药店卖药 实验报告

姓名：侯华玮

学号：202102001015

摘要

本实验根据药店卖药的具体情景，建立相应的数据结构，对药店卖药的各个环节进行了模拟。

针对卖药策略问题，先根据一般性的规则进行贪心求得次优解，再使用模拟退火算法对所得解进行迭代更新，最终获得较优的结果。

关键词：模拟，贪心算法，模拟退火算法

第 1 节 实验原理

1.1 实验内容

问题情景

你是一家药店的老板，这个月你从供货商手里收到了一批共 `50` 个药品，其中每个药品有独立的进价和保质期，其中 `进价` 的区间为 `[20,30]` 元，保质期的剩余天数为 `[1, 15]` 天。

你每天可以陈列出 `10` 个药品在商店中售卖，每天会有三个顾客各过来购买一个药品。药品的定价只能为商品进价加上 `{-1.5, -1, -0.5, 0, 2, 4, 6}` 元，不得随意定价。

每位顾客购买的逻辑是，买其中最便宜的药品，如果说最便宜的药品价格一致则购买保质期长的药品。三位顾客会依次购买药品。

药品如果没有陈列在商店中，而是存放在仓库时，会收取管理费，其中保质期低于 5 天的每天收取 1 元管理费，其余的每天收取 0.5 元。

你的目标是，10 天之后，你的利润（售出商品售价总和−售出商品进价总和−支付仓库的管理费用−10 天内过期/丢弃商品的进价）最大。

```C++
g_profit = g_sale_price - g_purchase_price -
           g_warehouse_price - g_delete_expired_price;
```

1.2 算法思路

1 建立相应的数据结构

C++

```
// 药品类与类方法
class Medicine {
private:
    int id;
    int status;
    float origin_price;
    float price;
    int expire_day;
    int delete_day;
public:
    enum {
        IN_STORE = 1,
        IN_WAREHOUSE = 2,
        INIT = 0,
        SOLD = -1,
        DELETED = -2,
        EXPIRED = -3
    };
    Medicine(int id, float origin_price, int expire_day) {
        this->id = id;
        this->origin_price = origin_price;
        this->price = origin_price;
        this->expire_day = expire_day;
        this->status = IN_WAREHOUSE;
        this->delete_day = -1;
    }
...
}
// 策略结构体
struct Strategy{
    int mid;
    int delta_id;
    float delta_price;
};
// 删除结构体
struct Delete{
    int day;
    int mid;
};
```

2 药店模拟流程

- 解析命令行参数，读入并解析相应的文件；
- 逐天进行模拟：

1. 执行删除操作，更新删除损失变量；
2. 选取相应的药品放入药店；
3. 按规则卖出 3 种药品，更新收益变量；
4. 更新管理费用，删除过期药品，更新药品状态；

- 结算总收益。

3 寻找最优卖药策略流程

由于利润为：

售出商品售价总和 – 售出商品进价总和 – 支付仓库的管理费用 – 10 天内过期/丢弃商品的进价

想要提高 10 天内的总利润，直观地想法是提高售出药品的价格，减少仓库管理费，减少过期和丢掉的损失，下面将就此分析一般的规则，用于贪心算法的求解

贪心算法：

主要规则：

1. 尽量少地扔掉药品（包括因过期扔掉与主动扔掉），因为扔掉药品会亏损药品进价和存放该药品的管理费；
2. 对于主动扔掉的药品，一定是在 10 天内一定过期且无法卖出的药品，如：

   有 8 个保质期为 2 天的药品，则 2 天最多只能卖出 $2 \times 3 = 6$ 个保质期为 2 天的药品，所以必须尽早扔掉过期的药品。即选择在第 1 天的开始，就扔掉所有一定会过期的药品。

