华中科技大学

2022

算法设计与分析实践报告

专　　业：计算机科学与技术

班　　级：计科 2108 班

学　　号：U202115586

姓　　名：殷梓达

完成日期：2022 年 12 月 23 日

目 录

1. 完成情况

1.1 已做的题目

1000 1005 1042 1050 1084 1088 1185 1328 1475 1700 1753 2366 2387 2449 2503 2586 3040 3169 3233 3295 3660 3714 1636 1201 1062

共计 25 题。

1.2 AC 的题目

1000 1005 1042 1050 1084 1088 1185 1328 1475 1700 1753 2366 2387 2449 2503 2586 3040 3169 3233 3295 3660 3714

共计 22 题在 poj 平台上 AC。

图 1：AC 的题目

1.3 已做但未 AC 的题目

1201 题目已做但是在 poj 平台上提交后 TLE 因此未 AC。 1636 题及 1062 题未通过。

2. 3295 解题报告

2.1 题目分析

WFF 'N PROOF 是一款用骰子玩的逻辑游戏。每个骰子有六个面，代表可能符号 K、A、N、C、E、p、q、r、s、t 的某个子集。格式良好的公式 (WFF)

是这些符号的任何字符串，它们遵循以下规则：

（1）p、q、r、s 和 t 是 WFF

（2）如果 w 是 WFF，则 N w 是 WFF

（3）如果 w 和 x 是 WFF，则 K wx、A wx、C wx 和 E wx 是 WFF。

WFF 的含义定义如下：

（1）p、q、r、s 和 t 是逻辑变量，它们的值可能为 0（假）或 1（真）。

（2）K、A、N、C、E 表示和，或，不，暗示和等于。

重言式是不管其变量的值的具有值 1（真）WFF。例如，ApNp 是一个重言式，因为无论 p 的值如何，它都为真。另一方面，ApNq 不是，因为对于 p=0, q=1，它的值为 0。

第一眼看到这道题脑子里想到了使用栈来做，即遍历 WFF 式，若遇到变量的枚举值则将其入栈，若遇到运算符则根据其是否为一目或多目运算符从栈中弹出数字进行运算，后将结果重新入栈。

2.2 算法设计

用一个栈 S1 来保存变量，s1len 表示栈顶的位置，从最后一个字向前遍历如果遇到变元就将其入栈，遇到运算符就将弹出一个或两个变元参与运算，将结果压入栈中。如果最终栈中只有一个变元且值为 1 说明是重言式。

遍历每个句子的所有的变元取值情况，如果每种取值都是重言式，说明这个句子一定是重言式。

2.3 性能分析

对于每个句子都要判断 2^5=32 种情况，每个句子还要遍历一次来判断是不是重言式，因此时间复杂度为 O(n)，n 是句子的长度。

代码段 1: POJ3295

```
#include <iostream>
#include <string.h>
using namespace std;
char str[1000];
int p, q, r, s, t; //枚举五个变量 p,q,r,s,t
```

```
int f()
{
    bool S1[100]; //构造一个栈存储变量
    int s1len = 0;
    int len = strlen(str);
    for (int i = len - 1; i >= 0; i--)
    {
        switch (str[i])
        {
        case 'p':
            S1[s1len++] = p;
            break;
        case 'q':
            S1[s1len++] = q;
            break;
        case 'r':
            S1[s1len++] = r;
            break;
        case 's':
            S1[s1len++] = s;
            break;
        case 't':
            S1[s1len++] = t;
            break;
        case 'K':
            s1len--;
            S1[s1len - 1] = (S1[s1len] && S1[s1len - 1]);
            break;
        case 'A':
            s1len--;
            S1[s1len - 1] = (S1[s1len] || S1[s1len - 1]);
            break;
        case 'N':
            S1[s1len - 1] = (S1[s1len - 1] == 1) ? 0 : 1;
            break;
        case 'C':
            s1len--;
            S1[s1len - 1] = (S1[s1len] == 0 || S1[s1len - 1] == 1);
            break;
        case 'E':
            s1len--;
            S1[s1len - 1] = (S1[s1len] == S1[s1len - 1]);
            break;
```

```
        }
    }

    return S1[0] && s1len == 1;
}
int solve()
{
    p = q = r = s = t = 0;
    for (p = 0; p <= 1; p++)
    {
        for (q = 0; q <= 1; q++)
        {
            for (r = 0; r <= 1; r++)
            {
                for (s = 0; s <= 1; s++)
                {
                    for (t = 0; t <= 1; t++)
                    {
                        int ans = f();
                        if (ans == 0)
                            return 0;
                    }
                }
            }
        }
    }
    return 1;
}
int main()
{
    scanf("%s", &str);
    while (str[0] != '0')
    {
        int ans = solve();
        if (ans)
            cout << "tautology\n";
        else
            cout << "not\n";
        scanf("%s", &str);
    }
    return 0;
}
```

