数据结构与算法课程设计题目

1．Skip List 的实现及分析

(1) 问题描述

Skip List 作为有序链表结构的一种扩展，如下图所示，其中 a 是普通的单链表；而 b 是在次基础上加上第二层（level 2）的额外指针，这些额外的指针指向间隔为 2 的下一个结点，skip list 因此得名；类似的 c 是加上 level 3 后的 skip list；d 是加上 level 4 后的 skip list。

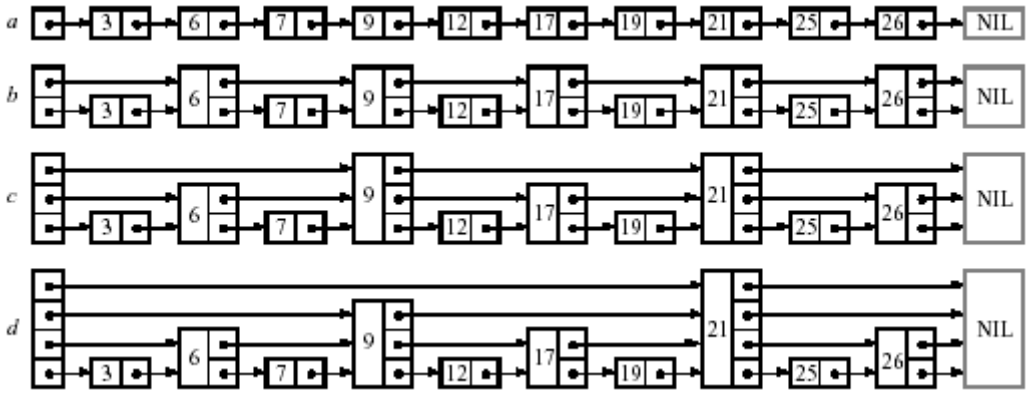


图 6-8 Skip List 的基本结构示意图

Skip List 上查找的基本思想是先从最高的 Level 层上查找，找到 key 所在的范围后，再从较低的层次继续重复查找操作，直到 Level 1。

Skip List 上的插入操作如下图所示。

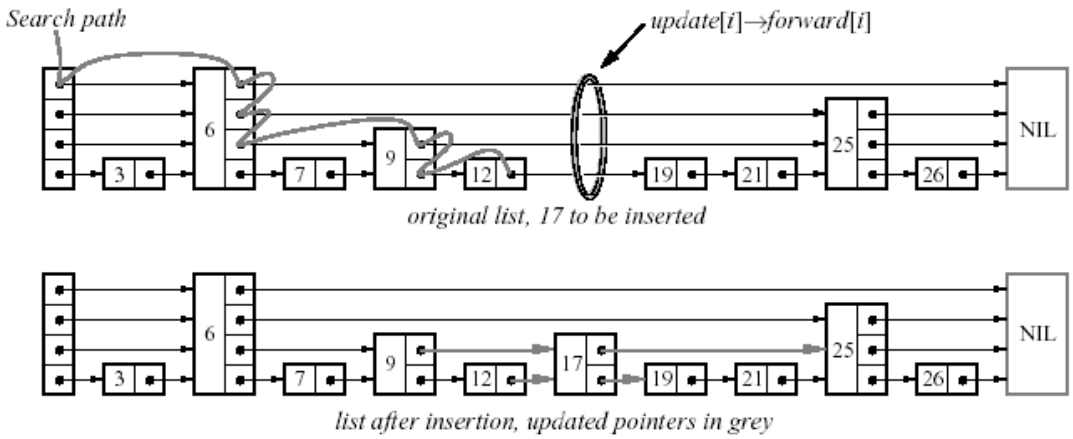


图 6-9 Skip List 上插入操作的示意图

Skip List 上的删除操作只需直接删除元素即可（包括指针的修整）。

构造并实现 Skip List 的 ADT，同时实现双向 Skip List 的 ADT 及循环 Skip List 的 ADT。

(2) 课程设计目的

认识并应用 Skip List 数据结构，并体会线性结构的变形。

(3) 基本要求

① ADT 中应包括初始化、查找、插入、删除等基本操作。

② 分析各基本操作的时间复杂性。

③ 针对一个实例实现 Skip List 的动态演示(图形演示)。

2. B-Trees 的实现及分析

(1) 问题描述

B-Trees 是一类满足特殊条件的 M 路查找树。首先说明 M 路查找树，M 路查找树是二元查找树的一般化，其结构如下图所示的 3 路查找树：M 路查找树中的任一结点至多存放 M-1 个数据，并至多拥有 M 棵子树；每个结点中的数据按升序排列 V1 < V2 < ... Vk (k <= M-1)，每个数据 Vi 都存在一棵左子树和一棵右子树，如果左子树不空的话，该子树中所有结点的值都小于 Vi,如果右子树不空的话，该子树中所有结点的值都大于 Vi。

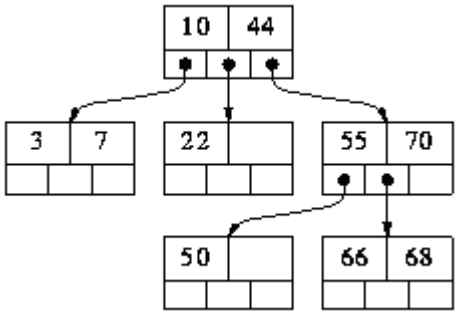


图 6-10 3-路查找树的结构示意图

B-Trees 是满足如下两个条件的 M 路查找树：

① 所有叶结点的高度相同。

② 除根之外的所有结点都至少是半满的，即该结点包含 M/2 或更多的值。

下图是一个 B-树的实例。

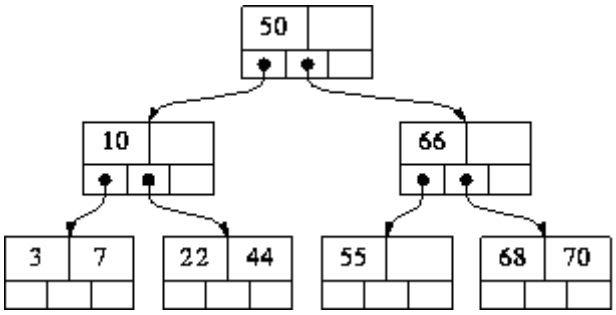


图 6-11 3-路 B-树的结构示意图

(2) 课程设计目的

认识并分析 B-树。

(3) 基本要求

① 实现在 B-树上的查找，并分析其时间复杂性。

② 实现 B-树的 ADT，包括其上的基本操作：结点的加入和删除。

③ 要求 B-树结构中的 M=3 或 5，实现其中的一种即可。

④ 实现基本操作的演示。

(4) 实现提示

主要考虑结点的分裂和和并。

3. AVL Tree 的实现及分析

(1) 问题描述

AVL(Adelson-Velskii and Landis)树是一株平衡的二元查找树。一株平衡的二元树就是指对其每一个结点，其左子树和右子树的高度之差不超过 1。下图就是一个 AVL 树的实例。

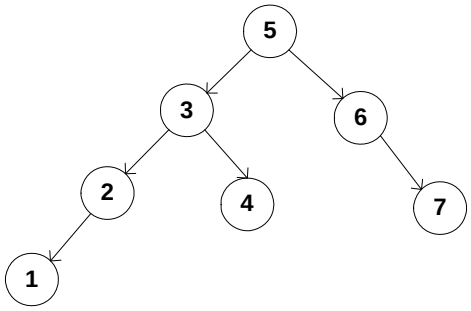


图 6-12 AVL 树的结构示意图

(2) 课程设计目的

认识 AVL Tree 数据结构。

编写程序实现 AVL 树的判别；并实现 AVL 树的 ADT，包括其上的基本操作：结点的加入和删除。BST 和 AVL 的差别就在平衡性上，所以 AVL 的操作关键要考虑如何在保持二元查找树定义条件下对二元树进行平衡化。

