

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TP HCM

Khoa công nghệ thông tin



BÁO CÁO ĐỒ ÁN ALGORITHMS

Sinh viên thực hiện

: Võ Minh Tân

18120552

Nguyễn Quốc Thái

18120554

Lớp : 18CTT5A

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TP HCM

Khoa công nghệ thông tin



NỘI DUNG : Xây dựng chương trình hỗ trợ học tập toàn rời rạc và đại số tuyến tính.

Giảng viên hướng dẫn : Nguyễn Thành An

MỤC LỤC

PHẦN 1 : Hướng dẫn sử dụng và chạy chương trình.....	4
PHẦN 2 : Phân chia công việc.....	9
PHẦN 3 : Giải thích phương thức , thuộc tính.....	10
3.1 Công thức đa thức tối thiểu hàm bool.....	10
+Sơ đồ lớp.....	10
+Giải thích phương thức và thuộc tính.....	12
+Tìm tế bào lớn.....	16
+Tìm công thức đa thức tối thiểu.....	19
+Lưu đồ thuật toán.....	21
3.2 Các phép toán trên vector.....	24
+Sơ đồ lớp.....	24
+Giải thích phương thức và thuộc tính.....	25
+Cộng hai vector.....	29
+Nhân một vector với một số alpha.....	29
+Lưu đồ thuật toán.....	31
3.3 Các phép toán trên ma trận.....	33
+Sơ đồ lớp.....	33
+Giải thích các phương thức và thuộc tính.....	34
+Tính định thức.....	35
+Tìm ma trận phụ hợp.....	37
+Tìm ma trận nghịch đảo.....	40
+Nhân hai ma trận.....	41
+Tìm hạng của ma trận.....	43
+Tìm nghiệm của hệ phương trình tuyến tính.....	46
+Lưu đồ thuật toán : cuối mỗi bài đều có một lưu đồ.	

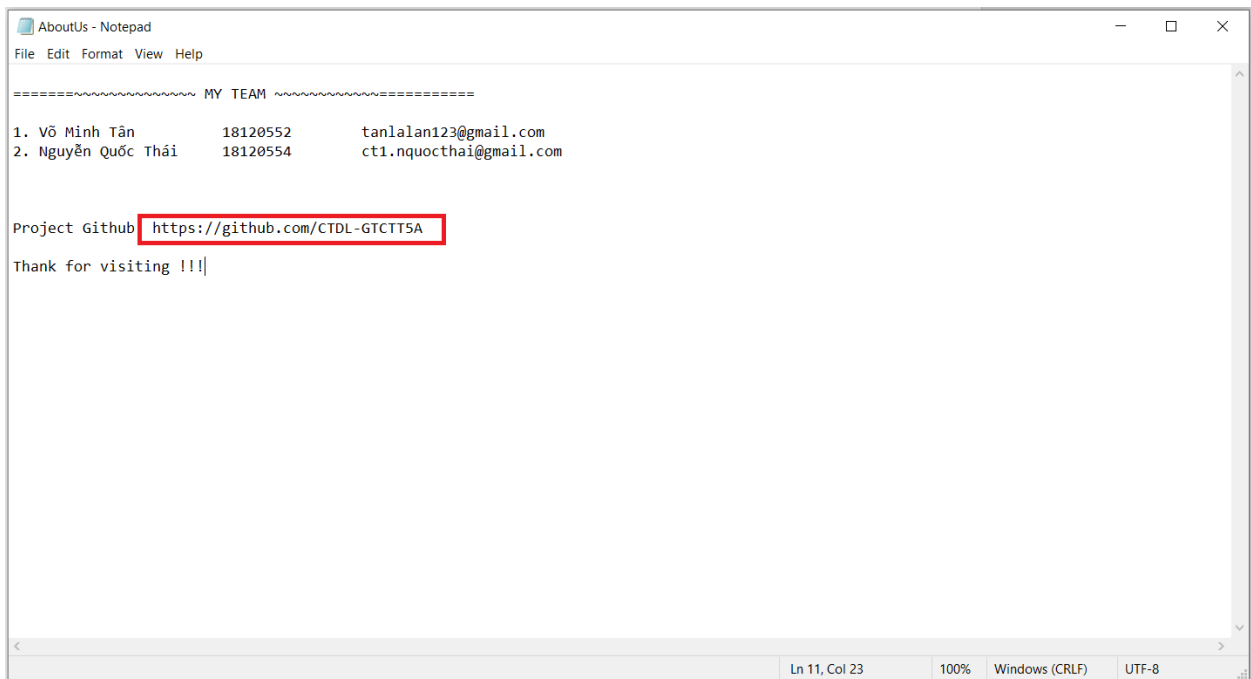
PHẦN 1 : HƯỚNG DẪN SỬ DỤNG VÀ CHẠY CHƯƠNG TRÌNH

A.Hướng dẫn sử dụng :

Video :

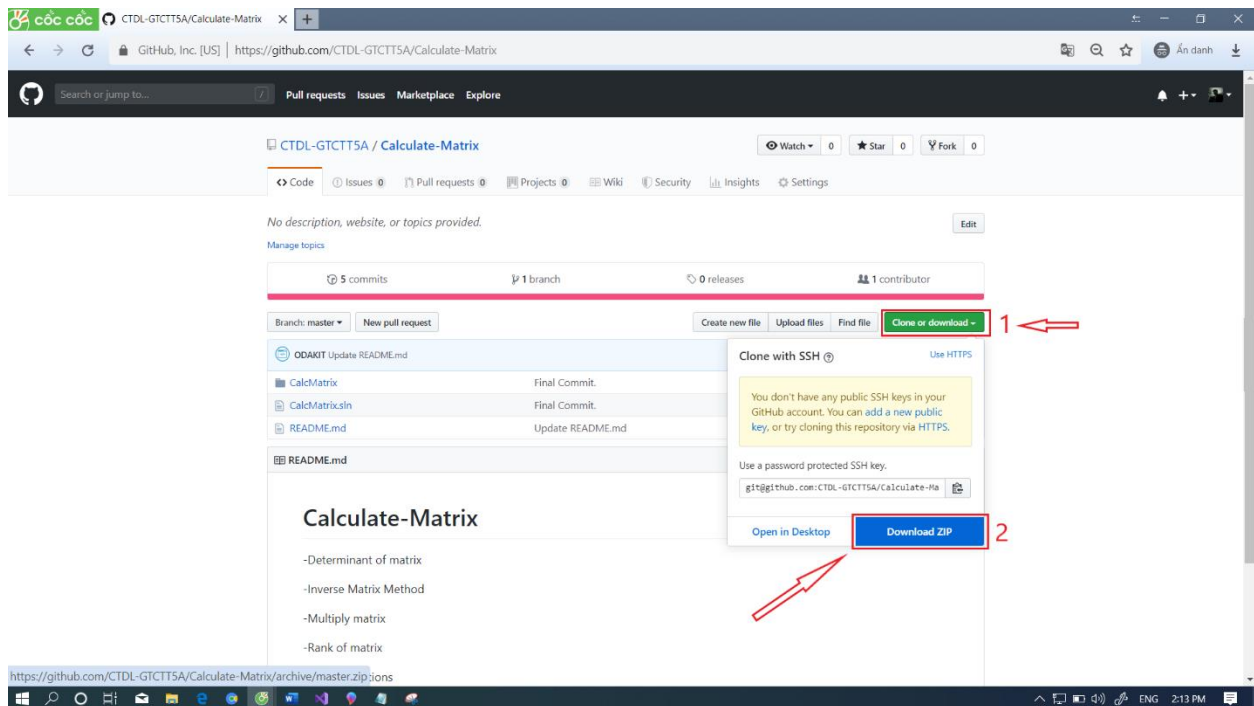
B.Hướng dẫn chạy chương trình:

Step 1: truy cập vào link Github đã được ghi trong tệp tin AboutUs nộp kèm theo qua hệ thống Moodle FIT-HCMUS hoặc truy cập [tại đây](https://github.com/CTDL-GTCTT5A).

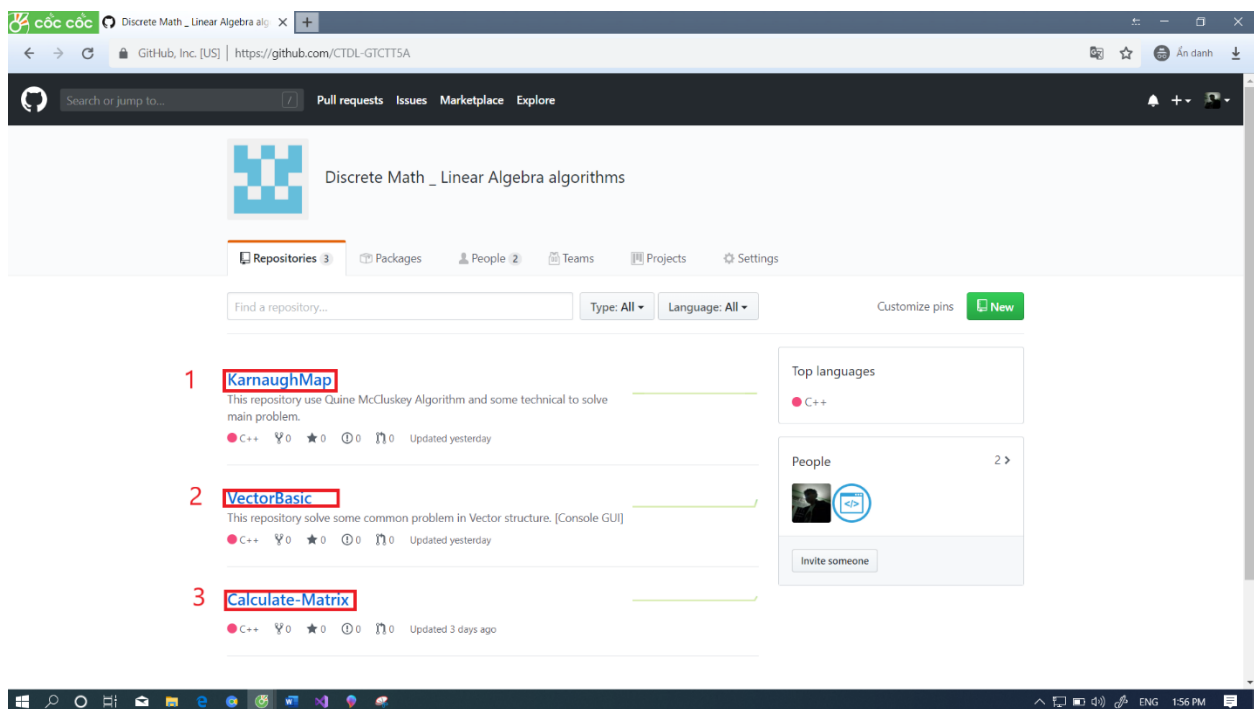


```
=====  
MY TEAM  
=====  
1. Võ Minh Tân      18120552      tanlalan123@gmail.com  
2. Nguyễn Quốc Thái 18120554      ct1.nquocthai@gmail.com  
  
Project Github https://github.com/CTDL-GTCTT5A  
Thank for visiting !!!
```

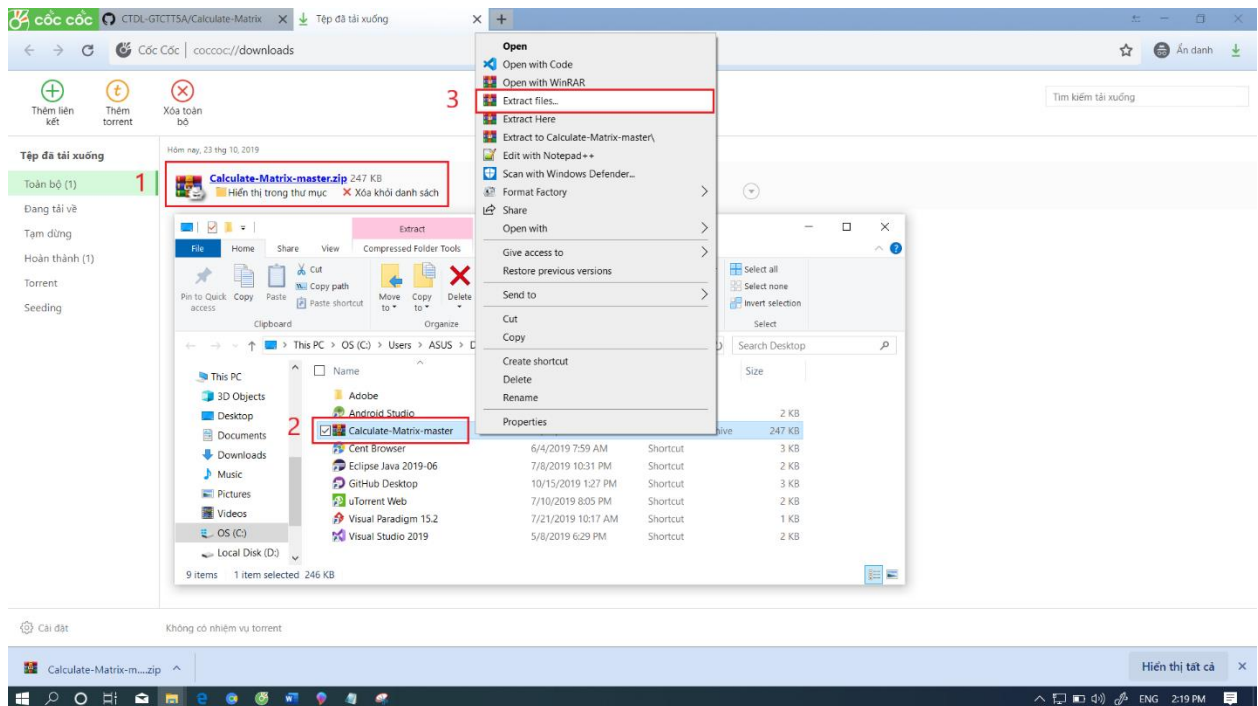
Step 2: Lựa chọn project muốn download. Sau đó click chọn nó.



