2006 年下半年软件设计师下午试题

（考试时间 14:00～16:30 共 150 分钟）

请按下述要求正确填写答题纸

1.在答题纸的指定位置填写你所在的省、自治区、直辖市、计划单列市的名称。

2.在答题纸的指定位置填写准考证号、出生年月日和姓名。

3.答题纸上除填写上述内容外只能写解答。

4.本试卷共 7 道题，试题一至试题四是必答题，试题五至试题七选答 1 道。每题 15 分，满分 75 分。

5.解答时字迹务必清楚，字迹不清时，将不评分。

6.仿照下面例题，将解答写在答题纸的对应栏内。

例题

2006 年下半年全国计算机技术与软件专业技术资格（水平）考试日期是（1）月（2）日。

因为正确的解答是“11 月 4 日”，故在答题纸的对应栏内写上“11”和“4”（参看下表）。

例题	解答栏
（1）	11
（2）	4

试题一至试题四是必答题

试题一（共 15 分）

阅读以下说明以及数据流图，回答问题 1 至问题 5，将解答填入答题纸的对应栏内。

[说明]

某银行已有一套基于客户机/服务器模式的储蓄系统 A 和一套建帐软件。建帐软件主要用于将储蓄所手工处理的原始数据转换为系统 A 所需的数据格式。该建帐软件具有以下功能：

（1）**分户帐录入**：手工办理业务时建立的每个分户帐数据均由初录员和复录员分别录

入，以确保数据的正确性；

（2）**初录/复录比对**：将初录员和复录员录入的数据进行一一比较，并标记两套数据 是否一致；

（3）**数据确认**：当上述两套数据完全一致后，将其中任一套作为最终进入系统 A 的 原始数据；

（4）**汇总核对和打印**：对经过确认的数据进行汇总，并和会计账目中的相关数据进行核对，以确保数据的整体正确性，并打印输出经过确认的数据，为以后核查可能的错误提供依据；