(3) 基本要求

① 编写 AVL 树判别程序，并判别一个二元查找树是否为 AVL 树。二元查找树用其先序遍历结果表示，如：5，2，1，3，7，8。

② 实现 AVL 树的 ADT，包括其上的基本操作：结点的加入和删除；另外包括将一般二元查找树转变为 AVL 树的操作。

(4) 实现提示

主要考虑树的旋转操作。

4. N-ary Trie 的实现和分析

(1) 问题描述

哈夫曼算法输出的结果就是一个二元 Trie，在二元 Trie 中，每个左子树分支用 0 表示，右子树分支用 1 表示。如下图就是就是一个二元 Trie 的示例图。

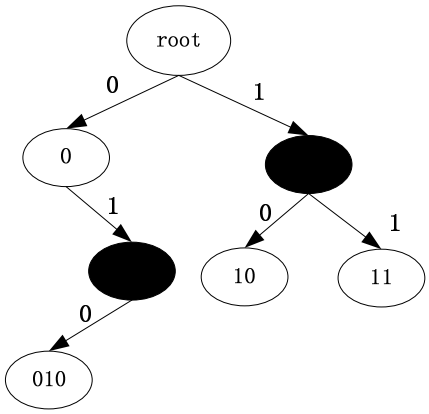


图 6-13 二元 Trie 的结构示意图

可以将二元 Trie 扩展到 N-ary Trie。在 N-ary Trie 中，每个结点都有 0~N 之间任何个儿子结点，其中每个分支都用一个相应的符号表示（在 N-ary Trie 中有 N 个不同的符号）。

例如一个电话本可用一个 N-ary Trie 表示，其中 N=10，分别表示 0~9 的十个数字，具体情况如下图所示：

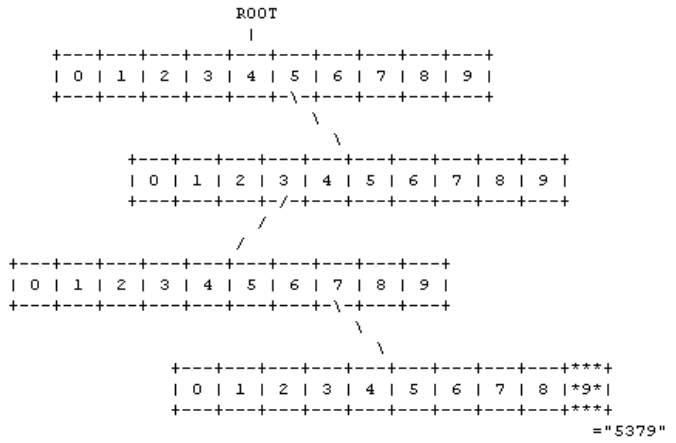


图 6-14 *N* 元 Trie 的结构示意图

(2) 课程设计目的

认识 N-ary Trie 结构，并能应用该结构解决实际问题。

(3) 基本要求

① 设计并实现 N-ary Trie 的 ADT（N=26，建立在英语上的 Trie），该 ADT 包括 Trie 的组织存储以及其上的基本操作：包括初始化，查找，插入，删除等。

② 应用 Trie 结构实现文本文档的索引化。首先扫描文本文档(存放在 txt 文件中)，然后在此基础上用 Trie 登记单词出现的位置（行号），最后在 Trie 上实现查询。

③ 用户输入的查询可以是针对一个单词的，也可是针对一个字符序列的。

(4) 实现提示

在树结构的基础上进行适当的修改。

5．Red-Black Tree 的实现和分析

(1) 问题描述

一棵具有红黑特征的树是一棵特殊的二元查找树，它在每个结点上增加一种额外属性——颜色（只能是红或黑）。

红黑树（Red-Black Tree）是一棵具有如下特征的二元查找树：

1．每一个结点或者是红色或者是黑色。

2．每个叶结点都是黑色。

3．如果一个结点是红色，那么它的两个孩子结点都是黑色。

4．从任何一个结点到所有以该结点为子树的叶结点的简单路径上拥有相同的黑色结点。

如下图就是一个红黑树的实例：

一般叶结点是不存放数据的，它作为哨兵用来表示已经到达叶结点。

从一个结点 x 到 x 子树中叶结点的路径（不包括 x）上黑色结点的个数称为 x 的黑色高度（*black-height*），标记为 **bh(x)**。

对于红黑树我们可以证明如下定理：任何一个具有 ***n*** 个内部结点的红黑树其高度至多为 2**log(*n*+1)**。该定理决定了红黑树可以保证查找时间复杂性一定时 **O(logn)**（具有和平衡树类似的性质）。

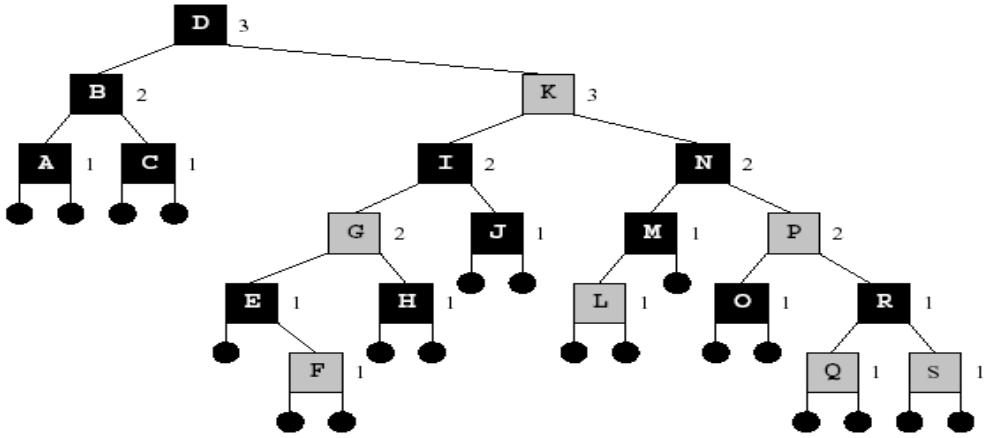


图 6-15 红黑树的结构示意图

(2) 课程设计目的

认识 Red-Black Tree 结构，并能依据其优点解决实际问题。

(3) 基本要求

① 设计并实现 Red-Black Tree 的 ADT，该 ADT 包括 Tree 的组织存储以及其上的基本操作：包括初始化，查找，插入和删除等。并分析基本操作的时间复杂性。

② 实现 Red-Black Tree ADT 的基本操作演示（要求应用图形界面）。

③ 演示实例为依次插入 A L G O R I T H M 并依同样的次序删除。

(4) 实现提示

主要考虑树的旋转。

6. Binomial heaps 的实现和分析

(1) 问题描述

二项堆是二项树的集合，而二项树是基于递归定义的，即二项树 B_k 是一棵在 B_{k-1} 的基础上递归定义的有序树，它由两个 B_{k-1} 形成链接形成：一棵 B_{k-1} 树的根是另一棵 B_{k-1} 根的最左儿子。下图给出了图示：

