Họ tên: Ngô Phù Hữu Đại Sơn
MSSV: 18120078

BÀI TẬP LÝ THUYẾT 2

Câu 1:

Gọi $T(n)$ là chi phí để tạo ra mọi hoán vị của n phần tử. Hệ thức truy hồi là:

$$T(n) = nT(n-1) + n \text{ và } T(1) = 0$$

Ta có:

$$T(n) = nT(n-1) + n$$
$$= n[(n-1)T(n-2) + (n-1)] + n$$
$$= n(n-1)[(n-2)T(n-3) + (n-2)] + n + n(n-1)$$
$$= n(n-1)(n-2)[(n-3)T(n-4) + (n-3)] + n + n(n-1) + n(n-1)(n-2)$$
$$= n(n-1)\dots(n-i+1)T(n-i) + n + n(n-1) + \cdots + n(n-1)(n-2)\dots(n-i+1)$$

Đặt: $n - i = 1 \Rightarrow n - i + 1 = 2$

$$\Rightarrow T(n) = n!\,T(1) + n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2)\dots 3.2$$
$$= n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2)\dots 3.2$$

$$\Rightarrow \frac{T(n)}{n!} = \frac{1}{(n-1)!} + \frac{1}{(n-2)!} + \frac{1}{(n-3)!} + \cdots + \frac{1}{2!} + 1 \approx e$$

$$\Leftrightarrow T(n) \approx e.n! \in \Theta(n!)$$

Câu 2:

Ý tưởng:

- Xây dựng mảng deg với $\deg(u)$ là bậc của đỉnh u, từ đó ta có:
  - Dạng vòng: tất cả các đỉnh có bậc 2
  - Dạng sao: có duy nhất 1 đỉnh có bậc (n-1) và các đỉnh còn lại có bậc 1
  - Dạng lưới đầy đủ: tất cả các đỉnh có bậc (n-1)

```
void DetectGraphType(int a[N][N], int n){
    int * deg = getDeg(a, n); //Get degree of each node
    cout << "Do thi: ";
    if(isFullyConnected(deg ,n)){
        cout << "dang luoi day du" << endl;
    }
    else if(isRing(deg,n)){
        cout << "dang vong" << endl;
    }
    else if(isStar(deg, n)){
        cout << "dang sao" << endl;
    }
    else{
        cout << "khong co dang" << endl;
    }
}
```

```
int * getDeg(int a[N][N], int n){
    int * deg = new int[n];
    for(int i = 0;i<n;i++){
        for(int j = 0;j<n;j++){
            deg[i] += a[i][j];
        }
    }
    return deg;
}
```

```
bool isFullyConnected(int * deg, int n){
    for(int i = 0;i<n;i++){
        if(deg[i] == n - 1){
            continue;
        }
        return false;
    }
    return true;
}
```

```
bool isRing(int * deg, int n){
    for(int i = 0;i<n;i++){
        if(deg[i] == 2){
            continue;
        }
        return false;
    }
    return true;
}
```

```
bool isStar(int * deg, int n){
    int countFullyConnectedNode = 0;
    for(int i = 0;i<n;i++){
        if(deg[i] == 1)
            continue;
        if(deg[i] == n - 1){
            countFullyConnectedNode++;
            if(countFullyConnectedNode > 1){
                return false;
            }
        }
        else{
            return false;
        }
    }
    if(countFullyConnectedNode == 1){
        return true;
    }
    return false;
}
```

Chi phí:

- Xây dựng bậc của các đỉnh (*getDeg(int a[][], int n)*):

- Kích thước dữ liệu nhập: n^2
- Phép toán cơ sở: $\deg[i] += a[i][j]$
  - $f(n) = n^2 \in \Theta(n^2)$

- Kiểm tra đồ thị có dạng sao không? (*isStar(int *deg, int n)*):

- Kích thước dữ liệu nhập: n
- Phép toán cơ sở: ==
- Trường hợp tốt nhất: đỉnh đầu tiên của đồ thị có bậc khác 1 và khác (n-1)

$$B(n) = 2 \in \Theta(1)$$

- Trường hợp xấu nhất: đồ thị có dạng sao

$$W(n) = 2n \in \Theta(n)$$

- Kiểm tra đồ thị có dạng vòng không? (*isRing(int *deg, int n)*):
  - Kích thước dữ liệu nhập: n
  - Phép toán cơ sở: ==
  - Trường hợp tốt nhất: đỉnh đầu tiên của đồ thị có bậc khác 2

$$B(n) = 1 \in \Theta(1)$$

  - Trường hợp xấu nhất: đồ thị có dạng vòng

$$W(n) = n \in \Theta(n)$$

- Kiểm tra đồ thị có dạng lưới đầy đủ không? (*isFullyConnected(int *deg, int n)*):
  - Kích thước dữ liệu nhập: n
  - Phép toán cơ sở: ==
  - Trường hợp tốt nhất: đỉnh đầu tiên của đồ thị có bậc khác 2

$$B(n) = 1 \in \Theta(1)$$

  - Trường hợp xấu nhất: đồ thị có dạng vòng

$$W(n) = n \in \Theta(n)$$

Vậy thuật toán có chi phí là:

$$B(n) = n^2 + 2 + 1 + 1 = n^2 + 4 \in \Theta(n^2)$$
$$W(n) = n^2 + 2n + n + n = n^2 + 4n \in \Theta(n^2)$$

