**CS 591 Lab 1**

**We gave two simple heuristics for the 8-puzzle: Manhattan distance and misplaced tiles.**

**Several heuristics in the literature purport to improve on this—see, for example, Nilsson (1971), Mostow and Prieditis (1989), and Hansson et al. (1992).**

1. **Test these claims by implementing the heuristics and comparing the performance of the resulting algorithms.**

**Answer:**

**8-puzzle issue:**

The beginning state and objective state for the 8-puzzle issue are demonstrated as follows:

|  |  |  |
| --- | --- | --- |
| 7 | 2 | 4 |
| 5 |  | 6 |
| 8 | 3 | 1 |

START STAE

|  |  |  |
| --- | --- | --- |
|  | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

GOAL STATE

The beginning state comprises of unordered tiles while the objective state comprises of requested tiles.

The two basic heuristics executed are:

h1= the quantity of lost tiles that is '8'.

h2= entirety of the distinction of separations of the tiles to the objective state.

=3+1+2+2+2+3+3+2

=18

Investigation of the exhibition for every calculation:

**1. Best-first search:**

• The best-first pursuit is the general methodology like Tree-search calculation where a hub is chosen and extended by utilizing an assessment work . The capacity is a gauge of attractive quality for every hub, since it extends the most alluring unexpanded hub.

• The calculation likewise incorporates a part h(n) for f(n). The segment h(n) is called as Heuristic capacity.

• The heuristic capacity h(n)=0 when 'n' is an objective hub.

**2. Greedy Best-first search:**

• It grows the nearest hub to the objective and assesses the hub f(n) by utilizing the heuristic capacity h(n).

The calculation isn't ideal in light of the fact that the calculation stuck insider savvy and takes O(b^m) space intricacy and time unpredictability. 'm' is most extreme profundity of search space.

The great heuristic capacity can diminish the space intricacy by setting all hubs in memory.

**3. A\* algorithm:**

• The calculation adds the expense to arrive at the hub g(n) and h(n).So, the assessment work f(n)=g(n)+h(n).

• It is finished, ideal pursuit calculation on the grounds that the calculation never overestimates the expense of arrangement along the present way for a hub.

• The downside of A\* calculation is that it comes up short on space right on time, before it uses up all available time.

• Thus, the calculation isn't successful for tackling 15-puzzle issue.

**4. Recursive-best first Search:**

• The calculation is like best-first hunt calculation however utilizes just the straight space.

• The calculation utilizes f - limit variable so as to monitor f - esteem.

• It monitors the f - estimation of the best elective way from any precursors of the present hub.

• When the present hub surpasses f - limit , it loosens up back to the substitute way.

• When the recursion loosens up, the calculation replaces the f - estimation of every hub along the way with a sponsored-up esteem, which is the best f - esteem among every one of its youngsters.

• Thus, it is ideal technique for taking care of 8-puzzle issue.

**Heuristics:**

Every heuristic capacities' productivity relies upon the underlying and final states. Henceforth some heuristic can be proficient in a given issue and some in another.

