دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

4/17/2020

Report Midterm Project

Advanced Programming

Mohammad Javad Amin (9523008)
AMIRKABIR UNIVERSITY OF TECHNOLOGY

به نام خدا

هدف پروژه ؛ طراحی برنامه ایی که یک روبیک ۲*۲ درهم را حل کند .

ابتدای کار class Rubik را تعریف می‌کنیم که شامل ویژگی‌های یک مکعب روبیک است.

این class شامل member variable و همچنین member function هایی است که در ادامه به تعریف و بررسی تک تک آن ها می پردازیم.

member variable ها به صورت private :

۱)colors : شامل رنگ‌های مربع های کوچک روبیک است.

۲) user_color : شامل رنگ هایی است که کاربر برای هر سطح روبیک انتخاب کرده است.

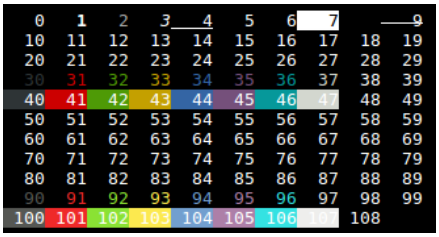
۳)numbers : نشان دهنده مربع های کوچک روبیک است.

کل class Rubik را در زیر قابل مشاهده است.

```
class Rubik
{
private:                                                    //member variable
    std::vector<std::string> colors;                        //to save  elements color of rubik
    std::vector<std::string> user_color;                    //to save user colors for rubik
    std::vector<size_t> numbers;                            //to save elements of rubik
                                                            //member function
    void introduce();                                       //show some information
    void set_rubik();                                       //get a rubik from user
    void set_color();                                       //save colors the rubik
    std::string which_color(const std::string& input_color);//to recognition color
    void counter(const size_t& i);                          //to check input rubik
    void counter_color(const std::string& s);               //to check input color rubik


public:
    Rubik();                                                                        //default constructor
    Rubik(const std::vector<size_t>& input_vector);                                 //constructor
    Rubik rotate(const size_t& face, size_t repeat, const std::string& rotate_name);//Rotate rubik
    bool solve_check();                                                             //check rubik is solve or not
    void show_rubik_index();                                                        //show rubik elements
    void show_rubik_color();                                                        //show rubik  elements color

    bool operator==(Rubik A);                                                       //operator ==
};
```

قبل از پرداختن به member function ها یک enum class و یک تابع کمکی تعریف می‌کنیم.

enum class ansi_color_code : در این enum class کد های ANSI برای رنگ های مختلف (برای رنگ نارنجی باید از شیوه دیگر استفاده کرد) را تعریف می‌کنیم . جدول کدهای ANSI در صفحه بعدی قابل مشاهده است.




```cpp
enum class ansi_color_code : int {                    //color

    red = 101,
    green = 102,
    yellow = 103,
    blue = 104,
    white = 107,
    orange = 0,
};
```

حال تابع print_as_color را تعریف میکنم که کار چاپ کردن عبارات داده شده به آن در رنگ خواسته شده است که به صورت زیر پیاده سازی می‌شود.

```cpp
template<ansi_color_code color, typename printable>
inline std::string print_as_color(printable const& value)    //save string for print each elements in color
{
    std::stringstream sstr{};
    if (color == ansi_color_code::orange)
        sstr << "  \033[48;2;255;165;0m" << value << "\033[0m";
    else
        sstr << "\033[1;" << static_cast<int>(color) << "m" << value << "\033[0m";
    return sstr.str();
}
```

: member functions

ابتدا تابع هایی که به صورت private تعریف می‌شوند را بررسی می‌کنیم. دلیل اینکه این توابع را به صورت private تعریف کرده‌ایم این است که لزومی ندارد کاربر به آن ها دسترسی پیدا کند.

۱)introduce() : کار این تابع نشان داده اطلاعات رنگ ها وگرفتن رنگ های هر سطح روبیک است و همچنین نشان دهنده‌ی روش‌های جستجو برای حل روبیک و ... است و به صورت زیر پیاده سازی می‌شود.

```cpp
void Rubik::introduce()                    //show some information
{
    std::cout << "Hi!" << std::endl;
    std::cout << "introduce colors " << std::endl;
    std::cout << std::setw(15) << "green -> g" << std::endl;
    std::cout << std::setw(15) << "white -> w" << std::endl;
    std::cout << std::setw(15) << "orange -> o" << std::endl;
    std::cout << std::setw(16) << "yellow -> y " << std::endl;
    std::cout << std::setw(16) << "blue -> b " << std::endl;
    std::cout << std::setw(15) << "red -> r" << std::endl;
    Rubik introduce{ std::vector<size_t>{1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6 } };
    introduce.show_rubik_index();
    std::cout << "chose your color for each face (g/w/o/y/b/r) " << std::endl;
    for (size_t i{}; i < 6; i++)                    //get user colors
    {
        std::cout << "face " << i + 1 << " : ";
        std::cin >> user_color_r[i];
        std::cout << std::endl;
    }
    introduce.user_color = user_color_r;          //save user color
    introduce.show_rubik_color();                 //show rubik

    std::cout << std::endl << "we have four search methods here " << std::endl;
    std::cout << std::setw(15) << "Breadth-first search -> BFS" << std::endl;
    std::cout << std::setw(15) << "Depth-limited search -> DLS" << std::endl;
    std::cout << std::setw(15) << "Iterative-deepening search -> IDS" << std::endl;
    std::cout << std::setw(15) << "Bidirectional -> BI" << std::endl << std::endl;
    std::cout << "thanks for your attention " << std::endl;
    std::cout << "programed by Mohammad Javad Amin" << std::endl;
    std::cout << "spring 2020" << std::endl << std::endl << std::endl;
}
```

۲) set_rubik : این تابع ابتدا تابع introduce را اجرا کرده(قبلا کار این تابع شرح داده شده است) و سپس نشان می‌دهد نحوه واردکردن المان های هر سطح روبیک چه گونه است سپس تمام المان های روبیک را ازکاربر می‌گیرد و ذخیره میکند. و به صورت زیر پیاده‌سازی می‌شود.

```
void Rubik::set_rubik()                    //get a rubik from user
{
    introduce();                           //show some information

    std::cout << "for enter elements for each face put space between them and end of each face press enter and go to next face" << std::endl;
    std::cout << "enter your numbers in range 1 to 6" << std::endl;

    size_t input{};

    for (size_t i = 0; i < 6; i++)         //get 6 face
    {
        std::cout << "face " << i+1 << " :" << std::endl;

        for (size_t i = 0; i < 4; i++)
        {
            std::cin >> input;             //get a elements of face

            if (input > 0 && input < 7)    //check the range of input
                numbers.push_back(input);  //save elements of rubik
            else
                throw std::invalid_argument("out of the range number please replace it");
        }
    }
    for (size_t i{ 1 }; i < 7; i++)        //check input rubix
        counter(i);
    user_color = user_color_r;             //save user colors
    set_color();                           //save elements color of rubik
}
```

