代码
源代码结构
petri net类
MarkingNode类
Reachability Graph类
例子
例一：H2O
例二：自定义的简单网络

代码

源代码结构

- main.cpp
- petri_net.h
- petri_net.cpp
- reachability_graph.h
- reachability_graph.cpp

petri net类

```
class PetriNet {
private:
  //PN={P,T,I,O,Mi}
  Eigen::VectorXi place_; // 库所
  Eigen::VectorXi transition_;// 变迁
  unsigned int num_of_place_ = 0;
  unsigned int num_of_trans_ = 0;
  Eigen::SparseMatrix<int> input_matrix_;// 前置矩阵
  Eigen::SparseMatrix<int> output_matrix_;//后置矩阵
  Eigen::SparseMatrix<int> trans_matrix_;
  std::vector<int> firable_transition_;//可激发的变迁保存到成员变量中
  Eigen::VectorXi marking_;// 状态标识
  static PetriNet *instance_;  //单例对象指针

  // 构造和析构成为私有的, 禁止外部构造和析构
  PetriNet(Eigen::VectorXi &p, Eigen::VectorXi &t,
           Eigen::MatrixXi &i, Eigen::MatrixXi &o,
           Eigen::VectorXi &m_0);
  PetriNet() = default;
  ~PetriNet() = default;
  // 更新可用激发
  void FreshFirableTransition();
public:
  // 禁止外部拷贝和赋值
  PetriNet(const PetriNet &) = delete;
  const PetriNet &operator=(const PetriNet &) = delete;
  // 实例化创建。获得本类实例的唯一全局访问点
  static PetriNet *GetInstance(Eigen::VectorXi &p, Eigen::VectorXi &t,
```

```
                                Eigen::MatrixXi &i, Eigen::MatrixXi &o,
                                Eigen::VectorXi &m_0);
  // 获取当前petri_net的Marking
  const Eigen::VectorXi &GetMarking() const;
  // 设置marking
  void SetMarking(const Eigen::VectorXi &marking);
  // 获取可用的激发
  const std::vector<int> &GetFirableTransition() const;
  // 激发一个transition，并改变petri net的状态
  void FiringATransition(int t);
};
#endif //PETRI_NET_H
```

MarkingNode类

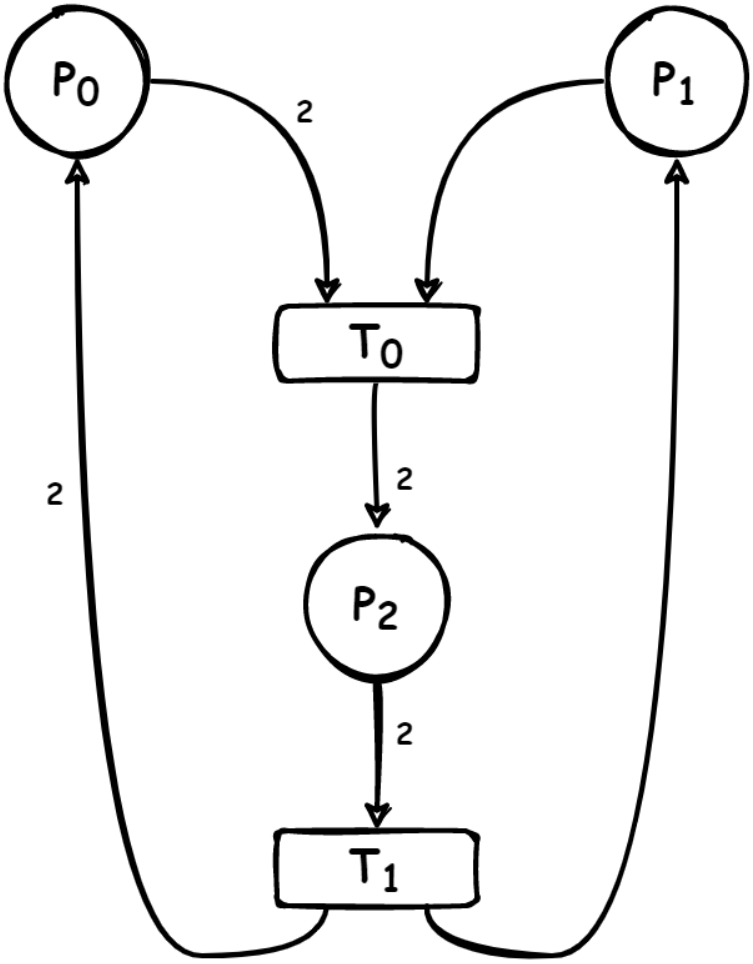
```
class MarkingNode {
public:
  int node_name_ = -1;// 状态节点的编号（Name）
  Eigen::VectorXi marking_;// 本节点状态，Mi = {P1,P2,P3...}
  std::vector<std::pair<int, int>> transition_to_son_;// pair<which_transition,
son_node>
  MarkingNode(int node_name, Eigen::VectorXi marking);
  MarkingNode() = default;
};
```