1. **Construct the search tree to represent the problem.**

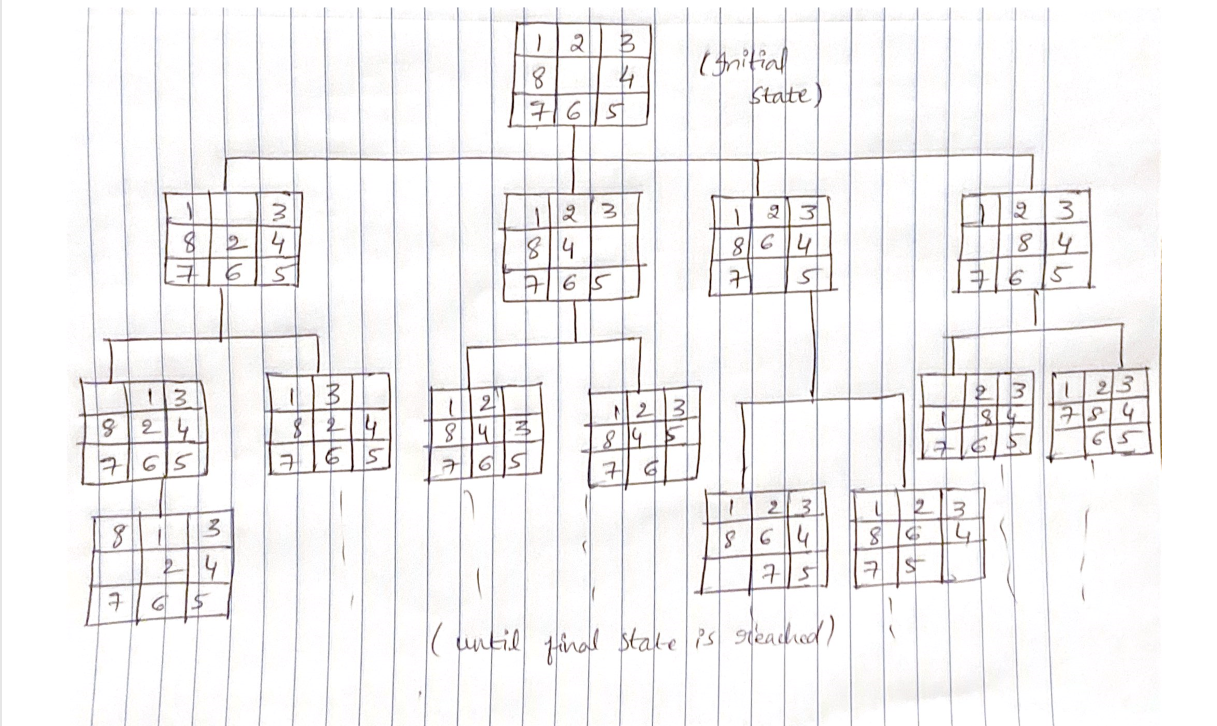
**Answer:**

The search tree which speaks to the 8-riddle issue thinking about an underlying state. This beginning state can be taken according to your benefit. Search tree begins with introductory state as root.

There are 4 potential ways from every hub, one for every activity top, right, base, left all together.

Be that as it may, each hub probably won't have potential for every one of the four activities as can be found in second layer of tree

where we have just two ways from the hub. We likewise don't perform tasks which return the state to any of the past state in way from root to that hub.



1. **Write in C++ language to solve the 8-puzzle problem.**

**Answer:**

#include <iostream>

#include <vector>

#include <set>

#include <fstream>

#include <Pstream>

using namespace std;

struct Statespace

{

      int Puzzle[3][3];

      int Spare[2];

      int Eval;

      bool operator< (Statespace) const;

      int operator- (Statespace) const;

      bool operator== (Statespace) const;

      void operator= (Statespace);

};

class Equal

{

      bool Greater;

      public:

      Equal(const bool& Lesser=1)

      {

           Greater=Lesser?0:1;

      }

      bool operator() (const Statespace& src, const Statespace& dest) const

      {

           if(Greater)

              return (src<dest?0:1);

           else

              return (src<dest);

      }

};

istream& operator>> (istream& input, Statespace& s)

{

      for(int i=0; i<3; i++)

      {

          for(int j=0; j<3; j++)

          {

              input>>s.Puzzle[i][j];

              if(s.Puzzle[i][j]==0)

              {

                  s.Spare[0]=i;

                  s.Spare[1]=j;

              }

          }

      }

      return input;

}

ostream& operator<< (ostream& output, Statespace& s)

{

     for(int i=0; i<3; i++)

     {

         for(int j=0; j<3; j++)

         {

             output<<s.Puzzle[i][j]<<" ";

         }

         output<<endl;

     }

     return output;

}

bool Statespace::operator< (Statespace s) const

{

     return (Eval<s.Eval);

}

int Statespace::operator- (Statespace s) const

{

    int count=0;

    for (int i=0; i<3; i++)

    {

        for(int j=0; j<3; j++)

        {

             if(Puzzle[i][j]!=s.Puzzle[i][j] && Puzzle[i][j]!=0)

                count++;

        }

    }

    return count;

}

bool Statespace::operator== (Statespace s) const

{

     if((\*this)-s==0)

        return 1;

     else

        return 0;

}

void Statespace::operator= (Statespace s)

{

     for(int i=0; i<3; i++)

     {

          for(int j=0; j<3; j++)

          {

              Puzzle[i][j]=s.Puzzle[i][j];

          }

     }

     Spare[0]=s.Spare[0];

     Spare[1]=s.Spare[1];

     Eval=s.Eval;