   一定会过期的药品的选择：对于天数 day ，从第 1 天到第 day 天，最多能卖出 $3 \times day$ 个药品。则分别计算保质期小于等于 day 的药品数 num_{day} ，若 $num_{day} > 3 \times day$ ，则选择价格最高的 $num_{day} - 3 \times day$ 个药品扔掉，即在第一天开始前扔掉。
3. 由于保质期越短，仓库所收管理费越高，且更容易过期。所以率先出售保质期更短的药；
4. 对于保质期相同的药品，率先出原价较低的药品，因为原价较低的药品的价格调控区间更；
5. 对于每一天选择放到药店的药品，根据前面的规则，先选出 3 个优先级最高的药品，作为“推荐出售药品”；再选择 7 个“补充药品”，补充药品的定价都为最高定价，以此有利于提高“推荐出售药品”的售价和出售机会；

算法流程：

- 初始化 `g_strategy` 策略全局变量，`store_medp` 药店内药品向量等；
- 计算求得一定会过期的药品，在第一天开始前扔掉，并更新 `g_delete_expired_price` ；
- 进入每一天的贪心与模拟中：

    - 根据上述规则，选出“推荐出售药品”，放入药店；
    - 根据上述规则，选出“补充药品”，放入药店；
    - 根据上述规则，对药店中的药品进行定价，并记录定价策略；
    - 根据顾客买药规则，选择并卖出 3 个药品，更新 `g_purchase_price` 和 `g_sale_price`；
    - 更新管理费 `g_warehouse_price`；
    - 检查所有剩余药品的保质期，扔掉过期药品，更新 `g_delete_expired_price`；
- 贪心模拟完毕，计算最终收益，获得次优解。

模拟退火算法

模拟退火算法是一种基于随机搜索的全局优化算法，用于在复杂问题中寻找近似最优解。针对贪心算法获得的次优解进行模拟退火算法的执行流程如下：

1. **初始化参数**：设定初始温度 t 和初始解，通常初始解是随机生成的，此处使用贪心算法获得的次优解。初始解对应问题的一个解决方案，即药店卖药的上架与定价策略，初始温度决定了在搜索过程中允许的状态变化程度。
2. **随机调整解**：在模拟退火的核心过程中，通过随机操作生成新的解，然后计算新状态下的盈利情况。这些随机操作有两个维度：
    - 交换任意两天确定要售出的药品中的一个。
    - 在一天中选择确定要售出的药品和放置在仓库里的药品或上架但不售出的药品进行交换。在进行交换时要考虑药品的保质期限制，以避免产生新的需要丢弃的药品。同时，按照贪心解的方法重新确定药品的定价以及上架方案。
3. **计算收益情况**：计算新状态的收益情况，类似于模拟中部分计算收益的情况。收益情况的计算与具体问题相关，对于药店问题，可以根据药品的售价和进价以及管理费用来计算收益。
4. **选择接受新状态**：根据收益的变化情况来选择是否接受新状态。
    - 如果新状态的收益较原先状态更高，则一定接受新状态。
    - 如果新状态的收益较原先状态较低，根据一个概率来选择是否接受新状态。这个概率通过计算指数函数 $e^{\frac{d}{t}}$ 得到，其中 d 表示新状态与原先状态的收益差异，t 表示当前的温度。
5. **降温**：在每次迭代后降低温度。温度的降低可以使用线性递减或指数递减方式。降温的目的是逐渐减小接受较差解的概率，使算法逐步趋向于收敛到全局最优解。
6. **重复迭代**：重复执行步骤 2 到步骤 5，直到达到最低温度。最低温度是算法停止的条件，可以是事先设定的一个阈值或者迭代次数。

7. **进一步逼近全局最优解**：为了进一步提高结果的质量，可以进行多次模拟退火过程，每次使用不同的初始解，并记录每次模拟退火的最优状态。最后从多次迭代中选择收益最高的状态作为近似的全局最优解。

模拟退火算法通过随机搜索和接受差解的策略，允许在搜索空间中跳出局部最优解，以期望找到更好的解决方案。随着温度的降低，算法逐渐趋于稳定，最终收敛到全局最优解或近似最优解。

