古典密碼學作業

程式碼GITHUB: HTTPS://GITHUB.COM/AKA2210/INFORMATION-SECURITY

文字加密

```
Enter Encode for encryption and Uncode for decryption: Encode
Please enter the plain text to be encrypted:ILoveYou
Ciphertext:XBNQXFSJ
key:QPtsmGgl
```

文字解密

```
Enter Encode for encryption and Uncode for decryption: Uncode
Please enter the cipher text to be decoded:XBNQXFSJ
Please enter the key corresponding to the ciphertext:QPtsmGgl
plaintext: ILoveYou
```

報告內容

- 我的加密方法(與tree有關)
- 我的解密方法(與tree有關)
- 我的程式碼(無chatgpt幫忙)


```
string Encode(string plaintext, string& key)
```

```
string Uncode(string Ciphertext, string key)
```

```
struct TreeNode
{
    char val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(char x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(char x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
};
```

(利用當前所學的資料結構)

加密	加密	加密	加密
• 將傳入的字串以tree的方式利用BFS進行排列，若傳入abcde，則排列為: a / \ b c / \ d e	• 在BFS的同時將每個字元的ASCII碼+15並且再加上i值(i值為當前字元在字串中的index->0,1,2,3...)，加完後若字母超過Z則會-=26。	• 第三步則是前序遍歷tree使得遇到的字元被存入，例如:(abcde)->(abdec)，且若遇到的字元為大寫則在key中隨機推入一個大寫字母，否則則隨機推入一個小寫字母。	• 輸出Ciphertext與key使後續可以解密。

解密	解密	解密	解密
• 將傳入的字串以DFS的方式排列成tree，若傳入abdec則排列為: a / \ b c / \ d e	• 利用BFS將每個字元的ASCII碼減15並且再減i值(i值為當前字元在字串中的index->0,1,2,3...)，加完後若字母低於A則會+=26，最後依靠key[i]的大小寫判斷當前node->val中字母的大小寫(isupper(node->val) == isupper(key[i]))。	• 在BFS的同時將解密後的字母存入plaintext，BFS跑完後plaintext就是正解，可直接return至main。	• cout << plaintext;使使用者得到明文。

程式碼展示
Encode and Tree

```cpp
struct TreeNode
{
    char val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(char x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(char x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
};
```

```cpp
void SequentialReplacement(TreeNode* node, string& curr)
{
    if(node == NULL)
        return;

    curr += node->val;
    SequentialReplacement(node->left, curr);
    SequentialReplacement(node->right, curr);
    return;
}
```

```cpp
string Encode(string plaintext, string& key)
{
    TreeNode *root = new TreeNode();
    TreeNode *node = root;
    string Ciphertext = "";
    srand(time(0));

    string temp = "";
    for(int i = 0; i < plaintext.size(); i++)
    {
        int judge = (int)plaintext[i] + 15 + i;
        if(isupper(plaintext[i]))
        {
            key += (65 + rand() % 26);
            while(judge > 'Z')
                judge = 'A' + (judge - 'Z') - 1;
        }
        else
        {
            key += (97 + rand() % 26);
            while(judge > 'z')
                judge = 'a' + (judge - 'z') - 1;
        }

        temp += toupper((char)judge);
    }
    plaintext = temp;

    queue<TreeNode*> q;
    q.push(root);
    root->val = plaintext[0];
    int index = 1;

    while(!q.empty())
    {
        int count = q.size();

        for(int i = 0; i < count; i++)
        {
            if(index < plaintext.size())
            {
                q.front()->left = new TreeNode(plaintext[index]);
                q.push(q.front()->left);
                index++;
            }

            if(index < plaintext.size())
            {
                q.front()->right = new TreeNode(plaintext[index]);
                q.push(q.front()->right);
                index++;
            }

            if(index < plaintext.size())
            {
                q.front()->right = new TreeNode(plaintext[index]);
                q.push(q.front()->right);
                index++;
            }

            q.pop();
        }
    }

    SequentialReplacement(root, Ciphertext);
    return Ciphertext;
}
```