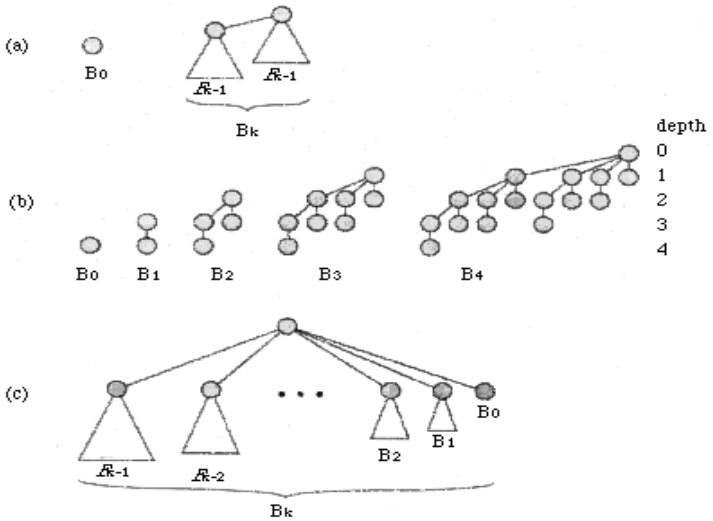


图 6-16 二项堆的结构示意图

图(*a*)说明二项树的递归定义，图(*b*)给出 B_0~B_4 的图例，图(*c*)表明了 B_k 的另一种表现形式。

二项堆 H 是满足如下二项堆特征的二项树集合：

1．H 中的每棵二项树都满足堆特征，即每个结点的值都大于等于其父结点的值。

2．H 中的每棵二项树根结点的度互不相同。

其中性质 2 意味着对于任何 k，H 中只可能是存在或不存在 B_k。如果 H 中存在 B_k，就会对 H 贡献 2^k 个结点，用二进制表示就是在第 k 为 1，所以对于 n 个结点的 H，n 的二进制表示$[b_{\log n}, b_{\log n-1} \ldots b_0]$中第 i 为 b_i 就表示了 H 中是否存在 B_i，相反的事实也成立。根据这一事实也可以看出 n 个结点的二项堆至多$\lfloor \log n \rfloor+1$ 个二项树。

下图为一二项堆的实例：

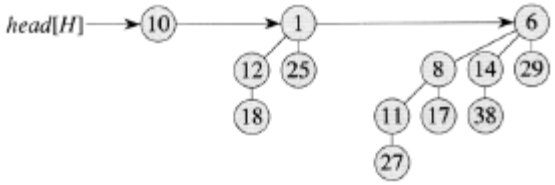


图 6-17 一二项堆实例

(2) 课程设计目的

认识二项树、二项堆数据结构，并能应用该结构解决实际问题。

(3) 基本要求

① 设计二项堆 ADT，其上的基本操作包括：**Make Heap (x), Find-Min, Union，Insert，Extract-Min，Decrease Key (x)，Delete**。

② 实现二项堆 ADT，包括实现二项堆的存储结构以及其上的基本操作，并分析基本操作的时间复杂性。

③ 实现二项堆 ADT 的基本操作演示（要求应用图形界面）。

(4) 实现提示

其中的 UNION 是关键操作，部分其它操作可在此基础上构造。UNION 操作可类似二进制加法进行。

7．遗传算法的模拟

(1) 问题描述

遗传算法是以达尔文生物进化论为基础，借鉴自然界中物种进化原理，依据优胜劣汰而达到优化的规律而创建的一种数学模型和算法，如下图所示。

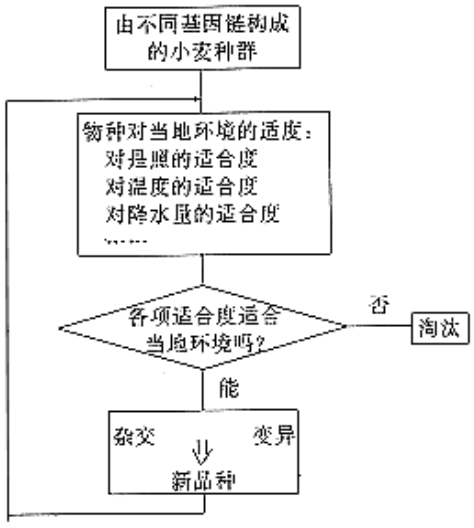


图 6-18 遗传算法的生物学背景

该图以小麦品种改良的过程为例说明遗传算法（也可称为基因算法）的基本原理：优化问题的可能解被称为是个体（*individuals*），首先考虑可能解（个体）组成的集合，即群体（*population*）；然后依据环境特征（优化问题特征）评定各个体的优劣（其适应度（*fitness*）来定义）；对适应度较差的个体进行淘汰，选取适应度好的个体（类比生物选择），在其上进行杂交，变异等操作形成新的群体；最后再进入下一轮遗传进化，上述过程不断迭代，直到群体满足了某条件，此时出现了满足要求的优化解。

将上述过程形式化后就得到如下图所示的遗传算法结构框图：

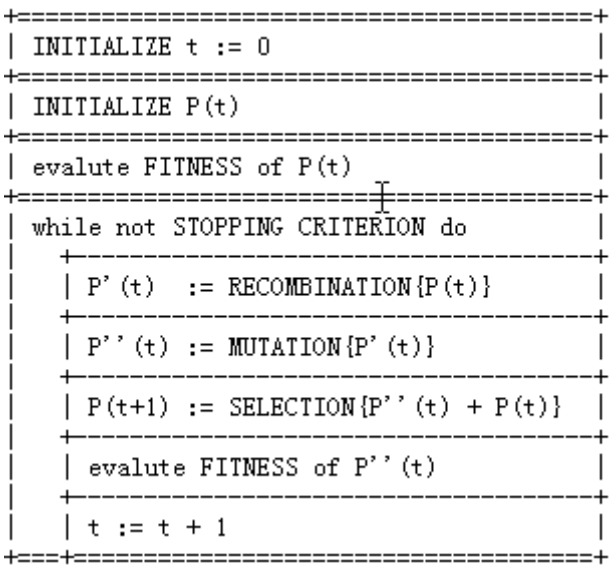


图 6-19 遗传算法基本结构

其中 P(t)为 t 时刻的父代，而 P"(t)为 t 时刻的子代。

对于计算机问题而言，一般要将问题的解进行编码，编码成二进制字符串，杂交和变异就在这些字符串上进行。

遗传算法可以很好解决很多的优化问题。

(2) 课程设计目的

体会遗传算法思想，能够设计并编写遗传算法的相关操作函数，并能够应用遗传算法求解具体问题。

(3) 基本要求

① 编写遗传算法的基本操作函数，包括选择，变异，交叉等。

② 应用遗传算法实现求解如下函数的极值

$$f(x)=x*\sin(10*\pi*x)+1.0 \quad x\in[-1,2]$$

③ 结果精度要求在小数点后六位。

④ 给出算法效率分析的实验结果。

(4) 实现提示

根据精度要求确定个体的二进制编码位数，同时要确定该编码和[-1,2]间数的换算规则，由于是求最大值的问题，所以适应度函数就可选为 f(x)，值越大的个体适应度越好。关于交叉和变异的方法参阅相关资料。

8．基于细胞自动机(Cellular Automata)实现 Gossip 模型的模拟与分析

(1) 问题描述

细胞自动机是一个时间和空间都离散的动态系统，每个细胞自动机都是由细胞单元

(cell)组成的规则网格，每个细胞元都有 k 个可能的状态，其当前状态由其本身及周围细胞元的前一状态共同决定。

例如：*The Game of Life*

1. 细胞元形成二维数组。
2. 每个细胞元有两个状态：living 和 dead。
3. t+1 时刻的状态由 t 时刻的状态决定。
4. 状态变迁规则 1：一个 living 的细胞元依旧保持存活，如果其周围存在 2 或 3 个 living 细胞元，否则死亡。
5. 状态变迁规则 1：一个 dead 的细胞元依旧保持死亡，除非其周围恰好 3 个 living 细胞元。