Câu 3:

Ý tưởng: gọi sum là tổng các phần tử trong tập. Ta cần tìm ra 1 tập con mà có tổng là sum/2. Duyệt qua tất các các chuỗi nhị phân từ 1 đến 2^n. Với từng chuỗi trên (trạng thái), những bit nào bằng 1 sẽ được đưa bao tập con. Nếu tổng các phần tử trong tập con đó đúng bằng sum/2 thì cập nhật kết quả.

```
void partition(int * a,int n){
    int sum = 0;
    for(int i = 0;i < n;i++){
        sum += a[i];
    }
    int currentSum = 0;
    for(int state = 1;state <= (1 << n);state++){
        currentSum = 0;
        for(int i = 0;i<n;i++){
            if(getBit(state, i) == true){
                currentSum += a[i];
            }
        }
        if(currentSum * 2 == sum){
            printResult(a, n, state);
        }
    }
}
```

```
void printResult(int * a,int n, int state){
    for(int i = 0;i < n;i++){
        if(getBit(state, i) == true) cout << a[i] << " ";
    }
    cout << " <---> ";
    for(int i = 0;i < n;i++){
        if(getBit(state, i) == false) cout << a[i] << " ";
    }
    cout << endl;
}

bool getBit(int x, int pos){
    return (x >> pos) & 1;
}
```

Câu 4:

a) Gọi t là tổng cần tìm.
Ta có tổng tất cả các dòng trong ma phương: tn
Mặc khác tổng trên cũng là tổng các số từ 1 đến n^2: $\frac{n^2(n^2+1)}{2}$

$$\Rightarrow tn = \frac{n^2(n^2+1)}{2}$$

$$\Leftrightarrow t = \frac{n(n^2+1)}{2} \ (đpcm)$$

b) Ý tưởng: tìm tất cả hoán vì từ 1 đến n^2. Với mỗi hoán vị tìm được, kiểm tra xem đó có phải là ma phương cần tìm không.

```
void magicSquare(int * p, int pivot, int n){
    int n2 = n * n;
    if(pivot >= n2 && isMagicSquare(p, n)){
        printResult(p, n);
    }
    for(int i = pivot;i<n2;i++){
        swap(p[i], p[pivot]);
        magicSquare(p, pivot + 1, n);
        swap(p[i], p[pivot]);
    }
}
```

```
bool isMagicSquare(int *p, int n){
    int sum = (n * (n * n + 1)) / 2;
    if(validRow(p,n, sum) && validCol(p, n, sum)
        && validMainDiag(p, n, sum) && validSecondDiag(p, n, sum)){
        return true;
    }
    return false;
}
```

```cpp
void printResult(int *p, int n){
    for(int i = 0;i<n;i++){
        for(int j = 0;j<n;j++){
            cout << p[computeIndex(i, j, n)] << " ";
        }
        cout << "\n";
    }
    cout << "\n";
}
```

```cpp
bool validRow(int * p, int n, int sum){
    int rowSum;
    for(int row = 0;row < n; row++){
        rowSum = 0;
        for(int col = 0; col < n;col++){
            rowSum += p[computeIndex(row, col, n)];
        }
        if(rowSum != sum) return false;
    }
    return true;
}
```

```cpp
bool validCol(int * p,int n, int sum){
    int colSum;
    for(int col = 0;col < n;col++){
        colSum = 0;
        for(int row = 0;row < n;row++){
            colSum += p[computeIndex(row, col, n)];
        }
        if(colSum != sum) return false;
    }
    return true;
}
```

```c
bool validMainDiag(int * p, int n, int sum){
    int mainDiagSum = 0;
    for(int row = 0;row < n;row++){
        mainDiagSum += p[computeIndex(row, row, n)];
    }
    return mainDiagSum == sum;
}

bool validSecondDiag(int * p, int n, int sum){
    int secondDiagSum = 0;
    for(int row = 0;row < n;row++){
        secondDiagSum += p[computeIndex(row, n - row - 1, n)];
    }
    return secondDiagSum == sum;
}

int computeIndex(int row, int col, int n){
    return row * n + col;
}
```