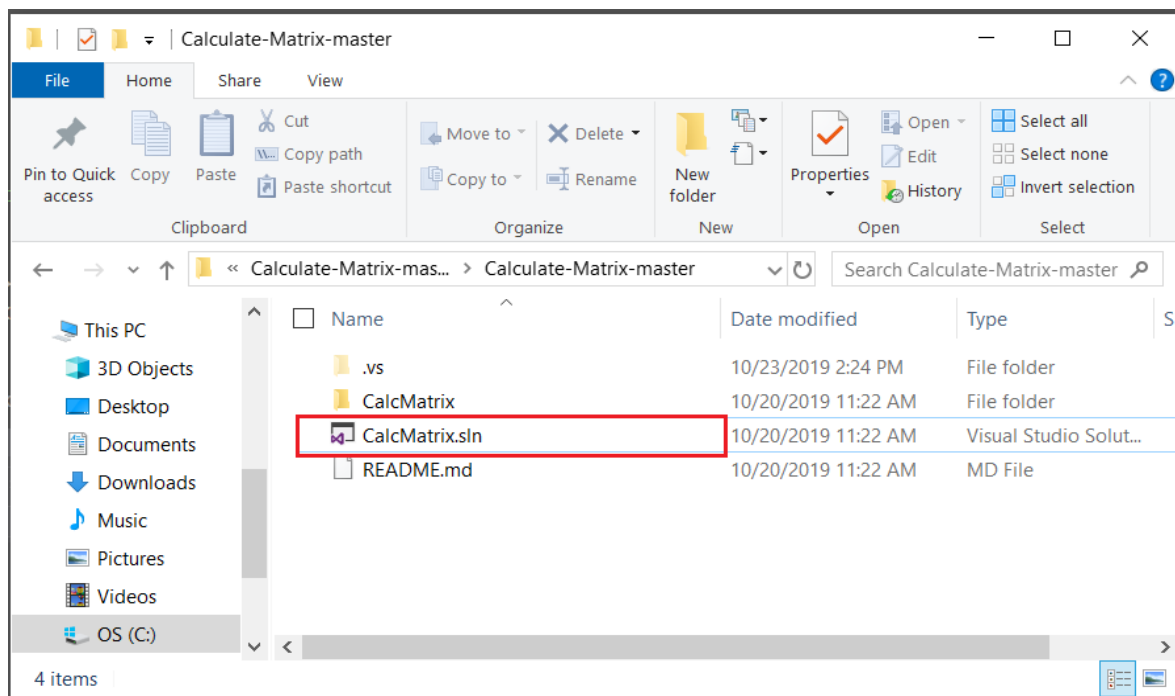
Step 3: Click Clone or Dowload và chọn dowload ZIP.



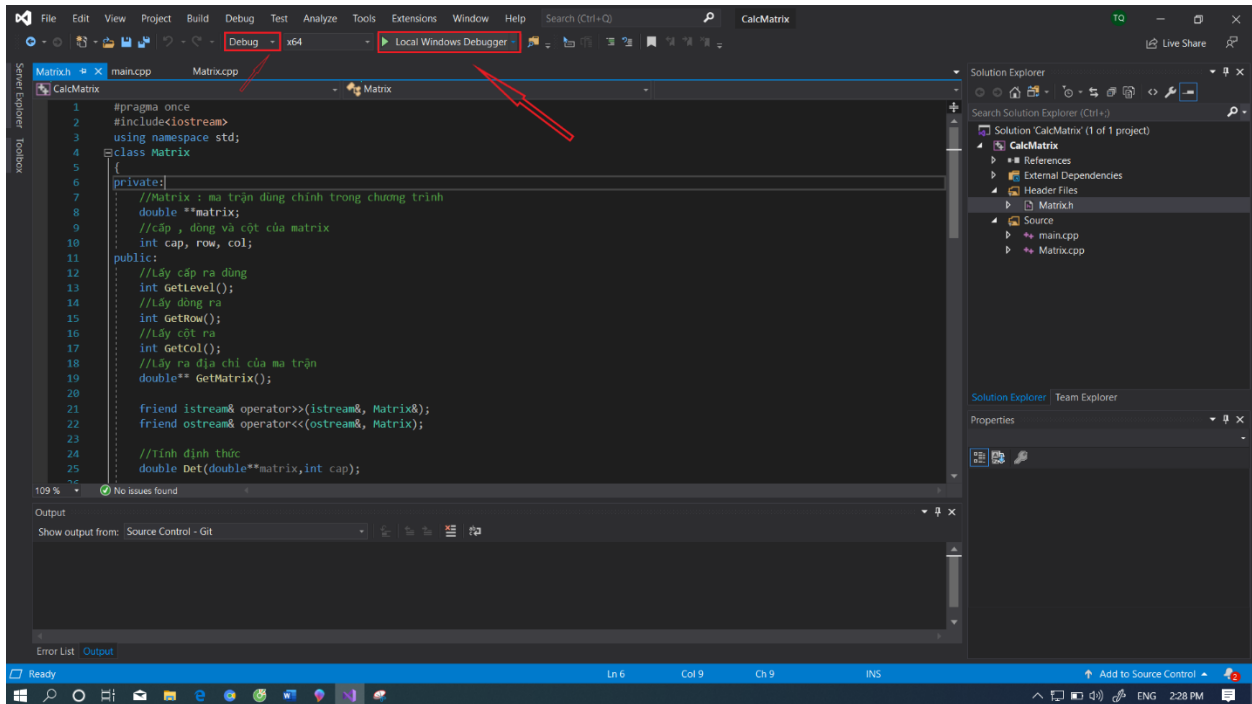
Step 4: Tìm mục vừa tải về và giải nén ra thư mục.



Step 5: Vào thư mục vừa giải nén và chạy file Solution của Project



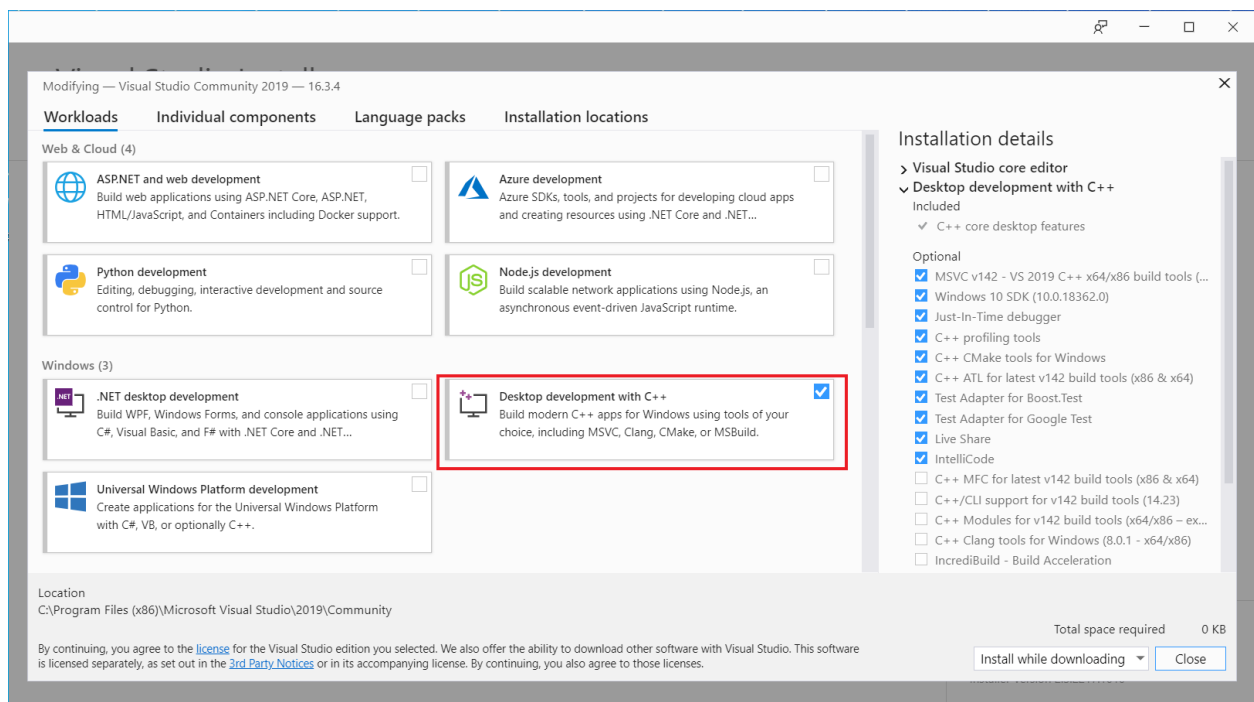
Step 6: Chọn chế độ Build ở dạng Debug hoặc Release. Tiếp đó click chọn Local Windows Debugger để chạy chương trình.



*LƯU Ý:

- Để chạy máy tính build được code C++ bạn cần phải cài đặt một IDE để biên dịch code. Recommend: [Visual Studio](#), [Visual Studio Code](#), [DevC++](#),...
- Đối với Visual Studio Code bạn cần phải Install thêm các Plug-in để có thể Compile thành công.
- Đối với Visual Studio sau khi cài đặt thành công bạn phải cần phải cài đặt thêm môi trường để có thể khởi chạy chương trình :

Cách cài đặt môi trường: Mở Visual Studio Installer:



Ở đây mình đã cài rồi, còn nếu bạn chưa cài thì chỉ việc click vào ô check box rồi ấn Modify và chờ đợi quá trình hoàn tất.

Đây là tài liệu hướng dẫn Dowload và chạy chương trình. Chi tiết việc sử dụng chương trình sẽ được giới thiệu qua Video mô tả.

PHẦN 2 : NỘI DUNG CÔNG VIỆC VÀ KẾT QUẢ ĐẠT ĐƯỢC

Mỗi màu tương ứng với một bài trong đề án.

CÔNG VIỆC	NGƯỜI THỰC HIỆN	ĐÁNH GIÁ
TÌM TẤT CẢ CÁC CÔNG THỨC ĐA THỨC TỐI TIỂU CỦA HÀM BOOL	NGUYỄN QUỐC THÁI	100%
PHÉP TOÁN CỘNG 2 VECTOR	NGUYỄN QUỐC THÁI	100%
PHÉP TOÁN NHÂN VECTOR VỚI MỘT SỐ ALPHA	NGUYỄN QUỐC THÁI	100%
TÌM ĐỊNH THỨC CỦA MA TRẬN	VÕ MINH TÂN	100%
TÌM NGHỊCH ĐẢO CỦA MA TRẬN	VÕ MINH TÂN	100%
TÍCH CỦA HAI MA TRẬN	VÕ MINH TÂN	100%
TÌM HẠNG CỦA MA TRẬN	VÕ MINH TÂN	100%
HỆ PHƯƠNG TRÌNH TUYẾN TÍNH	VÕ MINH TÂN	100%

PHẦN 3 : GIẢI THÍCH PHƯƠNG THỨC VÀ THUỘC TÍNH

MÔ TẢ CHỨC NĂNG CỦA CÁC THÀNH PHẦN TRONG PROJECT KARNAUGH-MAP

Sơ đồ lớp PROJECT KARNAUG



QMCluskey

// Attribute:

```
- num_var: int
- num_cell : int
- arr_n_dim_vector : *double

+ vector<int> input_arr
+ vector<string> nepi;
+ vector<string> pi
+ vector<string> epi;
+ vector<string> answer

+ int diff_bit(string a, string b, int bit);
+ vector<string> duplication(vector<string>& a, vector<string>& b);
+ void combine(vector<string>& combine_mt, vector<string>
               binary_mt, int bitnum);
+ bool isChecked(string pi, string binary, int bit);
```

// Method:

```
+ void Input();
+ void QM_Method();
+ void Minimizer();
```

// Constructor:

```
+ McCluskeyAlgorithm(int num_var);
```

// Destructor:

```
+ ~McCluskeyAlgorithm();
```

File MapGuild.h [Phục vụ việc khai báo Class]

1. Thuộc tính:

```
private:
    // Lấy hàng và cột tương ứng với số lượng biến Karr đầu vào
    int c_row;
    int c_col;

    // Số biến của biểu đồ Karr
    int num_var;

    // Map guild
    int c_map[MAX][MAX];
```

Gồm các thuộc tính:

- + **num_var**: số lượng biến của biểu đồ Kar. Trong chương trình này ta chỉ xét tới số lượng num_var từ 2 đến 6. Tất nhiên ta có thể mở rộng nó đến n nhưng do nhu cầu của người dùng cơ bản chưa cần thiết tới mức đó.
- + **c_row**: số hàng của một map khi có số lượng num_var tương ứng.
- + **c_col**: số cột của một map khi có số lượng num_var tương ứng.

2. Phương thức:

```
public:
    // Init map ban đầu:
    void set_Map();

    // Constructor:
    MapGuild(int num_var);

    // Method:
    void show();           // Showmap
    void update(int in);   // Update mỗi lần có giá trị nhập vào
```

- + **void set_map()**: thiết lập một map_guild hướng dẫn người dùng nhập liệu bằng cách đánh số cho các ô từ 0 đến $(2^{\text{num_var}} - 1)$.
- + **void show()**: hiển thị map ra cho người dùng dễ thao tác.
- + **void update(int in)**: hàm này nhận một giá trị đầu vào là số ô mà người dùng nhập vào sau đó dò trong map_guild và đổi giá trị đó thành -1. Ghi nhận quá trình nhập liệu. Hạn chế thiếu sót đầu vào.