（5）**数据转换**：将经过确认的数据转换为储蓄系统 A 需要的中间格式数据；

（6）数据清除：为加快初录和复录的处理速度，在数据确认之后，可以有选择地清除初录员和复录员录入的数据。

该软件的数据流图如图 1-1～图 1-3 所示。图中部分数据流数据文件的格式如下：

初录分户帐 = 储蓄所号 + 帐号 + 户名 + 开户日 + 开户金额 + 当前余额 + 性质

复录分户帐 = 储蓄所号 + 帐号 + 户名 + 开户日 + 开户金额 + 当前余额 + 性质

初录数据 = 手工分户帐 + 一致性标志

复录数据 = 手工分户帐 + 一致性标志

会计账目 = 储蓄所号 + 总户数 + 总余额

操作结果 = 初录操作结果 + 比对操作结果 + 复录操作结果

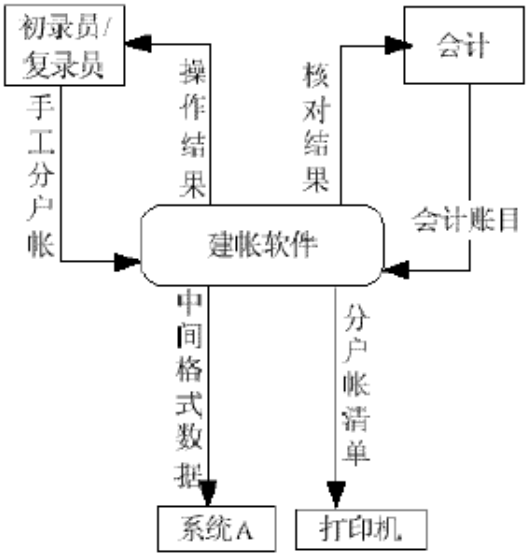

图 1-1 建帐软件顶层数据流图

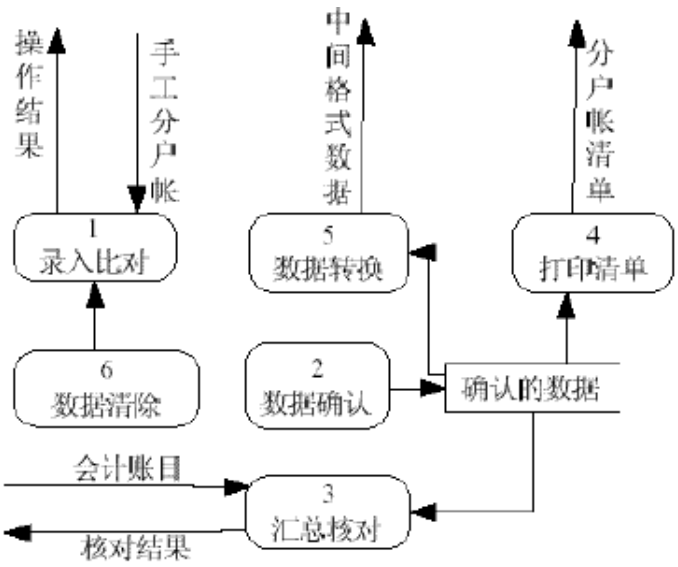

图 1-2 建帐软件第 0 层数据流图

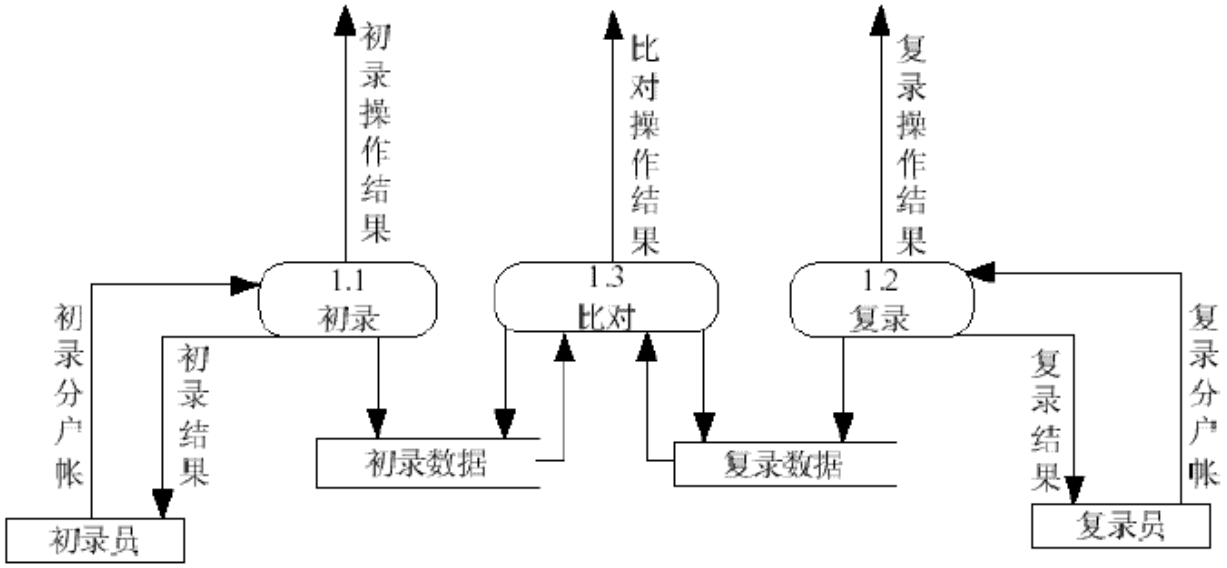

图 1-3 建帐软件第 1 层数据流图

软件需要打印的分户帐清单样式如表 1-1 所示：

表 1-1 分户帐清单样式表

储蓄所	帐号	开户日	户名	其他分户帐数据
储蓄所 1				

	…	…	…	…
储蓄所 1 合计	共 XXX 户，总余额 9999999.99 元			
储蓄所 2				
	…	…	…	…
储蓄所 2 合计	共 XXX 户，总余额 9999999.99 元			

[问题 1]（4 分）

请采用说明中的词汇，给出数据确认处理所需的数据流在第 1 层图中的全部可选起点（第 0 层图和第 1 层图中均未给出）。

[问题 2]（2 分）

不考虑数据确认处理（加工 2），请指出数据流图中存在的错误。

[问题 3]（2 分）

打印分户帐清单时，必须以下列哪一组数据作为关键字进行排序，才能满足需求？请从下面选项中选择，并将对应序号填入答题纸相应栏。

① 储蓄所 ② 帐号 ③ 开户日 ④ 总户数和总余额

[问题 4]（4 分）

加工 1（录入比对处理）除能够检查出初录数据和复录数据不一致外，还应当检测出下列哪些错误，请将对应序号填入答题纸对应栏。

① 输入的无效字符 ② 输入的半个汉字 ③ 显示器无法显示

④ 初录员重复录入同一帐户 ⑤ 汇总数据与会计账目不符 ⑥ 打印机卡纸

[问题 5]（3 分）

请使用数据字典条目定义形式，给出第 0 层 DFD 中的“手工分户帐”数据流和第 1 层 DFD 中的“初录分户帐”、“复录分户帐”的关系。

试题二(共 15 分)

阅读以下说明，回答问题 1 至问题 4，将解答填入答题纸的对应栏内。

[说明]

某宾馆需要建立一个住房管理系统，部分的需求分析结果如下：

(1) 一个房间有多个床位，同一房间内的床位具有相同的收费标准。不同房间的床位收费标准可能不同。

(2) 每个房间有房间号（如 201、202 等）、收费标准、床位数目等信息。

(3) 每位客人有身份证号码、姓名、性别、出生日期和地址等信息。

(4) 对每位客人的每次住宿，应该记录其入住日期、退房日期和预付款额信息。

(5) 管理系统可查询出客人所住房间号。

根据以上的需求分析结果，设计一种关系模型如图 2-1 所示：

图 2-1 住房管理系统的实体联系图

[问题 1]（1 分）

根据上述说明和实体-联系图，得到该住房管理系统的关系模式如下所示，请补充住 宿关系。

房间(房间号，收费标准，床位数目)

客人(身份证号，姓名，性别，出生日期，地址)

住宿(__（1）__,入住日期，退房日期，预付款额)

