**ЗВІТ**

до лабораторної роботи №1

на тему “ Рішення задачі «гра у 8» за допомогою методу пошуку у ширину”

**Мета роботи:** Ознайомитись з не інформативними методами пошуку рішення задач в просторі станів. Навчитись застосовувати на практиці алгоритм пошуку в ширину

**Індивідуальне завдання:**

1)Розв’язати задачу «гра у 8» за допомогою методу пошуку у ширину до третього рівня дерева пошуку (початковий стан –перший рівень) ручним способом. У звіті навести дерево пошуку.

2)Розробити алгоритм і програму рішення задачі «гра у 8» за допомогою методу пошуку у ширину.

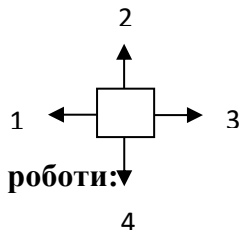
3) У звіті навести загальну кількість згенерованих станів, кількість станів занесених в базу станів, кількість відкинутих станів, глибину дерева пошуку, на якій знайдено рішення.

6) У звіті навести блок-схему алгоритму і роздрук програми.

**Заданий стан:**

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

**Алгоритм пересування пустої фішки:**




**Хід роботи:**

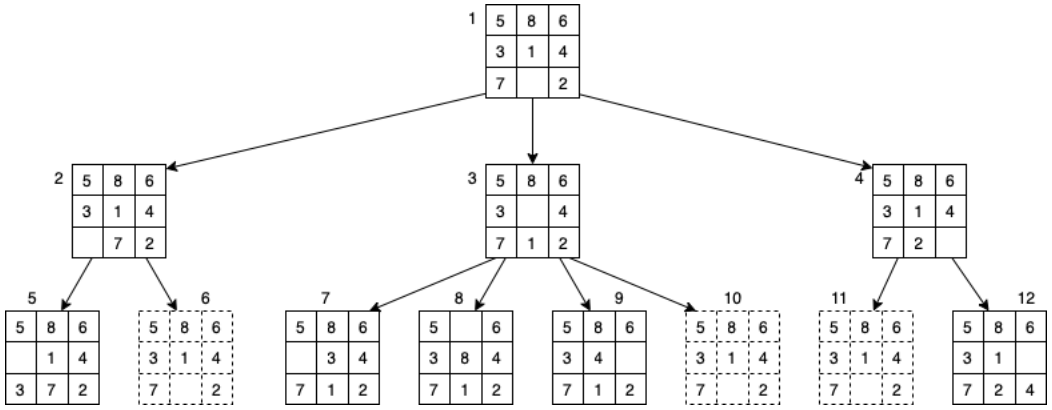



**Рисунок 1.** Дерево пошуку

**Код програми:**

```
#include <iostream>
#include<vector>
#include<queue>
#include<stack>
#include <unordered_set>
#include<fstream>
#define N 3 //number of row/coll
using namespace std;
vector<vector<int>> result={{1,2,3},{8,0,4},{7,6,5}}; // result matrix, 0-represent empty position


long long matrix_to_ll(vector<vector<int>> &matrix)
{
    long long res=matrix[0][0];
    for(int i=0;i<matrix.size();i++)
    {
        for(int j=0;j<matrix.size();j++)
        {
            res=res<<4;
            res=res | matrix[i][j];
        }
    }
    return res;
}


vector<vector<int>> ll_to_matrix(long long var)
{
    vector<vector<int>> res(N,vector<int>(N));
    int multiplayer=0;
    for(int i=res.size()-1;i>=0;i--)
    {
        for(int j=res.size()-1;j>=0;j--)
        {
            res[i][j]=(var>>(4*multiplayer)) & 0xf; //0xf-read 4-bite 0x7- read 3-bite 0x3- read 2-bite
            multiplayer++;
        }
    }

    return res;
}
```

```cpp
vector<long long> find_possible_position(vector<vector<int>> &matrix)
{
    vector<long long> res;
    int ind1,ind2;
    for(int i=0;i<matrix.size();i++)
    {
        for(int j=0;j<matrix.size();j++)
        {
            if(matrix[i][j]==0)
            {
                ind1=i;
                ind2=j;
                break;
            }

        }
    }
    if(ind2-1>=0) //check left
    {
        vector<vector<int>> temp=matrix;
        temp[ind1][ind2]=temp[ind1][ind2-1];
        temp[ind1][ind2-1]=0;
        res.emplace_back(matrix_to_ll(temp));
    }
    if(ind1-1>=0)  //check up
    {
        vector<vector<int>> temp=matrix;
        temp[ind1][ind2]=temp[ind1-1][ind2];
        temp[ind1-1][ind2]=0;
        res.emplace_back(matrix_to_ll(temp));
    }
    if(ind2+1<=matrix.size()-1) //check right
    {
        vector<vector<int>> temp=matrix;
        temp[ind1][ind2]=temp[ind1][ind2+1];
        temp[ind1][ind2+1]=0;
        res.emplace_back(matrix_to_ll(temp));
    }
    if(ind1+1<=matrix.size()-1) //check down
    {
        vector<vector<int>> temp=matrix;
        temp[ind1][ind2]=temp[ind1+1][ind2];
        temp[ind1+1][ind2]=0;
        res.emplace_back(matrix_to_ll(temp));
    }


    return res;
}


vector<long long>bfs(vector<vector<int>>& start)
{
    queue<long long> q;
    unordered_set<long long> visited;
    vector<long long> vis;
    q.push(matrix_to_ll(start));
    long long res= matrix_to_ll(result);
    ofstream f("ai_1.txt", ios::app);
    f.clear();
    visited.insert(matrix_to_ll(start));
    vis.push_back(matrix_to_ll(start));
    int iter=0;
    int ind=0;
    while(!q.empty())
```

```
    {
