华中科技大学

2023

算法设计与分析实践报告

专　　业：　计算机科学与技术

班　　级：　CS2201

学　　号：　U202215357

姓　　名：　王文涛

完成日期：　2024.1.6

华 中 科 技 大 学 课 程 设 计 报 告

目 录

1. 完成情况

在整个实验中，共做了 25 道题。提交 60 次，一共通过 25 道题。

Compare wwt2333 and GO

Rank:	36421	Solved Problems List
Solved:	25	1000 1005 1042 1050 1062 1088 1185 1201 1324 1328 1380 1700 1753 1860 2228 2366 2387 2503 2506 3040 3169 3233 3295 3660 3714
Submissions:	60	
School:	华中科技大学	
Email:	2380169004@qq.com	

2. POJ 1753 解题报告

2.1 题目分析

题目大意为，有 4*4 的正方形，每个格子要么是黑色，要么是白色，当把一个格子的颜色改变(黑->白或者白->黑)时，其周围上下左右(如果存在的话)的格子的颜色也被反转，问至少反转几个格子可以使 4*4 的正方形变为纯白或者纯黑？

输入：四行四列字符表示棋盘， "b"表示黑子，"w" 表示白子。

输出：如果可以达到目标，输出反转格子次数，否则输出"Impossible"。

2.2 算法设计

使用枚举的思想，对于每个格子，它要么反转 0 次，要么反转 1 次。由于只有 16 个格子,考虑计算 C_{16}^{i}，也就是 16 个选 i 个,选择翻转的棋子用 1 表示，比如要选择第 1 个、第 3 个、第 5 个和第 6 个，那就是 11 0101 的状态。由于只有 16 位，可以用一个 int 整数来对棋盘进行状态压缩，分别使用 0 和 1 表示白子和黑子。

对翻棋子的操作进行打表，用 state[i]来表示对从前到后编号为 i 的棋子进行翻转

代码段 1 POJ2503

```
#include<iostream>
#include <stdio.h>

int field;//将棋盘压缩成一个 int
int state[] = {19, 39, 78, 140, 305, 626, 1252, 2248, 4880, 10016, 20032, 35968, 12544,
29184, 58368, 51200};

void read()
{
	for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < 4; j++)
		{
			field <<= 1;
			if (getchar() == 'b')
				field |= 1;
		}
		getchar();
	}
}

bool check()
{
	return field == 0x0000 || field == 0xFFFF;
}

int minn = 0xFF;
void work()
{
	for (int flip = 0; flip <= 0xFFFF; flip++)
	{
		int temp = field, cnt = 0;
		for (int i = 0; i < 16; i++)
			if (flip & (1 << i))
			{
				field ^= state[i];
				++cnt;
			}
		if (check() && minn > cnt)
			minn = cnt;
		field = temp;
	}
}
```

```
void print()
{
	if (minn == 0xFF)
		puts("Impossible");
	else
		printf("%d\n", minn);
}

int main()
{
	read();
	work();
	print();
	return 0;
}
```

2.3 性能分析

由于输入数据的规模是一定的，所以时间复杂度为 $0(2^{16})$。

2.4 运行测试

User	Problem	Result	Memory	Time	Language	Code Length
wwt2333	1753	Accepted	152K	157MS	C++	981B

3. POJ 1860 题报告

3.1 题目分析

题目大意：找到一个二维数组的最大子数组和。

输入：一个代表数组的大小的数字 N，接着N^2个数字代表数组元素。

输出：最大子数组的和。

3.2 算法设计

使用动态规划。在一个子矩阵中，把同一列的元素相加，当作一个元素使用，然后，二维数组中的问题就回归到了一维数组，这样便减小了问题的规模，所以现在的问题就是，遍历这个矩阵，分析所有的子矩阵，求出所需要的 max。用这样的思想，首先将第一行当作一个求最大子段和的问题，求出第一行的最大值，接着，按次序遍历其他行以及连续行组成，即，1，12，123，1234，23，234…这样用计算得到的最大值不断更新 max，就能获得最终解。

```
#include<iostream>
```

```
int arr[101][101];
int temp_row[101];
int main()
{
    int n;
    std::cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            std::cin >> arr[i][j];
    int sum = -128, temp;
    for (int i = 0; i < n; i++)
    {
        for (int j = i; j < n; j++)
        {
            memset(temp_row,0,sizeof(int)*n);
            temp = 0;
            for (int c = 0; c < n; c++)
                for (int k = i; k <= j; k++)
                    temp_row[c] += arr[k][c];
            for (int k = 0; k < n; k++)
                sum = std::max(sum, temp = temp > 0 ? temp + temp_row[k] :
temp_row[k]);
        }
    }
    std::cout << sum << '\n';
    return 0;
}
```