一个演化图例如下：

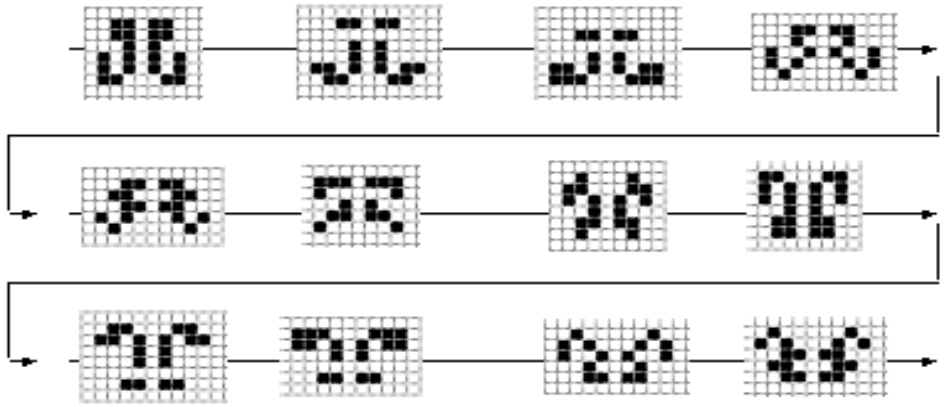


图 6-20 细胞自动机演示图例

细胞自动机从某种程度上反映了生物群落的演化，也可应用到许多计算机问题中。

(2) 课程设计目的

了解细胞自动机的基本原理，并能实现对细胞自动机的模拟以及在此基础上做有价值的分析。

(3) 基本要求

① 编写 Gossip 模型的模拟(最好有图形界面，可用 C 的图形库)。

② 该模型是一个二维模型，并按如下条件处理边界问题：右边界的右邻居是左边界，同理，左边界、上边界、下边界也一样，

③ 该模型的基本规则为：如果你的邻居中有人得知某消息，那么你将有 5%的概率从其中获知消息的一个邻居那里获得该消息（即如果只有一个邻居有消息，得知概率为 5%，有两个邻居就是 10%，依此类推）；如果你已经得知该消息，那么你将保存。

④ 依据上述模拟得出 Gossip 模型较为全面的实验结果：消息覆盖率随时间的变化；不同概率值的影响等。

(4) 实现提示

可用数组实现自动机的存储。

9．基于细胞自动机(Cellular Automata)实现 tribute 模型的模拟与分析

(1) 问题描述

细胞自动机是一个时间和空间都离散的动态系统，每个细胞自动机都是由细胞单元(cell)组成的规则网格，每个细胞元都有 k 个可能的状态，其当前状态由其本身及周围

细胞元的前一状态共同决定。其具体细节在上题中已有所描述，此处略去。

(2) 课程设计目的

了解细胞自动机的基本原理，并能实现对细胞自动机的模拟以及在此基础上做有价值的分析。

(3) 基本要求

① 编写 tribute 模型的模拟（最好有图形界面，可用 C 的图形库）。

② 该模型是一个一维模型，包含 N 个参与者（细胞元，N 可配置，缺省为 10，此处就用 10 说明问题），将其命名为 actors。这 10 个 actors 组织成一个环形，每个 actor 都有一定的初始化财富量（可随机确定，在 300~500 之间均匀分布，也可设定），标记为 W。

③ 该模型的基本规则为：

1. Actors 只可以和它的邻居（左邻居和右邻居）进行交互，只有两种可能的交互动作：一是向其邻居索取供品(demand tribute)，二是和其邻居结为联盟(alliance)。
2. 一个离散的时间片（表示 1 年）内，3 个随机选择的 Actors 一个接一个的被激活，每个 Actor(A)会向他的其中一个邻居(T)索取 Tribute，T 可以 pay，也可以拒绝并 fight，如果 pay，T pay A $\min(250,W_T)$；如果 fight，交战双方都会造成财富损失，每方损失的是对方财富量的 25%，如果其中一方的财富不足以承担这一数量时，双方的损失将成比例缩减，此时财富不足的那一方（B）将失去全部财富，引起另一方（A）的财富损失为 0.25*(B/(0.25*A))*B。
3. A 选取 T 的原则是使得 $W_T * (W_A–W_T)/W_A$ 最大，但如果没有 T 使得该值大于 0，A 选择不动作。
4. T 选取 pay 或 fight 的哪一动作依据的是哪一种动作花费更少。
5. 另外，每一年，每个 Actors 都有少量额外的财富（20）补充进来。
6. 作为 A 和 B 交互的副产品，会产生一个 A 对 B 的承诺值（Commitments），简称 C，C 会在如下三种情况下增加：A pays tribute to B；A receives tribute from B；和 A fights on the same side as B。而当 A fight on opposite sides with B 时 C 会降低。
7. C 值的范围为 0~1，每次增加及减少幅度都是 0.1。
8. 初始化时，各 Actors 对其它 Actor 的 C 值为 0。又由于规则 6 的对称性，所以任何时刻 A 对 B 的 C 值和 B 对 A 的 C 是相同的。
9. 引入 Commitments 后，A 选取 T 的范围扩大，不仅仅是其邻居 Actor，可以是任何一个 Actor。当 A 收取 T 的贡品时，位于 A 和 T 之间的 Actor 会根据其与 A 和 T 的 C 值大小决定加入 C 值较大的一方，如果两 C 值相同，则保持中立。只有当 A 和 T 之间的所有 Actor（环形的两个方向均可）都加入 A 方，A 才能向 T 收取贡品。对于存在若干个满足上述条件的 T，选取原则将按照规则 3 进行扩展。
10. 规则 9 中，如果 K 要加入 A(T)，仅当 K 和 A(T)相邻，或 K 和 A(T)之间的所有 Actor 都加入 A(T)。
11. A 形成的联盟的财产计算为：$W= W_A+C(A1,A)*W_{A1}+\ldots$，其中 C(A1,A)为 A1 对 A 的承诺值，A1 为联盟中的一名成员，W_{A1} 为 A1 的财产。收取的贡品仅归 A 所有，而战斗的负担由联盟共同承担，其费用比例就是各 Actor 的 C 值。

12. 所有 Actor 的 C 值、W 值对任何一个 Actor 都是可见的，在任何计算中都可使用。

④ 上述规则描述了 Tribute 模型，规则 1 到 5 是未引入 Commitments 时的基本模型，基本模型的模拟是本设计的要求内容；加上规则 6 到 12 是引入 Commitments 后的扩展模型，扩展模型的模拟是本设计的选做内容。

⑤ 模拟时间应超过 1000 年。

⑥ 根据上述模拟得出 Tribute model 较为全面的实验结果：财富的变化规律；各参数对结果的影响等。

(4) 实现提示

可用循环队列实现自动机的存储。

10. 蚁群算法在旅行商问题中的应用

(1) 问题描述

蚁群算法是在对真实蚂蚁的观测基础上提出的，单个蚂蚁是不具智能的，但生活在一起的蚁群却总是能够在蚁穴和食物间找到一条几乎是最短的路径。这就要靠蚁群的智能，蚁群算法就基于这一智能。下面的图例给出蚁群智能的基本思想。

蚂蚁会在自己走过的路径上留下生化信息素，同时它也会选择地面上信息素较多的路径行走。对于下图的第二种情况，因为路上有了障碍物，所以蚂蚁需要绕过障碍物，该咋么样绕过障碍物，由于没有信息素作为指导，所以起先只能是随机的选择从左或从右走（第三个图所示）；但是随着行走次数的增加，较短的路径上留下的信息素就会较多，就有更多的蚂蚁行走，信息素进一步增多，最终较短的路就成为了选择的路（第四个图）。这就是蚁群智能的基本原理。