File MapGuild.cpp [Phục vụ định nghĩa các phương thức khai báo Class]

1. Constructor:

```
// Constructor:
MapGuild::MapGuild(int num_var)
{
    this->num_var = num_var;

    switch (num_var)
    {
    case 2: {
        this->c_row = 2;
        this->c_col = 2;

        break;
    }
    case 3: {
        this->c_row = 2;
        this->c_col = 4;

        break;
    }
    }
```

Ứng với số lượng biến num_var đầu vào khi truyền tham số thì ta sẽ gán các dòng và cột của biểu đồ theo giá trị tương ứng.

2. Set_map():

```
/*
Set map Init ban đầu ứng với số lượng biến của biểu đồ Karr
Đầu ra là một matrix đánh số
*/
void MapGuild::set_Map()
{
    switch (this->num_var)
    {
    case 2: {
        c_map[0][0] = 0;    c_map[0][1] = 1;    c_map[1][0] = 2;    c_map[1][1] = 3;
        break;
    }
    case 3: {
        c_map[0][0] = 0;    c_map[0][1] = 1;    c_map[0][2] = 3;    c_map[0][3] = 2;
        c_map[1][0] = 4;    c_map[1][1] = 5;    c_map[1][2] = 7;    c_map[1][3] = 6;
        break;
    }
    }
```

Gán mặc định giá trị các ô cho biểu đồ phục vụ việc nhập và lưu dữ liệu.

3. Show():

```
/*
Đầu ra là matrix với các ô đã được chọn sẽ được gán thành giá trị -1
*/
void MapGuild::show()
{
    this->num_var = num_var;

    switch (num_var)
    {
    case 2: {
        string emt[2] = { " 0", " 1" };

        cout << setw(3) << " 0" << " ";
        cout << setw(3) << " 1" << " " << endl;
        cout << "_____" << endl;

        for (size_t i = 0; i < this->c_row; i++)
        {
            for (size_t j = 0; j < this->c_col; j++)
            {
                cout << setw(3) << this->c_map[i][j] << " |";
            }
            cout << " " << emt[i];
            cout << endl;
        }

        break;
    }
}
```

Thiết lập căn chỉnh và xuất các giá trị đã được gán của map.

4. Update():

```
/*
Với giá trị in người dùng đặt vào. Ta sẽ thay ô nào trong matrix có giá trị in thành -1
*/
void MapGuild::update(int in)
{
    for (size_t i = 0; i < this->c_row; i++)
    {
        for (size_t j = 0; j < this->c_col; j++)
        {
            if (this->c_map[i][j] == in) {
                this->c_map[i][j] = -1;
                break;
            }
        }
    }
}
```

Với giá trị ở ô tương ứng người dùng nhập vào thì ta sẽ tiến hành thay thế nó thành giá trị -1.

File McluskeyAlgorithm.h [Phục vụ việc khai báo Class]

1. Thuộc tính:

```
private:
// Các mảng và phương thức private phục vụ việc tìm tế bào lớn và rút gọn Minimizer
int num_var;           // Số biến của hàm bool
int num_cell;          // Số ô đã tô của biểu đồ Kar

vector<int> input_arr;  // Mảng đầu vào của người dùng
vector<string> nepi;    // Lưu tế bào lớn [Lọc lấy những tế bào khác nhau trong mảng pi và epi]
vector<string> pi;      // Mảng lưu các tế bào sau khi đã ghép cặp
vector<string> epi;     // Lưu tế bào nhỏ (1 ô) hoặc các tế bào không được combine. Dùng để kiểm tra xem nó đã nằm trong tế bào lớn đã ghép cặp hay chưa.
vector<string> answer;  // Mảng kết quả các tế bào lớn cuối cùng
```

2. Phương thức nội bộ Class:

```
int diff_bit(string a, string b, int bit);
vector<string> duplication(vector<string>& a, vector<string>& b);
void combine(vector<string>& combine_mt, vector<string>& binary_mt, int bitnum);
bool isChecked(string pi, string binary, int bit);
```

- + int diff_bit(): Đếm số lượng khác nhau của 2 chuỗi bit
- + vector duplication() : Loại bỏ các phần tử trùng trong mảng a (có trong mảng b)
- + void combine(): Kết hợp 2 dãy bit chỉ khác nhau 1 kí tự.
- + void isChecked(): Kiểm tra 2 dãy bit không thể kết hợp xem khối này có nằm trong khối kia hay không.

3. Phương thức:

```
public:
// Constructor:
McCluskeyAlgorithm(int num_var);

// Destructor:
~McCluskeyAlgorithm();

void Input();           // Nhập liệu từ người dùng
void QM_Method();       // Phương thức xử lý chính: lấy tất cả các tế bào lớn của biểu đồ Kar
void Minimizer();       // Phương thức rút gọn tế bào lớn lấy công thức đa thức tối thiểu
```

- + Constructor khởi tạo với tham số đầu vào là số lượng biến của biểu đồ Kar.
- + void input(): Phương thức nhập liệu từ người dùng.
- + void QM_Method(): Xử lý ghép cặp và trả về mảng các tế bào lớn của biểu đồ Kar.
 - + void Minimizer(): Từ mảng các tế bào lớn trả về ta tiến hành xử lý để nhận về công thức đa thức tối thiểu.

File MccluskeyAlgorithm.cpp [Phục vụ định nghĩa các phương thức khai báo Class]

1. Phương thức MC_Method(): Tìm tế bào lớn:

```
// minterm = origin_minterm

for (int i = 0; i < num_of_mt; i++) // Duyệt mảng minterm (có num_of_mt phần tử)
{
    int origin_minterm = minterm[i];

    while (minterm[i] > 1) // lấy mã nhị phân của các ô mà người dùng nhập vào
    {
        if (minterm[i] % 2 == 1)
            s += '1';
        else
            s += '0';
        minterm[i] /= 2;
    }

    if (origin_minterm == 0)
        s += '0';
    else
        s += '1';

    if (s.size() != bitnum) // Thêm các bit 0 để độ dài chuỗi nhị phân bằng với số lượng biến của hàm bool
    {
        while (bitnum != s.size())
        {
            s += '0';
        }
    }
    reverse(s.begin(), s.end()); // Đảo ngược chuỗi nhị phân và đưa vào mảng binary_minterm
    binary_minterm.push_back(s);
    s.clear();
}
```

Đầu tiên ta chuyển các giá trị đầu vào của người dùng thành mã nhị phân và lưu vào mảng `binary_minterm`. Đồng thời hiệu chỉnh chiều dài của mỗi chuỗi cho đồng bộ và bằng số lượng biến `num_variable`.

```
for (int i = 0; i < bitnum; i++)
{
    // Tiến hành ghép cặp và rút gọn:
    // Giả sử sau vòng for đầu (i=2) ta có 2 giá trị -001 và -101 thì vòng for tiếp theo 2 giá trị này có thể gặp nhau và ghép lại thành --01
    // Tiến hành ghép như thế để tạo tế bào lớn

    combine(combine_minterm, binary_minterm, bitnum);
}
```

Tiếp theo ta duyệt for `i` từ 0 đến chiều dài của `num_variable`. Ta tiến hành ghép cặp các chuỗi bit có `i` bit 1 và cập nhật vào mảng `combine_minterm`.

***Trong đó phương thức `combine` được định nghĩa như sau:**

```
void MccluskeyAlgorithm::combine(vector<string>& combine_mt, vector<string>& binary_mt, int bitnum){};
```



```

// Duyệt mảng và tiến hành ghép cặp
for (int i = 0; i < size - 1; i++)
{
    for (int j = i + 1; j < size; j++)
    {
        // Nếu 2 chuỗi chỉ khác nhau đúng 1 bit và bit đó ở vị trí tương ứng nhau thì thay vị trí khác đó thành dấu - và nhét chuỗi bit đó vào mảng temp_tm
        if (diff_bit(combine_mt[i], combine_mt[j], bitnum) == 1)
        {
            for (int k = 0; k < bitnum; k++)
            {
                if (combine_mt[i][k] != combine_mt[j][k])
                {
                    // Đánh dấu đã kết hợp
                    check[i] += 1; check[j] += 1;
                    string temp = combine_mt[i];
                    temp[k] = '-';
                    temp_mt.push_back(temp);
                    // Đẩy nó vào mảng temp_tm
                }
            }
        }
    }
}

```

Ta tiến hành ghép cặp đôi một và check xem nó có chỉ khác nhau đúng 1 vị trí, và số bit 1 của nó có chênh nhau đúng 1 bit hay không. Nếu có ta thanh thành dấu – tại vị trí đó.

**** Phương thức diff_bit() được định nghĩa như sau:**

```

// Đếm số lượng bit khác nhau của 2 số nguyên truyền vào [đã đổi thành mã nhị phân]
int McCluskeyAlgorithm::diff_bit(string a, string b, int bit)
{
    int diff_bit_num = 0;
    for (int i = 0; i < bit; i++)
    {
        if (a[i] != b[i] && (a[i] != '-' || b[i] != '-'))
            diff_bit_num++;
    }
    return diff_bit_num;
}

```

```

// Cập nhật lại mảng combine sau mỗi lần kết hợp
combine_mt = temp_mt;

// Kiểm tra xem trong quá trình ghép cặp thì các giá trị nào chưa sử dụng thì lưu nó vào mảng pi
for (int i = 0; i < check.size(); i++)
{
    if (check[i] == 0)
        pi.push_back(priv_mt[i]);
}

```

Ta cập nhật lại mảng combine sau mỗi lần ghép cặp và check xem ô nào không ghép cặp được ta đưa vào mảng pi.

```

for (int i = 0; i < binary_minterm.size(); i++)
{
    for (int j = 0; j < pi.size(); j++)
    {
        /*
        Kiểm tra nếu thằng còn lại chưa ghép nếu nó đã nằm trong thằng lớn hơn rồi thì gán nhãn là 1
        Ví dụ chỗ gọi hàm combine thì có 1 hoặc nhiều phần tử trong mảng combine_minterm không thể ghép cặp được do số lượng bit 1 chênh nhau khác 1
        thì ở bước này ta xét nếu ô đó nằm trong cặp đã kết hợp thì bỏ qua nó.
        combine_minterm có 3 con 0001, 001-, 011- thì cặp 001- sẽ kết hợp với cặp 011- thành 0-1- và ô 0001 đã nằm trong khối này rồi.
        */
        if (isChecked(pi[j], binary_minterm[i], bitnum))
            epi_check_board[i][j] = 1;
    }
}

```

Kế tiếp ta kiểm tra trong mảng pi xem các chuỗi bit không kết hợp được do khác số lượng bit 1 lớn hơn 1 xem nó có nằm trong các ô đã được combine hay không. Nếu có thì ta gán flag là 1 đồng thời đẩy vào mảng epi.

*** Trong đó phương thức isChecked được định nghĩa như sau:**

```
// Kiểm tra sự kết hợp của 2 chuỗi nhị phân
bool McCluskeyAlgorithm::isChecked(string pi, string binary, int bit)
{
    int same_bit_num = 0;
    int count = 0;

    for (int i = 0; i < bit; i++)
    {
        if (pi[i] == '-')
            count++;
    }

    // Nếu vị trí tương ứng bằng nhau chưa được thay thế thì tăng biến đếm lên
    // Ví dụ pi = -0-1, binary = -011 trả về true. Điều này thể hiện khối -011 nằm trong khối -0-1
    for (int i = 0; i < bit; i++)
    {
        if (pi[i] == binary[i] && (pi[i] != '-'))
            same_bit_num++;
    }

    if ((bit - count) == same_bit_num)
        return true;

    else
        return false;
}
```

Phương thức này kiểm tra chuỗi string chưa được kết hợp trong mảng pi với các khối đã kết hợp và xem có có được bao bọc trong khối hay không. Nếu có return true;

```
// Lọc những phần tử dư thừa trong mảng pi và lưu vào mảng nepi
nepi = duplication(pi, epi);
```

Sau đó ta lọc các phần tử trùng nhau trong mảng pi và epi rồi đưa vào mảng nepi. Mảng pi chứa tế bào sau khi ghép cặp thì giờ đã chuyển sang cho mảng nepi.