Reachability Graph类

```
class ReachabilityGraph {
private:
  // 标识当前共有多少个状态
  int marking_number_ = 0;
  // 维护v_new_，保存已出现但未激发的状态，因为v_new_需遍历/增删首尾元素，用vector较合理
  std::vector<Eigen::VectorXi> v_new_;
  // 维护v_old_，反向检索某状态是否已出现过，因v_old_只需添加/检索无需遍历，用
unordered_map/set更快
  std::unordered_map<std::string, std::pair<bool,int>> v_old_;
  // 维护nodes_结构体数组表示可达图
  std::vector<MarkingNode> nodes_;
  bool AddNode(const int &node_name,
               const Eigen::VectorXi &marking,
               const std::vector<std::pair<int, int>> &transition_to_son);
  bool AddNode(const int &node_name,
               const Eigen::VectorXi &marking);
  static std::string Vector2String(Eigen::VectorXi Eigen_vector_int);
  int GetNodeNumberInVOld(const Eigen::VectorXi &mark) const;
  bool GetNodeStatusInVOld(const Eigen::VectorXi &mark) const;
  bool SetNodeFiredInVOld(const Eigen::VectorXi &mark);
public:
  bool BuildReachabilityGraph(PetriNet *petri_net);
  // 外部调用GetNodes()获取成员变量nodes_即可以描述可达图
  const std::vector<MarkingNode> &GetNodes() const;
};
```