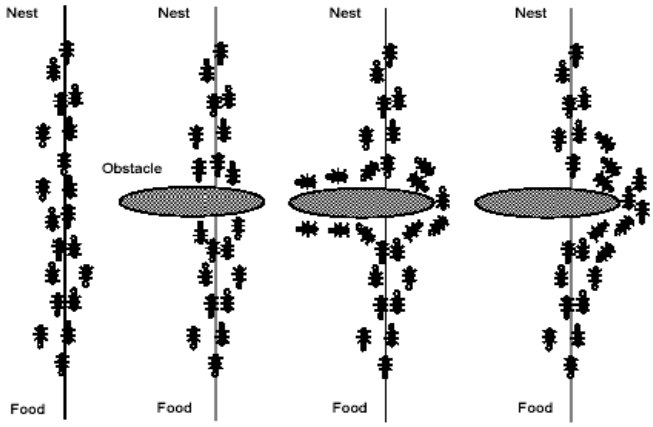


图 6-21 蚁群算法的基本思想

蚁群算法就是应用蚁群智能实现的算法，可以用它来实现在解空间上进行最优解的搜索，很多计算机问题可以转化为搜索最优解的问题。如旅行商问题（求解遍历全部城市的最短路经），就可用蚁群算法求解，在旅行商问题中，蚁穴到食物就对应遍历全部城市，蚂蚁就对应旅行商。

(2) 课程设计目的

了解蚁群算法的思想，并能应用蚁群算法求解具体问题。

(3) 基本要求

① 应用蚁群算法求解 TSP 问题。

② TSP 中的城市数量不少于 30 个，组成完全图，边上的权值自定。

③ 蚂蚁数量可配置，迭代次数可配置。

④ 给出较全面的实验结果：结果路经及长度；蚁群算法执行时间；不同参数值（蚂蚁数量，迭代次数）的影响等。

⑤ 迭代过程需要用图形界面表示(可用 C 的图形库)。

11. 实现计算几何软件包

(1) 问题描述

计算几何处理的基本对象是二维平面上的点和线段，以及靠这些点和线段形成的各种图形如凸多边形。在计算几何中，线段及其上的基本操作时计算几何的基础。

线段的基本操作如下：

① 叉积

对于 $p_1=(x_1, y_1)$ 和 $p_2=(x_2, y_2)$，p_1 和 p_2 的叉积 $p_1\times p_2$ 定义为：

$$\begin{aligned} p_1 \times p_2 &= \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1y_2 - x_2y_1 \\ &= -p_2 \times p_1 . \end{aligned}$$

如果 $p_1\times p_2$ 为正数，相对于原点来说，p_1 在 p_2 的顺时针方向上，如为负数，则在逆时针方向上。如果是对于公共端点 p_0，则计算叉积（p_1-p_0）×（p_2-p_0）就可判断线段 p_0p_1 在 p_0p_2 的顺时针方向还是逆时针方向。

② 连续线段是向左转还是向右转

该问题考虑的是连续线段 p_0p_1 和 p_1p_2 在 p_1 处是向左转还是向右转，如下图所示：

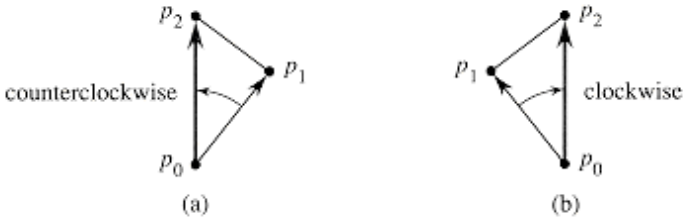


图 6-22 连续线段转向问题

可用叉积（p_2-p_0）×（p_1-p_0）来计算该值，如果该叉积为负，则在 p_1 点要向左转，如果该叉积为正，则在 p_1 点要向右转。本课程设计的基本内容是实现上述基本操作并据此解决如下两个基本问题：

① 确定两条线段是否相交

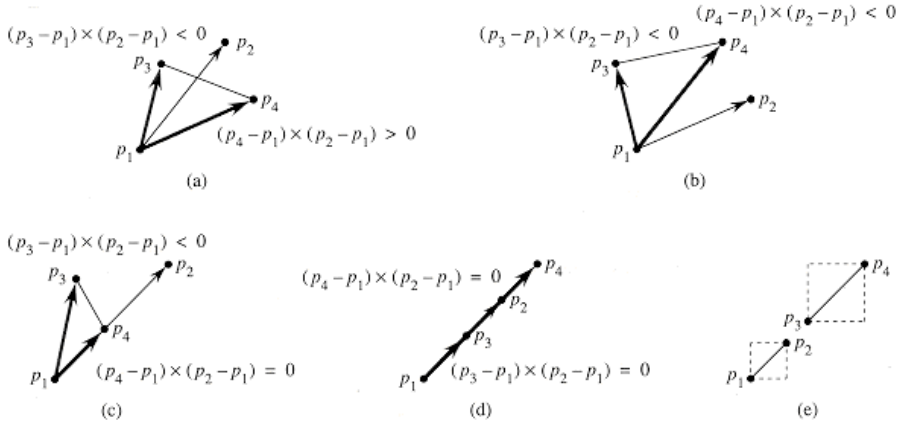


图 6-23 两条线段的相交问题

② 实现寻找凸包的 Graham 扫描法

该算法的基本思想如下图所示：

算法的伪代码如下图所示：

```
GRAHAM-SCAN(Q)
1  let p0 be the point in Q with the minimum y-coordinate, or the leftmost such point
in case of a tie
2  let ⟨p1, p2, . . . , pm⟩ be the remaining points in Q, sorted by polar angle in
counterclockwise order around p0 (if more than point has the same angle, remove all
but the one that is farthest from p0)
3  top[S] ← 0
4  PUSH(p0, S)
5  PUSH(p1, S)
6  PUSH(p2, S)
7  for i ← 3 to m
8        do while the angle formed by points NEXT-TO-TOP(S), TOP(S), and pi makes
a nonleft turn
9              do POP(S)
10           PUSH(S, pi)
11 return S
```