***Trong đó phương thức duplication được định nghĩa như sau:**

```
// Xóa những phần tử dư thừa trong a [phần tử có cả trong a và b]
vector<string> McCluskeyAlgorithm::duplication(vector<string>& a, vector<string>& b)
{
    vector<string>::iterator iter;
    vector<string>::iterator iter_b;
    vector<string> c = a;

    for (iter_b = b.begin(); iter_b != b.end(); iter_b++) {
        for (iter = c.begin(); iter != c.end(); ) {
            if (*iter == *iter_b)
                iter = c.erase(iter);
            else
                iter++;
        }
    }

    return c;
}
```

Nếu phần tử nào có trong a và có cả trong b thì ta sẽ xóa phần tử đó ở trong a.

```
// Sắp xếp các tế bào lớn tăng dần. Thay dấu - thành kí tự khác để dùng hàm sort
for (int i = 0; i < epi.size(); i++)
{
    for (int j = 0; j < bitnum; j++)
    {
        if (epi[i][j] == '-')
            epi[i][j] = '2';
    }
}

for (int i = 0; i < nepi.size(); i++)
{
    for (int j = 0; j < bitnum; j++)
    {
        if (nepi[i][j] == '-')
            nepi[i][j] = '2';
    }
}

sort(epi.begin(), epi.end());
sort(nepi.begin(), nepi.end());
```

Ta tiến hành sort 2 mảng epi và nepi bằng cách lợi dụng hàm sort theo kí tự alphabet. Tức là ta sẽ đổi vị trí của kí tự '-' thành kí tự '2' rồi mới tiến hành sort. Sau quá trình này ta sẽ trả nó về như cũ.

```
// Lọc trùng và đẩy vào mảng kết quả cuối cùng mảng answer
epi.erase(unique(epi.begin(), epi.end()), epi.end());
for (int i = 0; i < epi.size(); i++)
    answer.push_back(epi[i]);

for (int i = 0; i < nepi.size(); i++)
    answer.push_back(nepi[i]);

// Xuất kết quả tất cả các tế bào lớn tìm thấy:
cout << "All of big cell: ";
for (int i = 0; i < answer.size(); i++)
    cout << answer[i] << ' ';
cout << endl;
```

Kế tiếp ta sẽ lọc trùng lần cuối và đưa tất cả tế bào từ 2 mảng trung gian vào mảng kết quả cuối cùng [mảng answer] và xuất kết quả ra màn hình.

2. Phương thức Minimizer() Tìm công thức đa thức tối thiểu:

```
for (int i = 0; i < answer.size() - 1; i++) {
    sortByNumDash(answer);
    for (int j = i + 1; j < answer.size(); j++) {
        string cell = "";

        for (int k = 0; k < this->num_var; k++) {
            if (answer[i].at(k) != answer[j].at(k) && answer[i].at(k) != '-' && answer[j].at(k) != '-') {
                goto a;
            }
        }

        for (int k = 0; k < this->num_var; k++) {
            if (answer[i].at(k) != answer[j].at(k)) {
                if (answer[i].at(k) == '0' && answer[j].at(k) == '-') {
                    cell.append("1");
                }
                if (answer[i].at(k) == '1' && answer[j].at(k) == '-') {
                    cell.append("0");
                }
                if (answer[i].at(k) == '-') {
                    string temp = answer[j].substr(k, 1);
                    cell.append(temp);
                }
            }
        }
        else {
            string temp = answer[j].substr(k, 1);
            cell.append(temp);
        }
    }

    if (cell == answer[j]) {
        answer.erase(answer.begin() + j);
        j--;
    }
}
```

Từ mảng các tế bào lớn ta tiến hành match từng cặp với nhau. Với ý tưởng thuật toán thay thế theo công thức:

+ Duyệt từ $i = 0$ tới $i = \text{num_var} - 1$. Nếu có vị trí nào mà 2 bit khác nhau và 2 bit đó phải khác ' - ' thì ta ngưng việc match cặp đó và tiến hành với cặp kế tiếp. Vì 2 phần tử đó không hề chồng lên nhau nên ta không cần phải trừ phần chồng lấp.

Lấy ví dụ: $a = 00--$, $b = 01-0$ thì rõ ràng ta thấy $i = 1$ thì 0 khác 1 nên 2 khối này không liên qua gì nhau cả.

+ Nếu $a[i] \neq b[i]$ thì:

$++ a[i] = 0 \mid 1$ và $b[i] = '-'$. Ta thay $b[i] = (a[i] + 1) \% 2$.

$++ a[i] = '-'$ và $b[i] = 0 \mid 1$. Ta giữ nguyên $b[i] = b[i]$

Lấy ví dụ: $a = 0---$, $b = -1-0$ thì $b = 1100$

+ Nếu $a[i] == b[i]$ thì: ta ghi lại.

+ Nếu sau khi biến đổi mà b không đổi thì chứng tỏ b nằm trong a

Lấy ví dụ: $a = 0---$, $b = 0--1$ thì $b = 0--1$. Rõ ràng b nằm trong a . Lúc này ta sẽ xóa b đi.

Cứ thế mà ghép cho tới khi không còn có thể hiệu trừ nhau nữa thì dừng thuật toán. Trong này có dùng một hàm `sortByNumDash()`. Hàm này có tác dụng sắp xếp các phần tử trong mảng tế bào lớn theo số lượng dấu gạch giảm dần. Phục vụ việc match chính xác hơn.

***Hàm `sortByNumDash()`: được định nghĩa như sau:**

```
/*
Hàm sort mảng tế bào lớn theo số lượng dấu gạch
*/
inline void sortByNumDash(vector<string> &answer) {
    for (int i = 0; i < answer.size() - 1; i++) {
        for (int j = i + 1; j < answer.size(); j++) {
            if (countDash(answer[i]) < countDash(answer[j])) {
                swap(answer[i], answer[j]);
            }
        }
    }
}
```

File Source.cpp [Phục vụ việc gọi phương thức xử lý]

```
void McCluskeyAlgorithm::Main_Menu()
{
    int choose = 0;

    while (1) {
        int n;

        this->free_all();

        system("cls");

        cout << "Enter num variable of Karnaugh map: ";
        cin >> n;

        system("cls");

        this->num_var = n;

        cout << "-----" << endl;

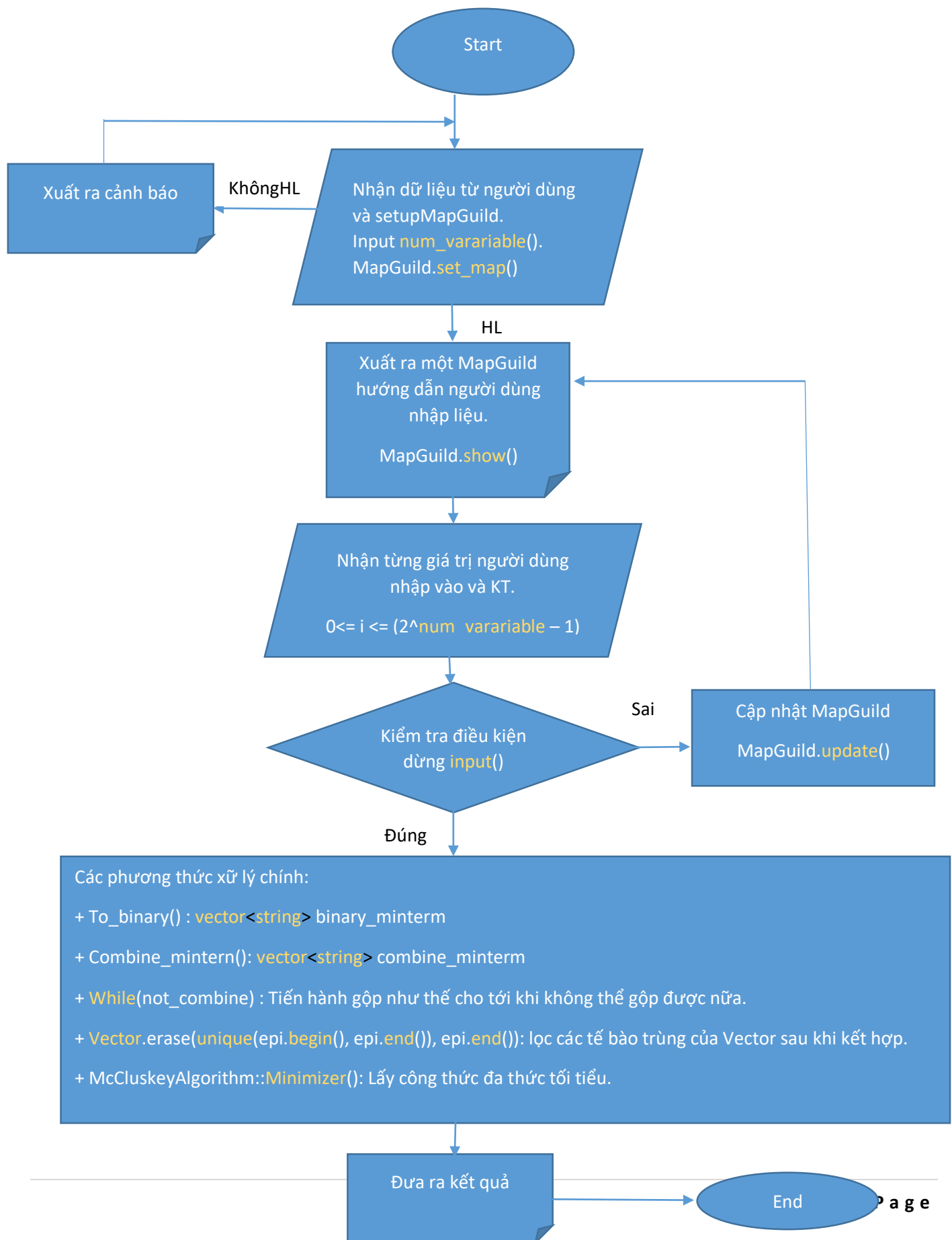
        this->Input();
        this->QM_Method();
        this->Minimizer();

        cout << endl;
        cout << "Do you want to continues ? (1. Yes, 2. No) ";
        fflush(stdin); cin.ignore();

        cin >> choose;
    }
}
```

Gọi phương thức Main_menu() để vào giao diện chương trình xử lý.

LƯU ĐỒ THUẬT TOÁN PROJECT KARNAUGH-MAP



Diễn tả lưu đồ:

Bắt đầu thuật toán

B1: Nhận số lượng biến của biểu đồ Kar từ người dùng.

- Hợp lệ nếu số lượng biến từ 2 đến 6. Thực hiện các bước tiếp theo.
- Không hợp lệ [còn lại]. Quay lại B1.

Thực chất ta có thể giải bài toán với số lượng biến là n bất kì. Nhưng do giới hạn về khả năng chính xác của thuật toán và nhu cầu của người dùng chưa cao nên ta đặt mức giới hạn như thế là vừa đủ. Ta có thể mở rộng thêm sau này đồng thời ta cũng có thể cải thiện khả năng chính xác của thuật toán bằng việc thanh lọc dữ liệu đầu vào cũng như dò các test case và xử lý ngoại lệ.

B2: Show ra một map hướng dẫn người dùng nhập liệu. Đó là một ma trận với chiều dài và chiều rộng được quy ước trước theo số lượng biến. Người dùng sẽ nhập vào các số nguyên từ 0 đến $2^n - 1$ (với n là số lượng biến của biểu đồ Kar).

B3: Kiểm tra điều kiện dừng:

- Chưa nhận thấy kí tự kết thúc quá trình nhập liệu của người dùng. Tiếp tục bước tiếp theo.
- Nhận thấy kí tự kết thúc. Chuyển sang bước 5.

B4: Cập nhập lại MapGuild theo từng giá trị mà người dùng nhập vào. Các ô mà người dùng chọn sẽ được thay thế thành giá trị -1. Quay lại B2.

B5: Các phương thức xử lý chính:

- Chuyển đổi các giá trị nguyên người dùng nhập vào thành các mã nhị phân.
- Tính chỉnh chiều dài các dãy bit nhị phân.
- Tiến hành gom nhóm các bit trên theo tiêu chí số lượng bit 1 (0 bit 1 đến n bit 1)
- Tiến hành compile các nhóm bit với nhau theo tiêu chí: 2 dãy bit phân biệt chỉ khác nhau đúng 1 bit và bit đó phải nằm ở vị trí tương ứng. Đồng thời thay thế vị trí đó thành 1 kí tự đặt biệt. (-).
- Tiếp tục quá trình đó cho tới khi không còn cặp bit nào có thể combine với nhau được nữa.
- Sau bước trên ta sẽ nhận được một Vector các tế bào cực đại của biểu đồ Kar.
- Tiến hành cộng gộp các tế bào trên, hiệu lấy phần dư, chia tế bào đồng thời sắp xếp và gom nhóm chúng để tạo công thức đa thức tối thiểu.

Trên đây là List mô tả các quá trình sẽ diễn ra khi chạy thuật toán. Cụ thể về quá trình xây dựng cũng như tác dụng của các phương thức, thuộc tính sẽ được giải thích trong file báo cáo chi tiết các mô tả dự án.

B6: Đưa ra kết quả cho người dùng.

Kết thúc thuật toán.

MÔ TẢ CHỨC NĂNG CỦA CÁC THÀNH PHẦN TRONG PROJECT VECTOR-BASIC

SƠ ĐỒ LỚP PROJECT VECTOR



File Vector.h: [Phục vụ việc khai báo Class]

1. Thuộc tính:

```
private:
    // Attribute:
    int n_dim;
    double* arr_n_dim_vector;
```

Khởi tạo thuộc tính của một vector bao gồm:

- + n_dim: Số chiều của một Vector.
- + *arr_n_dim_vector: mảng động chứa giá trị các chiều của vector n chiều.

2. Getter:

```
// Getter:
int get_dim();
double* get_arr_vector();
double get_at_arr_vector(int _i);
```

Sinh các Getter phục vụ cho việc truy vấn dữ liệu.

- + get_dim(): trả về số chiều của Vector
- + get_arr_vector(): trả về mảng động các chiều của vector
- + get_at_arr_vector(int i): trả về giá trị của chiều thứ i của vector

3. Constructor và Destructor:

```
// Constructor:
Vector();

// Destructor
~Vector();
```

Sinh các Constructor và Destructor.

4. Overloading một số phép tính:

```
// Overloading
Vector& operator = (const Vector &);
friend istream& operator >> (istream& in, Vector&);
friend ostream& operator << (ostream& out, const Vector&);
Vector& operator + (Vector &);
Vector& operator * (int k);
```

Việc Class có sử dụng con trỏ nên ta cần phải định nghĩa trước minh lại phương thức Khởi tạo sao chép, Phương thức hủy đối tượng, Toán tử gán bằng. Đồng thời Overloading thêm một số toán tử để việc code trở nên gọn gàng và dễ hiểu hơn. [Toán tử >>, <<, +, *]

5. Phương thức xử lý chính:

```
// Method  
void main_menu();
```

Đây là một hàm lặp vô tận có tác dụng nhận lệnh từ người dùng đồng thời thực hiện yêu cầu và trả về kết quả.

