图与网络Lab 3：启发式算法程序报告

小组成员：
黎锦灏 518021910771
沈刘 518030910114
程欣雯 518021910031

本次启发式算法实验编写的程序代码均已在GitHub上开源，GitHub repo 链接如下：
https://github.com/ljh2000/MATH6010_Graph-and-Network

目录

一、均匀图分割

1.1、问题描述

Uniform graph partition：

一个完全图有 2n 条边，cost 函数为 $E \to Z^+ \cup \{0\}$

寻找一个均匀图分割，使得 $c([X_0, X_1]) = \sum_{\{u,v\} \in E, u \in X_0, v \in X_1} cost(u, v)$ 最小

1.2、算法流程

Step 1:
生成包含 $2n$ 个节点的完全图，使用邻接矩阵存储边的权重信息

Step 2:
随机生成一个初始结果，计算当前情况下 $c([X_0, X_1])$ 的值

Step 3:
遍历所有分割图中的所有点对（u, v），如果交换点对会使得交换前的值 $c([X_0, X_1])$ 大于交换后的值 $C(X_0 \backslash \{u\} \cup \{v\}, X_1 \backslash \{v\} \cup \{u\})$，则交换点对（$u, v$）。

Step 4:
遍历所有点对，无论怎么交换分割图中的点对都不会使得 $c([X_0, X_1])$ 降低，则找到最优解。

1.3、程序框架

程序主要分为以下几个部分

1. 生成和初始化完全图函数 `initVault()`
   随机生成有 2n 条边的完全图，使用邻接矩阵 `valueTable` 存储每条边的权重。
   随机生成一个分割图，使用两个长度为 `n` 的 `vector` 进行存储

```
void initValue(int n){

    // resize Value Table
    valueTable.resize(2 * n);
    for(int i = 0; i < 2* n; i++ ){
        valueTable[i].resize(2*n);
    }
    // init ValueTable
    for(int i = 0; i < 2 * n; i++){
        valueTable[i][i] = 0;
        for(int j = i + 1; j < 2 * n; j++ ){
            valueTable[i][j] = randomInt(1,10);
            valueTable[j][i] = valueTable[i][j];
        }
    }
    // resize solution
    SolutionSetU.resize(n);
    SolutionSetV.resize(n);

    // init Solution
    for (int i = 0; i < n; i++){
        SolutionSetU[i] = i;
        SolutionSetV[i] = i + n;
    }
}
```

2. 求解函数 `solveUniformGraphPartition()` 和 `needExchange()`
   函数 `solveUniformGraphPartition` 用于求解图分割问题。这个函数中遍历所有点对，如果存在使得 $c([X_0, X_1])$ 降低的情况，则交换当前点对，函数 `needExchange` 用于计算交换当前点对是否会使得 $c([X_0, X_1])$ 降低。

```
void solveUniformGraphPartition(){
    for(int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            if (needExchange(i,j)){
                Exchange(i,j);
            }
        }
    }
}


bool needExchange(int a,int b){
    int now = calculateSum(SolutionSetU[a],SolutionSetV) + calculateSum(SolutionSetV[b],SolutionSetU) - valueTable[SolutionSetU[a]][SolutionSetV[b]] * 2;
    int after = calculateSum(SolutionSetU[a],SolutionSetU) + calculateSum(SolutionSetV[b],SolutionSetV);
    return after < now;
}
```

3. 校验函数 `checkMin()`
   这个函数会再次遍历所有点对，检验是否已经达成均匀图分割，如果函数返回 `false`，说明函数还没执行完成，还有需要交换的点对，则还需执行求解函数 `solveUniformGraphPartition()`

```
bool checkMin(){
    for(int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            if (needExchange(i,j)){
                // cout<<"Not min, exchange "<<SolutionSetU[i]<<" and "<<SolutionSetV[j]<<endl;
                // cout<<"before "<<sum()<<endl;
                // Exchange(i,j);
                // cout<<"after "<<sum()<<endl;
                return false;
            }
        }
    }
    // cout <<"MIN !"<<endl;
    return true;
}
```

