, alttaki 8Puzzle kodunda gerekli olmasa da yer alıyor, eğer kodu değiştirip denemek isterseniz yararlı olacaktır.

KOD:

```
#include<cstdio>
#include<cstdlib>
#include<string>
#include<queue>
#include<map>

using namespace std;

int num[9][2],move;
int a[4][3]={{0,1,'R'},{1,0,'D'},{-1,0,'U'},{0,-1,'L'}};
// Bu array hamleleri simgeliyor 0,1 -1,0 gibi olan ilk 2 elemanı
// yapılan hamlenin koordinat ekseninde sırayla x,y de yapacağı değişiklik
```

```

// 3. Eleman olan 'R','U' lar ise o hamleniyi gösteren karakter ('U' Up
oluyor)
string final;
// Son durumun stringi
int mut(int x)
{
    // Mutlak deger almak için, daha şık yapılabilir
    return x<0?-x:x;
}

int convert(char x)
{
    // Bu fonksiyon inputu okurken yararlı, karakterleri sayıya çeviriyor
    // nokta yani boşluk 8 oluyor, '1' ise integer olarak 0 oluyor
    if(x=='.') return 8;
    else return x-'1';
}

// alttaki classımız bir tahta durumunu temsil ediyor
class state{
public:
    string array,path;
    // array tek satir halinde gösterilen tahta durumu
    // path , durum için yapılan hamleleri tutuyor
    int value,hamle;
    // F(D)=value olacak
    void evaluate()
    {
        // Bu fonksiyon F(D) yi hesapliyor
        value=hamle;
        for(int i=0,x=0,y=0;i<9;i++)
        {
            if(y==3) {x++; y=0;}
            value+=abs(num[convert(array[i])][0]-
x)+abs(num[convert(array[i])][1]-y);
            y++;
        }
        // Bu prosedürün yaptığı iş, her sayinin ve boşluğun olması
        // olması gereken yere olan uzaklığını hesaplamak
    }
};

bool operator< (state a,state b)
{
    if(a.value>b.value) return 1;
    return 0;
    // classımız için < işlemini overload ediyoruz
}

priority_queue<state> Q;
// queuemuz
map<string,int> used;
// durumları işaretlemek için mapimiz

```

```

state find()
{
    // Arama algoritmamiz
    char array[3][3],tmp;
    // Her aşamada tahtanın tutulacağı array
    int k,i,x,y,chamle,x0,y0,ok,xx,yy;
    string cpath;
    state current;
    // current her aşamada işlediğimiz hamle, cpath ise bu hamlenin pathi
    while(!Q.empty())
    {
        move++;
        // performansi ölçmek için move değişkenini ekledim
        current=Q.top();
        // queue dan eleman alıyor
        Q.pop();
        // elemanı queue dan çıkartıyor
        chamle=current.hamle+1;
        // yeni hamle sayımız
        cpath=current.path;
        ok=1;
        // eğer ok=1 kalırsa sonucu bulduk demektir
        for(i=0,x=0,y=0;i<9;i++)
        {
            if(y==3) {x++; y=0;}
            array[x][y]=current.array[i];
            if(array[x][y]=='.') {x0=x; y0=y;}
            // x0 ve y0 boşluğun x,y si
            if(array[x][y]!=final[i]) ok=0;
            // eğer herhangi bir noktada son durumdan farklılık varsa
            // ok=0
            y++;
        }
        if(ok) return current;
        for(k=0;k<4;k++)
        {
            // olası 4 hamleyi hesaplayıp queue ya atıyoruz
            x=x0+a[k][0];
            y=y0+a[k][1];
            // yön arrayimizden boşluğun gideceği yeri hesaplıyoruz
            if(!(0<=x && x<3 && 0<=y && y<3)) continue;
            // bu if boşluğun gideceği yer tahta içinde mi diye bakıyor
            state ekle;
            // eklenecek state
            ekle.hamle=chamle;
            tmp=array[x0][y0];
            array[x0][y0]=array[x][y];
            array[x][y]=tmp;
            for(i=0,xx=0,yy=0;i<9;i++)
            {
                if(yy==3) {xx++; yy=0;}
                ekle.array+=array[xx][yy];
                yy++;
            }
            tmp=array[x0][y0];
            array[x0][y0]=array[x][y];
            array[x][y]=tmp;
        }
    }
}

```

```

        // state i olması gerektiği gibi doldurduk
        ekle.path=cpath+(char)a[k][2];
        ekle.evaluate();
        // F(D) sini hesaplattık
        if(used[ekle.array]) continue;
        // kullanılmış mı diye bakıyoruz, default olarak mapta bir
        // item yoksa ona tekabül eden değer 0 oluyor
        used[ekle.array]=1;
        // işaretledik
        Q.push(ekle);
        // Queue ya ekledik
    }
}
current.path="YAZDIRILAMAZ";
return current;
}

int main()
{
    FILE *input=fopen("yapboz.gir","r"),*output=fopen("yapboz.cik","w");
    char x;
    state ilk,sonuc;
    for(int i=0;i<9;i++) { fscanf(input," %c",&x); ilk.array+=x; };
    for(int i=0,xx=0,yy=0;i<9;i++) {
        if(yy==3) {yy=0; xx++;}
        fscanf(input," %c",&x);
        final+=x;
        num[convert(x)][0]=xx;
        num[convert(x)][1]=yy;
        yy++;
    };
    // input okuma
    ilk.hamle=0;
    ilk.evaluate();
    Q.push(ilk);
    used[ilk.array]=1;
    // ilk durumu ekledik
    sonuc=find();
    fprintf(output,"%d\n%s\n",sonuc.path.size(),sonuc.path.c_str());
    printf("%d times tried\n",move);
    return 0;
}

```

Elbet kodda anlamayacağınız noktalar olacaktır fakat genel iskeleti kullanarak kendi kodunuzu yazmanızı öneriyorum, bu sayede öğrendiklerinizi pekiştirmiş olursunuz

Hazırlayan Kaan Soral