while內為BFS

程式碼展示
Encode and Tree

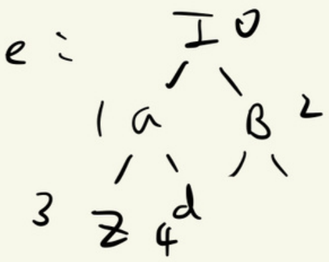
```cpp
string Uncode(string Ciphertext, string key)
{
    int index = 0;
    TreeNode* root = buildTree(Ciphertext, index, 0);
    string plaintext = "";
    queue<TreeNode*> q;
    q.push(root);
    index = 0;

    while(!q.empty())
    {
        int count = q.size();
        for(int i = 0; i < count; i++)
        {
            if(index < Ciphertext.size())
            {
                int judge = (int)q.front()->val - 15 - index;
                while(judge < 'A')
                        judge = 'Z' - ('A' - judge) + 1;
                    if(isupper(key[index]))
                        plaintext += (char)judge;
                    else
                        plaintext += tolower((char)judge);
                    index++;
                }

                if(q.front()->left != NULL)
                    q.push(q.front()->left);

                if(q.front()->right != NULL)
                    q.push(q.front()->right);

                q.pop();
            }
        }
        return plaintext;
}
```

```cpp
TreeNode* buildTree(string& s, int& index, int curr) {
    if (curr >= s.size()) {
        index--;
        return nullptr;
    }

    TreeNode* node = new TreeNode(s[index]);

    index++;
    node->left = buildTree(s, index, 2 * curr + 1);

    index++;
    node->right = buildTree(s, index, 2 * curr + 2);


    return node;
}
```

手寫算法(加密)

Let plaintext = IaBZd

tree :



Ciphertext = ILvueoyo中每個字元 +15+(他們在 tree 中的編號)

I+15+0 = 73+15+0 = 88 = X

a+15+1 = 97+16 = 113 = q

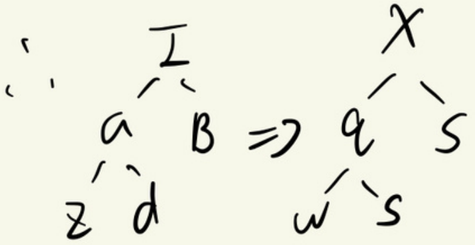
B+15+2 = 66+17 = 83 = S

E+15+3 = 69+18 = 87 = W

d+15+4 = 90+19 = 109

∵ 109 ≠ 字母 ⇒ A+(109−Z)−1 = A+18 = 83 = S

∴



∴ Ciphertext = XQWSS

Key = AZPGm

Key為 AbAAb

(A、b 可為任意相同形態字母)

e.x. GaBZy

返回議程頁面

手寫算法(解碼)

Cipher text = XQWSS

key = AzPGm

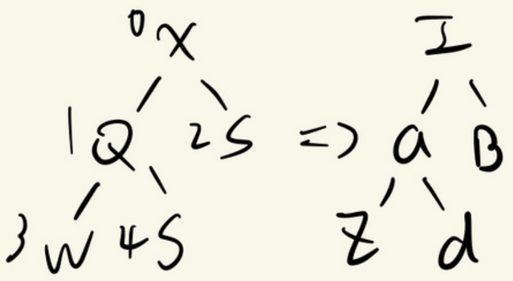
X-15-0=I , key[0] 是大寫 ⇒ I

Q-15-1=A , key[0] 是小寫 ⇒ a

S-15-2=B , key[2] 是大寫 ⇒ B

W-15-3=Z , key[3] 是大寫 ⇒ Z

S-15-4=d , key[4] 是小寫 ⇒ d



A = plaintext = IaBZd