4. 主函数 `main()`
   主函数用于接收用户输入，生成初始化图分割，执行求解函数的功能。如果用户没有指定n的大小，则程序自动指定 `n` 的大小为10。

```
int main(int argc, char** argv){
    srand(time(0));

    if (argc == 1)
        n = 10;
    else
        n = stoi(argv[1]);

    initValue(n);
    printValueTable();

    while(!checkMin()){
        solveUniformGraphPartition();
    }

    printSolution();
    cout << endl ;

    return 0;
}
```

1.4、结果分析

编写批量测试脚本，依此测试n 从1到50的情况

```
# complie
g++ UniformGraphPartition.cpp -o UniformGraphPartition

# test
for num in {1..50}
do
./UniformGraphPartition $num
done
```

运行完测试脚本大约花费 2s 时间，运行截图如下图，经过穷举验算得知，使用启发式算法可以求解出均匀图分割问题的最优解。

```
n = 4, Value Table:
0   4   2   5   4   1   4   7
4   0   3   7   8   3   9   2
2   3   0   3   4   7   7   1
5   7   3   0   4   7   7   4
4   8   4   4   0   5   4   7
1   3   7   7   5   0   7   2
4   9   7   7   4   7   0   9
7   2   1   4   7   2   9   0
solution set U:
5   6   2   3
solution set V:
0   4   1   7
Min CIJ : 67

n = 5, Value Table:
0   4   2   5   4   1   4   7   3   7
4   0   8   3   9   2   3   4   7   7
2   8   0   1   4   7   7   4   5   4
5   3   1   0   7   7   2   9   7   6
4   9   4   7   0   2   1   7   7   4
1   2   7   7   2   0   7   8   8   7
4   3   7   2   1   7   0   5   7   1
7   4   4   9   7   8   5   0   2   6
3   7   5   7   7   8   7   2   0   3
7   7   4   6   4   7   1   6   3   0
solution set U:
9   7   0   3   4
solution set V:
5   6   2   8   1
Min CIJ : 102
```

二、背包问题

2.1、问题描述

Knapsack problem：
一共有 N 件物品，第 i 件物品的价值为 p_i，重量为 w_i。背包的容量为 b。
计算在总重量不超过背包容量的情况下，能够装入背包物品的最大价值 $\sum_{x_i \in Y} p_i, Y \in X$ 。

2.2、算法流程

贪心算法

将物品价值与重量的比值认为是物品的性价比，即 v_i/w_i,从大到小排序，优先放入性价比高的物品。

动态规划

利用状态转移定义子问题，将庞大的搜索空间转化为第i个物品放入或不放入的两种状态，依据前$i-1$件物品所能带来的最大总价值，从两种状态中选出第i件物品所能带来的最优方案。利用极端情况和逆向枚举，以较低的复杂度得到准确的最优解。

模拟退火算法

Step 1：
随机生成一个满足约束条件（即总重量不超过背包容量）的初始解，计算初始解的总价值。设置初始温度，开始退火。

Step 2：
扰动初始解产生新解，计算新解的总价值。

Step 3:
将新解与当前解进行对比，根据Metropolis准则接受新解。
其中，Metropolis准则指当温度为T时，若新解的总价值高于当前解，则选择接受新解，否则，以$\exp\left(-\frac{V_{\text{new}}-V_{\text{cur}}}{T}\right)$的概率接受新解。接受新解的同时修正当前的总价值，舍弃新解时，便在当前解的基础上继续实验。

Step 4:
按照设定的退火速度开始降温，在温度降至终止温度前，在每个新温度上都进行充分的搜索，即重复Step2及Step3至总价值变化不大或达到一定的次数。

禁忌搜索算法

Step 1:
随机生成一个满足约束条件（即总重量不超过背包容量）的初始解，计算初始解的总价值。

Step 2:
通过一定方式扰动解，随机提取解的部分邻居作为候选集合。
若候选集合中的最优解没有被“禁忌”，且“禁忌表”没有达到规定的长度，则将当前解更新为该最优邻居解，并将该解列入“禁忌表”，若此时“禁忌表”已达到规定的长度，则将最早进入的解出队，再将该解放入“禁忌表”中。若该最优邻居解比最优解的总价值更高，则更新最优解。