第 2 节 实验结果

2.1 实验结果展示

1 药店模拟流程测试结果

测试集	结果
Data1	−181.5000
Data2	−254.8684
Data3	−254.0029
Data4	−341.8511
Data5	−345.7287
Data6	−417.6857
Data7	−435.3131
Data8	−430.9775
Data9	−494.2748
Data10	−546.2876

2 寻找最优卖药策略流程结果

测试集	教辅收益	我的收益	收益差
Data1	45.5	45	-0.5
Data2	44.5	42.5	-2
Data3	45.5	4	-5.5
Data4	42.5	43	0.5
Data5	41.5	38.5	-2
Data6	41	21	-20
Data7	36.5	35	-1.5
Data8	32	32	0
Data9	6.0688	5.5	-0.5688
Data10	-12.2548	-5.7548	6.5

2.2 实验结果分析

第 3 节 源代码解析

1 药店模拟流程测试

C++

```
int sim_day(int day) {
#ifdef DEBUG
    cout << "------- Sim day: " << day << " ------" << endl;
#endif
    // delete medicine
    for (auto d_mid : g_delete[day]) {
        auto it = g_medicine.begin();
        while (it != g_medicine.end()) {
            if (it->get_id() == d_mid) {
                if (it->get_status() == Medicine::SOLD) {
                    cout << "delete sold medicine" << endl;
                    return 0;
                }
                if (it->get_status() == Medicine::DELETED) {
                    cout << "delete deleted medicine" << endl;
                    return 0;
                }
                if (it->get_status() == Medicine::EXPIRED) {
                    cout << "delete expired medicine" << endl;
                    return 0;
                }
                it->set_status(Medicine::DELETED);
```

```
                it->set_delete_day(day);
                g_delete_expired_price += it->get_origin_price();
#ifdef DEBUG
                cout << "- delete medicine:" << endl;
                it->print();
#endif
                it = g_medicine.erase(it);
            }
            else {
                ++it;
            }
        }
    }
    // choose medicine to sell on the store
    vector<Medicine*> medicine_p_in_store;
    for (auto strategy : g_strategy[day]) {
        int mid = strategy.mid;
        float delta_price = strategy.delta_price;
        for (auto& medicine : g_medicine) {
            if (medicine.get_id() == mid) {
                medicine.set_price_delta(delta_price);
                medicine.set_status(Medicine::IN_STORE);
                medicine_p_in_store.push_back(&medicine);
                break;
            }
        }
    }
    // sort medicine_in_store
    sort(medicine_p_in_store.begin(), medicine_p_in_store.end(), buyer_cmp_price_ptr);
#if DEBUG
    cout << "medicine_p_in_store: " << endl;
    cout << "medicine_p_in_store size: " << medicine_p_in_store.size() << endl;
    for (auto medicine : medicine_p_in_store) {
        medicine->print();
    }
#endif
    int medicine_in_store_num = medicine_p_in_store.size();
    // sell medicine
    for (int i = 0; (i < CUSTOMER_NUM) && (i < medicine_in_store_num); i++) {
        medicine_p_in_store[i]->set_status(Medicine::SOLD);
        // medicine_p_in_store[i]->set_sale_day(day);
        g_sale_price += medicine_p_in_store[i]->get_price();
        g_purchase_price += medicine_p_in_store[i]->get_origin_price();
    }

    auto it = g_medicine.begin();
    while (it != g_medicine.end()) {
        if (it->get_status() == Medicine::SOLD) {
            // print_g_medicine();
#ifdef DEBUG
            cout << "- erase sold medicine:" << endl;
            it->print();
#endif
            it = g_medicine.erase(it);
        }
```

```
        else {
            if (it->get_status() == Medicine::IN_WAREHOUSE) {
                // update g_warehouse_price
                float warehouse_price = it->get_expire_day() - day < 5 ? 1 : 0.5;
                g_warehouse_price += warehouse_price;
            }
            else if (it->get_status() == Medicine::IN_STORE) {
                it->set_status(Medicine::IN_WAREHOUSE);
            }

            if (it->get_expire_day() == day) {
                it->set_status(Medicine::EXPIRED);
#ifdef DEBUG
                cout << "- erase expired medicine:" << endl;
                it->print();
#endif
                g_delete_expired_price += it->get_origin_price();
                it = g_medicine.erase(it);
            }
            else {
                ++it;
            }
        }
}
#ifdef DEBUG
    cout << "------ end sim day: " << day << " -------" << endl;
#endif
    return 1;
}
```