3.3 性能分析

将二维数组压缩成一维时间复杂度为$O(n^2)$，动态规划求一维数组最大子数组时间复杂度为$O(n)$，所以总的时间复杂度为$O(n^3)$。

3.4 运行测试

4. POJ 3040 解题报告

4.1 题目分析

题目大意：约翰要给他的牛贝西发工资，每天不得低于 C 元，约翰有 N 种面值的钱币，第 i 种的面值为V_i，数量有B_i。问这些钱最多给贝西发多少天的工资。注意，每种面值的金钱都是下一种的面值的倍数。

输入：第一行输入 N 和 C 接下来 N 行输入V_i和B_i。(1 <= N <= 20，1 <= C <= 100,000,000，1 <= V <= 100,000,000，1 <= B <= 1,000,000)。

输出：天数。

4.2 算法设计

使用贪心策略。 分成三部解决。

1. 首先面额大于 C 的硬币属于没办法节约的类型，先统统发掉。
2. 再按面值从大到小取，凑近 C，可以小于等于 C，但不能大于 C。
3. 最后从小到大取，凑满 C，这里的凑满可以等于大于 C。然后将上述 2,3 步取到的面值全部取走，再转入步骤 2，这样每次找到的取法就是当前最优取法，直到所剩下的金币总价值不够 C 结束。

```
#include <iostream>
#include <algorithm>
const int maxn = 21;
int need[maxn];
struct node
{
	int v, b;
	bool operator<(const node &x) const
	{
		return v > x.v;
	}
} coin[maxn];

int main()
{
	int n, c, v, b, ans = 0, m = 0;
	std::cin >> n >> c;
	for (int i = 1; i <= n; i++)
	{
		std::cin >> v >> b;
```

```
            if (v > c) // 面值大于 C 的硬币直接付工资
                ans += b;
            else
                coin[m].v = v, coin[m++].b = b;
        }
        std::sort(coin, coin + m); // 将硬币按面值从大到小排序
        int allowance = c;
        while (true)
        {
            for (int i = 0; i < m; i++) // 从大到小选硬币
            {
                if (allowance > 0 && coin[i].b > 0)
                {
                    int t = std::min(coin[i].b, allowance / coin[i].v);
                    allowance -= t * coin[i].v;
                    coin[i].b -= t;
                }
            }
            for (int i = m - 1; i >= 0; i--) // 从小到大选硬币
                if (allowance > 0 && coin[i].b > 0)
                {
                    int t = std::min(coin[i].b, (int)ceil((double)allowance /
(double)coin[i].v));
                    allowance -= t * coin[i].v;
                    coin[i].b -= t;
                }
            if (allowance > 0)
                break;
            ans++;
            allowance = c;
        }

        std::cout << ans << "\n";
        return 0;
}
```

4.3 性能分析

在最差的情况下，N 种硬币面值都小于 C，排序时间复杂度为$O(\mathrm{NlogN})$，在循环中，每次循环最少使用 2 枚硬币，循环内时间复杂度为$O(\mathrm{N})$,循环的次数最高为 B/2，$\mathrm{B}=\sum \mathrm{B_i}$ ，所以总时间复杂度为$O(\mathrm{NlogN}+\mathrm{NB})$。

4.4 运行测试

User	Problem	Result	Memory	Time	Language	Code Length
wwt2333	3040	Accepted	156K	0MS	C++	1314B

5. POJ 3040 解题报告

5.1 题目分析

题目大意：有给定 n 种货币，且存在 m 种交易方式，对于每一种交易方式，有税率与佣金等参数。例如现存在一种交易方式:从货币 A 到货币 B，原有货币 A 的数量为 Sa,税率为 rate,佣金为 dec，则可得货币 B 的数量为 Sb=(Sa-dec)*rate。问经过交易能否使原有货币价格升高。

输入：第一行为四个数，货币的种类数 N，交换点数目 M，Nick 拥有的钱 S，Nick 拥有的钱的数目 V。接下来 M 行，每行 6 个数字，依次表示货币 A,货币 B, A 到 B 的汇率R_{AB}, A 到 B 的佣金C_{AB}, B 到 A 的汇率 R_{BA} 和 B 到 A 的佣金 C_{BA}。