[问题 2]（4 分）

请给出问题 1 中住宿关系的主键和外键。

[问题 3]（4 分）

若将上述各关系直接实现为对应的物理表,现需查询在 2005 年 1 月 1 日到 2005 年 12 月 31 日期间，在该宾馆住宿次数大于 5 次的客人身份证号，并且按照入住次数进行降序 排列。下面是实现该功能的 SQL 语句，请填补语句中的空缺。

```
SELECT 住宿.身份证号,count(入住日期) FROM 住宿，客人
WHERE 入住日期 >= ‘20050101’AND 入住日期 <= ‘20051231’
  AND 住宿.身份证号 = 客人.身份证号
GROUP BY （2）
（3） count(入住日期) > 5
```

（4）

[问题 4]（6 分）

为加快 SQL 语句的执行效率,可在相应的表上创建索引。根据问题 3 中的 SQL 语句，除主键和外键外，还需要在哪个表的哪些属性上创建索引，应该创建什么类型的索引，请说明原因。

试题三(共 15 分)

阅读以下说明和图，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

[说明]

S 公司开办了在线电子商务网站,主要为各注册的商家提供在线商品销售功能。为更 好地吸引用户，S 公司计划为注册的商家提供商品(Commodity)促销(Promotion)功能。商品的分类(Category)不同，促销的方式和内容会有所不同。

注册商家可发布促销信息。商家首先要在自己所销售的商品的分类中,选择促销涉及 的某一具体分类，然后选出该分类的一个或多个商品（一种商品仅仅属于一种分类），接着制定出一个比较优惠的折扣政策和促销活动的优惠时间，最后由系统生成促销信息并将 该促销信息公布在网站上。

商家发布促销信息后,网站的注册用户便可通过网站购买促销商品。用户可选择参与 某一个促销(Promotion)活动,并选择具体的促销商品(Commodity),输入购买数量等购买信息。系统生成相应的一份促销订单(POrder)。只要用户在优惠活动的时间范围内，通过 网站提供的在线支付系统，确认在线支付该促销订单（即完成支付），就可以优惠的价格完成商品的购买活动，否则该促销订单失效。

系统采用面向对象方法开发，系统中的类以及类之间的关系用 UML 类图表示，图 3-1 是该系统类图中的一部分；系统的动态行为采用 UML 序列图表示，图 3-2 是发布促销的序列图。