File Vector.cpp: [Phục vụ việc định nghĩa các phương thức trong Class]

1. Sinh các Getter:

```
// Sinh các getter của Object
int Vector::get_dim()
{
    return this->n_dim;
}

double* Vector::get_arr_vector()
{
    return this->arr_n_dim_vector;
}

double Vector::get_at_arr_vector(int _i)
{
    return this->arr_n_dim_vector[_i];
}
```

2. Khởi tạo Constructor không đối số:

```
// Constructor Default
Vector::Vector()
{
    this->n_dim = -1;
}
```

Constructoc khởi tạo n_dim = -1. Để phục vụ việc kiểm tra sau này trước khi tính toán.

3. Nạp chồng toán tử:

+ Toán tử gán bằng:

```
// Overloading Operator =
Vector& Vector::operator=(const Vector& _v2)
{
    // TODO: insert return statement here
    if (this->arr_n_dim_vector != _v2.arr_n_dim_vector) {
        this->n_dim = _v2.n_dim;

        delete[] this->arr_n_dim_vector;
        arr_n_dim_vector = new double[this->n_dim];

        for (int i = 0; i < this->n_dim; i++) {
            arr_n_dim_vector[i] = _v2.arr_n_dim_vector[i];
        }
    }
    return *this;
}
```

+ Toán tử >>

```
// Overloading Operator >>
istream& operator>>(istream& in, Vector& v1)
{
    // TODO: insert return statement here
    cout << "_____ In Vector _____" << endl;
    cout << "Enter dim of Vector V: ";
    in >> v1.n_dim;

    v1.arr_n_dim_vector = new double[v1.n_dim];

    cout << ">> Enter Vector V:" << endl;

    for (size_t i = 0; i < v1.n_dim; i++) {
        cout << "Value of dim " << i + 1 << ": ";
        in >> v1.arr_n_dim_vector[i];
    }

    cout << "Done !\n" << endl;

    return in;
}
```

+ Toán tử <<

```
// Overloading Operator <<
ostream& operator<<(ostream& out, const Vector& v1)
{
    // TODO: insert return statement here
    if (v1.n_dim == -1) {
        cout << "_____ Warnings _____" << endl;
        cout << "You must enter V1 before using this method !\n" << endl;

        return out;
    }

    out << "_____ Out Vector _____" << endl;
    out << ">> [";

    int i;

    for (i = 0; i < v1.n_dim - 1; i++) {
        out << v1.arr_n_dim_vector[i] << ", ";
    }

    out << v1.arr_n_dim_vector[i];
    out << "]" << endl << endl;

    return out;
}
```

+ Toán tử +

```
/*
Overloading +
Đầu ra: một Vector result = v1+v2
*/
Vector& Vector::operator+(Vector& v2)
{
    // TODO: insert return statement here
    if (this->n_dim == -1) {
        cout << "Warnings" << endl;
        cout << "You must enter V1 before using this method !\n" << endl;

        cout << "Dim of V1: " << this->n_dim << endl;
        cout << "Dim of V2: " << v2.n_dim << endl << endl;
        return v2;
    }

    cin >> v2;

    if (v2.n_dim != this->n_dim) {
        cout << "Warnings" << endl;
        cout << "Dim of two Vector are not similar !" << endl << endl;

        cout << "Dim of V1: " << this->n_dim << endl;
        cout << *this;

        cout << "Dim of V2: " << v2.n_dim << endl;
        return v2;
    }

    Vector *result = new Vector();
    result->n_dim = this->n_dim;
    result->arr_n_dim_vector = new double[this->n_dim];

    for (int i = 0; i < this->n_dim; i++) {
        result->arr_n_dim_vector[i] = this->arr_n_dim_vector[i] + v2.arr_n_dim_vector[i];
    }

    return *result;
}
```

Điều kiện if(1) để kiểm tra xem người dùng có nhập Vector v1 vào chưa. Nếu chưa thì xuất ra cảnh báo và dùng phương thức.

Điều kiện if(2) để kiểm tra xem số chiều của Vector 1 và Vector 2 có bằng nhau không. Nếu không thì không thể thực hiện phép toán cộng 2 vector.

Đầu ra của phương thức là một Vector $\text{result} = v1 + v2$;

+ Toán tử *

```
Vector& Vector::operator*(int k)
{
    // TODO: insert return statement here
    if (this->n_dim == -1) {
        cout << "Warnings" << endl;
        cout << "You must enter V1 before using this method !\n" << endl;

        return *this;
    }

    for (int i = 0; i < this->n_dim; i++) {
        this->arr_n_dim_vector[i] *= k;
    }

    return *this;
}
```

Điều kiện if(1) để kiểm tra xem người dùng có nhập Vector v1 vào chưa. Nếu chưa thì xuất ra cảnh báo và dùng phương thức.

Đầu ra của phương thức là một Vector $*this = *this * k$;

4. Phương thức xử lý chính: Main_menu()

```
cout << "          MENU          \n"
<< "1. Enter vector: \n"
<< "2. Show vector: \n"
<< "3. Add two vector: \n"
<< "4. Multiple this vector with value k: \n"

<< "          \n"
<< "Enter any others to exit !\n"
<< "You choose ? ";

cin >> choose;
```

Nhận lệnh từ người dùng và xử lý theo yêu cầu.

File Source.cpp [Phục vụ việc gọi phương thức xử lý chính đã định nghĩa trong file Vector.cpp]

```
#include "Vector.h"
#include <iostream>
using namespace std;

int main()
{
    Vector* v1 = new Vector();

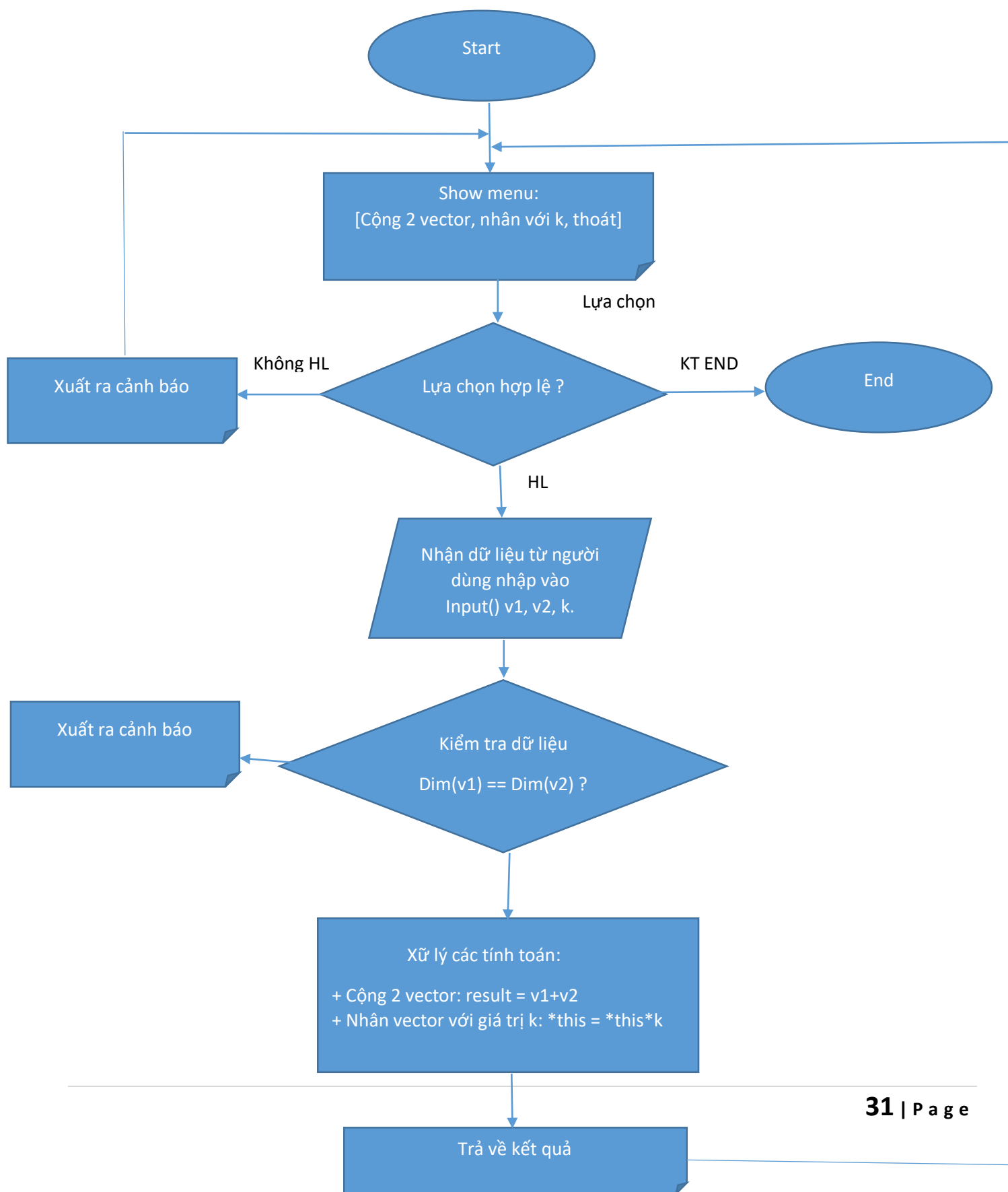
    v1->main_menu();

    return 0;
}
```

Gọi

main_menu() vào giao diện người dùng.

LƯU ĐỒ THUẬT TOÁN PROJECT VECTOR



DIỄN TẢ LƯU ĐỒ:

Bắt đầu chương trình:

B1: Trình diễn Menu thông báo các chức năng hỗ trợ của chương trình.

B2: Nhận lựa chọn từ người dùng:

- TH1: Lựa chọn sai. Thực hiện 2 việc:
 1. Xuất ra cảnh báo
 2. Quay lại B1
- TH2: Lựa chọn đúng:
 1. Lựa chọn thoát [Dừng chương trình]
 2. Thực hiện các bước kế tiếp

B3: Nhận dữ liệu đầu vào từ người dùng và lưu trữ nó.

B4: Kiểm tra dữ liệu đầu vào:

- TH1: Dữ liệu không hợp lệ
 1. Số chiều của 2 vector nhận vào không giống nhau => Không thể thực hiện phép tính cộng 2 vector.
 2. Số chiều của vector < 0

Quay lại B3.

- TH2: Dữ liệu hợp lệ: Thực hiện các bước kế tiếp.

B5: Xử lý các tính toán: Cộng 2 vector, nhân vector với một giá trị k bất kì.

B6: Trả về kết quả và quay lại B2.

Ý tưởng của lưu đồ là dùng một vòng lặp vô tận để nhận các yêu cầu từ người dùng sau đó xử lý các dữ liệu nếu hợp lệ định dạng và trả về kết quả. Chương trình kết thúc khi được lệnh từ phía người dùng hoặc đóng giao diện console đi.

MÔ TẢ CHỨC NĂNG CỦA CÁC THÀNH PHẦN TRONG

PROJECT CALCULATE-MATRIX

Để dễ hình dung ta có sơ đồ lớp sau :

Matrix

```
- matrix : double**
- cap    : int
- row    : int
- col    : int

//Lấy cấp ra dùng
+ int GetLevel();
//Lấy dòng ra
+ int GetRow();
//Lấy cột ra
+ int GetCol();
//Lấy ra địa chỉ của ma trận
+ double** GetMatrix();
//Nạp chồng toán tử nhập(Nhập ma trận)
+ friend istream& operator>>(istream&, Matrix&);
//Nạp chồng toán tử xuất(Xuất ma trận)
+ friend ostream& operator<<(ostream&, Matrix&);

//Tính định thức
+ double Det(double**matrix,int cap);

//Ma trận phụ hợp
+ Matrix FindAdj();

//Ma trận nghịch đảo
+ Matrix inverse();

//Dùng để nhân 2 ma trận
+ Matrix operator*(const Matrix b);

//Hạng của ma trận
+ int Rank();
//Hàm gọi lại tái sử dụng
+ static int Rank(double **matrix , int d , int c);

//He phương trình tuyến tính
+ void TimNghiemHePTTT();

//Menu thao tác với người dùng
+ void MainMenu();
//Constructor
+ Matrix();
+
+ Matrix(int n);
//Destructor
+ ~Matrix();
```

- **A.Các thuộc tính có trong bài**

```
private:
    //Matrix : ma trận dùng chính trong chương trình
    double **matrix;
    //cấp , dòng và cột của matrix
    int cap, row, col;
```

- Matrix dùng trong bài là một ma trận 2 chiều động vì vậy phải khai báo con trỏ cấp 2
- Cap , row , col : tương ứng với cấp ma trận , dòng và cột cấp ở đây thể hiện nó là một ma trận vuông . Khi row = col thì mới tồn tại khái niệm cấp
- Cấp được dùng chính trong các thao tác tính toán với ma trận vuông(Định thức , nghịch đảo..)
- Ngoài ra các thuộc tính phải là private đảm bảo tính đóng gói

- **B.Các phương thức CHÍNH dùng trong bài:**

-GetLevel() , GetRow() , GetCol() , GetMatrix() : Các method này để lấy giá trị của thuộc tính ra dùng

-Nạp chồng toán tử dùng để đơn giản hoá việc nhập và xuất

- TÍNH ĐỊNH THỨC : `double Det(double**matrix,int cap).`

(Ảnh minh họa : tài Liệu ĐH Bách Khoa Hà Nội)

2.2.1 Đ/n1: Cho ma trận $A=[a_{ij}]$ vuông cấp n . Phần phụ đại số của a_{ij} , kí hiệu là A_{ij} , được xác định như sau

$$A_{ij} = (-1)^{i+j} \det M_{ij}$$

trong đó M_{ij} là ma trận có được từ ma trận A bằng cách bỏ đi hàng i , cột j .