图 6-24 Graham 的伪代码

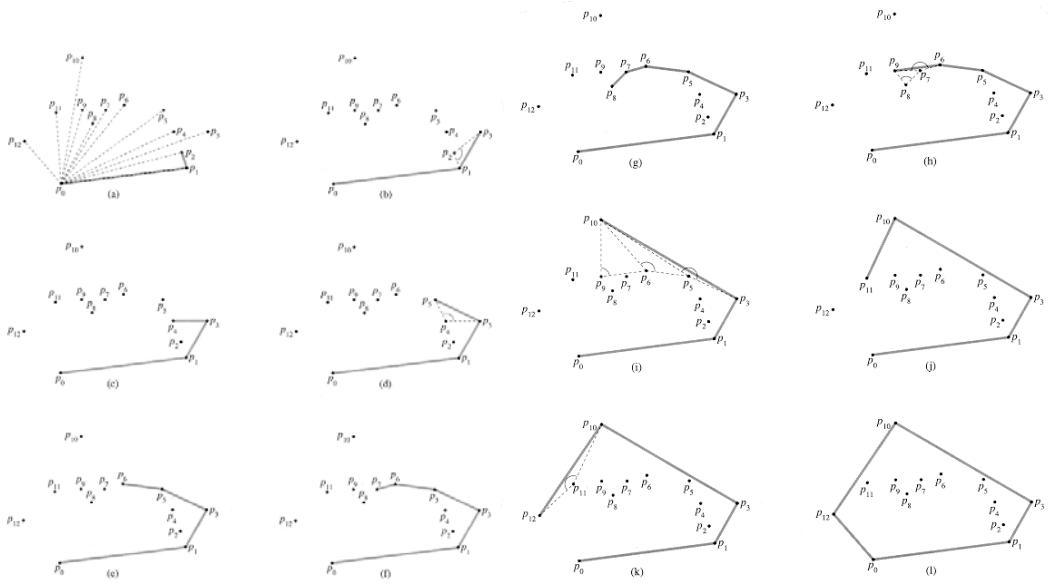


图 6-25 Graham 扫描算法

(2) 课程设计目的

应用数据结构与算法基本知识解决实际问题，对计算几何建立初步的认识。

(3) 基本要求

① 定义合适的数据结构，实现叉积、方向判断、相交判断、寻找凸包四个算法。

② 算法演示中的点和线段在二维平面随机生成。

③ Graham 扫描法要给出分析（最好是证明）算法的正确性，并分析算法的时间复杂性。

④ 最好用图形界面说明算法结果。

(4) 实现提示

需要查阅相关资料。

12. 长整数的代数计算

(1) 问题描述

设计数据结构完成长整数的表示和存储，并编写算法来实现两长整数的加、减、乘、除等基本代数运算。

(2) 课程设计目的

能够应用线性数据结构解决实际问题。

(3) 基本要求

① 长整数长度在一百位以上。

② 实现两长整数在取余操作下的加、减、乘、除操作，即实现算法来求解 a+b mod n, a-b mod n, a×b mod n, a÷b mod n。

③ 输入输出均在文件中。

④ 分析算法的时空复杂性。

(4) 实现提示

需将长整数的加法转化为多个一般整数加法的组合。

13. 实现并对比三种基本的字符串匹配算法

(1) 问题描述

字符串匹配问题是：假定文本时一个长度为 n 的数组 T[1..n]，模式是一个长度为 m≤n 的数组 P[1..m]，如果 0≤s≤n-m，并且 T[s+1..s+m] = P[1..m]，则称模式 P 完成了和 T 的匹配，且 P 在 T 中出现的位移为 s，如下图所示：

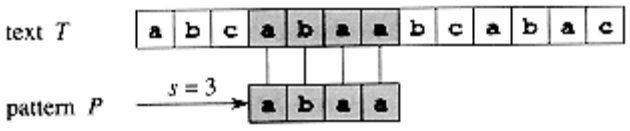


图 6-26 字符串匹配问题

本课程设计要实现三种基本的字符串匹配算法：朴素匹配算法；Rabin-Karp 算法；KMP 算法。

① 朴素匹配算法

是最简单的匹配算法，可以形象地看成是用一个包含模式 P 的模板沿文本滑动，同时对每个位移注意模板的上的字符是否与文本中的相应字符相等，如下图所示：

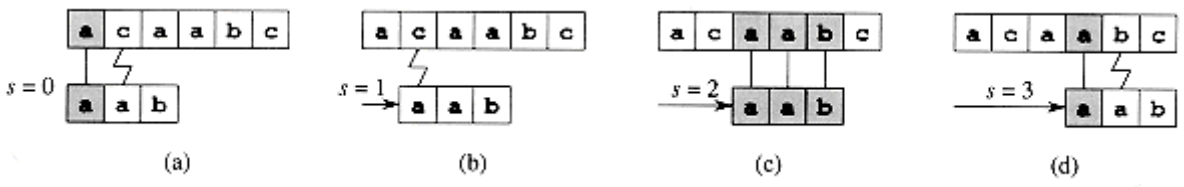


图 6-27 朴素字符串匹配算法示意图

② Rabin-Karp 算法

基本思想是将模板 P 用一个数 p（通常是十进制）表示，如果字符串中的每位都是 0-9 的整数：

p = P[m] + 10（P[m-1] + 10（P[m-2]+ … + 10(P[2] + 10P[1])…））

同时将 T 中每个 m 位连续的串上计算出同样的正数值，有 n-m 个：t_1，t_2，…，t_{n-m}。一种较为巧妙的方法是在 t_s 基础上可按如下方式计算 t_{s+1}：

$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$

由于这样的值非常大，可能会导致溢出，通常的处理办法是计算 p mod q 和 t_s mod q，计算时会用到性质 a+b mod n = a mod n + b mod n；a * b mod n = a mod n * b mod n。

Rabin-Karp 算法的具体过程如下图所示：

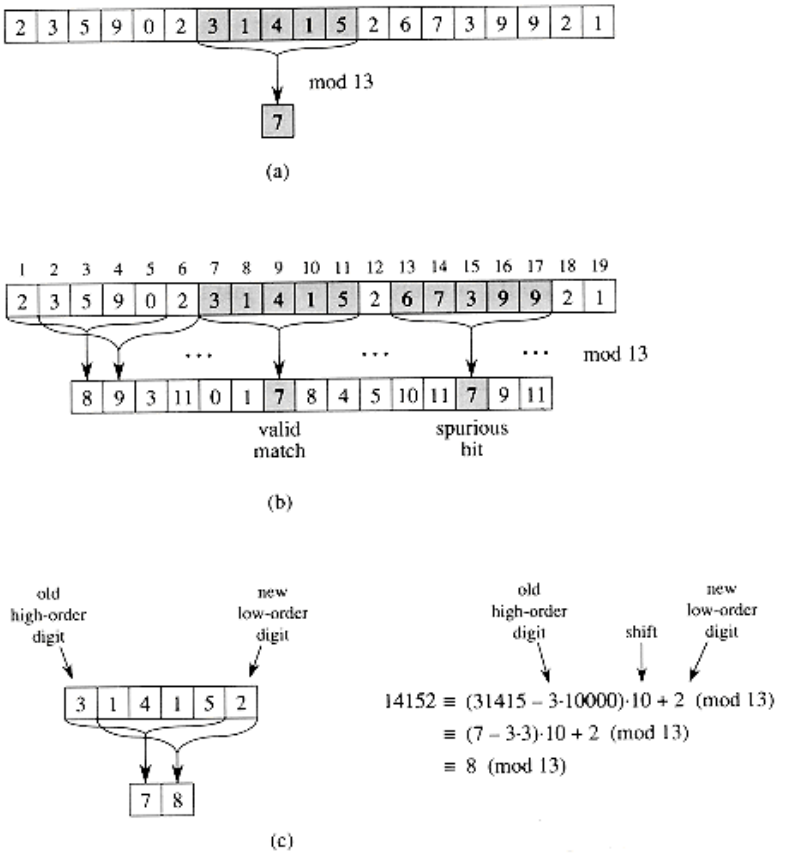


图 6-28 Rabin-Karp 字符串匹配算法示意图

③ KMP 算法

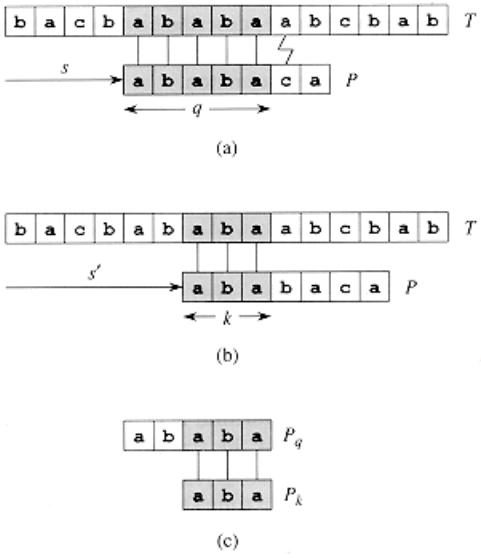


图 6-29 模式匹配中的信息重利用

KMP 算法是一种设计得很巧妙但也较复杂的一种算法，是应用自动机思想实现的匹配算法。其基本思想是如果已经完成了 P 部分字符（一个前缀 q）的匹配，那么在匹配 T 后面的字符时可以应用 P 模式的信息减少比较次数，如图 6-29 所示。