}

Statespace Initial, Goal, get();

int t=0;

Statespace st;

ofstream ofs("out.txt");

set<Statespace, Equal> Openlist;

vector<Statespace> Explored;

//Move Fuction, To move blocks

inline int Move(char pos)

{

     int p, q;

     int r, c;

     int s1=st.Spare[0];

     int s2=st.Spare[1];

     if(pos == 'R')

     {

         p=0;

         q=1;

     }

     else if(pos == 'L')

     {

          p=0;

          q=-1;

     }

     else if(pos == 'U')

     {

          p=1;

          q=0;

     }

     else if(pos == 'D')

     {

          p=-1;

          q=0;

     }

     r = s1+p;

     c = s2+q;

     swap(st.Puzzle[s1][s2], st.Puzzle[r][c]);

     if(find(Explored.begin(),Explored.end(),st) != Explored.end())

     {

          st=Initial;

          return 0;

     }

     st.Spare[0]=r;

     st.Spare[1]=c;

     get();

     Openlist.insert(st);

     st=Initial;

     return 0;

}

Statespace get()

{ st.Eval=Goal-st; }

int BFS()

{

    set<Statespace, Equal>::iterator l=Openlist.begin();

    st=\*l;

    if(st==Goal)

    {

      return t;

    }

  ofs<<st<<endl;

    cout<<st<<endl;

    Explored.push\_back(st);

    Openlist.erase(l);

    Initial=st;

    if(st.Spare[1]<2)

       Move('R');

    if(st.Spare[1]>0)

       Move('L');

    if(st.Spare[0]<2)

       Move('U');

    if(st.Spare[0]>0)

       Move('D');

    t++;

    BFS();

}

int main()

{

    ifstream inf("A2in1.txt");

    inf>>Initial>>Goal;

    Openlist.insert(Initial);

    BFS();

    ofs<<st<<endl;

    cout<<st<<endl;

    ifstream infile("out.txt");

    ofstream ofile("A2P1out1.txt");

    int arr[10], arr1[10], arr2[10], arr3[10], arr4[10];

    int l=0, m=0, n=0, p=0, q=0;

    for(int i=0; i<9; i++)

    {

      infile>>arr[i];

      if(arr[i]==0)

      {

         l=i;

      }

    }

    for(int i=0; i<9; i++)

    {

      infile>>arr1[i];

      if(arr1[i]==0)

      {

         m=i;

      }

    }

    for(int i=0; i<9; i++)

    {

      infile>>arr2[i];

      if(arr2[i]==0)

      {

         n=i;

      }

    }

    for(int i=0; i<9; i++)

    {

      infile>>arr3[i];

      if(arr3[i]==0)

      {

         p=i;

      }

    }

    for(int i=0; i<9; i++)

    {

      infile>>arr4[i];

      if(arr4[i]==0)

      {

         q=i;

      }

    }

    if(m==l+1)

    {

       ofile<<arr[m]<<" left"<<endl;

    }

    else if(m==l+3)

    {

       ofile<<arr[m]<<" up"<<endl;

    }

    if(n==m+1)

    {

        ofile<<arr1[n]<<" left"<<endl;

    }

    else if(n==m+3)

    {

        ofile<<arr1[n]<<" up"<<endl;

    }

    else if(n==m-3)

    {

        ofile<<arr1[n]<<" down"<<endl;

    }

    else if(n==m-1)

    {

        ofile<<arr1[n]<<" right"<<endl;

    }

    if(p==n+1)

    {

        ofile<<arr2[p]<<" left"<<endl;

    }

    else if(p==n+3)

    {

        ofile<<arr2[p]<<" up"<<endl;

    }

    else if(p==n-3)

    {

        ofile<<arr2[p]<<" down"<<endl;

    }

    else if(p==n-1)

    {

        ofile<<arr2[p]<<" right"<<endl;

    }

    if(q==p+1)

    {

        ofile<<arr3[q]<<" left"<<endl;

    }

    else if(q==p+3)

    {

        ofile<<arr3[q]<<" up"<<endl;

    }

    else if(q==p-3)

    {

        ofile<<arr3[q]<<" down"<<endl;

    }

    else if(q==p-1)

    {

        ofile<<arr3[q]<<" right"<<endl;

    }

                ofile << "Goal state found." << endl;

                cout<<"Goal state found."<<endl;

    return 0;

}

**Result:**