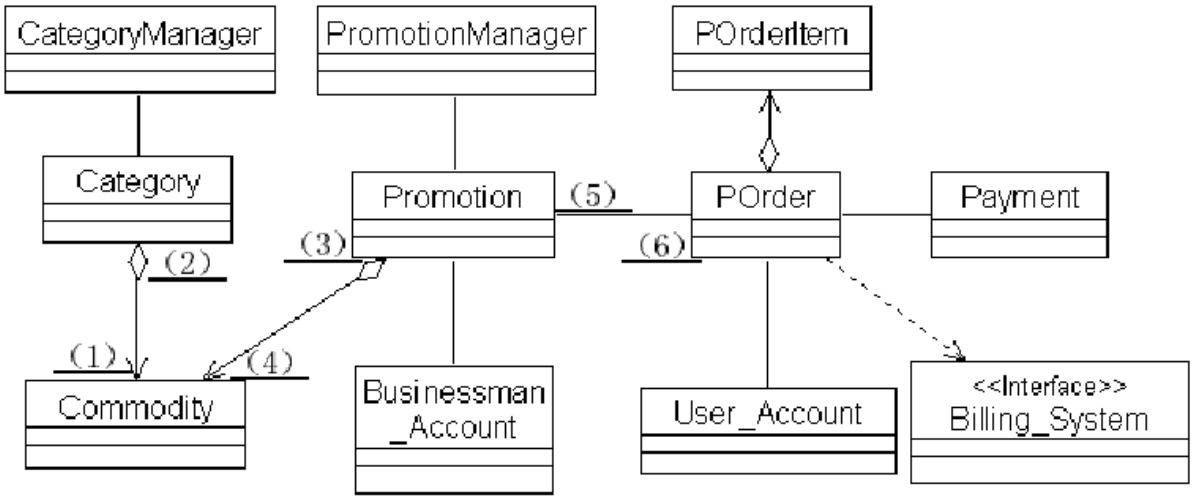

图 3-1 在线促销系统部分类图

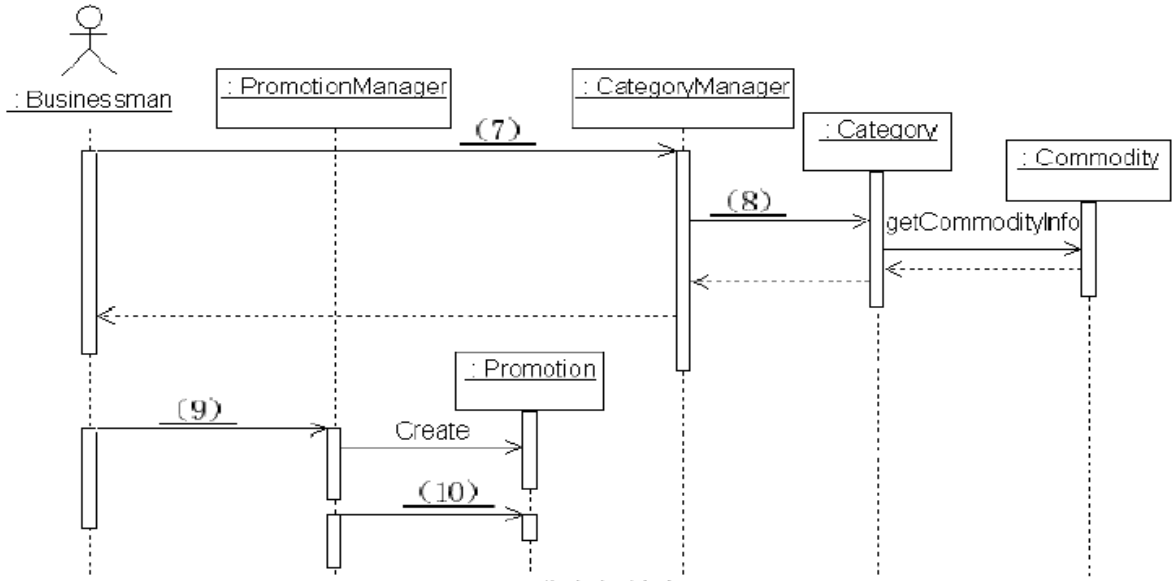

图 3-2 发布促销序列图

[问题 1]（6 分）

识别关联的多重度是面向对象建模过程中的一个重要步骤。根据说明中给出的描述，完成图 3-1 中的(1)～(6)。

[问题 2]（4 分）

请从表 3-1 中选择方法，完成图 3-2 中的(7)～(10)。

表 3-1 可选消息列表

功能描述	方法名
向促销订单中添加所选的商品	buyCommodities
向促销中添加要促销的商品	addCommodities
查找某个促销的所有促销订单信息列表	getPromotionOrders
生成商品信息	createCommodity
查找某个分类中某商家的所有商品信息列表	getCommodities
生成促销信息	createPromotion
生成促销订单信息	createPOrder
查找某个分类的所有促销信息列表	getCategoryPromotion
查找某商家所销售的所有分类列表	getCategories
查找某个促销所涉及的所有商品信息列表	getPromotionCommodities

[问题 3]（5 分）

关联(Association)和聚集（Aggregation）是 UML 中两种非常重要的关系。请说明关联和聚集的关系，并说明其不同点。

试题四(共 15 分)

阅读以下说明和图，填补流程图中的空缺，将解答填入答题纸的对应栏内。

[说明]

某汽车制造工厂有两条装配线。汽车装配过程如图 4-1 所示,即汽车底盘进入装配线，零件在多个工位装配，结束时汽车自动完成下线工作。

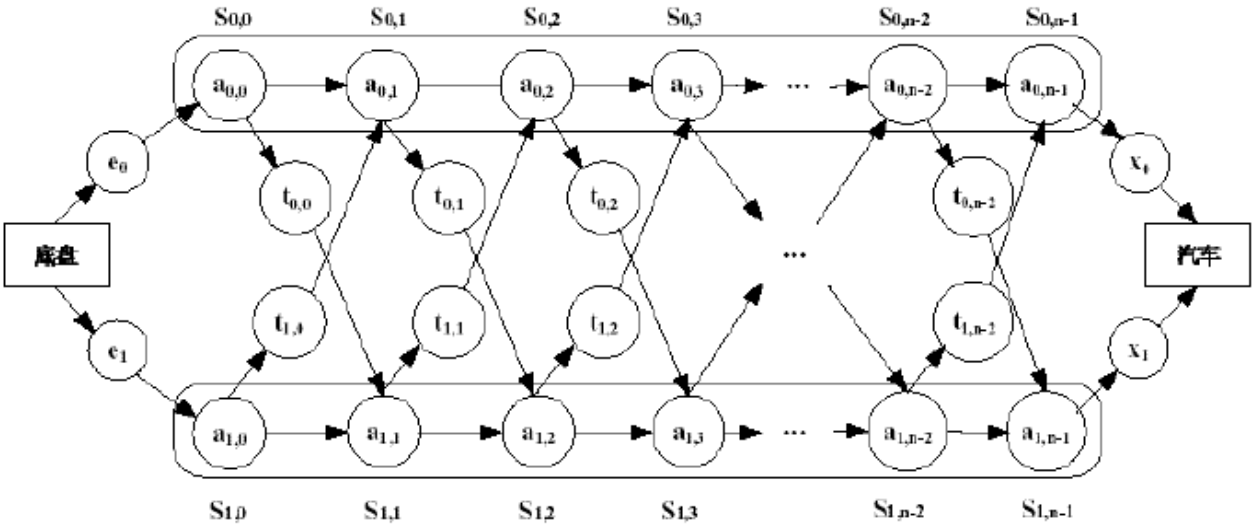

图 4-1 汽车装配线

(1) e0 和 e1 表示底盘分别进入装配线 0 和装配线 1 所需要的时间。

(2) 每条装配线有 n 个工位，第一条装配线的工位为 S0,0, S0,1, …, S0,n-1, 第二条装 配线的工位为 S1,0, S1,1, …, S1,n-1。其中 S0,k 和 S1,k（0≤k≤n-1）完成相同的任务，但所需时间可能不同。