Ta giải bài này bằng cách áp dụng pp khai triển định thức theo cột , cột ta chọn = 0 (tức là cột thứ nhất)

Thay vì tính trực tiếp ta sẽ hạ bậc từ từ bằng đệ quy cho đến khi nó cấp 2 là điều kiện dừng(hay nói cách khác là bỏ liên tục từng dòng , từng cột như ảnh trên)

Ta hạ cấp bằng cách bỏ từng dòng i và cột 0 để tính định thức tại cấp đó , sau đó cộng tất cả lại , ta có định thức.

Ví dụ ta có ma trận 4x4

khi ta bỏ dòng thứ 0 và cột thứ 0 thì nó ngay lập tức về ma trận 3x3

tương tự bỏ dòng thứ 1 và cột thứ 0 => ma trận 3x3

tương tự như thế cho đến khi hết dòng

Ta được 4 ma trận 3x3

Và $\text{Det}(A) = A[i][0] * (-1)^i + \text{Det}(A_i)$

với A_i là các ma trận ta vừa hạ cấp

Tóm lại : khi có ma trận cấp 5

-Ta hạ cấp được 5 ma trận cấp 4

-1 Ma trận cấp 4 hạ được 4 ma trận cấp 3 => 5 ma trận cấp 4 = 20 ma trận cấp 3

- 20 ma trận cấp 3 => 60 ma trận cấp 2 => trả được kết quả

=>Cho nên khi tính với cấp n ta chỉ cần gọi đệ quy là được

Trong file cpp đã giải thích rất rõ vấn đề này

Ta có source code(trong file cpp) :

```

for (int i = 0; i < cap; i++)
{
    cot = 0;
    for (int j = 1; j < cap; j++)
    {
        if (i != d)
        {
            subMatrix[dong][cot] = matrix[i][j];
            cot++;
        }
    }
    //Kiểm tra nếu nhập được thì mới tăng dòng lên tránh tăng 2 lần
    if (i != d)
    {
        dong++;
    }
}
if (d % 2 == 0)
    t = 1;
else
    t = -1;
det += matrix[d][0] * t * Det(subMatrix, cap - 1);
delete[] subMatrix;

```

Như đã nói ta sẽ bỏ lần lượt từng dòng và cột 0 => vòng for đầu tiên là xét từng dòng

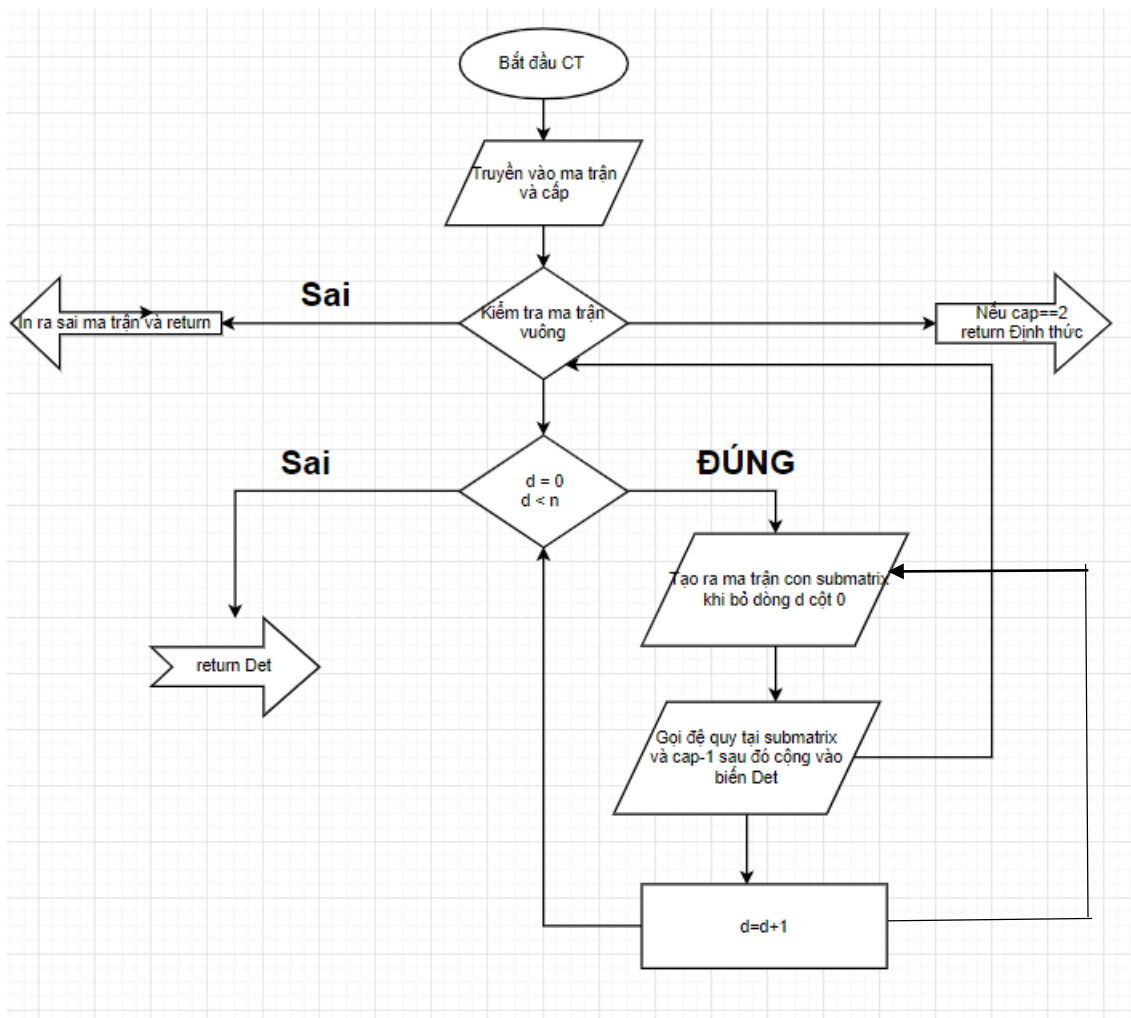
Và for bên trong là xét cột, vì ta luôn bỏ cột 0 nên ta xét j = 1, khi dòng thứ i mà khác dòng ta bỏ thì mới đưa vào submatrix (submatrix là ma trận phụ giúp ta đệ quy), sau khi tìm được ma trận bỏ dòng i - cột 0 thì ta gọi đệ quy tại đó tiếp tục và điều kiện dừng là cấp = 2 với công thức định thức:

```

if (cap == 2)
    return matrix[0][0] * matrix[1][1] - matrix[1][0] * matrix[0][1];

```

LƯU ĐỒ THUẬT TOÁN :



-TÌM MA TRẬN PHỤ HỢP : `Matrix FindAdj();`

Thuật toán tìm ma trận phụ hợp là :

$$\text{Adj}(A) = -1^{(i+j)} * D_{ij}$$

Tức là ta lần lượt bỏ từng dòng và cột của ma trận A sau đó Tính định thức của ma trận sau khi bỏ dòng và cột đó

rồi lấy định thức đó nhân với $-1^{(\text{dòng} + \text{cột})}$ là ra được phần tử thứ ij(phần tử tại vị trí dòng và cột đó)

ví dụ :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

[a31 , a32 , a33]]

Thì $\text{Adj}(A) = \begin{bmatrix} [A_{11} , A_{12} , A_{13}] \\ [A_{21} , A_{22} , A_{23}] \\ [A_{31} , A_{32} , A_{33}] \end{bmatrix}$

Với $A_{11} = -1^{(1+1)} * D_{11} = -1^{(1+1)} * \begin{bmatrix} [a_{22} , a_{23}] \\ [a_{32} , a_{33}] \end{bmatrix}$

$A_{12} = -1^{(1+2)} * D_{12} = -1^{(1+2)} * \begin{bmatrix} [a_{21} , a_{23}] \\ [a_{31} , a_{33}] \end{bmatrix}$

$A_{13} = -1^{(1+3)} * D_{13} = -1^{(1+2)} * \begin{bmatrix} [a_{21} , a_{22}] \\ [a_{31} , a_{32}] \end{bmatrix}$

với $D_{11} D_{12} D_{13}$ là định thức mà bỏ đi các dòng-cột : (1,1) , (1,2) , (1,3) ,
 ..
 Tương tự với A_{21} , A_{22} , \dots

Ta có source code thể hiện trong file cpp :

-Ban đầu ta cần bỏ đi từng dòng i,cột j ta có source:

```
for (int d = 0; d < cap; d++)
{
    for (int c = 0; c < cap; c++)
    {
        double** subMatrix;
        int dong = 0, cot = 0;
        //Khởi tạo địa chỉ cho subMatrix
        CreateNewMatrix(subMatrix, cap);
        for (int i = 0; i < cap; i++)
        {
            //nếu i không trùng với dòng bỏ thì thực hiện
            if (i != d)
            {
                cot = 0;
                for (int j = 0; j < cap; j++)
                {
                    //Nếu j không trùng với dòng bỏ thì thực hiện tìm ma trận con
                    if (j != c)
                    {
                        subMatrix[dong][cot] = matrix[i][j];
                        cot++;
                    }
                }
                dong++;
            }
        }
    }
}
```

- Ở source trên với 2 vòng for đầu biến d và c tương tự như là các dòng và cột ta bỏ lần lượt , bên dưới thì xét điều kiện để lấy ra các cột cần thiết và lưu nó vào submatrix(giống ở hàm Det)

- Sau khi đã lấy ra được từng dòng , như thế là chưa đủ so với những gì ta trình bày , phải tính Det từng cái đấy và xét vào một ma trận mới ta bổ sung(trong vòng for lồng) :

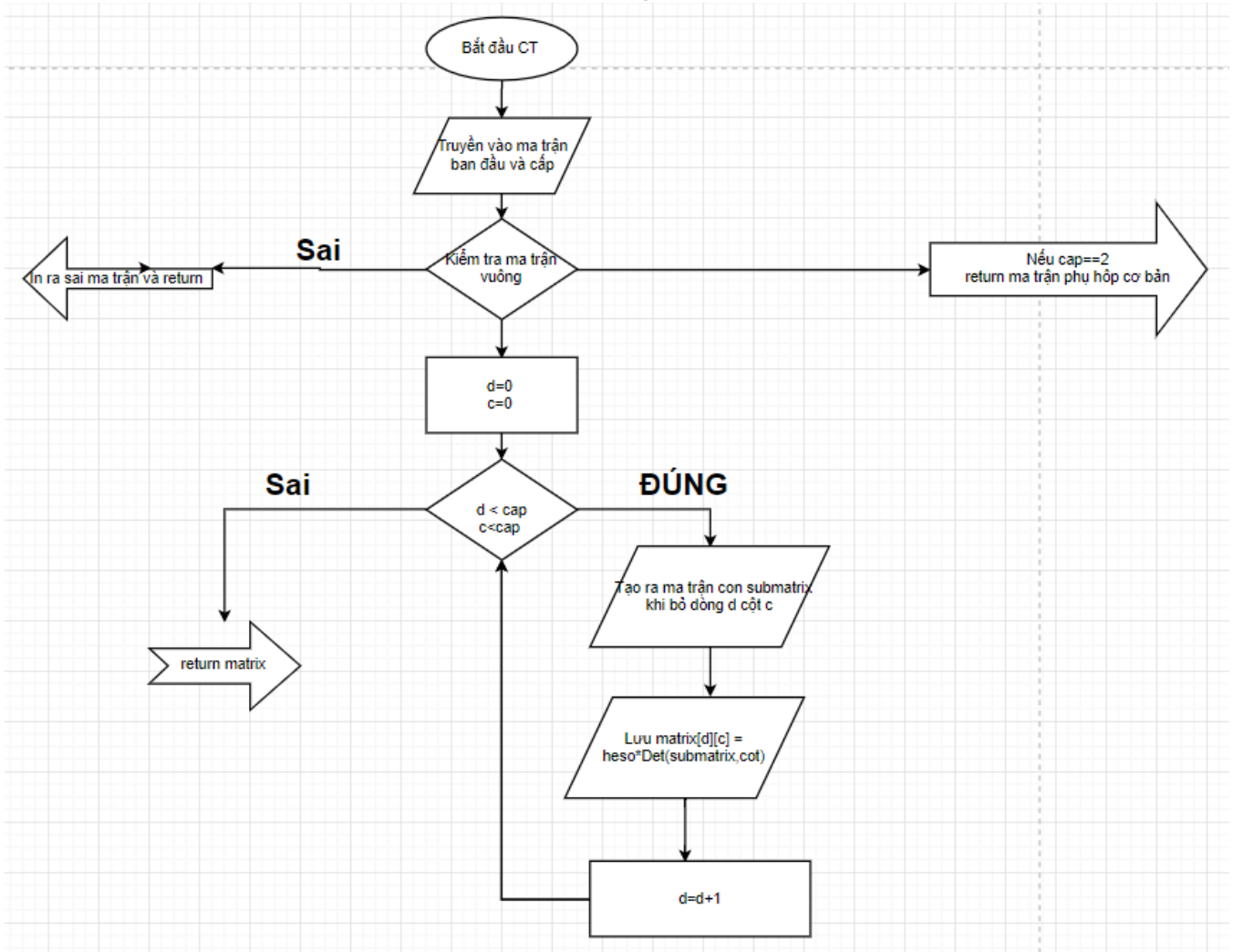
```

//Hệ số -1(i+j)
int somu = c + d;
int heso;
if (somu % 2 == 0)
    heso = 1;
else
    heso = -1;
//Trả về ma trận phụ hợp
temp.matrix[d][c] = heso * Det(subMatrix, cot);
delete[] subMatrix;

```

Với Matrix temp là kết quả trả về , d và c là các dòng cột tại vị trí ta bỏ => Ta đã lưu về dạng ma trận , gọi thêm 1 bước chuyển vị(hàm transform) thì sẽ thành ma trận phụ hợp.