2 寻找最优卖药策略流程

C++

```
int greedy() {
    // init g_strategy
    for (int day = 1; day <= SIM_DAY_NUM; day++) {
        for (int i = 0; i < STORE_SIZE; i++) {
            g_strategy[day][i].mid = -1;
            g_strategy[day][i].delta_id = INT_MAX;
            g_strategy[day][i].delta_price = INT_MAX;
        }
    }
    /*
    每天选出前3个推荐卖出并放到药店中，从剩余药品中选出价格最高的7个放到药店中（如果还有的话），称为补充药品
    定价规则：
    1. 记补充药品的最低价为min_origin_price
    2. 尝试对每一个推荐卖出药品进行定价：
        2.1 delta_price 为使 推荐药品price < min_origin_price 的最大的delta_price
        2.2 如果不存在这样的delta_price，则 delta_price = MIN delta_price
    3. 所有补充药品定价规则：delta_price = MAX delta_price
    4. 将在商店中的药品按照价格从高到低排序，价格相同，expire_day大的排在前面
```

```
    5. 根据题目规则，应卖出前3个药品，计算相应收益损失
    6. 遍历g_medicine，1将卖出的删除，2更新g_warehouse_price，3将过期的删除，4将在商店中的药品
放回仓库
    */
    // 仅维护一个全局的g_medicine，其他都用指针引用
    vector<Medicine*> store_medp;
    vector<Medicine*> warehouse_medp;
    vector<Medicine*> expired_medp;
    vector<Medicine*> deleted_medp;
    int near_expire_day = 1;
    // 初始化xxx_medp
    for (int i = 0; i < MEDICINE_NUM; i++) {
        warehouse_medp.push_back(&g_medicine[i]);
    }
    // 1. 每天选出前3个推荐卖出并放到药店中，从剩余药品中选出价格最高的7个放到药店中（如果还有的
话），称为补充药品
    for (int day = 1; day <= SIM_DAY_NUM; day++) {

        // cout << "Greedy day: " << day << endl;
        // 清空store_medp
        store_medp.clear();
        /* 1.1 选出前3个推荐卖出并放到药店中 */
        /* 1.1 选出3个推荐卖出：先在expire_day小于等于10中的选，如果不够，再选*/
        // sort warehouse_medp by expire_day, if expire_day equal, smaller origin_pric
e first
        sort(warehouse_medp.begin(), warehouse_medp.end(), cmp_expire_day_ptr);
#ifdef DEBUG
        cout << "warehouse_medp sorted by expire_day: " << endl;
        for (int i = 0; i < (int)warehouse_medp.size(); i++) {
            warehouse_medp[i]->print();
        }
#endif
        // choose the first 3 med put into store
        for (int i = 0;
            (i < 3) && (i < (int)warehouse_medp.size());
            i++)
        {
            store_medp.push_back(warehouse_medp.front());
            warehouse_medp.erase(warehouse_medp.begin());
        }
        int recommend_num = store_medp.size();
#ifdef DEBUG
        cout << "chooe 3 in store_medp: " << endl;
        for (int i = 0; i < (int)store_medp.size(); i++) {
            store_medp[i]->print();
        }
        cout << "delete first 3 in warehouse_medp: " << endl;
        for (int i = 0; i < (int)warehouse_medp.size(); i++) {
            warehouse_medp[i]->print();
        }
#endif
        /* 1.2 从剩余药品中选出原价格最高的7个放到药店中（如果还有的话），称为补充药品
                同价格选expire_day小的（商家）
        */
       /* new 1.2 选出7个补充药品
```

```
        推荐药品最大原价格为max_rec_oprice
        从剩余药品中的，原价格 > max_rec_oprice 的，选择expire_day小的
        如果不够，就不选了
        */



        // sort warehouse_medp by origin_price
        sort(warehouse_medp.begin(), warehouse_medp.end(), seller_cmp_oprice_ptr);
#ifdef DEBUG
        cout << "warehouse_medp sorted by origin_price: " << endl;
        for (int i = 0; i < (int)warehouse_medp.size(); i++) {
            warehouse_medp[i]->print();
        }
#endif
        // choose the first 7 med put into store
        // float min_origin_price_in_warehouse = warehouse_medp.back()->get_origin_pri
ce();
        for (int i = 0;
            (i < 7) && (i < (int)warehouse_medp.size());
            i++)
        {
            store_medp.push_back(warehouse_medp.front());
            warehouse_medp.erase(warehouse_medp.begin());
        }
        float min_oprice_supplement = store_medp.back()->get_origin_price();
        int store_med_num = store_medp.size();


        // int supplement_num = store_medp.size() - recommend_num;
#ifdef DEBUG
        cout << "chooe 7 in store_medp: " << endl;
        for (int i = 0; i < (int)store_medp.size(); i++) {
            store_medp[i]->print();
        }
        cout << "delete first 7 in warehouse_medp, size: " << warehouse_medp.size() <<
endl;
        for (int i = 0; i < (int)warehouse_medp.size(); i++) {
            warehouse_medp[i]->print();
        }
#endif


        // 2. 尝试对每一个推荐卖出药品进行定价：
        // 2.1 delta_price 为使 推荐药品price < min_origin_price + MAX delta price 的最大
的delta_price
        // 2.2 如果不存在这样的delta_price，则 delta_price = MIN delta_price
        float max_delta_price = g_delta_price[PRICE_NUM - 1];
        for (int i = 0; i < recommend_num; i++) {
            Medicine* rec_medp = store_medp[i];
            for (int delta_id = PRICE_NUM - 1; delta_id >= 0; delta_id--) {