(3) ai,j 表示在工位 Si,j 处的装配时间，其中 i 表示装配线（i=0 或 i=1），j 表示工位号（0≤j≤n-1）。

(4) ti,j 表示从 Si,j 处装配完成后转移到另一条装配线下一个工位的时间。

(5) x0 和 x1 表示装配结束后，汽车分别从装配线 0 和装配线 1 下线所需要的时间。

(6) 在同一条装配线上，底盘从一个工位转移到其下一个工位的时间可以忽略不计。

图 4-2 所示的流程图描述了求最短装配时间的算法，该算法的输入为：

n: 表示装配线上的工位数；

e[i]: 表示 e1 和 e2，i 取值为 0 或 1；

a[i][j]:表示 ai,j，i 的取值为 0 或 1，j 的取值范围为 0～n-1；

t[i][j]:表示 ti,j, i 的取值为 0 或 1，j 的取值范围为 0～n-1；

x[i]: 表示 x0 和 x1，i 取值为 0 或 1。

算法的输出为：

fi:最短的装配时间；

li:获得最短装配时间的下线装配线号（0 或者 1）。

算法中使用的 f[i][j]表示从开始点到 Si,j 处的最短装配时间。

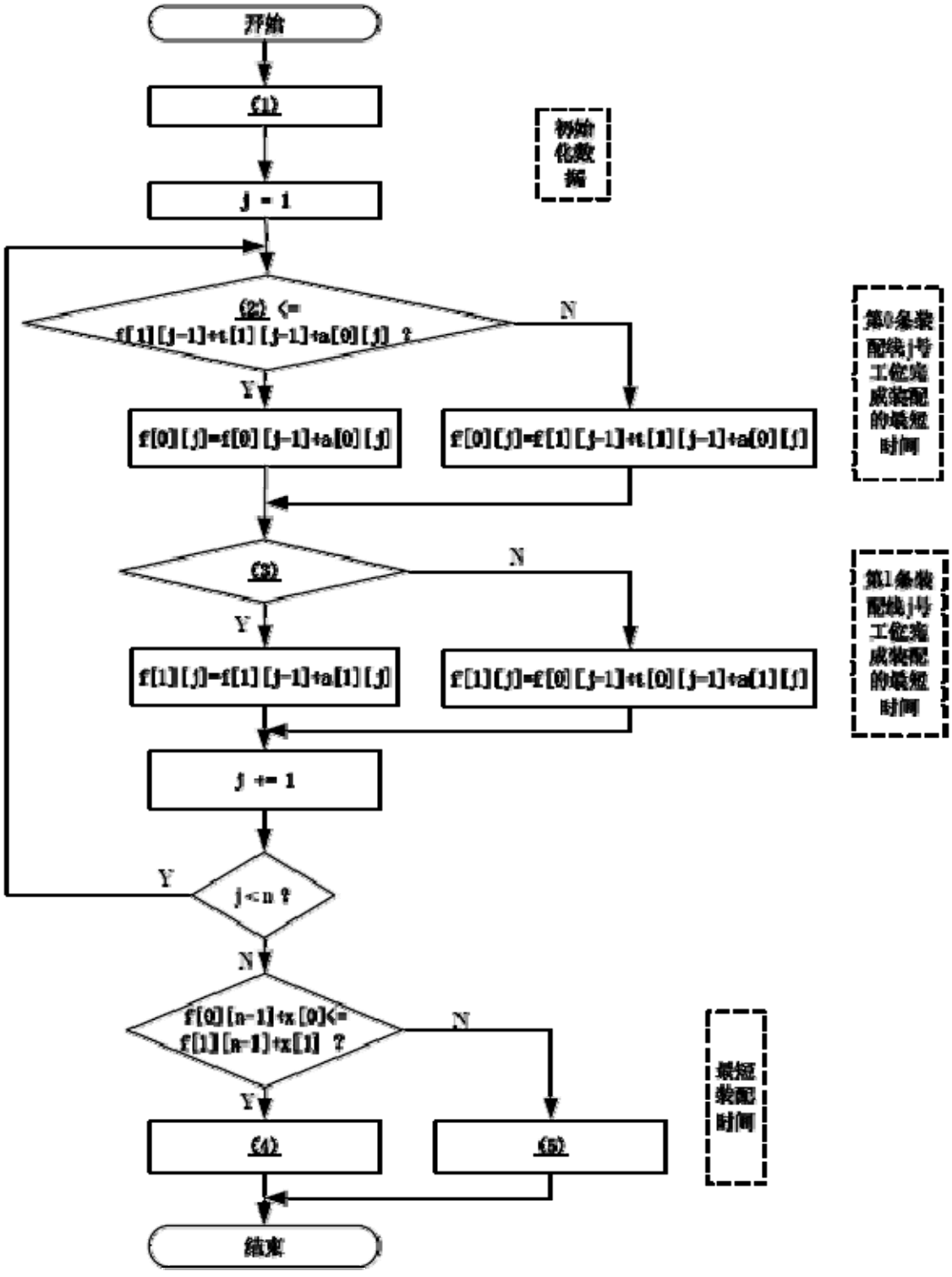

图 4-2 求最短装配时间算法

从下列的 3 道试题（试题五至试题七）中任选 1 道解答。

如果解答的试题数超过 1 道，则题号小的 1 道解答有效。

试题五(15 分)