例子

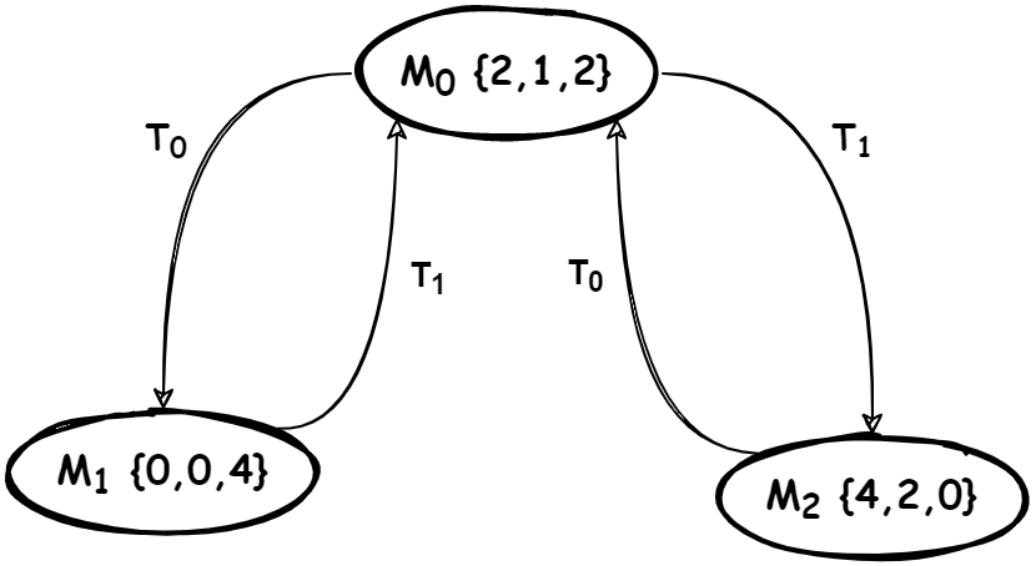
例一：H2O



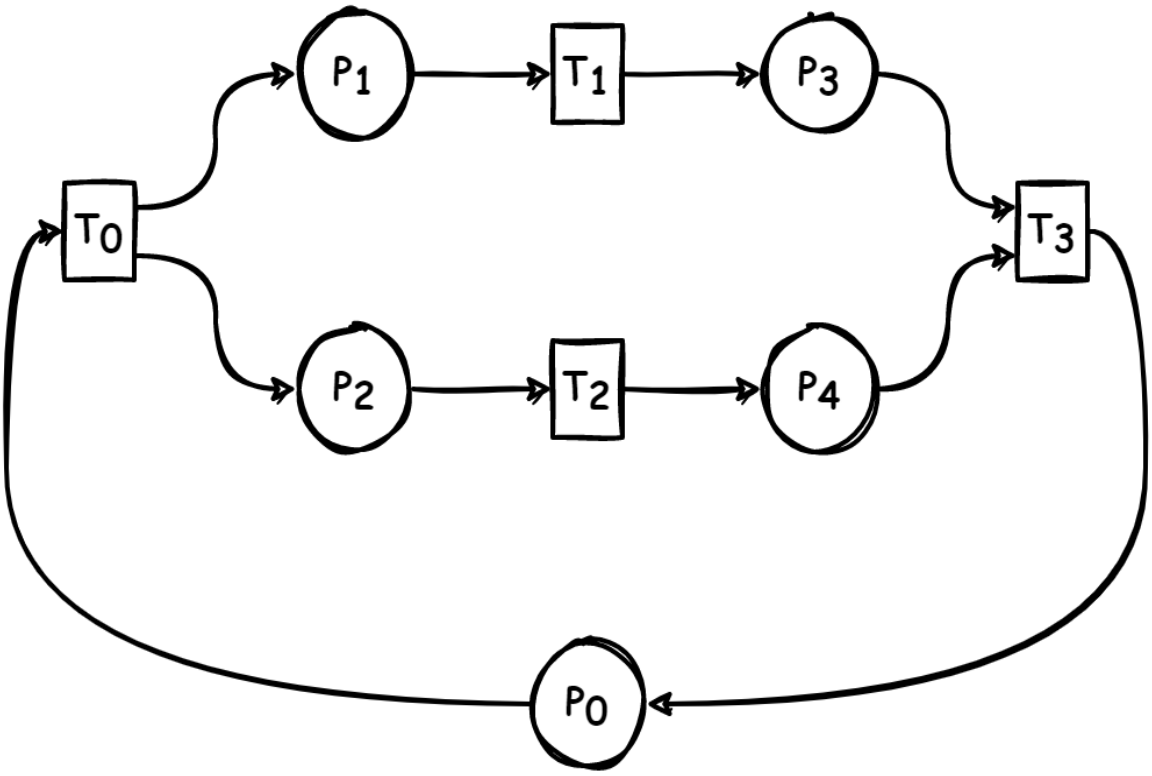
INPUT MATRIX		
	T0	T1
P0	2	0
P1	1	0
P2	0	2

OUTPUT MATRIX		
	T0	T1
P0	0	2
P1	0	1
P2	2	0

```
运行: Homework_1
C:\Users\LvMeng\Desktop\Petri_Homework\Homework_1\cmake-build-debug\Homework_1.exe
hello world!
node(0)
marking_0: 2 1 2
  --[T_0]--> node(1)
  --[T_1]--> node(2)

node(1)
marking_1: 0 0 4
  --[T_1]--> node(0)

node(2)
marking_2: 4 2 0
  --[T_0]--> node(0)


进程已结束,退出代码0
```