۳)counter : کار این تابع این‌که چک میکند المان ورودی به این تابع چند بار در روبیک تکرار شده است و اگر بیشتر یا کمتر از ۴ باشد پیغام میدهدکه اطلاعات ورودی روبیک غلط است و به صورت زیر پیاده سازی می‌شود.

```
void Rubik::counter(const size_t& i)   //check rubik is valid or not(check number of each rubik element)
{
    size_t count{};                    //counter

    for (size_t j = 0; j < 24; j++)    //count each element in rubik
        if(!(numbers[j]-i))
            count++;

    if (count < 4)                     //number of each must be 4 else it's incrroct
    {
        std::ostringstream oss{};
        oss << "Numbers of " << i << " less than 4 , it must be 4" << std::endl;
        throw std::invalid_argument(oss.str());
    }
    if (count > 4)
    {
        std::ostringstream oss{};
        oss << "Numbers of " << i << " bigger than 4 , it must be 4" << std::endl;
        throw std::invalid_argument(oss.str());
    }
}
```

۴)counter_color : این تابع چک میکندکه رنگ های وارد شده توسط کاربر درست هستند یانه مثلا دوبار یک رنگ را وارد نکرده باشد. پیاده سازی این تابع به صورت زیر است.

```
void Rubik::counter_color(const std::string& s)     //to check input color rubik
{
    size_t count{};                    //counter

    for (size_t j = 0; j < 6; j++)     //count each input color  from user
        if (s == user_color[j])
            count++;

    if (count < 1)                      //number of each color be 1 else it's incrroct
    {
        std::ostringstream oss{};
        oss << "Numbers of color " << s << " less than 1 , it must be 1" << std::endl;
        throw std::invalid_argument(oss.str());
    }
    if (count > 1)
    {
        std::ostringstream oss{};
        oss << "Numbers of color " << s << " bigger than 1 , it must be 1" << std::endl;
        throw std::invalid_argument(oss.str());
    }
}
```

۵)which_color : کار این تابع تشخیص رنگ ورودی و برگرداندن رشته مناسب برای رنگی چاپ کردن هر المان است که این کار را با تابع print_as_color انجام می‌دهد. پیاده سازی این تابع به صورت زیر است.

```
std::string  Rubik::which_color(const std::string& input_color)    //to recognition color and save it
{
    if (input_color == "g")
        return print_as_color<ansi_color_code::green>("g");
    else if (input_color == "r")
        return print_as_color<ansi_color_code::red>("r");
    else if (input_color == "w")
        return print_as_color<ansi_color_code::white>("w");
    else if (input_color == "y")
        return print_as_color<ansi_color_code::yellow>("y");
    else if (input_color == "b")
        return print_as_color<ansi_color_code::blue>("b");
    else if (input_color == "o")
        return print_as_color<ansi_color_code::orange>("o");
    else
        throw std::invalid_argument("wrong input color from user ");
}
```

۶)set_color : کار این تابع ابتدا چک میکندکه این اعداد وارد شده و رنگ های وارد شده توسط کاربر صحیح هستند یا نه این کار را به وسیله دو تابع counter و counter_color یا نه بررسی میکند. سپس رنگ هر المان روبیک را به وسیله تابع which_color تشخیص و ذخیره می‌کند . این تابع به صورت صفحه بعد پیاده سازی می‌شود.

```cpp
void Rubik::set_color()
{
    for (size_t i{ 1 }; i < 7; i++)      //check rubik is valid or not
        counter(i);
    for (size_t i{}; i < 6; i++)         //check rubik is valid or not
        counter_color(user_color[i]);

    for (size_t i{}; i < 24; i++)        //set color for rubik elements
    {
        if (numbers[i] == 1)
            colors[i] = which_color(user_color[0]);
        else if (numbers[i] == 2)
            colors[i] = which_color(user_color[1]);
        else if (numbers[i] == 3)
            colors[i] = which_color(user_color[2]);
        else if (numbers[i] == 4)
            colors[i] = which_color(user_color[3]);
        else if (numbers[i] == 5)
            colors[i] = which_color(user_color[4]);
        else if (numbers[i] == 6)
            colors[i] = which_color(user_color[5]);
        else
            throw std::invalid_argument("out of the range number please replace it");
    }
}
```

حال به سراغ member function هایی می‌رویم که به صورت public تعریف شده اند.

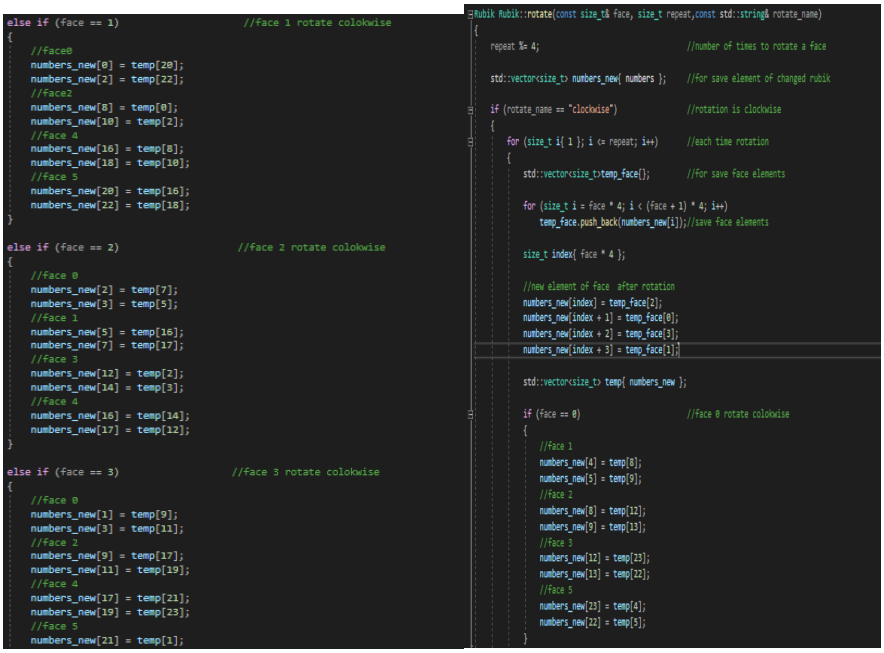
۱)ابتدا default constructor را به صورت زیر تعریف می‌کنیم. که در اینجا با تابع set_rubik روبیک را ازکاربر می‌گیریم سپس با تابعset_color رنگ های هر المان را ذخیره می‌کنیم.

```cpp
Rubik::Rubik()                              //default constructor
{
    colors = std::vector<std::string>(24);  //initialize colors
    user_color = user_color_r;              //initialize user colors
    set_rubik();                            //get a rubik
    set_color();                            //save elements color of rubik
}
```

۲)یک constructor دیگر تعریف میکنیم که ورودی آن به صورت یک وکتور از المان های روبیک است. در این مرحله چک میکنیم که اطلاعات وارد شده صحیح است یا نه .

```cpp
Rubik::Rubik(const std::vector<size_t>& input_vector)          //constructor
{
    if (input_vector.size() != 24)                             //number of inputs elements must be 24
        throw std::invalid_argument("wrong size input vector");

    colors = std::vector<std::string>(24);                     //initialize colors
    user_color = user_color_r;                                 //initialize user colors
    numbers = input_vector;                                    //sava input rubik
    for (size_t i{ 1 }; i < 7; i++)                            //check input rubix
        counter(i);
}
```