阅读以下说明、图和 C 代码，将应填入__（n）__ 处的字句写在答题纸的对应栏内。

[说明]

一般的树结构常采用孩子－兄弟表示法表示，即用二叉链表作树的存储结构，链表中结点的两个链域分别指向该结点的第一个孩子结点和下一个兄弟结点。例如，图 5-1(a) 所示的树的孩子－兄弟表示如图 5-1(b)所示。

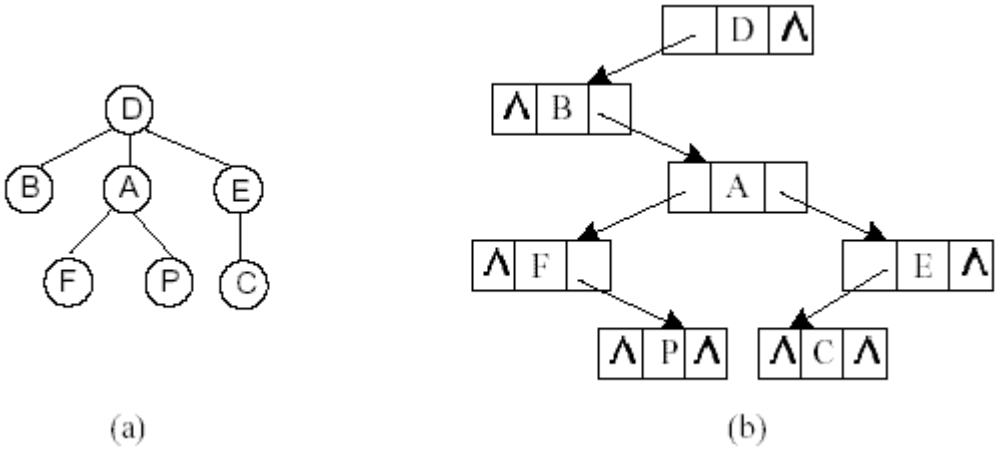

图 5-1 树及其孩子－兄弟表示示意图

函数 LevelTraverse()的功能是对给定树进行层序遍历。例如，对图 5-1 所示的树进行层序遍历时，结点的访问次序为：D B A E F P C 。

对树进行层序遍历时使用了队列结构，实现队列基本操作的函数原型如下表所示：

函数原型	说明
void InitQueue(Queue *Q)	初始化队列
Bool IsEmpty(Queue Q)	判断队列是否为空，若是则返回 TRUE,否则返回 FALSE

void EnQueue(Queue *Q,TreeNode p)	元素入队列
void DeQueue(Queue *Q,TreeNode *p)	元素出队列

Bool、Status 类型定义如下：

```
typedef enum {FALSE = 0,TRUE = 1} Bool;
typedef enum {OVERFLOW = -2,UNDERFLOW = -1,ERROR = 0,OK = 1} Status;
```

树的二叉链表结点定义如下：

```
typedef struct Node {
  char data;
  struct Node *firstchild,*nextbrother;
}Node,*TreeNode;
```

[函数]

```
Status LevelTraverse(TreeNode root)
{  /*层序遍历树，树采用孩子-兄弟表示法，root 是树根结点的指针*/
  Queue tempQ;
  TreeNode ptr,brotherptr;
  if (!root)
    return ERROR;
  InitQueue(&tempQ);
  (1) ;
  brotherptr = root -> nextbrother;
  while (brotherptr){ EnQueue(&tempQ,brotherptr);
    (2) ;
  }  /*end-while*/
  while ( (3) ) {
    (4) ;
    printf("%c\t",ptr->data);
    if ( (5) ) continue;
```

```
    (6) ;
    brotherptr = ptr->firstchild->nextbrother;
    while (brotherptr){ EnQueue(&tempQ,brotherptr);
      (7) ;
    }  /*end-while*/
  }  /*end-while*/
  return OK;
}/*LevelTraverse*/
```

试题六（共 15 分）

阅读以下说明和 C++代码，将应填入 （n） 处的字句写在答题纸的对应栏内。

[说明]

传输门是传输系统中的重要装置。传输门具有 Open（打开）、Closed（关闭）、Opening（正在打开）、StayOpen（保持打开）、Closing（正在关闭）五种状态。触发传输门状态转 换的事件有 click、complete 和 timeout 三种。事件与其相应的状态转换如图 6-1 所示。