2.4 运行测试

表 1：3295 题测试样例表

测试输入	预期输出	实际输出
ApNp ApNq 0	tautology not	tautology not

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
23904709	yinzida	3295	Accepted	376K	0MS	G++	1963B	2022-12-23 12:06:26

图 2：3295 题测试结果

3. 3233 解题报告

3.1 题目分析

题目大意：已知一个 n*n 的矩阵 A，计算 S = A + A^2 + A^3 + … + A^k，S 的值对 m 取模。

此题可以用分治法。因为 A^6=A^3*A^3。以 S(6)和 S(7)为例。

S(6)=A+A^2+A^3+A^3*(A+A^2+A^3)=(1+A^3)S(3)，

S(7)= A+A*(A+A^2+A^3)+A^4*(A+A^2+A^3)=A+(A+A^4)*S(3)。

因此每次计算都能将幂次降低一半，从而加快运算速度。

3.2 算法设计

使用结构体来保存二维矩阵能够使代码更加简洁。同时重载矩阵之间的加法乘法以及乘方运算，能够使得代码更加易读，当 n 为奇数时，S(n)=(1+A^(n/2))S(n/2)，当 n 为偶数时，S(n)=A+(A+A^((n+1)/2))*S(n/2)，使得时间复杂度降低到 logn 的水平。

3.3 性能分析

因为每次都将问题的规模缩小到一般，因此时间复杂度为 O(logn^2)。n 是矩阵 A 的维度。

代码段 2：POJ3233

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <string.h>
using namespace std;

int n, k, m;
struct node
{
    int matrix[50][50];
} a;

node operator+(node x, node y) //  矩阵 x+矩阵 y
{
    node ans;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            ans.matrix[i][j] = (x.matrix[i][j] + y.matrix[i][j]) % m;
    return ans;
}
node operator*(node x, node y) //  计算矩阵 x*y
{
    node c;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
        {
            int ans = 0;
            for (int p = 1; p <= n; p++) //
            {
                ans += (x.matrix[i][p] * y.matrix[p][j]) % m;
                ans %= m;
            }
            c.matrix[i][j] = ans % m;
        }
    return c;
}
```

```
node func(node x, int i) //  计算矩阵 x^i
{
    node temp, c;
    memset(temp.matrix, 0, sizeof(temp.matrix));
    for (int j = 1; j <= n; j++)
        temp.matrix[j][j] = 1;
    if (i == 0)
        return temp;
    if (i == 1)
        return x;
    c = func(x, i / 2);
    if (i % 2 == 0)
        return c * c;
    else
        return (c * c) * a;
}
node fun(node A, int x) //  计算 a^1+a^2+...+a^k
{
    if (x == 1)
        return A;
    node B = func(A, (x + 1) / 2);
    node C = fun(A, x / 2);
    if (x % 2 == 0)
        return (func(A, 0) + B) * C; // return B+mult(C,B);
    else
        return A + (A + B) * C; // B+mult(C,B)+C;
}
int main()
{
    scanf("%d %d %d", &n, &k, &m);

    int i, j;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &a.matrix[i][j]);
    node ans = fun(a, k);
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            printf("%d ", ans.matrix[i][j]);
        }
        printf("\n");
```

```
    }

    return 0;
}
```

3.4 运行测试

表 2：3233 题测试样例表

测试输入	预期输出	实际输出
2 2 4 0 1 1 1	1 2 2 3	1 2 2 3

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
23904732	yinzida	3233	Accepted	2212K	1688MS	G++	1767B	2022-12-23 12:48:00

图 3：3233 题测试结果

4. 1088 解题报告

4.1 题目分析

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度减小。因此此题目标是求最长下降子序列。

4.2 算法设计

用数组 a 存储每个顶点的高度，用数组 p 存储从该顶点出发的最长下降子序列长度。然后计算得出数组 p 中最大的数值就是所求的结果。

在计算 p[i][j]时，可以采用递归的方式，如果 p[i][j]大于零，则不需要计算。否则递归计算从该点向上下左右出发得到的最长下降序列长度，计算最大长度作为该点的 p[i][j]。

4.3 性能分析

因为是递归计算 p[i][j]，而且如果某一个点周围的点已经求解结束，那个这个点求解就会只需要一步计算。大大简化计算量。时间复杂度为 O(mnlog(mn))

代码段 3：POJ1088