۳) rotate : این تابع سه ورودی دارد که به ترتیب الف)سطح مورد نظر برای چرخش ب)تعداد چرخش پ)جهت چرخش است . کار تابع این است که سطح مورد نظر را به تعداد دفعات مشخص شده در جهت داده شده به آن بچرخاند. توجه شد اگر تعداد چرخش ها بشتر از ۳ بار باشد آن را بر ۴ تقسیم می‌کنیم چون مثلا ۵ بار چرخش با یک بار چرخش برابر است و نکته دیگر این است چرخش در جهت خلاف عقربه های ساعت درست ۳ بار چرخش در جهت عقربه های ساعت است. این تابع به صورت زیر پیاده سازی می‌شود.



```
else if (face == 1)                    //face 1 rotate colokwise
{
    //face0
    numbers_new[0] = temp[20];
    numbers_new[2] = temp[22];
    //face2
    numbers_new[8] = temp[0];
    numbers_new[10] = temp[2];
    //face 4
    numbers_new[16] = temp[8];
    numbers_new[18] = temp[10];
    //face 5
    numbers_new[20] = temp[16];
    numbers_new[22] = temp[18];
}

else if (face == 2)                    //face 2 rotate colokwise
{
    //face 0
    numbers_new[2] = temp[7];
    numbers_new[3] = temp[5];
    //face 1
    numbers_new[5] = temp[16];
    numbers_new[7] = temp[17];
    //face 3
    numbers_new[12] = temp[2];
    numbers_new[14] = temp[3];
    //face 4
    numbers_new[16] = temp[14];
    numbers_new[17] = temp[12];
}

else if (face == 3)                   //face 3 rotate colokwise
{
    //face 0
    numbers_new[1] = temp[9];
    numbers_new[3] = temp[11];
    //face 2
    numbers_new[9] = temp[17];
    numbers_new[11] = temp[19];
    //face 4
    numbers_new[17] = temp[21];
    numbers_new[19] = temp[23];
    //face 5
    numbers_new[21] = temp[1];
```

```
Rubik Rubik::rotate(const size_t& face, size_t repeat,const std::string& rotate_name)
{
    repeat %= 4;                                //number of times to rotate a face

    std::vector<size_t> numbers_new{ numbers };    //for save element of changed rubik

    if (rotate_name == "clockwise")             //rotation is clockwise
    {
        for (size_t i{ 1 }; i <= repeat; i++)       //each time rotation
        {
            std::vector<size_t>temp_face{};         //for save face elements

            for (size_t i = face * 4; i < (face + 1) * 4; i++)
                temp_face.push_back(numbers_new[i]);//save face elements

            size_t index{ face * 4 };

            //new element of face  after rotation
            numbers_new[index] = temp_face[2];
            numbers_new[index + 1] = temp_face[0];
            numbers_new[index + 2] = temp_face[3];
            numbers_new[index + 3] = temp_face[1];

            std::vector<size_t> temp{ numbers_new };

            if (face == 0)                      //face 0 rotate colokwise
            {
                //face 1
                numbers_new[4] = temp[8];
                numbers_new[5] = temp[9];
                //face 2
                numbers_new[8] = temp[12];
                numbers_new[9] = temp[13];
                //face 3
                numbers_new[12] = temp[23];
                numbers_new[13] = temp[22];
                //face 5
                numbers_new[23] = temp[4];
                numbers_new[22] = temp[5];
            }
```

```
            else if (face == 4)                 //face 4 rotate colokwise
            {
                //face 1
                numbers_new[6] = temp[21];
                numbers_new[7] = temp[20];
                //face 2
                numbers_new[10] = temp[6];
                numbers_new[11] = temp[7];
                //face 3
                numbers_new[14] = temp[10];
                numbers_new[15] = temp[11];
                //face 5
                numbers_new[20] = temp[15];
                numbers_new[21] = temp[14];
            }

            else if (face == 5)                 //face 5 rotate colokwise
            {
                //face 0
                numbers_new[0] = temp[13];
                numbers_new[1] = temp[15];
                //face 1
                numbers_new[4] = temp[1];
                numbers_new[6] = temp[0];
                //face 3
                numbers_new[15] = temp[18];
                numbers_new[13] = temp[19];
                //face 4
                numbers_new[18] = temp[4];
                numbers_new[19] = temp[6];
            }
        }
    }
    else if (rotate_name == "anticlockwise")     //rotation is anticlockwise
        return Rubik{ rotate(face, 4 - repeat, "clockwise") };
    else
        throw std::invalid_argument("wrong name rotate");

    return Rubik{ numbers_new };                //return rubik after rotation
```

در این تابع تغییر جای هر المان بسته به نوع چرخش هر وجه در آن حساب شده و جای المان ها تغییر می‌کنند و در آخر روبیک حاصل از چرخش را به عنوان خروجی برمی‌گرداند.

۴) solve_check : این تابع بررسی میکندکه آیا روبیک کامل است یا نه که این کار را با چک کردن رنگ هر چهار المان هر سطح انجام می‌دهد.

```
bool Rubik::solve_check()                    //check rubik is solve or not
{
    bool check_flag{ true };                  //to show rubik is solved or not

    set_color();                              //save element colors of rubik

    for (size_t i = 0; i < 24; i += 4)        //check each face of rubik has same elements or not
    {
        if (colors[i] == colors[i + 1] && colors[i + 1] == colors[i + 2] && colors[i + 2] == colors[i + 3])
            continue;
        else
        {
            check_flag = false;
            break;
        }
    }

    return check_flag;
}
```

۵)show_rubik_index : کار این تابع نشان دادن شکل باز شده روبیک با المان های عددی آن است و به صورت زیر پیاده‌سازی می‌شود.

```
void Rubik::show_rubik_index()                //show elements of the rubik
{
    std::cout << "Rubik's cube interal indexing is" << std::endl << std::endl;
    std::cout << std::setw(29) << "+---------+" << std::endl;
    for (size_t i{}; i < 4; i += 2)
        std::cout << std::setw(20) << "|" << std::setw(3) << numbers[i] << std::setw(3) << numbers[i + 1] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(20) << "+---------+" << "---------+" << "---------+" << std::endl;
    std::cout << std::setw(11) << "|" << std::setw(3) << numbers[4] << std::setw(3) << numbers[5] << std::setw(3) << "|";
    std::cout << std::setw(3) << numbers[8] << std::setw(3) << numbers[9] << std::setw(3) << "|";
    std::cout << std::setw(3) << numbers[12] << std::setw(3) << numbers[13] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(11) << "|" << std::setw(3) << numbers[6] << std::setw(3) << numbers[7] << std::setw(3) << "|";
    std::cout << std::setw(3) << numbers[10] << std::setw(3) << numbers[11] << std::setw(3) << "|";
    std::cout << std::setw(3) << numbers[14] << std::setw(3) << numbers[15] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(20) << "+---------+" << "---------+" << "---------+" << std::endl;
    for (size_t i{ 16 }; i < 20; i += 2)
        std::cout << std::setw(20) << "|" << std::setw(3) << numbers[i] << std::setw(3) << numbers[i + 1] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(29) << "+---------+" << std::endl;
    for (size_t i{ 20 }; i < 24; i += 2)
        std::cout << std::setw(20) << "|" << std::setw(3) << numbers[i] << std::setw(3) << numbers[i + 1] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(29) << "+---------+" << std::endl;
}
```