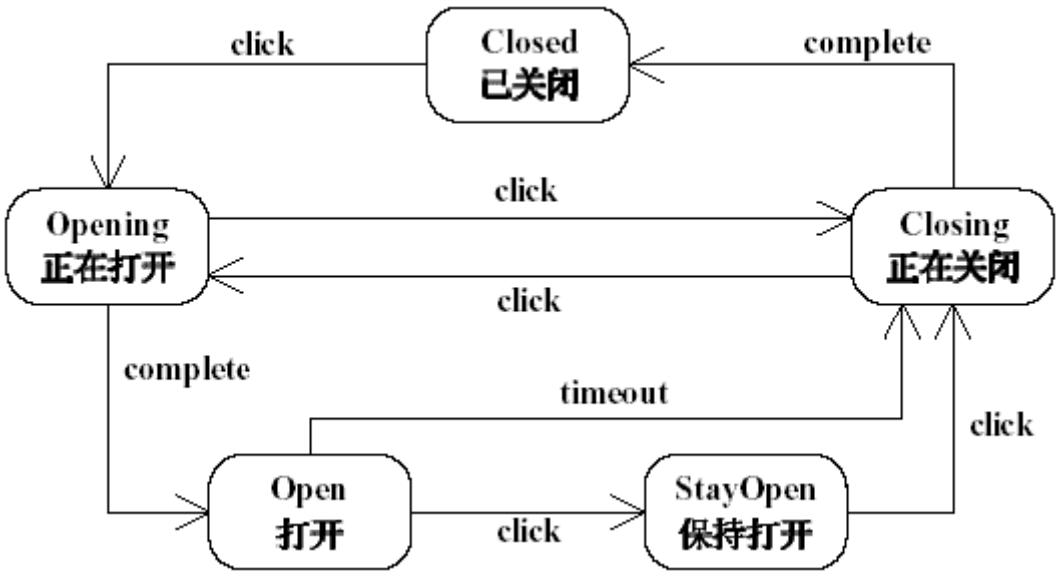

图 6-1 传输门响应事件与其状态转换图

下面的[C++代码 1]与[C++代码 2]分别用两种不同的设计思路对传输门进行状态模拟，

请填补代码中的空缺。

[C++代码 1]

```
const int CLOSED = 1; const int OPENING = 2;
const int OPEN = 3; const int CLOSING = 4;
const int STAYOPEN = 5; //定义状态变量，用不同整数表示不同状态

class Door {
private:
  int state; //传输门当前状态
  void setState(int state){ this->state = state; } //设置当前状态
public:
  Door():state(CLOSED){};
  void getState(){  //根据当前状态输出相应的字符串
    switch(state){
      case OPENING: cout <<"OPENING" << endl; break;
      case CLOSED: cout << "CLOSED" << endl; break;
      case OPEN: cout << "OPEN" << endl; break;
      case CLOSING: cout << "CLOSING" << endl; break;
      case STAYOPEN: cout << "STAYOPEN" << endl; break;
    }
  }
  void click() {  //发生 click 事件时进行状态转换
    if ( (1) ) setState(OPENING);
    else if ( (2) ) setState(CLOSING);
    else if ( (3) ) setState(STAYOPEN);
  }
  void timeout(){  //发生 timeout 事件时进行状态转换
    if (state == OPEN) setState(CLOSING);
  }
  void complete(){  //发生 complete 事件时进行状态转换
```

```
    if (state == OPENING) setState(OPEN);
    else if (state == CLOSING) setState(CLOSED);
  }
};
int main(){
  Door aDoor;
  aDoor.getState();  aDoor.click();  aDoor.getState();  aDoor.complete();
  aDoor.getState();  aDoor.click();  aDoor.getState();  aDoor.click();
  aDoor.getState(); return 0;
}
```

[C++代码 2]

```
class Door {
public:
  DoorState *CLOSED, *OPENING, *OPEN, *CLOSING, *STAYOPEN, *state;
  Door();
  virtual ~Door(){ ……    //释放申请的内存，此处代码省略};
  void setState(DoorState *state) { this->state = state; }
  void getState(){
    // 此处代码省略，本方法输出状态字符串，
    // 例如，当前状态为 CLOSED 时,输出字符串为“CLOSED”
  };
  void click();
  void timeout();
  void complete();
};


Door::Door(){
  CLOSED = new DoorClosed(this);        OPENING = new DoorOpening(this);
```

```
  OPEN = new DoorOpen(this);              CLOSING = new DoorClosing(this);
  STAYOPEN = new DoorStayOpen(this);      state = CLOSED;
}
void Door::click(){ (4) ;}
void Door::timeout(){ (5) ; }
void Door::complete(){ (6) ; }

class DoorState  //定义一个抽象的状态，它是所有状态类的基类
{
protected: Door *door;
public:
  DoorState(Door *door) { this->door = door; }
  virtual ~DoorState(void);
  virtual void click() {}
  virtual void complete() {}
  virtual void timeout() {}
};

class DoorClosed :public DoorState{  //定义一个基本的 Closed 状态
public:
  DoorClosed(Door *door):DoorState(door) {}
  virtual ~ DoorClosed (){}
  void click();
};
void DoorClosed::click() { (7) ; }
// 其它状态类的定义与实现代码省略

int main(){
  Door aDoor;
  aDoor.getState();  aDoor.click();    aDoor.getState();  aDoor.complete();
```

```
    aDoor.getState();  aDoor.timeout();  aDoor.getState();  return 0;
  }
```

试题七（共 15 分）

阅读以下说明以及 Java 程序，将应填入 （n） 处的字句写在答题纸的对应栏内。

[说明]

传输门是传输系统中的重要装置。传输门具有 Open（打开）、Closed（关闭）、Opening（正在打开）、StayOpen（保持打开）、Closing（正在关闭）五种状态。触发状态的转换事件有 click、complete 和 timeout 三种。事件与其相应的状态转换如图 7-1 所示。