LƯU ĐỒ THUẬT TOÁN :



-TÌM MA TRẬN NGHỊCH ĐẢO : `Matrix inverse()`:

$$A^{-1} = \frac{1}{\det(A)} C^T$$

Với C^T là ma trận chuyển vị của ma trận phụ hợp(hàm chuyển vị chỉ là hàm hỗ trợ không khai báo method của matrix)

Với hàm tìm ma trận phụ hợp ta đã có(FindAdj ở bên trên) => Chỉ cần bổ sung vào ma trận chuyển vị và đem từng phần tử trong ma trận chia cho Det(A) là xong.

Simple source code: Với adj là ma trận phụ hợp.

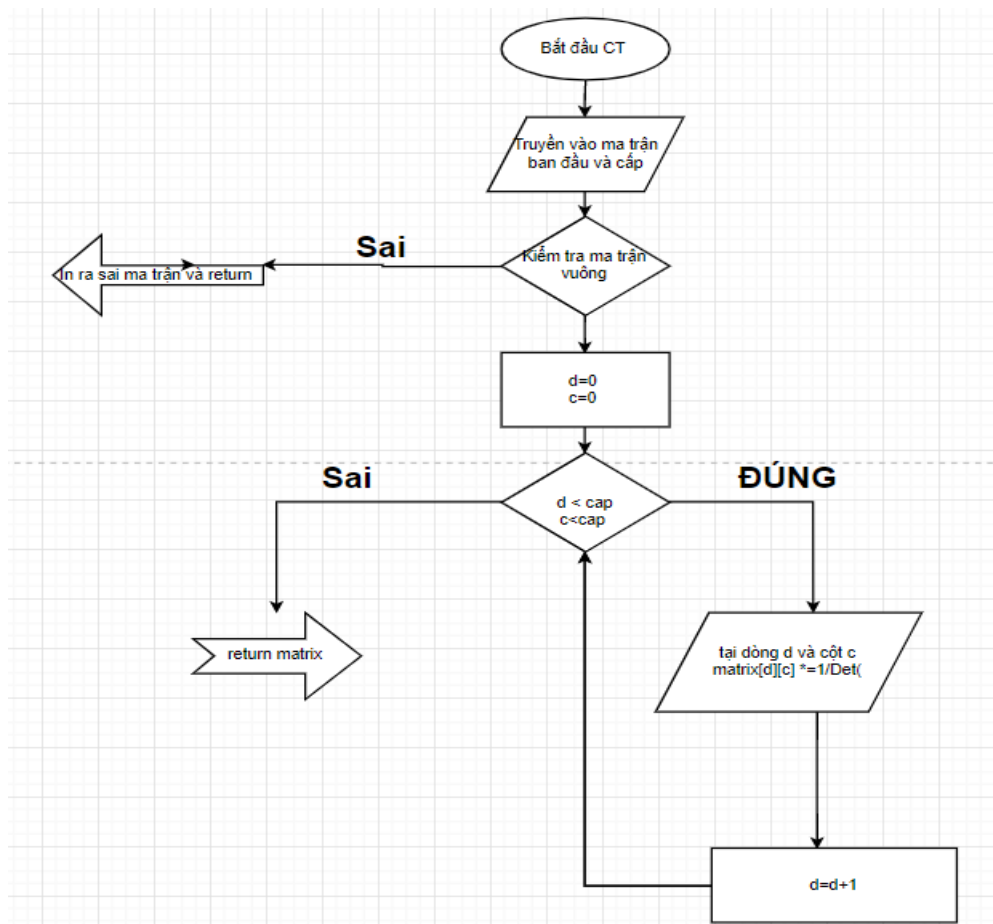
```
for (int i = 0; i < cap; i++)
{
    for (int j = 0; j < cap; j++)
    {
        //A^-1 = 1/Det|A| * Adj(A)
        adj.matrix[i][j] *= (double)1 / Det(matrix, cap);
    }
}
```

+Các hàm hỗ trợ tính ma trận nghịch đảo :

`Matrix FindAdj()`;; Như đã nói ở trên , tìm ma trận phụ hợp

`double**Transform(double**matrix,int dong , int cot)` : Tìm ma trận chuyển vị
Và trong hàm inverse gọi những hàm mà ta đã trình bày nãy giờ là xong.

LƯU ĐỒ THUẬT TOÁN :



-NHÂN 2 MA TRẬN : `Matrix operator*(const Matrix b)`

Trong bài em sử dụng nạp chồng toán tử `*` để cho nó thân thiện hơn với người dùng. Phép nhân 2 ma trận chỉ xảy ra khi cột ma trận `a` = dòng ma trận `b`. Và kết quả trả về là một ma trận với cấp là dòng. Phép nhân ma trận khá đơn giản bằng cách nhân các phần tử dòng `a` * các phần tử cột `b`. Sau đó cộng chúng lại, lần lượt như thế cho đến khi hết dòng hết cột. Độ phức tạp $O(n^3)$.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

A
B
C

Nguồn ảnh :
geeksforgeeks

A, B and C are square matrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

Ta có source code sau :

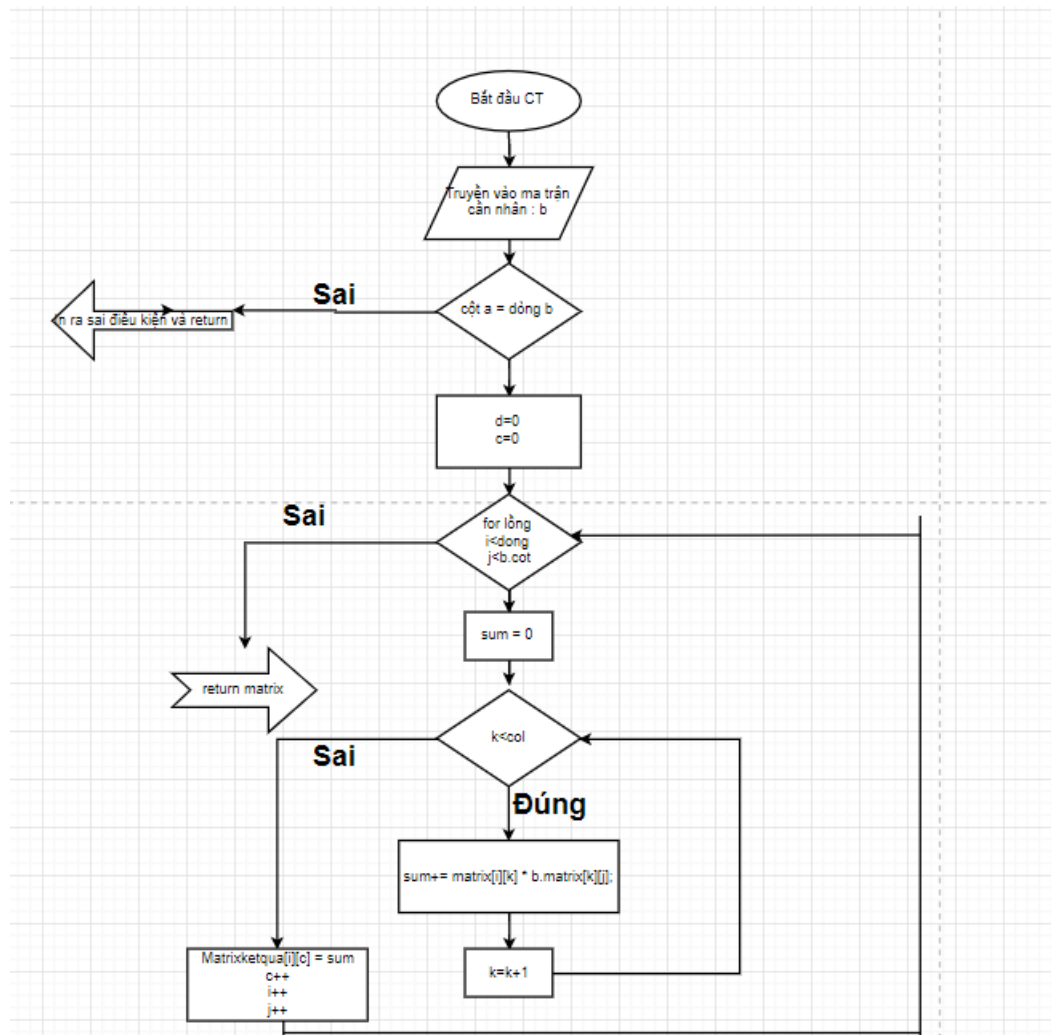
```

for (int i = 0; i < row; i++)
{
    for (int j = 0; j < b.col; j++)
    {
        int sum = 0;
        for (int k = 0; k < col; k++)
        {
            sum += matrix[i][k] * b.matrix[k][j];
        }
        kq.matrix[i][j] = sum;
        c++;
    }
    c = 0;
}
return kq;

```

Trong source , ta nạp chồng toán tử * , truyền vào một Matrix và trả về một matrix
2 vòng for lồng đầu tiên , ta xét từng dòng từng cột và vòng for thứ 3 là biến chạy lần lượt các dòng và cột.
Và chắc chắn còn xét điều kiện cột A = dòng B nữa.

LƯU ĐỒ THUẬT TOÁN :



-TÌM HẠNG CỦA MA TRẬN : `Matrix operator*(const Matrix b)`

Để làm được bài này ta phải chia nó ra làm nhiều bước nhỏ
 +Việc đầu tiên chúng ta phải luôn luôn ưu tiên các hàng ít số 0 hơn nằm bên trên để đảm bảo nó luôn là một dạng ma trận bậc thang
 +Bước thứ hai là biến đổi dòng ma trận thì nó sẽ đưa về một ma trận tương đương , vì vậy ta biến đổi làm để về ma trận bậc thang càng tốt

(Nguồn ảnh : *slideplayer*)

➤ Chương 1. Ma trận – Hệ PT ĐSTT

VD 18. $A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 4 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix}, I_n$

là các ma trận bậc thang;

$$C = \begin{pmatrix} 0 & 2 & 7 \\ 0 & 3 & 4 \\ 0 & 0 & 5 \end{pmatrix}, D = \begin{pmatrix} 2 & 3 & 5 \\ 0 & 0 & 0 \\ 0 & 1 & 3 \end{pmatrix}$$

không phải là các ma trận bậc thang.

Định lý

- Mọi ma trận đều có thể đưa về bậc thang bằng hữu hạn phép biến đổi sơ cấp trên dòng.

Thuật toán biến đổi cơ bản như sau : sau khi đẩy các dòng ít số 0 ưu tiên lên đầu , tại dòng thứ 2 , vị trí phần tử cơ sở biến đổi sao cho nó về 0 (biến đổi so với dòng 1) , tương tự dòng 3 , 4 ...

Lưu ý : như ta đã nói , sau mỗi bước biến đổi tiếp tục sort lại ma trận sao cho các dòng nhiều số 0 đưa lên đầu tiên

Cứ làm lại như thế cho đến khi dòng ta xét là dòng cuối cùng,

Và cuối cùng **rank = số dòng khác 0** của ma trận

+Các hàm con hỗ trợ hàm tính rank :

1.Hàm swap line để sort lại ưu tiên hàng ít số 0 lên đầu:

```
void SwapLine(double** matrix, int row, int col)
```

Simple source code:

```
for (int i = 0; i < row - 1; i++)
{
    for (int j = i + 1; j < row; j++)
    {
        if (CalcZero(matrix, col, i) > CalcZero(matrix, col, j))
            //Thực hiện swap line
            {
                for (int k = 0; k < col; k++)
                {
                    swap(matrix[i][k], matrix[j][k]);
                }
            }
    }
}
```

Về cơ bản đây chính thuật toán selection sort

Với phương thức CalcZero là tính lượng số 0 trong dòng đó.