例二：自定义的简单网络

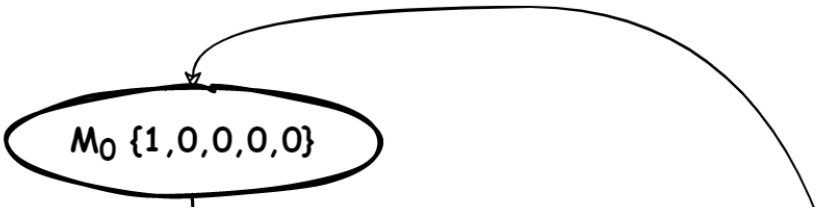

P1
T1
P3
T0
T3
P2
T2
P4
P0

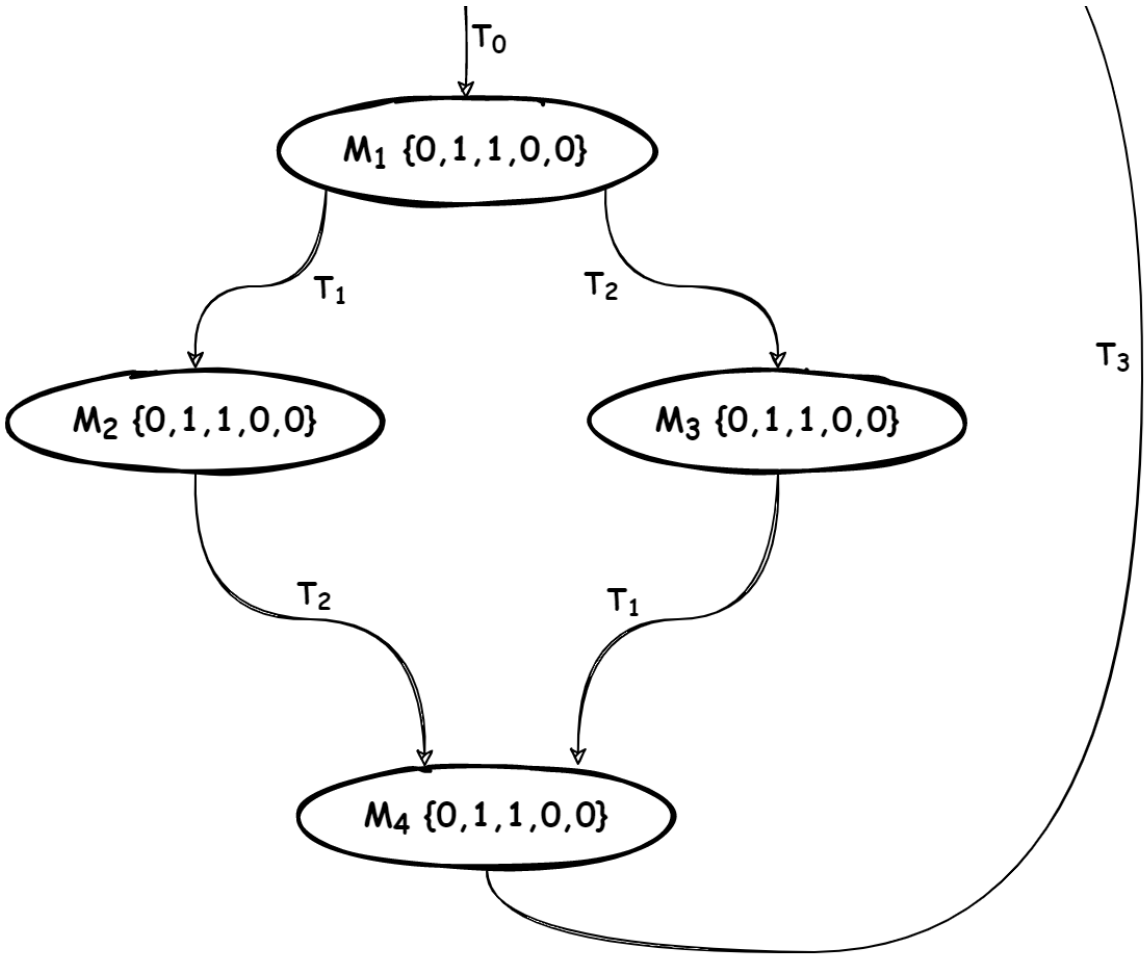
INPUT MATRIX				
	T_0	T_1	T_2	T_3
P_0	1	0	0	0
P_1	0	1	0	0
P_2	0	0	1	0
P_3	0	0	0	1
P_4	0	0	0	1

OUTPUT MATRIX				
	T_0	T_1	T_2	T_3
P_0	0	0	0	1
P_1	1	0	0	0
P_2	1	0	0	0
P_3	0	1	0	0
P_4	0	0	1	0

运行: Homework_1 ×

```
C:\Users\LvMeng\Desktop\Petri_Homework\Homework_1\cmake-build-debug\Homework_1.exe
hello world!
node(0)
marking_0: 1 0 0 0 0
  --[t_0]--> node(1)

node(1)
marking_1: 0 1 1 0 0
  --[t_1]--> node(2)
  --[t_2]--> node(3)

node(2)
marking_2: 0 0 1 1 0
  --[t_2]--> node(4)

node(3)
marking_3: 0 1 0 0 1
  --[t_1]--> node(4)

node(4)
marking_4: 0 0 0 1 1
  --[t_3]--> node(0)


进程已结束,退出代码0
```




T0
M1 {0,1,1,0,0}
T1
T2
T3
M2 {0,1,1,0,0}
M3 {0,1,1,0,0}
T2
T1
M4 {0,1,1,0,0}