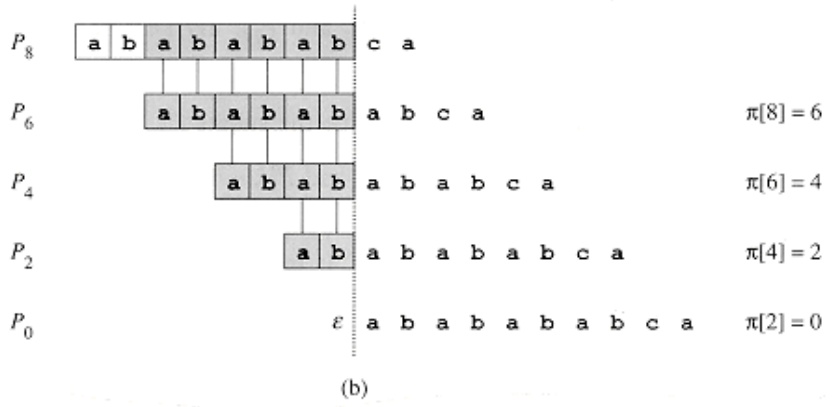
首先需要计算模式 P 的前缀函数 π：{1，2，…，m} → {0，1，…，m-1}满足：

$\pi[q] = \max\{k: k<q \text{ 且 } P_k \text{ 是 } P_q \text{的后缀}\}$

对于模式 abababababca 的完整前缀函数 π 如下图所示：

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

(a)



(b)

图 6-30 前缀函数 π

KMP 算法的基本思想是用函数 π 来确定匹配的状态迁移，算法的伪代码如下图所示：

```
KMP-MATCHER(T, P)
1 n ← length[T]
2 m ← length[P]
3 Π ← COMPUTE-PREFIX-FUNCTION(P)
4 q ← 0
5 for i ← 1 to n
6       do while q > 0 and P[q + 1] ≠ T[i]
7             do q ← Π[q]
8          if P[q + 1] = T[i]
9            then q ← q + 1
10          if q = m
11            then print "Pattern occurs with shift" i - m
12                 q ← Π[q]
```

图 6-31 KMP 算法伪代码

(2) 课程设计目的

应用数据结构与算法基本知识解决实际问题，对字符串匹配形成一定的认识。

(3) 基本要求

① 编程实现三种字符串匹配算法。

② 分析三种算法的时间复杂性，通过应用三种算法在一个较长英文文本中查找一个

子串的实验数据来对比三种算法的运行时间。

(4) 实现提示

需要查阅相关资料，尤其是 KMP 算法。

14. 文档集合上的查询

(1) 问题描述

设计数据结构完成在一个文档集合的存储，并构造算法实现其内容的查询。该设计包括三个部分：

(一) 应用数据结构完成文档集合的内容(基于单词的)存储，并为下一步的查询建立索引。

(二) 就单个单词的查询请求，设计算法进行查询。

(三) 对多个单词通过 AND 和 OR 构造的复杂查询进行处理(此处可只做两个单词的情况)。

具体情形如下面的例子：

Example

Doc1: I like the class on data structures and algorithms.

Doc2: I hate the class on data structures and algorithms.

Doc3: Interesting statistical data may result from this survey.

Here are the answers to some queries:

Query 1: data

Doc1, Doc2, Doc3

Query2: data AND structures

Doc1, Doc2

Query 3: like OR survey

Doc1, Doc3

图 6-32 文档集合上的查询实例

(2) 课程设计目的

用线性结构组织信息，查找算法的选择与应用。

(3) 基本要求

① 文档集合中的文档数不能少于 20 个。

② 数据结构的设计以及查找算法的构造应考虑如何最大程度的提高查询效率。

③ 查询效率的提高应是综合多种查询的，而不是只针对一种查询的优化。

④ 给出查询效率的模拟实验数据。

(4) 实现提示

AND 和 OR 查询可转变为单个单词查询结果的组合。

15. 应用堆实现一个优先队列并实现作业的优先调度

(1) 问题描述

优先队列 ***priority queue*** 是一种可以用于很多场合的数据结构，该结构支持如下几本操作：

•Insert (S, x) – 将元素 x 插入集合 S

•Minimum (S) – 返回 S 中最小的关键字

•Extract–Min (S) – 删除 S 中的最小关键字

•DecreaseKey(S,n,key) – 将 S 中的结点 n 的关键字减小为 key

要求以堆作为辅助结构设计并实现一个优先队列。要将堆结构嵌入到队列结构中，

以作为其数据组织的一部分。

应用该优先队列实现作业的优先调度：

一个作业 t_i =（s_i，e_i），s_i 为作业的开始时间（进入时间），e_i 为作业的结束时间（离开时间）。作业调度的基本任务是从当前在系统中的作业中选取一个来执行，如果没有作业则执行 nop 操作。本题目要求的作业调度是基于优先级的调度，每次选取优先级最高的作业来调度，优先级用优先数（每个作业一个优先数 p_i）表征，优先数越小，优先级越高。作业 t_i 进入系统时，即 s_i 时刻，系统给该作业指定其初始优先数 $p_i = e_i - s_i$，从而使越短的作业优先级越高。该优先数在作业等待调度执行的过程中会不断减小，调整公式为：$p_i = p_i - w_i$，其中的 w_i 为作业 t_i 的等待时间：w_i = 当前时间-s_i。一旦作业被调度，该作业就一直执行，不能被抢占，只有当前执行作业指向完成时，才产生下一轮调度。所以可以在每次调度前动态调整各作业的优先数。

编程实现这样一个作业调度系统。

(2) 课程设计目的

熟悉堆结构的应用，优先队列的构造，解决实际问题。

(3) 基本要求

① 给出优先队列的 ADT 描述，包括队列的逻辑结构及其上基本操作。

② 以堆结构为辅助结构实现优先队列的存储表示并实现其上的基本操作。

③ 作业集合中的各作业随机生成，根据作业的 s 属性和 e 属性动态调整作业队列，不断加入作业，作业结束删除作业。

④ 要对作业调度的结果给出清晰的输出信息，包括：何和作业进入，何时调度哪个作业，何时离开，每个作业等待多长时间，优先数的动态变化情况等。

(4) 实现提示

无。

16. 应用小大根交替堆实现双端优先队列

(1) 问题描述

双端优先队列是一个支持如下操作的数据结构：

•Insert (S, x) – 将元素 x 插入集合 S

•Extract –Min (S) –删除 S 中的最小关键字

•Extract –Max (S) –删除 S 中的最大关键字

可用小大根交替堆来实现对上述三个操作的支持。小大根交替堆是一个满足如下小大根交替条件的完全二元树：如果该二元树不空，那么其上的每个元素都有一个称为关键字的域，且针对该关键字，二元树按层次形成了小大根交替的形式，即对于小大根交替堆中的任何一个结点 x，如果 x 位于小根层次，那么 x 就是以 x 为根节点的二元树中键值最小的结点，并称该结点为一个小根结点。同样的道理，如果 x 位于大根层次，那么 x 就是以 x 为根节点的二元树中键值最大的结点，并称该结点为一个大根结点。在小大根交替堆中根结点位于小根层次。