```
#include <iostream>
#include <cmath>
using namespace std;
#define inf 999999
int a[105][105];
int p[105][105];
int r, c;
bool judge(int i, int j) //边缘判断
{
    if (i >= 1 && i <= r && j >= 1 && j <= c)
    {
        return 1;
    }
    else
        return 0;
}

int F(int i, int j) //判断（i,j)的最长路径
{
    if (p[i][j] > 0)
        return p[i][j];
    int shang = 1, xia = 1, zuo = 1, you = 1;
    if (judge(i - 1, j) && a[i - 1][j] < a[i][j])
    {
        shang = 1 + F(i - 1, j);
    }
    if (judge(i + 1, j) && a[i + 1][j] < a[i][j])
    {
        xia = 1 + F(i + 1, j);
    }
    if (judge(i, j - 1) && a[i][j - 1] < a[i][j])
    {
        zuo = 1 + F(i, j - 1);
    }
    if (judge(i, j + 1) && a[i][j + 1] < a[i][j])
    {
```

```
            you = 1 + F(i, j + 1);
        }
        int ans = max(shang, xia);
        ans = max(ans, zuo);
        return ans = max(ans, you);
}
int main()
{
        cin >> r >> c;
        for (int i = 1; i <= r; i++)
        {
            for (int j = 1; j <= c; j++)
            {
                cin >> a[i][j];
            }
        }
        int h = 0;
        for (int i = 1; i <= r; i++)
        {
            for (int j = 1; j <= c; j++)
            {
                p[i][j] = F(i, j);
                h = max(h, p[i][j]);
            }
        }
        cout << h;
        return 0;
}
```

4.4 运行测试

表 3：1088 题测试样例表

测试输入	预期输出	实际输出
5 5 1 2 3 4 5 16 17 18 19 6 15 24 25 20 7 14 23 22 21 8 13 12 11 10 9	25	25

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
23904822	yinzida	1088	Accepted	508K	125MS	G++	1226B	2022-12-23 14:24:40

图 4：1088 题测试结果

5. 1042 解题报告

5.1 题目分析

一个人打算在编号 1~n 的湖里钓鱼，钓鱼是单向走的，不能往回走。给你 n 个湖，每个湖初始鱼的数量 pi，每次每个湖钓鱼后鱼的减少量 di，第 i 个湖到第 i+1 湖的距离时间 ti（单位是 5min），可以在任何湖停止钓鱼。求如何钓鱼才能在 h 小时内钓鱼量最多。输出在每个湖钓鱼的时间。相同钓鱼量情况下，输出湖编号小的用时多的时间。

此题使用贪心算法。使用 maxans 保存最大捕获量，maxk 保存第 i 个湖时的得到最大捕获量。计算 n 次，每一次计算添加一个湖，如果计算出最大捕获量，就更新 maxans 和 maxk 值。

在计算前 n 个湖获得的最大捕获量时，每次都选择 5 分钟捕获量最大的湖而与湖的顺序无关。因此不妨使用优先队列保存湖的数据，每次计算弹出单位捕获量最大的湖，然后将该湖的捕获量减下降速率，存入优先队列。直到时间减少为 0 得到的数据就是前 i 个湖的最大捕获量。

5.2 算法设计

使用结构体来保存二维矩阵能够使代码更加简洁。同时重载矩阵之间的加法乘法以及乘方运算，能够使得代码更加易读，当 n 为奇数时，S(n)=(1+A^(n/2))S(n/2)，当 n 为偶数时，S(n)=A+(A+A^((n+1)/2))*S(n/2)，使得时间复杂度降低到 logn 的水平。

5.3 性能分析

每次计算时都加一个湖，因此时间复杂度为 O(n^2)。n 是湖的数量。

代码段 4：POJ1042

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#include <cmath>
using namespace std;

int h; //  表示小时数
int n; //  表示湖泊的数量
struct pool
{
    int id; //  表示湖泊的 id
    int fi; //  最初五分钟内的捕获鱼的数量
    int di; //  减小的速率
    friend bool operator<(pool a, pool b)
    {
        if (a.fi == b.fi)
            return a.id > b.id;
        return a.fi < b.fi;
    }
} pool[30];          //  记录每个湖的数据
int ti[30];          //  表示从湖泊 i 到湖泊 i+1 所需的时间
int times[30][30]; //  记录每个湖的钓鱼时间
int main()
{
    cin >> n;
    while (n)
    {
        memset(times, 0, sizeof(times));
        cin >> h;
        h = h * 12; //  表示分钟数除以 5，总的时间

        for (int i = 1; i <= n; i++)
        {
            cin >> pool[i].fi;
            pool[i].id = i;
        }
        for (int i = 1; i <= n; i++)
            cin >> pool[i].di;

        for (int i = 1; i < n; i++)
            cin >> ti[i];
```

```
        int maxans = 0;
        int maxk = 1;
        for (int i = 1; i <= n; i++) //  枚举前 i 个湖泊
        {
            int tc = 0;
            for (int j = 1; j < i; j++)
                tc += ti[j];
            priority_queue<struct pool> p;
            for (int k = 1; k <= i; k++)
                p.push(pool[k]);
            int ans = 0;
            int t = h - tc;
            for (int j = 1; j <= t; j++)
            {
                struct pool fo = p.top();
                ans += fo.fi;
                times[i][fo.id] += 5;
                p.pop();
                struct pool pl = {fo.id, max(fo.fi - fo.di, 0), fo.di};
                p.push(pl);
            }
            if (maxans < ans)
            {
                maxans = ans;
                maxk = i;
            }
        }
        for (int i = 1; i < n; i++)
            printf("%d, ", times[maxk][i]);
        printf("%d\n", times[maxk][n]);
        printf("Number of fish expected: %d\n\n", maxans);

        cin >> n;
    }
    return 0;
}
```

