# 中山大学数据科学与计算机学院本科生实验报告

课程名称：**算法设计与分析**　　任课教师：**张子臻**

| 年级 | **2018** | 专业（方向） | **软件工程** |
|---|---|---|---|
| 学号 | **18342075** | 姓名 | **米家龙** |
| 电话 | **18566916535** | Email | **781131011@qq.com** |
| 开始日期 | **2020-05-15** | 完成日期 | **2020-05-17** |

## 1.实验题目

* 1035 DNA matching

* 1198 Substring

## 2.实验目的

* 熟悉贪心算法的使用

* 学会使用 C++的 STL 库，并掌握相关的方法

* 加深数据结构和 C++的理解

## 3.程序设计

### 1035 DNA matching

思路：

使用循环。对于每一个 DNA 单链，在所有的未被匹配到的 DNA 单链集

合中进行寻找，若找到匹配的，则将匹配到的单链标记为已匹配，剔除该集合；若未找到匹配单链，则将该单链加入到该集合。

该过程代码：

```
int matchDNA(int n)
{
  static char strTemp[127]; // 用于获取目标匹配的单链
  int res = 0;              // 结果
  multiset<string> strSet;  // 储存未被匹配的单链
  string str;               // 输入的单链
  string rts;               // 匹配的单链

  strTemp['A'] = 'T';
  strTemp['T'] = 'A';
  strTemp['C'] = 'G';
  strTemp['G'] = 'C';

  for (int i = 0; i < n; i++)
  {
    str = "";
    rts = "";
    cin >> str;
    for (int j = 0; j < str.size(); j++)
    {
      rts += strTemp[str[j]]; // 获取到目标匹配的单链
    }
    if (strSet.count(rts)) // 如果存在目标单链，则在集合中删去并且结果+1
    {
      res++;
      strSet.erase(strSet.find(rts));
    }
    else
    {
      strSet.insert(str); // 否则插入
    }
  }

  return res;
}
```

## 1198 Substring

思路：

通过枚举，能够比较简单得得到结果。但需要注意的是，该解法是 O (n!) 的时间复杂度，在数据量庞大的情况下，程序运行状况会十分糟糕。

改进思路：

在组合过程中，将前面的字符串拼接而成得到的新字符串 A，将要与 A 拼接得到新字符串的字符串 B，如果 A+B > B+A，那么该循环过程可以直接停止，避免后续的无用计算；当 A+B < 当前最小的字符串，同样终止当前循环。

当前过程代码：

```
string minString(string strList[], int length)
{
  string res = "", tmp = "";
  for (int i = 0; i < length; i++) // 初始值
  {
    res += strList[i];
  }
  while (next_permutation(strList, strList + length)) // 全排列函数，方便组合
  {
    bool flag = true;
    tmp = "";
    for (int i = 0; i < length; i++)
    {
      if (strList[i] + tmp < tmp + strList[i] || tmp + strList[i] > res) // 这种情
况下，再继续拼接字符串没有意义了
      {
        flag = false; // 表示是否终止
        break;
      }
      else
      {
        tmp += strList[i];
      }
    }
    if (flag)
    {
      res = res < tmp ? res : tmp;
```

```
    }
  }

  return res;
}
```

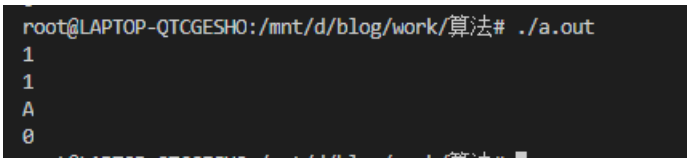
## 4.程序运行与测试

## 1035 DNA matching

样例测试：

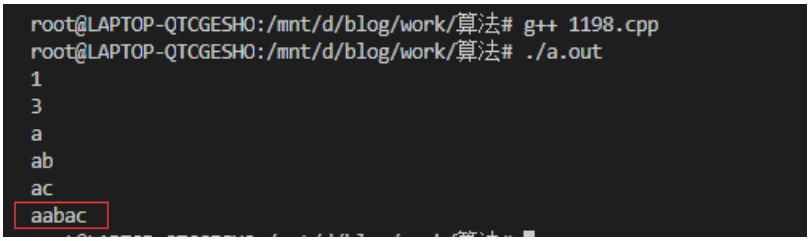

测试 1：


测试 2：

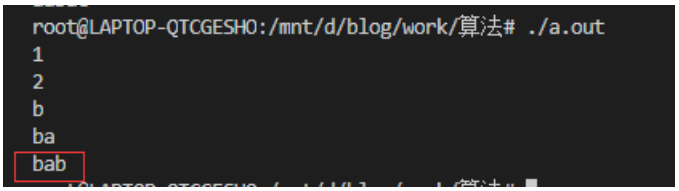


## 1198 SubString

样例测试：




测试 1：




测试 2：


## 5.实验总结与心得

这两题理论上都可以使用简单的循环解决，但是由于时间复杂度的问题，导致计算成本较高，但通过适当地判断来终止循环，或者通过数据结构来加速计算，都能够有效地减少时间成本，从某种意义上来说，这也是一种贪心。

## 附录、提交文件清单

### 1035.cpp

```cpp
#include <iostream>
#include <set>

using namespace std;

int matchDNA(int n)
{
  static char strTemp[127]; // 用于获取目标匹配的单链
  int res = 0;              // 结果
  multiset<string> strSet;  // 储存未被匹配的单链
  string str;               // 输入的单链
  string rts;               // 匹配的单链

  strTemp['A'] = 'T';
  strTemp['T'] = 'A';
  strTemp['C'] = 'G';
  strTemp['G'] = 'C';

  for (int i = 0; i < n; i++)
  {
    str = "";
    rts = "";
    cin >> str;
    for (int j = 0; j < str.size(); j++)
    {
      rts += strTemp[str[j]]; // 获取到目标匹配的单链
    }
    if (strSet.count(rts)) // 如果存在目标单链，则在集合中删去并且结果+1
    {
      res++;
      strSet.erase(strSet.find(rts));
    }
    else
    {
      strSet.insert(str); // 否则插入
    }
  }

  return res;
```

```
}

int main()
{
  int k = 0;
  cin >> k;

  for (int i = 0; i < k; i++)
  {
    // cout << i;
    int n = 0;
    cin >> n;
    cout << matchDNA(n) << endl;
  }

  return 0;
}
```

## 1198.cpp

```
#include <iostream>
#include <algorithm>

using namespace std;

string minString(string strList[], int length)
{
  string res = "", tmp = "";
  for (int i = 0; i < length; i++) // 初始值
  {
    res += strList[i];
  }
  while (next_permutation(strList, strList + length)) // 全排列函数，方便组合
  {
    bool flag = true;
    tmp = "";
    for (int i = 0; i < length; i++)
    {
      if (strList[i] + tmp < tmp + strList[i] || tmp + strList[i] > res) // 这种情
况下，再继续拼接字符串没有意义了
      {
        flag = false; // 表示是否终止
        break;
```

```
      }
      else
      {
        tmp += strList[i];
      }
    }
    if (flag)
    {
      res = res < tmp ? res : tmp;
    }
  }

  return res;
}

int main()
{

  int t = 0;
  cin >> t;
  while (t--)
  {
    int n = 0;
    cin >> n;
    string strList[n] = {};
    for (int i = 0; i < n; i++)
    {
      cin >> strList[i];
    }
    sort(strList, strList + n);
    cout << minString(strList, n) << endl;
  }
}
```