Step 3:
以一定的次数重复Step2，若出现当前解比所有的邻居解的总价值都高的情况，可以提前结束算法。

2.3、程序框架

本程序主要分为以下几个模块：

1. 主函数，指定物品个数、背包容量、最大重量和最大价值，依次使用贪心算法、动态规划、模拟退火算法和禁忌搜索算法求解同一个背包问题，对禁忌搜索算法进行参数搜索。在禁忌搜索的参数搜索中，根据参数所带来的计算代价排序，最后得到的是搜索范围内计算量最低的参数。

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description = "KnapSack Problem")
    parser.add_argument('-capacity',default = 50, type = int)
    parser.add_argument('-amount',default = 10, type = int)
    parser.add_argument('-max_val',default = 100, type = int)
    parser.add_argument('-max_weight',default = 30, type = int)
    parser.add_argument('-cooling_ratio',default = 0.95, type = float)
    parser.add_argument('-init_temper',default = 100, type = int)
    parser.add_argument('-terminal_temper',default = 2, type = int)
    parser.add_argument('-epoch',default = 100, type = int)
    args = parser.parse_args()

    knap = Knapsack(args.amount,args.capacity,args.max_weight,args.max_val)

    knap.Greedy()

    knap.DP()

    knap.simulated_annealing(args.cooling_ratio,args.init_temper,args.terminal_temper,args.epoch)

    # Choose Optimal Param for Tabu Search
    best_val = 0
    for epoch in [1000,500,200,100]:
        for near_num in [100,50,20,10,5,3,2,1]:
            for tabu_length in [50,20,10,5,3]:
                print("[epoch]: {},[near_num]:{},[tabu_length]:{}".format(epoch,near_num,tabu_length))
                val = knap.tabu_search(tabu_length,epoch,near_num)
                if val>best_val:
                    best_val = val
                    best_param = [epoch, near_num, tabu_length]

    print("best_param",best_param)
    knap.tabu_search(*best_param)
```

2. 给定物体总数 n 、背包容量 $capacity$、物体重量及价值的范围 max_w, max_v，初始化结构体KnapSack,并随机生成相应的背包问题。

```
class Knapsack(object):
    def __init__(self,amount,capacity,max_weight,max_val):
        self.capacity = capacity
        self.amount = amount
        self.value = np.random.randint(1,max_val,self.amount)
        self.weight = np.random.randint(1,max_weight,self.amount)
```

3. 贪心算法的实现，主要包括计算并排序每个物体的“性价比”。

```
# Greedy Algorithm
def Greedy(self):
    density = -self.value/self.weight
    sorted_ind = np.argsort(density)
    sorted_density = np.sort(density)
    sorted_cum = np.cumsum(sorted_density)
    greed_cap = self.capacity
    for i in range(self.amount):
        if sorted_cum[i]> -greed_cap:
            print('Put in {}th item [Value] {} [Weight] {}'.format(i,self.value[i],self.weight[i]))
            greed_cap -= self.weight[sorted_ind[i]]
        else:
            break

    in_bag_v = self.value[sorted_ind[:i]]
    in_bag_w = self.weight[sorted_ind[:i]]
    print("Greddy_Value",sum(in_bag_v))
```

4. 动态规划的实现，将背包问题分解为子问题递归计算。根据状态转移方程，$dp[i][j]$ 记录了当背包容量为j 时，前i个物品能够带来的最大价值。逐渐减少背包容量j，根据前$i-1$个物品的总价值及第i个物品的重量与价值，判断第i个物品的最优状态。