۶)show_rubik_color : کار این تابع نشان دادن شکل باز شده ی روبیک با المان های رنگی آن است. پیاده سازی این تابع به صورت زیر است.

```
void Rubik::show_rubik_color()            //show elements color of the rubik
{
    set_color();

    std::cout << "Rubik's cube interal colors is" << std::endl << std::endl;
    std::cout << std::setw(29) << "+---------+" << std::endl;
    for (size_t i{}; i < 4; i += 2)
        std::cout << std::setw(20) << "|" << std::setw(15) << colors[i] << std::setw(15) << colors[i + 1] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(20) << "+---------+" << "---------+" << "---------+" << std::endl;
    std::cout << std::setw(11) << "|" << std::setw(15) << colors[4] << std::setw(15) << colors[5] << std::setw(3) << "|";
    std::cout << std::setw(15) << colors[8] << std::setw(15) << colors[9] << std::setw(3) << "|";
    std::cout << std::setw(15) << colors[12] << std::setw(15) << colors[13] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(11) << "|" << std::setw(15) << colors[6] << std::setw(15) << colors[7] << std::setw(3) << "|";
    std::cout << std::setw(15) << colors[10] << std::setw(15) << colors[11] << std::setw(3) << "|";
    std::cout << std::setw(15) << colors[14] << std::setw(15) << colors[15] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(20) << "+---------+" << "---------+" << "---------+" << std::endl;
    for (size_t i{ 16 }; i < 20; i += 2)
        std::cout << std::setw(20) << "|" << std::setw(15) << colors[i] << std::setw(15) << colors[i + 1] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(29) << "+---------+" << std::endl;
    for (size_t i{ 20 }; i < 24; i += 2)
        std::cout << std::setw(20) << "|" << std::setw(15) << colors[i] << std::setw(15) << colors[i + 1] << std::setw(3) << "|" << std::endl;
    std::cout << std::setw(29) << "+---------+" << std::endl;
}
```

۷)operator==: که کار آن چک کردن این است که آیا دو روبیک مختلف در دوسمت آن یکی هستند یا نه .

```
bool Rubik::operator==(Rubik A)          //operator ==
{
    set_color();                         //save element colors of rubiks
    A.set_color();                       //compare two rubiks
    if (colors == A.colors)
        return true;
    else
        return false;
}
```

حال به سراغ تعریف class graphمیرویم که در آن الگوریتم های مختلف برای حل روبیک را در آن پیاده سازی می‌کینم.

این class شامل member variable و همچنین member function هایی است که در ادامه به تعریف و بررسی تک تک آن ها می پردازیم.

member variable ها به صورت private :

۱)rubiks : وکتوری از روبیک هایی که پس از هرچرخش ساخته می‌شود.

۲)limit : عمق است که جستجو در آن انجام می شود.

۳)limit_temp : عمقی که در روش بازگشتی DLS استفاده می‌شود.

۴)size : تعداد روبیک هایی است که ما بررسی کردیم ایم تا به جواب برسیم.

۵)rubik_rotation : شامل جهت چرخش در هر مرحله است.

۶)solution_rubik_rotations : شامل جهت هایی است که به جواب منتهی می‌شود.

۷)solution_rubik_rotations_bidirection : شامل جهت هایی است که در روش bidirection منتهی به جواب می‌شود.

کل class graph را در شکل زیر قابل مشاهده است.

```
class Graph
{
private:
    std::vector<Rubik> rubiks;    //rubiks
    size_t limit{};               //depth limit
    int limit_temp{};             //depth limit for DLS algorithm
    size_t size{};                 //size of nodes

    std::vector<std::string>rubik_rotations{ "" };    //to save rubik rotations
    std::vector<std::string>solution_rubik_rotations{ "" }; //to save solutions rubik rotations
    std::vector<std::string>solution_rubik_rotations_bidirectional{}; //to save solutions rubik rotations in bidirectional algotithm

    bool new_nodes_BFS();                                  //new nodes for each limit in BFS algorithm
    void save_solution(size_t limit_graph, size_t index_node); //for save Rotations for solve rubik
    void new_nodes_bidirectional(const bool& flag);        //new nodes for each limit in bidirectional algorithm
    void save_solution_bidirectional(Graph& Target, const std::array<int, 2>& find);//save rotations to how to solve rubik
    std::array<int, 2> chek_bidirectional(Graph& Target);//check we solved rubik or not
    void bidirectional();                                  //solve rubiks with BFS algorithm
    bool DLS(size_t l, Rubik B);         //DLS algorithm
    bool new_nodes_dls(Rubik A);        //new nodes for each limit in DLS algorithm
public:
    Graph(const Rubik& A);                    //constructors
    void show_solution_BFS();                 //solution with BFS algorithm
    void show_solution_bidirectional();       //solution with bidirectional algorithm
    void show_solution_DLS();                 //solution with DLS algorithm
    void show_solution_IDS();                 //solution with IDS algorithm
    size_t getlimit();                        //get depth limit
    size_t getsize();                         //get number of nodes
    Rubik operator[](const size_t& i);        //operator []
};
```

ابتدا یک تابع کمکی را تعریف می‌کنیم که کار آن حساب کردن تعداد کل node ها در limit داده شده به آن است.و به صورت زیر پیاده سازی می‌شود.

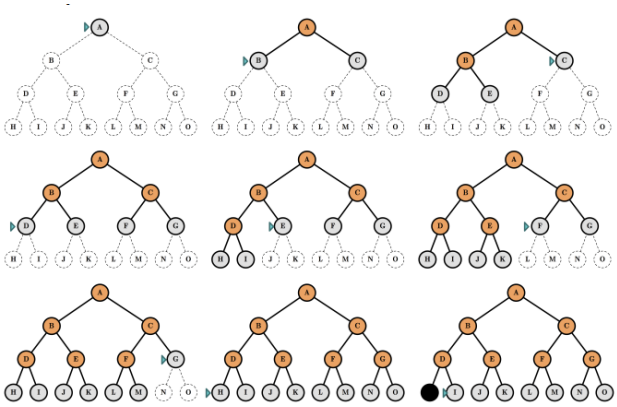
```
size_t number_of_node(int x)         //number of nodes in each limit in BFS algoritm
{
    if (x < 0)
        return 0;
    else
        return static_cast<size_t>((std::pow(12, x + 1) - 1) / 11) - 1;
}
```

: member function

ابتدا تابع هایی که به صورت private تعریف می‌شوند را بررسی می‌کنیم. دلیل اینکه این توابع را به صورت private تعریف کرده‌ایم این است که لزومی ندارد کاربر به آن ها دسترسی پیدا کند.

۱)new_nodes_BFS : کار این تابع ساختن حالت های جدید روبیک با استفاده از روبیک ها در limit بالاتر است. و پس ساخت هر حالت جدید نحوه چرخش آن نیز ذخیره می‌شود و بعد چک می شود که آیا به جواب رسیده ایم یا نه و در آخر تعداد node ها ذخیره می‌شود.

الگوریتم BFS به صورت صفحه بعد است.



```cpp
bool Graph::new_nodes_BFS()     //new nodes for each limit in BFS algorithm
{
    size_t numer_of_new_nodes{ static_cast<size_t>(std::pow(12,limit)) };  //number of total nodes in this limit
    bool find_solve{ false };                                               //for show find solution

    for (size_t i = size - numer_of_new_nodes; i < size; i++)              //creat new nodes
    {
        if (rubiks[i].solve_check())                                       //check rubik is solve or not
        {
            limit--;
            find_solve = true;
            break;
        }

        for (size_t j = 0; j < 12; j++)                                   //rotate rubik to crate new nodes
        {
            std::ostringstream oss{};

            if (j / 6 == 0)                                               //rotate clockwise
            {
                rubiks.push_back(rubiks[i].rotate(j % 6, 1, "clockwise"));//save new nodes
                oss << "face " << (j % 6) + 1 << "->  clockwise";
                rubik_rotations.push_back(oss.str());                     //save rotations
            }
            if (j / 6 == 1)                                               //rotate anticlockwise
            {
                rubiks.push_back(rubiks[i].rotate(j % 6, 1, "anticlockwise"));//save new nodes
                oss << "face " << (j % 6) + 1 << "-> anticlockwise";
                rubik_rotations.push_back(oss.str());                    //save rotations
            }
            if (rubiks[rubiks.size() - 1].solve_check())                //check rubik is solve or not
            {
                find_solve = true;
                break;
            }
        }
        if (find_solve)
            break;
    }
```