        long long cur=q.front();
        q.pop();
        vector<vector<int>> temp= ll_to_matrix(cur);
        vis.push_back(cur);
        f<<"Iteration number "<<ind++<<endl;
        for(int i=0;i<temp.size();i++)
        {
            for(int j=0;j<temp.size();j++)
            {
                f<<temp[i][j];
            }
            f<<endl;
        }
        f<<endl;
        if(cur==res )
            break;
        vector<long long> next= find_possible_position(temp);
        for(auto& elem:next)
        {
            auto it=visited.find(elem);

            if(it==visited.end())
            {
                q.push(elem);
                visited.insert(elem);
            }
            else
            {
               iter++;
            }
        }

    }
    f.close();
    vis.push_back(static_cast<long long>(iter));
    return vis;
}


void Menu(vector<vector<int>>&matrix )
{
    int key;
    vector<long long> visited=bfs(matrix);
    auto it=visited.begin();
    while(true)
    {
        cout<<"1-Print next state"<<endl;
        cout<<"2-Print last state"<<endl;
        cout<<"3-Exit"<<endl;
        cin>>key;
        if(key==1)
        {
        vector<vector<int>> temp(ll_to_matrix(*it));
        for(int i=0;i<temp.size();i++)
        {
            for(int j=0;j<temp.size();j++)
            {
                cout<<temp[i][j]<<" ";
            }
            cout<<endl;
        }
        cout<<endl;
        it++;
        if(it++==visited.end())
```

```
            break;
        else
            it--;
        }
        else if(key==2)
        {
            auto it_end=visited.end();
            it_end--;
            it_end--;
            vector<vector<int>> temp(ll_to_matrix(*it_end));
            for(int i=0;i<temp.size();i++)
            {
                for(int j=0;j<temp.size();j++)
                {
                    cout<<temp[i][j]<<" ";
                }
                cout<<endl;
            }
            cout<<endl;
            if(temp==result)
            {
                cout<<"Final state achieved"<<endl;
            }
            else
            {
                vector<long long> next= find_possible_position(temp);
                cout<<"It is impossible to achieve the final state"<<endl;
                cout<<"Next states are already in the state database"<<endl;
                for(auto elem:next)
                {
                  auto fnd=find(visited.begin(),visited.end(),elem);
                  int ind=fnd-visited.begin();
                  cout<<"Index in the set database: "<<ind<<endl;
                  vector<vector<int>> cur= ll_to_matrix(elem);
                    for(int i=0;i<cur.size();i++)
                    {
                        for(int j=0;j<cur.size();j++)
                        {
                            cout<<cur[i][j]<<" ";
                        }
                        cout<<endl;
                    }
                    cout<<endl;
                }
                cout<<"Number of generated state: "<<visited.size()-
1+visited[visited.size()-1]<<endl;
                cout<<"Number of state added to database: "<<visited.size()-
1<<endl;
                cout<<"Number of rejected state: "<<visited[visited.size()-
1]<<endl;
            }

        }
        else
            return;
    }
}



int main() {
    vector<vector<int>> matrix={{5,8,6},{3,1,4},{7,0,2}}; //start matrix, 0-
represent empty position
    Menu(matrix);
```

```
    return 0;
}
```

З даного початкового стану неможливо досягнути кінцевого стану. Тому в результаті роботи алгоритму генерується 181440 станів і додаються в базу станів після чого закінчуються можливі переходи і алгоритм закінчується.