5.4 运行测试

表 4：1042 题测试样例表

测试输入	预期输出	实际输出
2 1 10 1 2 5 2 4 4 10 15 20 17 0 3 4 3 1 2 3 4 4 10 15 50 30 0 3 4 3 1 2 3 0	45, 5 Number of fish expected: 31 240, 0, 0, 0 Number of fish expected: 480 115, 10, 50, 35 Number of fish expected: 724	45, 5 Number of fish expected: 31 240, 0, 0, 0 Number of fish expected: 480 115, 10, 50, 35 Number of fish expected: 724

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
23904855	yinzida	1042	Accepted	412K	313MS	G++	1887B	2022-12-23 14:56:25

图 5：1042 题测试结果

6. 总结

6.1 实验总结，可以写几点做题中学到程序技术和技巧

首先是学习了动态规划，贪心算法，最短路，最大流，以及搜索算法，使我解决新问题是的方法更加多样，也能有更优秀的算法来结局问题。在动态规划分治法甚至暴力求解方法解决问题时，多次使用递归的方法求解问题，使我对递归有了更深的理解对函数调用也有了更清晰的认识。

在解决动态规划问题时，会应用到使用二进制表示的情况，但是对于这种问题我需要较长时间 debug，因此我会加强这方面的练习，从而更加熟练。

在解决算法问题时，无论是贪心算法还是动态规划，都是先解决小规模的问题，然后扩大到解决大规模的问题，最终求解出所有问题，是我的思维方式也发

生了转变，我学会了先思考小规模的问题，然后得到的数据求解剩余问题。这种求解方式一般都是牺牲空间换取时间。

6.2 心得体会和建议

我在算法实验课能学到非常多的知识，对我未来的学习以至于工作都有十分重要的意义，在解决问题中，我能够回忆起做过的练习然后思考他们之间的共同点，最终解决新的问题。

第一章就是普通的算法题，第二章是分治算法，基本都用到了分治或者哈希的方法。其中 3714 题与书上的求解最小距离很像，应用排序以及分治的方法，但是我也调试了很久才做出来。第三章的 1088 题，我认为是最简单的动态规划题了，思路最清晰实现也比较简单。第四章贪心算法做起来比动态规划简单一下，但是要对题目以及数据做一些处理，有时候还需要一些特殊的数据结构，使我意识到应该重新学习一下数据结构，这样在遇到相关的题目时能够游刃有余的选择合适的数据结构，从而简化问题。

算法题完成后，我认识到了注释和在完成题目后做题解的重要性。时间久了之前完成过的代码细节大多都记不住了，而通过题解，哪怕是在代码上方通过注释小小的标注一下做题思想，再结果各语句的注释，重新理解自己做过的题目就不难了。之前在完成题目时，仅仅是按心情有选择地实现了一些题解，在最近重温题目完成实验报告的过程中，实现过题解的题目和未实现过题解的题目在题目的熟悉程度上有着天壤之别。

针对本次算法实验，我的建议是是否可以针对每章的经典算法例题提供一份详细的题解，因为各章的题目基本上都是各章知识的实现，若能在做题之前对相关算法实现有一个较为清晰的认识，可能在实现过程中对它的理解会更深。

感谢在实验中帮助过我的老师、助教和同学们。