```
#  Dynamic Process
def DP(self):
    dp = np.zeros(self.capacity+1)
    for i in  range(self.amount):
        for j in range(self.capacity,self.weight[i],-1):
            dp[j] = max(dp[j],dp[j-self.weight[i]]+self.value[i])
    print("DP_Value",dp[self.capacity])
```

5. 启发式算法中共用的生成初始解init_sol、扰动产生新解pert_sol及保证生成的解满足约束条件的函数take_out。

init_sol中，随机生成一串0-1编码，代表每个物体是否被放入背包，作为初始解。

```
def init_sol(self,init_val):
      rnd_ind = np.random.permutation(range(self.amount))
      init_sol = np.ones(self.amount)
      np.random.shuffle(rnd_ind)
      init_sol[rnd_ind[:random.choice(rnd_ind)]] = init_val
      init_sol = self.take_out(init_sol)
      return init_sol
```

物体有放入背包和不放入背包两种状态，因此通过改变物体状态产生新解。

pert_sol中，以两种方式对解进行扰动：① 随机反转某个位置上的编码，即改变某物体的状态。 ② 随机交换两个物体的状态。

```
def pert_sol(self,near_sol):
    if np.random.rand()>0.5:
            rnd_ind = np.random.randint(0,self.amount)
            near_sol[rnd_ind] = abs(near_sol[rnd_ind]-1)
            near_sol = self.take_out(near_sol)
    else:
        rnd_ind = np.random.randint(0,self.amount,2)
        near_sol[rnd_ind[0]],near_sol[rnd_ind[1]] =near_sol[rnd_ind[1]],near_sol[rnd_ind[0]]
        near_sol = self.take_out(near_sol)
    return near_sol
```

take_out中，当背包满时，不断地随机取出一个物品，直到满足物品重量不超过背包容量的约束。

```
# Reasonable Solution
def take_out(self,sol):
    while 1:
        if sum(sol*self.weight)>self.capacity:
            ind = np.argwhere(sol==1)
            sol[random.choice(ind)] = 0
        else:
            break
    return sol
```

6. 模拟退火算法的实现，包括外循环和内循环两个层次。内循环产生新解并根据Metropolis准则接受新解，内循环完成后以一定比率进行一次降温，降为终止温度时停止。

```
    def simulated_annealing(self,ratio,init_temper,terminal_temper,epoch=100):
        temper = init_temper
        cur_sol = self.init_sol(0)
        new_sol = self.init_sol(1)
        best_val = np.inf
        cur_val = np.inf
        best_sol = np.ones(self.amount)
        it = 0
        anneal_val = -self.value


        while temper>terminal_temper:
            it+=1
            for i in range(epoch):
                new_sol = self.pert_sol(new_sol.copy())
                new_val = sum(new_sol*anneal_val)
                if new_val<cur_val or np.random.rand()<np.exp(-(new_val-cur_val)/temper):
                    cur_val = new_val
                    cur_sol = new_sol
                else:
                    new_sol = cur_sol
                if cur_val < best_val:
                    best_val = cur_val
                    best_sol = cur_sol.copy()
            temper *= ratio
        best_val = -best_val
        print("Simulated Annealing Value",best_val)
```

7. 禁忌搜索算法的实现，根据给定的邻近解个数扰动当前解，若邻近中的最优解没有被禁忌，则更新当前解并将该解禁忌。若此时禁忌表达到了规定的长度，则解放表中被囚禁最久的解。

```python
# Tabu Search
def tabu_search(self,tabu_len,epoch,near_num,init_val=0):
    init_sol = self.init_sol(init_val)
    init_val = sum(init_sol*self.value)
    vals = [init_val]
    sols = [init_sol]
    tabu_list = []
    best_val = init_val
    best_sol = init_sol.copy()

    for i in range(epoch):
        nears = self.find_near(best_sol,near_num)
        near_vals = [sum(-near*self.value) for near in nears]
        val_ind = np.argsort(near_vals)
        for i in val_ind:
            if i not in tabu_list:

                best_sol = nears[i].copy()
                vals.append(near_vals[i])
                sols.append(nears[i])
                tabu_list.append(i)
                if len(tabu_list) > tabu_len:
                    tabu_list.pop(0)
                break
            elif near_vals[i]<min(near_vals):
                tabu_list.remove(i)
                tabu_list.append(i)
                vals.append(near_vals[i])
                sols.append(nears[i])
                best_sol = nears[i].copy()


    best_val = sum(best_sol*self.value)
    print("Tabu Search",best_val)
    return best_val
```

2.4、结果分析

1. 禁忌搜索参数搜索结果如下图所示。由于禁忌搜索对初始解有较强的依赖性，参数搜索时使用贪心算法得到的解进行统一的初始化。

```
--------------  Settings  ----------------
 [Number] 10 Items and A Knapsack with [Capacity] 50
 [Weight] [14  9 10 29 28  5 29 14 16 11]
 [Value] [52 88 72 65 95 93  2 62  1 90]
-----------------------------------------
 [ Greddy Value ] :  181
 [ Greedy Solution ] :  [0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]
 [ Dynamic Programming Value ] :  405.0
 [ Simulated Annealing Value ] :  405.0
 [ Simulated Annealing Solution ] :  [0. 1. 1. 0. 0. 1. 0. 1. 0. 1.]
[epoch]: 1000,[near_num]:100,[tabu_length]:50
 [ Tabu Search ] :  278.0
[epoch]: 1000,[near_num]:100,[tabu_length]:20
 [ Tabu Search ] :  405.0
[epoch]: 1000,[near_num]:100,[tabu_length]:10
 [ Tabu Search ] :  405.0
[epoch]: 1000,[near_num]:100,[tabu_length]:5
 [ Tabu Search ] :  405.0
[epoch]: 1000,[near_num]:100,[tabu_length]:3
 [ Tabu Search ] :  405.0
[epoch]: 1000,[near_num]:50,[tabu_length]:50
 [ Tabu Search ] :  240.0
[epoch]: 1000,[near_num]:50,[tabu_length]:20
 [ Tabu Search ] :  278.0
[epoch]: 1000,[near_num]:50,[tabu_length]:10
 [ Tabu Search ] :  278.0
[epoch]: 1000,[near_num]:50,[tabu_length]:5
 [ Tabu Search ] :  278.0
[epoch]: 1000,[near_num]:50,[tabu_length]:3
 [ Tabu Search ] :  405.0
[epoch]: 1000,[near_num]:20,[tabu_length]:50
 [ Tabu Search ] :  189.0
[epoch]: 1000,[near_num]:20,[tabu_length]:20
 [ Tabu Search ] :  224.0
[epoch]: 1000,[near_num]:20,[tabu_length]:10
 [ Tabu Search ] :  405.0
[epoch]: 1000,[near_num]:20,[tabu_length]:5
```

2. 四种算法的结果对比如下。在模拟退火算法中，初始温度为100,终止温度为2，以0.95的速率降温；在禁忌搜索算法中，每次扰动当前解生成10个邻居解，禁忌长度为10，迭代步数达到1000次后终止。

   10个物品，背包容量为50时：

```
--------------  Settings  ----------------
 [Number] 10 Items and A Knapsack with [Capacity] 50
 [Weight] [14  9 10 29 28  5 29 14 16 11]
 [Value] [52 88 72 65 95 93  2 62  1 90]
-----------------------------------------
 [ Greddy Value ] :  181
 [ Greedy Solution ] :  [0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]
 [ Dynamic Programming Value ] :  405.0
 [ Simulated Annealing Value ] :  405.0
 [ Simulated Annealing Solution ] :  [0. 1. 1. 0. 0. 1. 0. 1. 0. 1.]
 [ Tabu Search ] :  405.0
```

100个物品，背包容量为500时：