۲)save_solution : کار این تابع ذخیره کردن نحوه چرخش ها در روش BFSتا رسیدن به جواب است و به صورت زیر پیاده سازی می‌شود.

```cpp
void Graph::save_solution(size_t limit_graph, size_t index_node)  //for save Rotations for solve rubik
{
    if (limit_graph == 1)
        solution_rubik_rotations[0] = rubik_rotations[size - 1];
    else
    {
        size_t index{ ((index_node - number_of_node(limit_graph - 1)) / 12) + number_of_node(limit_graph - 2) };

        solution_rubik_rotations.push_back(rubik_rotations[index + 1]);
        save_solution(limit_graph - 1, index);
    }
}
```

۳)new_nodes_bidirectional : کار این تابع ساختن حالت های جدید روبیک با استفاده از روبیک ها در limit بالاتر است. و پس ساخت هر حالت جدید نحوه چرخش آن نیز ذخیره و بر اساس ورودی تابع مشخص می شود که این چرخش از سمت رویک هدف بوده چون در این حالت باید جهت چرخش ها برعکس شود و یا چرخش از جهت روبیکی است که کاربر وارد کرده است. و در آخر تعداد کل روبیک ها ذخیره می‌شود. پیاده سازی آن به صورت شکل صفحه بعد است.

```cpp
void Graph::new_nodes_bidirectional(const bool& flag) //new nodes for each limit in bidirectional algorithm
{
    size_t numer_of_new_nodes{ static_cast<size_t>(std::pow(12,limit)) }; //number of total nodes in this limit

    for (size_t i = size - numer_of_new_nodes; i < size; i++)       //creat new nodes
    {
        for (size_t j = 0; j < 12; j++)                             //rotate rubik to crate new nodes
        {
            std::ostringstream oss{};

            if (j / 6 == 0)                                         //rotate clockwise
            {
                rubiks.push_back(rubiks[i].rotate(j % 6, 1, "clockwise")); //save new nodes
                if (flag)                                           //change gole or input rubik
                    oss << "face " << (j % 6) + 1 << "-> anticlockwise";
                else
                    oss << "face " << (j % 6) + 1 << "-> clockwise";
                rubik_rotations.push_back(oss.str());               //save rotations
            }
            if (j / 6 == 1)                                                 //rotate anticlockwise
            {
                rubiks.push_back(rubiks[i].rotate(j % 6, 1, "anticlockwise"));    //save new nodes
                if (flag)                                                   //change gole or input rubik
                    oss << "face " << (j % 6) + 1 << "-> clockwise";
                else
                    oss << "face " << (j % 6) + 1 << "-> anticlockwise";
                rubik_rotations.push_back(oss.str());                       //save rotations
            }
        }
    }
    size = rubiks.size();                                           //save size of nodes
    limit++;
}
```

۴) check_bidirectional : کار این تابع این است که بررسی میشود ایا ما از سمت هدف و ایا از سمت روبیک وارده به یک حالت مشترک رسیده ایم یا نه و اگر رسیده ایم شماره node را برای هر دو طرف برمی‌گرداند.

```cpp
std::array<int, 2> Graph::chek_bidirectional(Graph& Target)  //check we solved rubik or not
{
    std::array<int, 2> find{ -1,-1 };              //to show find solution or not and save node in both side

    for (size_t i = number_of_node(limit - 1); i <= number_of_node(limit); i++)
    {
        for (size_t j = number_of_node(limit - 1); j <= number_of_node(limit); j++)
        {
            if (rubiks[j] == Target.rubiks[i])
            {
                find[0] = static_cast<int>(i);    //save node from gola side
                find[1] = static_cast<int>(j);    //save node from input side
                break;
            }
        }
    }
    return find;
}
```

۵) save_solution_bidirectional : کار این تابع ذخیره کردن جهت چرخش ها تا رسیدن به جواب است که ابتدا چرخش ها از طرف روبیک وارد شده از سمت کاربر ذخیره می‌شود سپس جهت چرخش ها از سمت روبیک هدف ذخیره شده وکل چرخش ها را در رشته solution_rubik_rotations_bidirection ذخیره میکنیم. پیاده سازی آن به صورت شکل صفحه بعد است.

```
void Graph::save_solution_bidirectional(Graph& Target, const std::array<int, 2>& find) //save rotations to how to solve rubik
{
    save_solution(getlimit(), find[1]);    //node input side  and save rotations
    Target.save_solution(getlimit(), find[0]);  //node gole side and save rotations

    for (int i = solution_rubik_rotations.size() - 1; i >= 0; i--)    //save rotations to how to solve rubik
    {
        if (solution_rubik_rotations[i] != "")
            solution_rubik_rotations_bidirectional.push_back(solution_rubik_rotations[i]);
    }
    for (size_t i = 0; i < Target.solution_rubik_rotations.size(); i++)
    {
        solution_rubik_rotations_bidirectional.push_back(Target.solution_rubik_rotations[i]);
    }
}
```

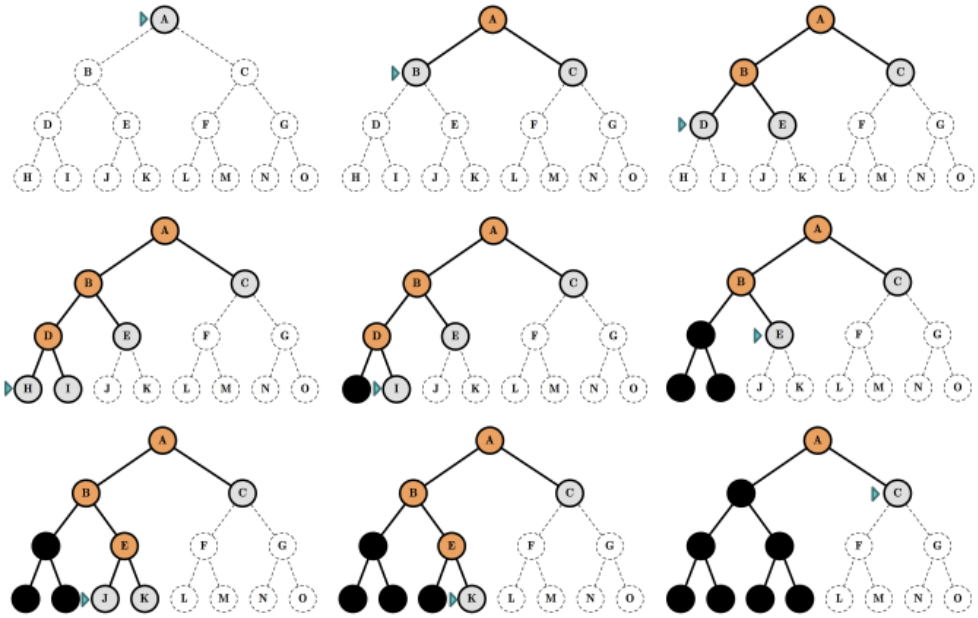
۶)bidirectional : کار این تابع حل روبیک به وسیله روش bidirectional است. الگوریتم آن به این صورت است که ما یکی از سمت هدف شروع به تغییر دادن روبیک میکنیم و یک بار از سمت روبیک وارده شروع به تغییر دادن روبیک میکنیم و این کار را تا جایی ادامه می دهیم که به یک حالت مشترک برسیم.

```
void Graph::bidirectional()          //solve rubiks with bidirectional algorithm
{
    Graph Target{ Rubik { std::vector<size_t>{1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6 } } };  //goal rubik
    std::array<int, 2> find{ -1,-1 };              //to show find solution or not and save node in both side

    while (true)                                  //solve rubiks with BFS algorithm
    {
        Target.new_nodes_bidirectional(true);
        new_nodes_bidirectional(false);
        find = chek_bidirectional(Target);
        if (find[0] >= 0)
            break;
    }
    size = find[1] + 1;                          //number nodes inputs side
    Target.size = find[0] + 1;                   //number node goal side
    save_solution_bidirectional(Target, find); //save rotations to how to solve rubik
}
```