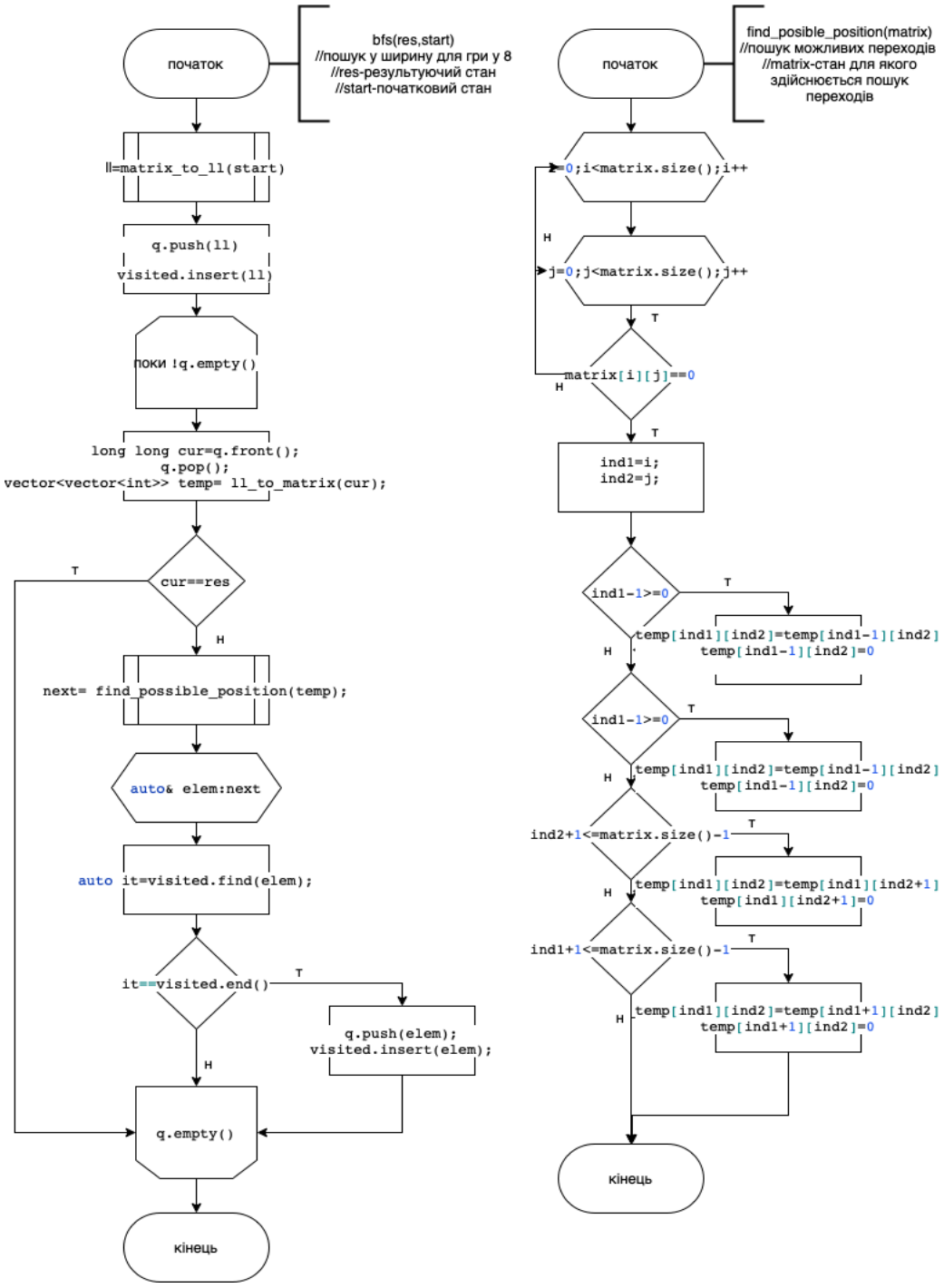



**Рисунок 2.** Блок-схема алгоритму

Вивід кількох перших станів:

```
5 8 6
3 1 4
7 0 2

Number of iteration 1

5 8 6
3 1 4
0 7 2

Number of iteration 2

5 8 6
3 0 4
7 1 2

Number of iteration 3

5 8 6
3 1 4
7 2 0

Number of iteration 4

5 8 6
0 1 4
3 7 2

Number of iteration 5

5 8 6
0 3 4
7 1 2
```

```
1-Print next state
2-Print last state
3-Exit
2
2 3 7
4 1 8
6 5 0

It is impossible to achieve the final state
Next states are already in the state database
Index in the set database: 181438
2 3 7
4 1 8
6 0 5

Index in the set database: 181427
2 3 7
4 1 0
6 5 8

Number of generated state: 483842
Number of state added to database: 181441
Number of rejected state: 302401
```

**Висновок:** В результаті виконання даної лабораторної роботи я ознайомився з не інформативними методами пошуку рішення задач в просторі станів. навчився застосовувати на практиці алгоритм пошуку в ширину.