                float delta_price = g_delta_price[delta_id];
                float price = rec_medp->get_origin_price() + delta_price;
                // cout << "price: " << price << ", min_origin_price_in_warehouse: " <
< min_oprice_supplement << ", max_delta_price: " << max_delta_price << endl;
```

```
                if (price < min_oprice_supplement + max_delta_price) {
                    rec_medp->set_price_delta(delta_price);
                    // set g_strategy
                    g_strategy[day][i].mid = rec_medp->get_id();
                    g_strategy[day][i].delta_id = delta_id;
                    g_strategy[day][i].delta_price = delta_price;
                    break;
                }
            }
            if (g_strategy[day][i].delta_id == INT_MAX) {
                printf("Error: cannot find delta_price for medicine %d\n", rec_medp->g
et_id());
                rec_medp->set_price_delta(max_delta_price);
                // set g_strategy
                g_strategy[day][i].mid = rec_medp->get_id();
                g_strategy[day][i].delta_id = PRICE_NUM - 1;
                g_strategy[day][i].delta_price = max_delta_price;
                // rec_medp->set_price_delta(g_delta_price[0]);
                // // set g_strategy
                // g_strategy[day][i].mid = rec_medp->get_id();
                // g_strategy[day][i].delta_id = 0;
                // g_strategy[day][i].delta_price = g_delta_price[0];
            }

        }
        // 3. 所有补充药品定价规则：delta_price = MAX delta_price
        for (int i = recommend_num; i < store_med_num; i++) {
            Medicine* sup_medp = store_medp[i];
            sup_medp->set_price_delta(max_delta_price);
            // set g_strategy
            g_strategy[day][i].mid = sup_medp->get_id();
            g_strategy[day][i].delta_id = PRICE_NUM - 1;
            g_strategy[day][i].delta_price = max_delta_price;
        }

        // 4. 将在商店中的药品按照价格从高到低排序，价格相同，expire_day大的排在前面
        sort(store_medp.begin(), store_medp.end(), buyer_cmp_price_ptr);
#ifdef DEBUG
        cout << "4. store_medp sorted by price: " << endl;
        for (int i = 0; i < (int)store_medp.size(); i++) {
            store_medp[i]->print();
        }
#endif
        // 5. 根据题目规则，应卖出前3个药品，计算相应收益损失
        for (int i = 0;
            (i < 3) && (i < store_med_num);
            i++)
        {
            Medicine* sold_medp = store_medp[i];
            g_purchase_price += sold_medp->get_origin_price();
            g_sale_price += sold_medp->get_price();
            store_medp.erase(store_medp.begin());
        }
#ifdef DEBUG
        cout << "5. delete first 3 in store_medp: " << endl;
```

```
        for (int i = 0; i < (int)store_medp.size(); i++) {
            store_medp[i]->print();
        }
#endif
        /* 一天结束，更新状态*/
        // 遍历在仓库中的
        auto medpp = warehouse_medp.begin();
        while (medpp != warehouse_medp.end()) {
            // 更新仓库管理费
            float warehouse_price = (*medpp)->get_expire_day() - day < 5 ? 1 : 0.5;
            g_warehouse_price += warehouse_price;

            // 更新过期药品
            if ((*medpp)->get_expire_day() == day) {
                expired_medp.push_back(*medpp);
                g_delete_expired_price += (*medpp)->get_origin_price();
                medpp = warehouse_medp.erase(medpp);
            }
            else {
                medpp++;
            }

        }
        // 遍历在药店中的
        medpp = store_medp.begin();
        while (medpp != store_medp.end()) {
            // 更新过期药品
            if ((*medpp)->get_expire_day() == day) {
                expired_medp.push_back(*medpp);
                g_delete_expired_price += (*medpp)->get_origin_price();
                medpp = store_medp.erase(medpp);
            }
            else {
                // 没过期，移动到仓库
                warehouse_medp.push_back(*medpp);
                medpp++;
            }
        }
        /*将药店中剩余药放回仓库 */
        // cout << "Greedy end day: " << day << endl;
    }

    g_profit = g_sale_price - g_purchase_price - g_warehouse_price - g_delete_expired_
price;
    // set precision
    cout << fixed << setprecision(5);
    cout << "g_sale_price: " << g_sale_price << endl;
    cout << "g_purchase_price: " << g_purchase_price << endl;
    cout << "g_warehouse_price: " << g_warehouse_price << endl;
    cout << "g_delete_expired_price: " << g_delete_expired_price << endl;
    cout << "g_profit: " << g_profit << endl;

    return 1;
}
```