۷)DLS : کار این تابع است که ابگوریتم DLS را به صورت بازگشتی پیاده سازی میکند یعنی هر دفعه به عمق پایین تر رفته تا limit_temp صفر شود و در هر عمق به وسیله تابع new_nodes_dls حالت های جدید تولد می‌شود . وقتی limit_temp=0 چک می‌کندکه آیا به جواب رسیده ایم یا نه اگر نرسیده بودیم به عمق بالاتر رفته و بقیه حالت های آن را چک می‌کنیم . در ابتدا این تابع چک می‌شود که روبیک داده شده به این تابع کامل است یا نه .

```
bool Graph::DLS(size_t li , Rubik input)     //DLS algorithm
{
    if (input.solve_check())     //check rubik is solved or not
        return true;

    if (li == 0)
    {
        bool find_solve{ false };   //to show we find solution or not

        limit_temp++;               //back to upper limit

        if (input.solve_check())   //check rubik is solved or not
            find_solve = true;

        return find_solve;
    }
    else
        return new_nodes_dls(input);
}
```

۸)new_nodes_dls : کار این تابع تولید حالا جدید در هر call تابع DLS است . در تابع وقتی یک حالت جدید تولید شود چرخش آن نیز ذخیره میشود . سپس آن روبیک تولید شده را به ورودی تابع DLS میدهیم و یک عمق پایین تر را بررسی میکنیم. الگوریتم این تابع به صورت شکل زیر است.



الگوریتم DLS با عمق ۳ در شکل بالا قابل مشاهده است.

پیاده سازی آن به صورت زیر است.

```
bool Graph::new_nodes_dls(Rubik A)    //new nodes for each limit in DLS algorithm
{
    size_t numer_of_new_nodes{ 12 };   //number of new nodes
    bool find_solve{ false };    //to show we find solution or not

    for (size_t j = 0; j < numer_of_new_nodes; j++)   //rotate rubik to crate new nodes
    {
        std::ostringstream oss{};

        if (j / 6 == 0)                    //rotate clockwise
        {
            rubiks.push_back(A.rotate(j % 6, 1, "clockwise")); //save new nodes
            oss << "face " << (j % 6) + 1 << "->  clockwise";
            rubik_rotations.push_back(oss.str());  //save rotations
        }
        if (j / 6 == 1)                     //rotate anticlockwise
        {
            rubiks.push_back(A.rotate(j % 6, 1, "anticlockwise"));  //save new nodes
            oss << "face " << (j % 6) + 1 << "-> anticlockwise";
            rubik_rotations.push_back(oss.str());          //save rotations
        }

        find_solve = DLS(--limit_temp, rubiks[rubiks.size() - 1]);  //do down  depth limit

        if (find_solve)
            break;
        else
            rubik_rotations.pop_back(); //if it not our solution so clear this rotation
    }
    size = rubiks.size();  //save size of nodes
    limit_temp++;

    return find_solve;
}
```

حال به سراغ member function هایی می‌رویم که به صورت public تعریف شده اند.

۱)ابتدا constructor را تعریف میکنیم که ورودی آن روبیکی است که کاربر وارد کرده است.

```
Graph::Graph(const Rubik& A)    //constructors
{
    rubiks.push_back(A);        //save input rubik
    size = rubiks.size();       //save size of nodes
    limit = 0;                  //depth limits
}
```

۲)show_solution_BFS : در این تابع ابتدا چک می‌شود که ایا روبیکی که کاربر وارد کرده است کامل است یا نه و اگر نبود سپس الگوریتم BFS را اجرا میکند و عمق را تا جایی زیاد می‌کند که به جواب برسیم . بعد مراحل رسیدن به جواب را به کاربر نشان میدهد . سپس در صورت تمایل کاربر تعداد node های بررسی شده و limit را نمایش می‌دهیم سپس در صورت تمایل کاربر شکل روبیک به صورت رنگی وقتی که به جواب رسیده ایم را نمایش می دهیم.

```
void Graph::show_solution_BFS()                                 //solution with BFS algorithm
{
    char ch;                                                    //for input answer user

    if (rubiks[0].solve_check())                                //chech rubik is solved or not
        std::cout << std::endl << std::endl << "you give me a solved rubik, why? !!" << std::endl;
    else
    {
        while (!new_nodes_BFS()) {};                             //solve rubiks with BFS algorithm
        save_solution(getlimit(), getsize() - 1);               //save rotations to how to solve rubik

        for (int i = solution_rubik_rotations.size() - 1; i >= 0; i--)   //show rotations to how to solve rubik
        {
            std::cout << std::endl << std::endl;
            std::cout << "step " << solution_rubik_rotations.size() - i << " : ";
            std::cout << solution_rubik_rotations[static_cast<size_t>(i)] << std::endl;;
        }

        std::cout << std::endl << std::endl;
        std::cout << "do you want to show limit and numbers of nodes? (y/n): ";

        std::cin >> ch;                         //user answer

        if (ch == 'y')                          //print limit and number of nodes
        {
            std::cout << "depth limit : " << getlimit() << std::endl;
            std::cout << "node : " << getsize() - 1 << std::endl;
        }
        else if (ch == 'n')
            std::cout << "ok" << std::endl;
        else
            throw std::invalid_argument("wrong input");
    }

    std::cout << std::endl << std::endl;
    std::cout << "do you want to show rubik? (y/n): ";

    std::cin >> ch;                         //user answer

    if (ch == 'y')                          //show rubik
        rubiks[size - 1].show_rubik_color();
    else if (ch == 'n')
        std::cout << "ok" << std::endl;
    else
        throw std::invalid_argument("wrong input");
}
```

۳)show_solution_bidirectional : در این تابع ابتدا چک می‌شود که ایا روبیکی که کاربر وارد کرده است کامل است یا نه و اگر نبود سپس تابع bidirectional را اجرا میکند و بعد از اجرای این تابع ؛ مراحل رسیدن به جواب را به کاربر نشان میدهد . سپس در صورت تمایل کاربر limit را نمایش می‌دهیم سپس در صورت تمایل کاربر شکل روبیک به صورت رنگی وقتی که به جواب رسیده ایم را نمایش می دهیم.

```
void Graph::show_solution_bidirectional()    //solution with bidirectional algorithm
{
    char ch;
    if (rubiks[0].solve_check())                  //check rubik is solved or not
        std::cout << std::endl << std::endl << "you give me a solved rubik, why? !!" << std::endl;
    else
    {
        bidirectional();                      //solve rubiks with bidirectional algorithm

        for (size_t i = 0; i < solution_rubik_rotations_bidirectional.size(); i++)  //show rotations to how to solve rubik
        {
            std::cout << std::endl << std::endl;
            std::cout << "step " << i + 1 << " : ";
            std::cout << solution_rubik_rotations_bidirectional[i] << std::endl;;
        }

        std::cout << std::endl << std::endl;
        std::cout << "do you want to show limit ? (y/n): ";

        std::cin >> ch;     //user answer

        if (ch == 'y')     //show depth limit
        {
            std::cout << "depth limit : " << getlimit() << std::endl;
        }
        else if (ch == 'n')
            std::cout << "ok" << std::endl;
        else
            throw std::invalid_argument("wrong input");

    }

    std::cout << std::endl << std::endl;
    std::cout << "do you want to show rubik? (y/n): ";
    std::cin >> ch;       //user answer

    Rubik gole{ std::vector<size_t>{1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6 } };

    if (ch == 'y')   //show rubik goal
        gole.show_rubik_color();
    else if (ch == 'n')
        std::cout << "ok" << std::endl;
    else
        throw std::invalid_argument("wrong input");
}
```