```
--------------  Settings  ---------------
[Number] 100 Items and A Knapsack with [Capacity] 500
[Weight] [19 10 22 25 28 22 23 10 23 13 28 28 24 23 20  5 28  9  2  4 14 14 14  8
19  1  5 15 21 18 23  6 14  2 21 18  8  5 22 17 23  3 26 24  1 13 12 11
16  9 28  1 22 20  1 21  4  4 12  4  7  5 24 29  3  5 28  5 19 21 18 13
16 19  4 26  1 13 13 13 28 22 29 19 18 17 16 23  9 10 23  2 18 15 14  1
25 21  3  3]
[Value] [52 88 72 65 95 93  2 62  1 90 46 41 93 92 37 61 43 59 42 21 31 89 31 29
31 78 83 29 86 94 11 31 15 62 70 77 12 53 76 33 53 51 22 27  8 17 88 93
44 60  1 32 31 19  2 12 47 28 48 88 80 18 70 55 30 12 13  6 56 89 73 19
 3 78 60 42  7 11 36 52 54 48 57 21 66 97 26 82 73 49  3 76 74 14 64 20
79 59 10  7]
-----------------------------------------
[ Greddy Value ] :  1248
[ Greedy Solution ] :  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1.
0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1.
0. 0. 0. 0.]
[ Dynamic Programming Value ] :  3063.0
[ Simulated Annealing Value ] :  3040.0
[ Simulated Annealing Solution ] :  [0. 1. 1. 0. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 0. 1. 0
. 0.
0. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 1. 1.
0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.
0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1.
0. 0. 1. 0.]
[ Tabu Search ] :  2795.0
```

3. 由上述结果可以看出，贪心算法只能得到较次的解，动态规划算法可以以较低的复杂度得到最优解，而模拟退火和禁忌搜索算法的性能与初始化解和参数有关。初始化和邻域解生成中引入的随机性增强了这两种算法的局部搜索能力，而接受劣质解的可能性赋予了这两种算法较强的“爬山”能力。因此，在参数合适时，上述两种启发式算法通常能找到全局的较优解。

三、爬山法求解 STS(V) 问题

1、问题描述

Steiner Triple Systems：

定义：一个集合系统 (V,B)，其中，B 表示三元组（block）集合，V 表示点集，$|V|=v$。对于任意一对点 i,j，满足 B 中有且仅有一个 $block$ 含有这对点 i,j，则 $STS(V,B)$ 称为 **Steiner Triple Systems。**

样例:
$STS(7)$：
V={1,2,3,4,5,6,7} ， B={{1,2,4} , {2,3,5} , {3,4,6}, {4,5,7} , {1,5,6} , {2,6,7} , {1,3,7}}

算法目标:
给定点集 V，满足 $|V|=v$，$v\equiv 1,3\ (mod\ 6)$，构造出满足要求的系统 $STS(V,B)$。

2、算法流程

Step 1:
初始时，设置 $PSTS(V)$ 为空集，每步执行 $SWITCH$ 操作：尝试添加一个新的 $block$，或者在 $block$ 总数不变的情况下，调整 $block$ 的构成。

Step 2:
$SWITCH$ 操作:
若当前没有任何一个 $block$ 包含点对 (x,y)，则称 (x,y) 为 $live\ pair$。
令点 x 为 $live\ point$，当且仅当，$r_x<\frac{v-1}{2}$即点 x 的出现次数小于 $\frac{v-1}{2}$。此时，对于点 x 而言，至少还有 2 个点未相遇，可设为 y,z:

- 若 (y,z) 是 $live\ pair$，则直接将 (x,y,z) 作为新 $block$ 加入，则此时 $blocks$ 总数加一；

- 若 (y,z) 不是 $live\ pair$，根据定义，$PSTS(V)$ 中存在一个 $block$ 包含点对 (y,z)，不妨设为：(w,y,z)，接着将 (w,y,z) 从 $PSTS(V)$ 中删去，并加入新的 $block$：(x,y,z)，此时 $blocks$ 总数不变。

Step 3:
若 $block$ 的数量 $NumBlocks < \frac{v(v-1)}{6}$，则继续执行 $SWITCH$ 操作直到 $block$ 数目达到要求，从而生成满足条件的 $SteinerTripleSystems$。

3、程序框架

本程序分为以下几个模块：

1. 主函数 `main`
   接收输入信息 v，若当前 $block$ 的数量 $NumBlocks < \frac{v(v-1)}{6}$，则循环执行 $SWITCH$ 操作直到 $block$ 数目达到要求。