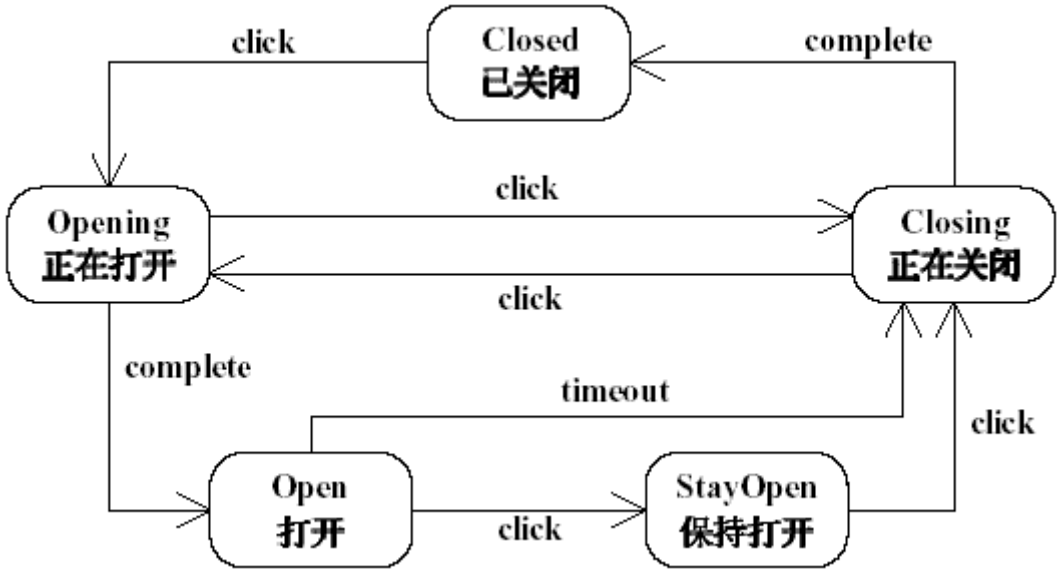

图 7-1 传输门响应事件与其状态转换图

下面的[Java 代码 1]与[Java 代码 2]分别用两种不同的设计思路对传输门进行状态模拟，请填补代码中的空缺。

[Java 代码 1]

```
public class Door {
  public static final int CLOSED = 1;    public static final int OPENING = 2;
  public static final int OPEN = 3;       public static final int CLOSING = 4;
  public static final int STAYOPEN = 5;  private int state = CLOSED;
  //定义状态变量，用不同的整数表示不同状态
  private void setState(int state){this.state = state;}  //设置传输门当前状
```

态

```
    public void getState(){
      // 此处代码省略，本方法输出状态字符串，
      // 例如，当前状态为 CLOSED 时,输出字符串为”CLOSED”
    }
    public void click() {    //发生 click 事件时进行状态转换
      if ( (1) ) setState(OPENING);
      else if ( (2) ) setState(CLOSING);
      else if ( (3) ) setState(STAYOPEN);
      //发生 timeout 事件时进行状态转换
      public void timeout(){ if (state == OPEN) setState(CLOSING); }
      public void complete(){  //发生 complete 事件时进行状态转换
      if (state == OPENING) setState(OPEN);
      else if (state == CLOSING) setState(CLOSED);
    }
    public static void main(String [] args){
      Door aDoor = new Door();
      aDoor.getState();    aDoor.click();      aDoor.getState();
aDoor.complete();
      aDoor.getState();    aDoor.click();      aDoor.getState();
aDoor.click();
      aDoor.getState();    return;
    }
  }
```

[Java 代码 2]

```
  public class Door {
    public final DoorState CLOSED = new DoorClosed(this);
    public final DoorState OPENING = new DoorOpening(this);
    public final DoorState OPEN = new DoorOpen(this);
```

```
    public final DoorState CLOSING = new DoorClosing(this);
    public final DoorState STAYOPEN = new DoorStayOpen(this);
    private DoorState state = CLOSED;

    //设置传输门当前状态
    public void setState(DoorState state){ this.state = state;}
    public void getState(){  //根据当前状态输出对应的状态字符串
      System.out.println(state.getClass().getName());
    }

    public void click(){ (4) ;}  //发生 click 事件时进行状态转换
    public void timeout(){ (5) ;}  //发生 timeout 事件时进行状态转换
    public void complete(){ (6) ;}  //发生 complete 事件时进行状态转换

    public static void main(String[] args){
      Door aDoor = new Door();
      aDoor.getState();  aDoor.click();    aDoor.getState();
aDoor.complete();
      aDoor.getState();  aDoor.timeout();  aDoor.getState();  return;
    }
  }

  public abstract class DoorState {  //定义所有状态类的基类
    protected Door door ;
    public DoorState(Door door) {this.door = door;}
    public void click() {}
    public void complete() {}
    public void timeout() {}
  }

  class DoorClosed extends DoorState{  //定义一个基本的 Closed 状态
    public DoorClosed(Door door) { super(door); }
```

```
    public void click() { (7) ; }
    //该类定义的其余代码省略
}
//其余代码省略
```