2. Hàm chuyển về bậc thang như trình bày ở trên

`double**ChuyenVeBacThang(double** matrix, int row, int col)`

short source code :

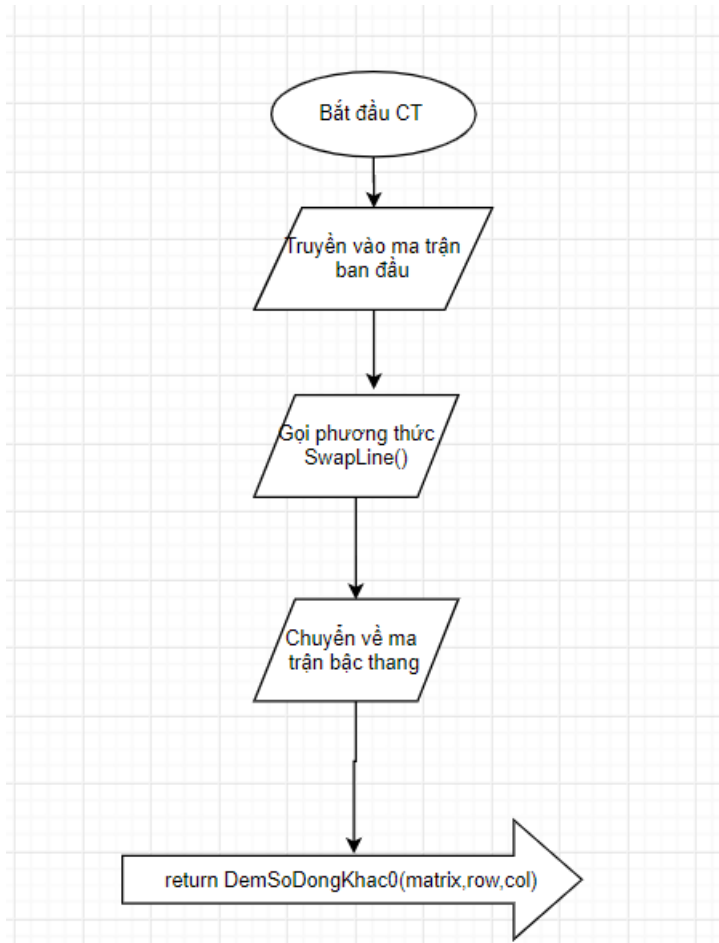
```
int cot = FirstCol(matrix, 0, col); //Tìm số đầu tiên khác 0 dòng đầu tiên
for (int i = 0; i < row - 1; i++)
{
    for (int j = i + 1; j < row; j++) //dòng chạy
    {
        //lấy giá trị dòng trên và dòng dưới
        int a = matrix[i][cot];
        int b = matrix[j][cot];
        for (int k = 0; k < col; k++)
        {
            //Tại mỗi cột thì trừ nhau cho mất đi giá trị đầu tiên(biến về ma trận bậc thang)
            matrix[j][k] = matrix[j][k] * a - matrix[i][k] * b;
        }
    }
    //Sau mỗi lần trừ, tránh trường hợp những dòng = 0 thì thuật toán sẽ bị sai.
    //Vì vậy sau mỗi lần trừ ta phải sort lại ma trận
    SwapLine(matrix, row, col);
    cot = FirstCol(matrix, i + 1, col);
}
return matrix;
```

Ban đầu ta phải gọi hàm swapline trước(như đã nói ở trên) sau đó ta gán `int cot = FirstCol(matrix,0,col)`. Tức là lúc này ta đang tìm phần tử cơ sở của dòng đầu tiên.

Sau khi tìm được thì thực hiện từng bước như đã trình bày

Sau mỗi lần thực hiện thì phải swapline lại một lần đảm bảo thuật toán luôn đúng.

LƯU ĐỒ THUẬT TOÁN:



-TÌM NGHIỆM CỦA HỆ PHƯƠNG TRÌNH TUYẾN TÍNH :

`void TimNghiemHePTTT()`

Một hệ ta có các trường hợp nghiệm sau

1.Hệ vô nghiệm:

Hệ vô nghiệm khi rank của hệ khi đưa về dạng ma trận < số biến
Và tại vị trí `matrix[row-1][col-1] !=0`.

2.Hệ có nghiệm duy nhất:

Hệ có nghiệm duy nhất khi `rank == dong`
Trường hợp này khá đơn giản chỉ cần đưa ma trận hệ về dạng ma trận bậc thang sau đó suy ra nghiệm theo kiến thức giải hệ pt lớp 7.

3.Hệ vô số nghiệm

Hệ vô số nghiệm khi rank của hệ khi đưa về dạng ma trận < số biến nhưng
`matrix[row-1][col-1]==0`

Lúc này hệ có số nghiệm tự do là `= dòng - rank`.

Ví dụ ta có hệ sau:

$$X + z = 1$$

$$y + z = 2$$

Đối với pt 3 biến thì phải có 3 phương trình mới giải được vì vậy ta phải thêm các dòng số 0 này vào đảm bảo tính đúng đắn của thuật toán

Đưa về ma trận :

$$1 \ 0 \ 1 \ 1$$

$$0 \ 1 \ 1 \ 2 \quad \Rightarrow \text{gọi } x_3 = t \text{ là nghiệm tự do}$$

$$0 \ 0 \ 0 \ 0 \quad \Rightarrow \quad x_2 = 2-t$$

$$x_1 = 1-t$$

VẬY THUẬT TOÁN ĐƯỢC THỂ HIỆN NHƯ THẾ NÀO?

Ta không bàn đến trường hợp vô nghiệm vì nó quá đơn giản , ở đây là cách giải quyết chung cho trường hợp có nghiệm tự do(vô số nghiệm) và có nghiệm duy nhất.

A.Hệ có nghiệm duy nhất(không có nghiệm tự do)

Như em đã trình bày ở trên , một hệ phương trình tuyến tính giải được khi hệ phương trình ở dạng ma trận có thể đưa về dạng ma trận bậc thang , vì vậy bước đầu tiên luôn luôn phải đưa về dạng bậc thang(bằng pp biến đổi dòng - cột)

Sau khi đưa về dạng bậc thang. Ta xét ví dụ sau ta có hệ pt 4 biến được đưa về ma trận như sau:

$$1 \ 1 \ 1 \ 1 \ 2$$

$$0 \ 3 \ 4 \ 5 \ 9$$

$$0 \ 0 \ 2 \ 3 \ 1$$

$$0 \ 0 \ 0 \ 1 \ 5$$

Để tìm được hết nghiệm ta phải đi từ dưới lên tức là tìm x_4 trước

$$\Rightarrow x_4 = 5$$

Sau khi tìm được x_4 ta mới lấy x_4 nhân với hệ số bên trên nó

$$\text{Đó là } 2.x_3 + 3.x_4 = 1$$

$$\Rightarrow x_3$$

Cứ như vậy ta thay x_3 và x_4 vào

$$\Rightarrow x_2$$

Tiếp tục thay x_2 x_3 x_4 vào

$$\Rightarrow x_1.$$

Như những cách ta trình bày , đây là trường hợp không có nghiệm tự do

Vậy ta chỉ cần tính nghiệm cuối cùng lưu nó vào 1 mảng

Tiếp tục tính nghiệm tiếp theo và lưu nó vào mảng , cứ như vậy cho đến khi lên tới dòng đầu tiên.

Lúc đây chỉ cần xuất mảng ra là xong.

Short code:

```
//Mảng a dùng để lưu các nghiệm
float* a = new float[dong];
a[0] = (float)matrix[dong - 1][cot - 1] / matrix[dong - 1][cot - 2];
int n = 1;
for (int i = dong - 2; i >= 0; i--)
{
    //Pos a là tại mỗi vị trí cần trừ để tìm x thì trừ đi những nghiệm đã có
    int posA = 0;
    for (int j = cot - 2; j >= 0; j--)
    {
        //Nếu tại cột j không phải là một phần tử cơ sở(phần tử đầu tiên tại dòng !=0), thì áp dụng quy tắc tìm x tính bình thường
        if (j != FirstCol(matrix, i, cot))
        {
            matrix[i][cot - 1] -= a[posA] * matrix[i][j];
            posA++;
        }
        //Sau khi đã đến vị trí cơ sở thì => nghiệm tại đó
        else
        {
            matrix[i][cot - 1] = (float)matrix[i][cot - 1] / matrix[i][j];
            a[n++] = matrix[i][cot - 1];
        }
    }
}
```

Tạo ra một ma trận a , và số nghiệm chắc chắn = số dòng

Ta khởi tạo a[0] là biến sau khi tính ở dòng cuối cùng.

Sau đó bắt đầu quét

vòng for đầu tiên quét từ từ vị trí kế dưới cùng đến dòng đầu

vòng for lồng tiếp theo quét từ cột = cot - 2 (tại vì cột cuối cùng là vế phải của phương trình)

Điều kiện là nếu cột đó không phải là vị trí cơ sở thì :

Vế phải - = biến * hệ số (của từng phần tử dòng)

Nếu nó là phần tử cơ sở thì chia cho hệ số => ta được biến

Lưu lại -> In ra

B. Hệ có nghiệm tự do

Nghiệm tự do có thể có rất nhiều nghiệm , việc ta cần thực hiện là làm sao để thể hiện được số nghiệm đó ví dụ:

Ta có hệ sau(đã đưa về ma trận)

```
-2  2  1  2  -1
 3 -1  0  1   4
 1 -1 -1 -1  -1
```

Giải nhanh , ta biết hệ có nghiệm tự do.

Ta gọi nghiệm đó là u1

Vậy giải nhanh thì hệ phương trình có nghiệm là:

X1 = -1-u1;

X2 = -1-2u1;

$$x_3 = 3;$$

Những gì chúng ta cần thực hiện là làm sao hiện thực được số nghiệm này lên màn hình.

Để in được nghiệm tự do , ta sẽ thao tác với chuỗi khá nhiều , vì vậy để tách hệ số và hệ số biến(ví dụ $5+6t$. 5 là hệ số còn 6 là hệ số biến)
ta sẽ phải gán cho $t = 0$; và tách ra để dễ xử lý hơn.

Trong bài mảng a sẽ lưu giá trị hệ số tự do

Mảng hesoT sẽ lưu các giá trị biến chứa tham số u_1 , u_2

Sau đó thao tác tương tự như với tìm nghiệm duy nhất :

Cần x_4 để tìm x_3 , cần x_3 để tìm x_2 , cần x_2 để tìm x_1 ==>>>

Ví dụ minh họa ta có ma trận sau:

Yêu cầu người dùng nhập vào số biến và số pt.

số biến : 4

số pt : 2

1 2 17 -29 0

0 1 -10 17 0

Nhưng tính chất của hệ , hệ chỉ giải được khi số pt = số ẩn

=> Ta phải thêm 2 dòng 0 (2 dòng vì dòng-rank = , vì thiếu 2 => thêm 2) vào ta được ma trận sau

1 2 17 -29 0

0 1 -10 17 0

0 0 0 0 0

0 0 0 0 0

B1 : Ban đầu ta thấy có 2 hàng trống (toàn số 0) => hệ có 2 nghiệm tự do(vì dòng = 4 và rank = 2 => nghiệm tự do = dòng - rank = 2)

lúc này ta có 2 biến tự do u_1 và u_2

B2:Tại vị trí dòng đầu tiên có số khác 0 ta có 2 số -10 và 17 là 2 hệ số của biến

Ta có mảng lưu hệ số :

HeSo[0] = 17 và HeSo[1] = -10 (Lưu lại nhằm xử lý , nhưng tại dòng đầu tiên này chưa cần dùng đến)

Và vì có 2 biến nên mảng giá trị biến $a[0]$ và $[1]$ đều = 0

Lúc này ta thực hiện vòng lặp để tìm nghiệm ở phương trình đầu tiên(dòng cuối trước):

0 1 -10 17 0

vì $a[0]$ và $a[1] = 0$ (ứng với nghiệm x_3 x_4)

$$\Rightarrow X_2 = (0 - \text{heso}[0]*0 - \text{heso}[1]*0) / (\text{hệ số của } x_2) \\ = (0 - 17*0 - 10*0) / (1)$$

$$\Rightarrow X_2 = 0$$

Đúng như những gì ta quy ước vì x_2 là một nghiệm chứa tham số nên ta gán nó = 0 để xử lý các tham số kia

Lúc này $a[2] = 0$

và

=> Ta in ra nghiệm trên màn hình là

$$X_2 = -17u_1 + 10u_2$$

Bằng cách chuyển -17 và 10 nhờ to_string và + u"thứ tự biến"

ở dòng đầu tiên này chưa có gì phức tạp , ta chỉ cần chuyển về đổi dấu và nhớ chia cho hệ số của x , ta thu được biến

```
x2 = 10u2 - 17u1;
```

Vì để thực hiện tiếp tục với các pt trên , ta gọi thêm 1 mảng hệ số tạm , lúc này mảng sẽ lưu giá trị hệ số của biến x2

ta lưu như sau(gọi for)

```
HeSoTam[0] = -17 = HeSoT[0]
```

```
HeSoTam[1] = 10 = HeSoT[1]
```

dùng vòng for vì số nghiệm tự do đã =2;

B3: Quay tiếp tục dòng trên

tại đây ta có pt : $x + 2(x2) + 17u2 - 29u1 = 0$

Lúc này mảng hệ số tiếp tục lưu 2 giá trị HeSo[0] = -29 và HeSo[1] = 17

=> tại $2*x2$ ta lấy 2 nhân từng hệ số trong HeSoTam : HeSoTam[0] = $2*HeSoTam[0]$
= -34 và HeSoTam[1] = $2*HeSoTam[1]$ = 20

sau đó cộng nó theo thứ tự để trùng nhau biến:

HeSo[0] + HeSoTam[0] = $-29 + -34 = -63$, HeSo[1] + HeSoTam[1] = 37;

=> Chia hệ số biến x và đổi dấu ta có

```
x = 63u1 - 37u2;
```

Đó là thuật toán.

(Nhắm vào hệ số tạm vì nó là thứ sẽ lưu lại nghiệm trước đó ta tính)

Quá trình in ra và tìm nghiệm sẽ xảy ra trong hàm :

```
void TimNghiem(double** matrix, int dong, int cot)
```

Và những bước tuần tự để đến bước tìm nghiệm sẽ được thao tác trong phương thức chính đó là phương thức

```
void Matrix::TimNghiemHePTTT()
```

LƯU ĐỒ THUẬT TOÁN

vì không thể hiện hết được bằng lưu đồ nên em chỉ tóm tắt một số phương thức cần gọi để thực hiện thao tác tìm nghiệm