۴) show_solution_DLS : در این تابع ابتدا پرسیده می‌شود که تا چه عمقی پیش برویم و در صورت عدم تمایل کاربر برای ورود ؛عمق ۸ را در نظر می‌گیریم. سپس برای یافتن جواب تابع DLS را اجرا می‌کنیم و به عنوان ورودی به آن روبیکی که کاربر وارد کرده و عمق را به آن می‌دهیم. بعد مراحل رسیدن به جواب را به کاربر نشان میدهد . سپس در صورت تمایل کاربر تعداد node های بررسی شده و limit را نمایش می‌دهیم سپس در صورت تمایل کاربر شکل روبیک به صورت رنگی وقتی که به جواب رسیده ایم را نمایش می دهیم. پیاده سازی آن به صورت شکل صفحه بعد است.

```
void Graph::show_solution_DLS()        //solution with IDS algorithm
{
    char ch;        //user answer
    std::cout << "do you want enter depth limit (y/n) :";
    std::cin >> ch;
    if (ch == 'y')  //get limit from user
    {
        std::cout << "please enter depth limit search : ";
        std::cin >> limit;
    }
    else if(ch=='n')
        limit = 8;                          //depth limit search
    else
        throw std::invalid_argument("wrong input");

    limit_temp = limit;
    rubik_rotations = {};
    if (DLS(limit, rubiks[0]))     //solve rubiks with DLS algorithm
    {
        for (size_t i{}; i < rubik_rotations.size(); i++)  //show rotations to how to solve rubik
        {
            std::cout << std::endl << std::endl;
            std::cout << "step " << i + 1 << " : ";
            std::cout << rubik_rotations[i] << std::endl;;
        }


        std::cout << std::endl << std::endl;
        std::cout << "do you want to show limit and numbers of nodes? (y/n): ";

        std::cin >> ch;                         //user answer

        if (ch == 'y')                          //print limit and number of nodes
        {
            std::cout << "depth limit : " << getlimit() << std::endl;
            std::cout << "node : " << getsize() - 1 << std::endl;
        }
        else if (ch == 'n')
            std::cout << "ok" << std::endl;
        else
            throw std::invalid_argument("wrong input");
```

```
        std::cout << std::endl << std::endl;
        std::cout << "do you want to show rubik? (y/n): ";

        std::cin >> ch;                             //user answer

        if (ch == 'y')                              //show rubik
            rubiks[size - 1].show_rubik_color();
        else if (ch == 'n')
            std::cout << "ok" << std::endl;
        else
            throw std::invalid_argument("wrong input");


    }
    else
        std::cout << "can not find solutin in depth limit " << limit << std::endl;
}
```

۵) show_solution_IDS : در این تابع ابتدا پرسیده می‌شود که تا چه حداکپرعمقی پیش برویم و در صورت عدم تمایل کاربر برای ورود عمق؛ ۱۰ را در نظر می‌گیریم.سپس از عمق صفر شروع میکنیم و برای هر عمق با روش DLS پیش میرویم اگر جواب را پیدا نکردیم از اول دوباره همه چی را پاک کرده و به عمق را یکی زیاد میکنیم و این کار را تا جایی ادامه می‌دهیم که به جواب برسیم یا به حداکثر عمق جستجو برسیم و سپس در صورت یافتن پاسخ مراحل رسیدن به جواب را به کاربر نشان میدهد . سپس

در صورت تمایل کاربر تعداد node های بررسی شده و limit را نمایش می‌دهیم سپس در صورت تمایل کاربر شکل روبیک به صورت رنگی وقتی که به جواب رسیده ایم را نمایش می دهیم. پیاده سازی آن به صورت زیر است.

```
void Graph::show_solution_IDS()      //solution with IDS algorithm
{
    char ch;        //user answer
    bool flag{ true };  //show we find solution or not

    size_t max_limit{};
    std::cout << "do you want enter maximum depth limit (y/n) :";
    std::cin >> ch;
    if (ch == 'y')   //get max limit from user
    {
        std::cout << "please enter maximum depth limit search : ";
        std::cin >> max_limit;
    }
    else if (ch == 'n')
        max_limit = 10;                         //max depth limit search
    else
        throw std::invalid_argument("wrong input");

    while (true)    //solve rubiks with IDS algorithm
    {                           //start new graph each depth limit
        Rubik input = rubiks[0];   //save input rubik
        rubiks = {};
        rubik_rotations = {};
        limit_temp = limit;
        rubiks.push_back(input);
        if (DLS(limit, rubiks[0]))  //chech we find solution or not in DLS algorithm
            break;
        limit++;           //increasing limit
        if (limit > max_limit)
        {
            std::cout << "can not find solutin in depth limit " << max_limit << std::endl;
            flag = false;
            break;
        }
    }
```

```
if (true)   //if we find solution
{
    for (size_t i{}; i < rubik_rotations.size(); i++)  //show rotations to how to solve rubik
    {
        std::cout << std::endl << std::endl;
        std::cout << "step " << i + 1 << " : ";
        std::cout << rubik_rotations[i] << std::endl;;
    }

    char ch;        //user answer
    std::cout << std::endl << std::endl;
    std::cout << "do you want to show limit and numbers of nodes? (y/n): ";

    std::cin >> ch;                         //user answer

    if (ch == 'y')                          //print limit and number of nodes
    {
        std::cout << "depth limit : " << getlimit() << std::endl;
        std::cout << "node : " << getsize() - 1 << std::endl;
    }
    else if (ch == 'n')
        std::cout << "ok" << std::endl;
    else
        throw std::invalid_argument("wrong input");


    std::cout << std::endl << std::endl;
    std::cout << "do you want to show rubik? (y/n): ";

    std::cin >> ch;                           //user answer

    if (ch == 'y')                            //show rubik
        rubiks[size - 1].show_rubik_color();
    else if (ch == 'n')
        std::cout << "ok" << std::endl;
    else
        throw std::invalid_argument("wrong input");
}
```

۶)getlimit : عمق را برمیگرداند.