```
int main(int argc, char** argv)
{
    srand(time(NULL));

    if (argc == 1)
        v = 7;
    else
        v = atoi(argv[1]);

    while(NumBlocks < v*(v-1)/6)
        Switch_op();
```

2. $SWITCH$ 操作函数 `Switch_op`
   首先遍历得到当前的 $live\ points$，并随机选取一个 $live\ point\ x$，并寻找两个 $live\ pairs$：(x,y)、(x,z)。

```
void Switch_op(){
    int tot = 0;
    for(int i = 1;i <= v; i++)
        if(deg[i] < (v-1)/2)
            live_points[ ++tot ] = i;

    random_shuffle(live_points+1,live_points+tot+1);

    int x = live_points[1];

    int y,z;
    // find y
    for(int i = 1;i <= v;i++) {
        if(i == x) continue;
        if(bel[x][i]) continue;
        y = i;
        break;
    }

    // find z
    for(int i = y+1; i <= v;i++) {
        if(i == x) continue;
        if(bel[x][i]) continue;
        z = i;
        break;
    }

    triple_cnt++;
    Block[triple_cnt].x = x;
    Block[triple_cnt].y = y;
    Block[triple_cnt].z = z;
    Block[triple_cnt].in = true;
```

然后分类讨论，若 (y,z) 是 $live\ pair$，则直接将 (x,y,z) 作为新 $block$ 加入；若 (y,z) 不是 $live\ pair$，根据定义，$PSTS(V)$ 中存在一个 $block$ 包含点对 (y,z)：(w,y,z)，接着将 (w,y,z) 从 $PSTS(V)$ 中删去，并加入新的 $block$：(x,y,z)。

```
// case 1:
if(bel[y][z] == 0) {
    deg[y] ++;
    deg[z] ++;
    NumBlocks++;
}
// case 2:
else {
    int old_triple = bel[y][z];
    int w;

    if(Block[old_triple].y == y && Block[old_triple].z == z) w = Block[old_triple].x;
    if(Block[old_triple].y == z && Block[old_triple].z == y) w = Block[old_triple].x;
    if(Block[old_triple].x == y && Block[old_triple].z == z) w = Block[old_triple].y;
    if(Block[old_triple].x == z && Block[old_triple].z == y) w = Block[old_triple].y;
    if(Block[old_triple].x == y && Block[old_triple].y == z) w = Block[old_triple].z;
    if(Block[old_triple].x == z && Block[old_triple].y == y) w = Block[old_triple].z;


    Block[old_triple].in = false;
    bel[w][y] = bel[w][z] = bel[y][w] = bel[z][w] = 0;

    deg[w] --;
}

deg[x] ++;
bel[x][y] = bel[y][x] = bel[y][z] = bel[z][y] = bel[x][z] = bel[z][x] = triple_cnt;
```

3. 输出函数 `output`

   最后输出构造得到的满足要求的 $STS(v)$。

```
    printf("STS(%d):\n",v);
    printf("V={");
    for(int i = 1;i <= v;i++) {
        if(i != 1) printf(",");
        printf("%d",i);
    }
    printf("}\n");

    printf("B={");
    int sort_block[3];
    for(int i = 1;i <= triple_cnt;i++) {
        if(!Block[i].in) continue;
        sort_block[0] = Block[i].x;
        sort_block[1] = Block[i].y;
        sort_block[2] = Block[i].z;
        sort(sort_block,sort_block+3);
        printf("{%d %d %d}",sort_block[0],sort_block[1],sort_block[2]);
        if(i != triple_cnt) printf("\n");
    }
    printf("}\n\n");
    return 0;
```

4、结果分析

测试数据分别输入$v=9,15,181,199$，输出结果信息如下：

$v = 9$:

```
C:\Users\ljh2000\Desktop\2021-2022-2\图与网络\MATH6010_Graph-and-Network\Lab3\SteinerTripleSystems>.\SteinerTripleSystems.exe 9
STS(9):
V={1, 2, 3, 4, 5, 6, 7, 8, 9}
B={{4  6  7}
{1  5  7}
{2  3  7}
{1  4  8}
{5  6  9}
{1  2  9}
{1  3  6}
{2  6  8}
{7  8  9}
{2  4  5}
{3  4  9}
{3  5  8}}
```

$v = 15$:

```
C:\Users\ljh2000\Desktop\2021-2022-2\图与网络\MATH6010_Graph-and-Network\Lab3\SteinerTripleSystems>.\SteinerTripleSystems.exe 15
STS(15):
V={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
B={{3  4  15}
{1  4  8}
{5  7  12}
{1  6  9}
{8  12  13}
{1  2  12}
{3  9  12}
{10  12  15}
{1  3  14}
{4  5  14}
{2  5  15}
{6  11  15}
{1  5  11}
{5  6  13}
{1  10  13}
{4  7  13}
{9  13  15}
{5  8  9}
{3  5  10}
{3  6  8}
{4  6  12}
{1  7  15}
{2  6  10}
{6  7  14}
{7  8  10}
{9  10  14}
{4  10  11}
{2  4  9}
{7  9  11}
{8  14  15}
{11  12  14}
{2  8  11}
{2  13  14}
{2  3  7}
{3  11  13}}
```

$v = 181$:（$STS(181)$过长，截取输出结果前几行如下）

```
C:\Users\ljh2000\Desktop\2021-2022-2\图与网络\MATH6010_Graph-and-Network\Lab3\SteinerTripleSystems>.\SteinerTripleSystems.exe 181
STS(181):
V={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64
, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119
, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181}
B={{20  59  94}
{6  101  121}
{41  70  103}
{29  59  102}
{19  98  109}
{15  113  123}
{44  78  100}
{3  136  163}
{24  77  103}
{24  113  122}
{37  82  103}
{46  91  110}
{3  44  134}
{39  83  113}
{21  44  95}
{13  42  108}
{48  67  113}
{30  96  107}
{50  98  108}
{26  71  133}
{46  101  113}
{13  96  124}
{26  77  88}
{37  118  123}
{6  125  149}
{50  99  113}
{10  96  129}
{30  113  131}
{27  105  127}
{7  116  142}
{4  75  129}
{32  91  112}
{2  129  147}
{5  105  110}
```

$v = 199$: ($STS(199)$过长，截取输出结果前几行如下)

```
C:\Users\ljh2000\Desktop\2021-2022-2\图与网络\MATH6010_Graph-and-Network\Lab3\SteinerTripleSystems>.\SteinerTripleSystems.exe 199
STS(199):
V={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64
, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119
, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199}
B={{38 84 114}
{5 83 102}
{8 78 133}
{20 128 137}
{30 106 127}
{14 106 120}
{53 95 123}
{6 30 136}
{8 126 142}
{23 60 103}
{52 87 118}
{34 65 113}
{8 24 95}
{27 88 145}
{8 63 122}
{41 103 137}
{24 54 91}
{14 113 148}
{37 76 122}
{56 132 137}
{5 142 150}
{24 93 142}
{11 96 110}
{32 123 144}
{5 55 118}
{34 54 123}
{24 79 121}
{50 73 92}
{45 116 139}
{23 49 120}
{72 112 131}
{14 34 135}
{11 84 145}
{13 30 100}
{66 112 126}
{69 109 128}
{23 68 115}
{13 48 137}
{19 97 150}
{56 108 133}
{76 108 123}
{27 53 125}
{4 48 94}
{8 125 131}
```

由此可见，程序能根据给出的 v 构造输出满足要求的 $Steiner\ Triple\ Systems$:$STS(v) = (V, B)$