C++

```cpp
// 调整退火的新状态
vector<int> change(vector<int> s) {
    vector<int> new_state = s;
    for (int i = 0; i <= rand() % 5; i++) {
        while (1) {
            int op = rand() % 4;
            if (op == 0) { // 与剩余的药品进行交换
                int x = rand() % 10 + 1, y = rand() % 3, z = rand() % s.size();
                if (g_medicine[s[z]].status >= x && g_medicine[s[x * 3 + y]].status >
10) {
                    swap(new_state[z], new_state[x * 3 + y]);
                    break;
                }
            }
            else { // 内部交换顺序
                int x = rand() % 10 + 1, y = rand() % 3, u = rand() % 10 + 1, v = rand
() % 3;
                if (g_medicine[s[x * 3 + y]].status >= u && g_medicine[s[u * 3 + v]].s
tatus >= x) {
                    swap(new_state[x * 3 + y], new_state[u * 3 + v]);
                    break;
                }
            }
        }
    }
    return new_state;
}

// 模拟退火求解
void simulatedAnnealing() {
    double T = 10000, dT = 0.993, eps = 1e-6;
    vector<int> state; // 初始状态
    // 初始化 state
    // ...
    double ans = calculate(state);
    while (T > eps) {
        vector<int> new_state = change(state);
        double tmp = calculate(new_state);
        double delta = tmp - ans;
        if (delta > 0 || exp(delta / T) * RAND_MAX > rand()) {
            ans = tmp;
            state = new_state;
        }
        T *= dT;
    }
    // 输出结果
    // ...
}
```