۷)getsize : تعدادکل node های تولید شده را برمی‌گرداند.

```
size_t Graph::getlimit()            //get depth limit
{
    return limit;
}

size_t Graph::getsize()             //get size of nodes
{
    return size;
}
```

۸)operator[] : روبیک در node وارده را برمی‌گرداند.

```
Rubik Graph::operator[](const size_t& i)   //operator []
{
    return rubiks[i];
}
```

بعد از اتمام این class در main.cpp تابع solve_rubik را تعریف میکنیم. که این تابع شروع به معرفی کلی برنامه و نحوه وارد کردن روبیک میکند سپس روبیک را ازکاربرگرفته(به وسیله توابعی که در constructor کلاس rubikاست انجام می‌شود) سپس ازکاربر شیوه جستجو را میخواهیم وارد کند سپس در آخر با توجه به شیوه جستجوکاربر مراحل حل کردن روبیک چاپ می‌شود و سوالات مربوطه که در class graph مطرح کردیم پرسیده می‌شود . در آخر هم ازکاربر پرسیده می‌شود که میخواهد یک روبیک دیگر را حل کند یا نه و اگرگفت بله کل مراحل از اول تکرار می‌شود.

```
void solve_rubik();

int main()
{
    solve_rubik();

    system("pause");
    return 0;
}

void solve_rubik()
{
    try
    {
        while (true)
        {
            Rubik input_rubik{};
            Graph solution{ input_rubik };
            std::string  user_answer;
            char ch;

            std::cout << "Search method (BFS/DLS/IDS/BI) : ";
            std::cin >> user_answer;
            if (user_answer == "BI")
                solution.show_solution_bidirectional();
            else if (user_answer == "BFS")
                solution.show_solution_BFS();
            else if (user_answer == "IDS")
                solution.show_solution_IDS();
            else if (user_answer == "DLS")
                solution.show_solution_DLS();
            else
                throw std::invalid_argument("input search method wrong");

            std::cout << std::endl << std::endl;
            std::cout << "do you want to try another Rubik? (y/n): ";
            std::cin >> ch;
            if (ch == 'y')
                continue;
            else if (ch == 'n')
                break;
            else
                throw std::invalid_argument("wrong input");
        }
        std::cout << std::endl << std::endl << "ok,see you!" << std::endl;
    }
    catch (std::invalid_argument& e)
    {
        std::cerr << std::endl << e.what() << std::endl;
    }
}
```

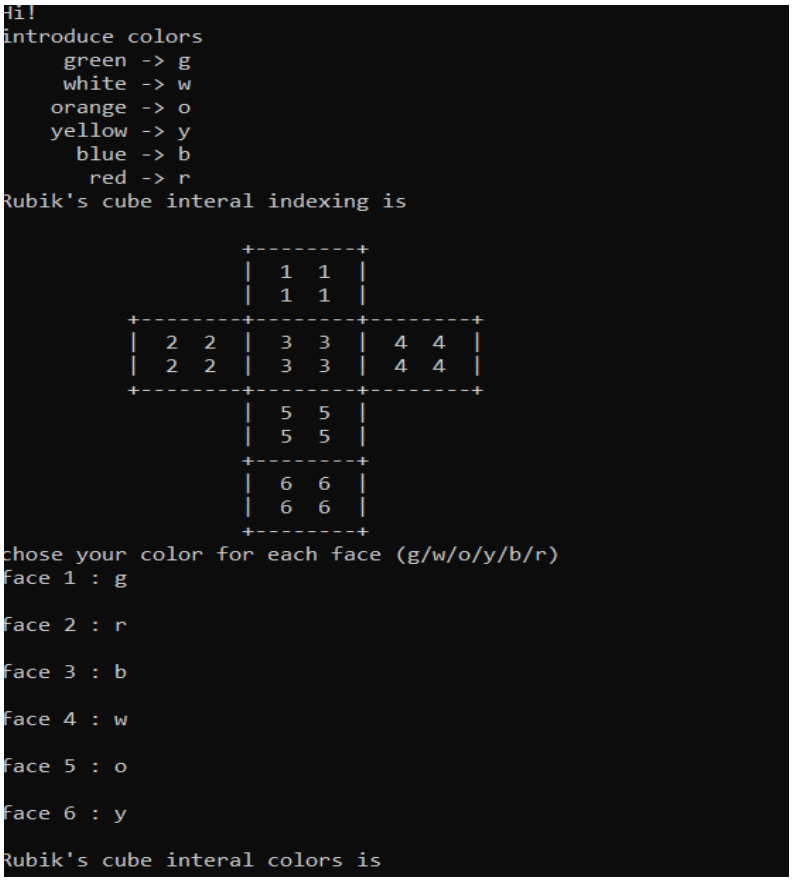
توجه۱ : در ورودی بعضی از تابع ها ؛ دلیل اینکه ورودی را به صورت refernce می‌دهیم این است که باعث افزایش performance و صرفه‌جویی در حافظه می‌شود. و در برخی موارد ورودی را به صورت const هم می‌دهیم که این کار برای جلوگیری از اجازه تغییر در تابع است.

توجه۲: توضیحات مربوط به پیاده سازی الگوریتم ها در داخل کد به صورت کامنت نوشته شده است و در این گزارش کار سعی شده در مورد الگوریتم توابع توضیح داده شود.

توجه ۳: سعی شده تمام جاهایی که در برنامه احتمال بروز خطا است شناسایی و با یک پیغام مناسب کاربر را از بروز خطا آگاه کنیم.(توابع متعدد شناسایی خطا در برنامه همانطورکه گفته شده وجود دارد)

توجه ۴: توضیحات بیشتر راجب الگوریتم های حل مسئله در پاورپوینت موجود است.

توجه۵: این پروژه همانطورکه در تصاویر مشخص است ابتدا در visual studio 2019 نوشته و سپس در visual studio code به وسیله docker اجرا شده و جواب ها موفقیت آمیزه بوده و در صفحات بعدی موجود است.



```
Rubik's cube interal colors is

                +--------+
                |  g  g  |
                |  g  g  |
       +--------+--------+--------+
       |  w  w  |  o  o  |  y  y  |
       |  w  w  |  o  o  |  y  y  |
       +--------+--------+--------+
                |  b  b  |
                |  b  b  |
                +--------+
                |  r  r  |
                |  r  r  |
                +--------+

we have four search methods here
Breadth-first search -> BFS
Depth-limited search -> DLS
Iterative-deepening search -> IDS
Bidirectional -> BI

thanks for your attention
programed by Mohammad Javad Amin
spring 2020


for enter elements for each face put space between them and end of each face press enter and go to next face
enter your numbers in range 1 to 6
face 1 :
5 6 3 4
face 2 :
2 1 5 4
face 3 :
2 6 3 6
face 4 :
1 5 1 4
face 5 :
1 2 3 5
face 6 :
4 6 3 2
Search method (BFS/DLS/IDS/BI) : DLS
```

```
step 1 : face 2-> anticlockwise


step 2 : face 3-> anticlockwise


step 3 : face 1-> anticlockwise


step 4 : face 2-> anticlockwise


step 5 : face 1-> anticlockwise


do you want to show limit and numbers of nodes? (y/n): y
depth limit : 5
node : 174471


do you want to show rubik? (y/n): y
Rubik's cube interal colors is

                +--------+
                |  g  g  |
                |  g  g  |
       +--------+--------+--------+
       |  w  w  |  o  o  |  y  y  |
       |  w  w  |  o  o  |  y  y  |
       +--------+--------+--------+
                |  b  b  |
                |  b  b  |
                +--------+
                |  r  r  |
                |  r  r  |
                +--------+


do you want to try another Rubik? (y/n): j

wrong input
sh: 1: pause: not found
root@fa2789514c45:/usr/src/app# exit
logout
```

توجه ۶: از هرگونه لغو زبان و سایر اشتباهات دستوری درگزارش کار وکامنت ها عذر خواهی می‌شود و قطعا این برنامه وگزارش کار خالی از اشتباه نیست.

تمام