输出：若 Nick 的财富能增加，输出 YES，否则输出 NO。

5.2 算法设计

对于这些货币以及货币间的交易方式，可以抽象成图的节点和边，每一条边连接两个节点，且存在税率与佣金等参数。那么要寻找到能使手中资金不断升高的交易圈，即在当前图中找到一个最大正环。

使用 Dijkstra 算法采用邻接表存图，定义节点结构体，记录节点编号以及当前节点下能获得的最大货币价值，使优先队列优先弹出当前最大价值节点。检查所有边，若存在一条边使得经过这条边的交易方式后货币价值仍能够提高，则返回真，输出 YES，反之即输出 NO。

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
const int INF = 0x3f3f3f3f;
const int N = 500;
struct edge
{
    int u, v;
    double rate, dec;
    edge(int a, int b, double c, double d)
```

```
    {
        u = a;
        v = b;
        rate = c;
        dec = d;
    }
};
vector<edge> G[N]; // 邻接表存图
struct s_node
{
    int id;
    double val;
    s_node(int a, double b)
    {
        id = a;
        val = b;
    }
    bool operator<(const s_node &a) const
    {
        return val < a.val;
    }
};
int n, m, s;
double V;
bool dijkstra()
{
    double dis[N]; // 记录所有点到起点能够产生的最大价值
    bool vis[N];
    for (int i = 1; i <= n; i++)
    {
        // 与最短路相反，初始化定义为 0，即当前无价值
        dis[i] = 0;
        vis[i] = false;
    }
    // 对于起点，本存在有价值 V
    dis[s] = V;
    priority_queue<s_node> q;
    q.push(s_node(s, dis[s]));
    while (!q.empty())
    {
        // 优先弹出目前已产生价值中有最大价值的节点
        s_node u = q.top();
        q.pop();
```

```
        if (vis[u.id])
            continue;
        vis[u.id] = true;
        // 检查从该节点出发的所有交易方式
        for (int i = 0; i < G[u.id].size(); i++)
        {
            // 交易方式 y
            edge y = G[u.id][i];
            if (vis[y.v])
                continue;
            // 通过 y 的交易之后若得到更大价值，则拓展新邻居
            if (dis[y.v] < (u.val - y.dec) * y.rate)
            {
                dis[y.v] = (u.val - y.dec) * y.rate;
                q.push(s_node(y.v, dis[y.v]));
            }
        }
    }
    // 检查每一条边
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j < G[i].size(); j++)
        {
            edge x = G[i][j];
            if ((dis[x.u] - x.dec) * x.rate > dis[x.v])
                return true;
        }
    }
    return false;
}
void solve()
{
    cin >> n >> m >> s >> V;
    for (int i = 1; i <= m; i++)
    {
        int u, v;
        cin >> u >> v;
        double x, y;
        cin >> x >> y;
        G[u].push_back(edge(u, v, x, y));
        cin >> x >> y;
        G[v].push_back(edge(v, u, x, y));
    }
```

```
    if (dijkstra())
        cout << "YES" << endl;
    else
        cout << "NO" << endl;
}
int main()
{
    solve();
    return 0;
}
```

5.3 性能分析

法使用 Dijkstra 算法，产生O(NM)的时间复杂度，而优先队列的排序会产生O(NlogN)的时间复杂度，所以总的时间复杂度为O(NM + NlongN)

5.4 运行测试

User	Problem	Result	Memory	Time	Language	Code Length
wwt2333	1860	Accepted	236K	0MS	C++	1291B

6. 总结

6.1 实验总结

从这次实验中完成的 25 道题中，我学习到了非常多的技术和技巧。比如说状态压缩技术，可以将一个只有两个状态的二维图用 0 和 1 来表示，将每一行都转换成一个整数。还有动态规划时，如果只会用到相邻两个状态。可以只用 dp_0 和 dp_1 来存储，将空间复杂度从 O(n)降为 O(1)。

6.2 心得体会和建议

在这次实验中，我完成了 25 道题，其中有很多题目的难度比较大，经过研究网上其他人发的题解，最后明白了解题的思路。题目与我们学习的各种算法息息相关，经过联系，我对各种算法的理解更加深刻，逐渐能够比较快地判断出问题需要使用的算法。同时在研究他人的题解时，我学习到了许多其他的技巧，可以运用到类似的问题上，让我的解题能力得到了提高。