下图给出了一个具有 12 个顶点的小大根交替堆实例。每个结点中的数值就是该结点对应元素的键值。

假设要在该图的堆中插入键值为 5 的元素。和普通堆一样，小大根交替堆上的插入算法也要针对从新插入结点到根结点的路径进行处理。对于上面的例子，需要比较键值 5 和其父结点键值（此处是 10）的大小关系，此处由于 10 位于小根层次且 5<10，就决定了 5 一定比从新加结点到根结点路径上的所有大根结点都要小，也即 5 无论放在那里都不影响大根层次的大根性质，所以插入 5 时只需调整从新加结点到根结点路径上的小根结点就能使得整个堆满足小大根交替性质。具体的调整过程为：首先，由于 5<10，

所以 10 下移填充到新加结点中；接着，5 和 7 比较，由于 5<7，所以 7 下移填充到原来结点 10 的位置；最后将结点 5 填入根结点。

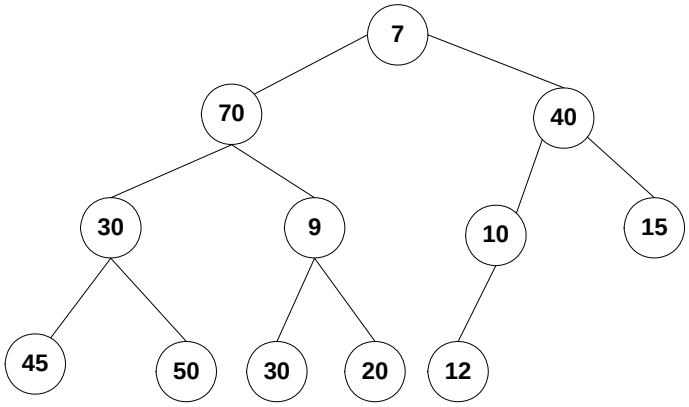


图 6-33 一小大根交替堆实例

下面我们将考虑在上图中插入键值为 80 的元素的情况。此时 10 仍处于小根层次，但由于 80>10，就有 80 大于从新加结点到根结点路径上的所有小根结点，也就决定了插入 80 时只需调整从新加结点到根结点路径上的大根结点就能使得整个堆满足小大根交替性质。上图中只有一个满足条件的结点——键值为 40 的结点。由于 80>40，所以 40 将下移填入新加结点中，而 80 将填入原 40 结点所占据的位置。

如果我们要删除该堆中键值最小的元素，那么删除的显然就是根结点，对于上图所示的小大根交替堆，就是删除结点 7。删除了结点 7 以后，小大根交替堆中就剩下了 11 个元素，并拟用最后一个元素（键值为 12）重填根结点。和普通的小根（大根）堆类似，重填的动作将引起从根到叶的一遍检查。

考虑在堆中实现元素 item 对根结点的重填，需分析如下两种情况：

(1) 如果根没有儿子结点。

此时 item 直接填入根结点即可。

(2) 如果根至少存在一个儿子结点。

此时，小大根交替堆中键值最小的结点一定出现在根结点的儿子结点或孙子结点中，并且可以很容易的找到该结点，将其标记为 k。然后再分成如下几种情况进行考虑：

a) item.key <= heap[k].key。

此时 item 直接插入根结点处，因为 item 就是堆中键值最小的结点。

b) item.key > heap[k].key 且 k 是根结点的儿子。

由于 k 是大根结点，所以其后代结点中不可能有键值大于 heap[k].key 的结点，也就没有键值大于 item.key 的结点。此时将 heap[k]的元素移到根结点，然后将 item 插入 k 结点所在位置即可。

c) item.key > heap[k].key 且 k 是根结点孙子。

对于此种情况也是先将 heap[k]的元素移到根结点，将 item 插入 k 结点所在位置。然后考察 k 的父结点 parent，如果 item.key>heap[parent].key，就将 heap[parent]和 item 元素互换，这一交换保证了 parent 结点处的键值是以 parent 为根结点的子堆中最大的，即 parent 满足了大根条件。此时需考虑如何将 item 插入到以 k 为根的子堆中，现在该子小大根交替堆的根结点为空，显然此时的情况和初始情况完全相似，因此可以重复上述处理过程。

对于本例有 item.key=12，且根结点的所有儿子和孙子中的最小键值为 9，设 9

所对应的结点用 k 标记，其父结点用 p 标记。由于有 9<12 且 k 是根的孙子结点，属于情形 2（c）。所以键值 9 对应的元素（即 h[k]）将移到根结点处，又由于条件 item.key=12<70=h[p].key，所以不需要交换 x 和 h[p]。现在需将 item 插入到以 k 为根的子小大根交替堆中，由于 k 的所有儿子和孙子结点中最小的键值为 20，将上条件 12<20，属于情形 2（a），将 item 插入到 h[k]中即可，至此完成了插入过程。

(2) 课程设计目的

掌握小大根交替堆，并用它实现双端优先队列的构造。

(3) 基本要求

① 给出双端优先队列的 ADT 描述，包括优先队列的逻辑结构及其上基本操作。

② 给出小大根交替堆的 ADT 描述，并实现该 ADT。

③ 以小大根交替堆结构为辅助结构实现双端优先队列的存储表示并实现其上的基本操作。

④ 应用双端优先队列的 ADT 实现依据学生成绩实现对学生信息的查询。

⑤ 学生信息存放在文本文件中（格式自定，内容自行输入）。

(4) 实现提示

出队、入队需考虑优先性。

17. 模拟文件目录系统

(1) 问题描述

本设计需完成两部分工作：一个是定义并实现一称为 CatalogTree 的 ADT，用它来表达字符串集合组成的有序树；另一个是一个 Shell 的应用程序，用它来模拟文件目录系统，并提供模拟操作界面。

CatalogTree 的组织结构如下图(带父结点指针的儿子—兄弟链树)：

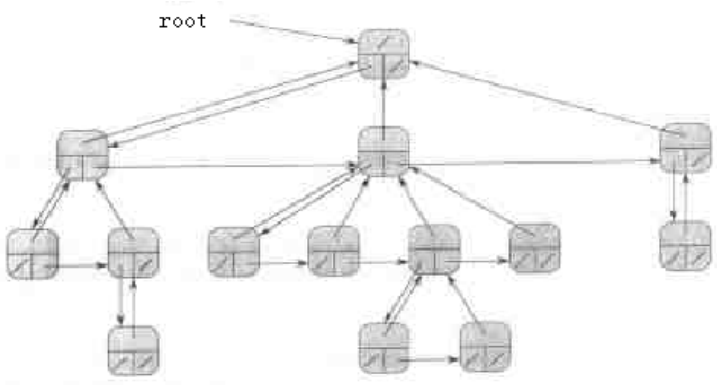


图 6-34 CatalogTree 的结构示意图

针对于目录系统，CatalogTree 的结点存放的数据内容为字符串，每个结点对应一个目录项，该目录项可以是目录，也可以是文件，如果是目录就可以再存放其它目录或文件，即非叶结点；如果是文件就是叶结点。从根结点到该结点路经所有结点的字符串用“/”进行组合后就是该目录项的绝对路径，用来唯一的标识该目录。例如：/usr/li/email/student/。

目录系统具有如下基本操作：

1) dir ——列出当前目录下的所有目录项

2) cd ——打出当前目录的绝对路经

3) cd ..——当前目录变为当前目录的父目录

4) cd str——当前目录变为 str 所表示路径的目录

5) mkdir str ——在(当前目录下)创建一个子目录(名为 str)

6) mkfile str ——在(当前目录下)创建一个文件(名为 str)

7) delete str ——删除(当前目录下)名为 str 的目录或文件

(2) 课程设计目的

应用树知识模拟一个目录管理系统。

(3) 基本要求

① 描述并实现 CatalogTree 的 ADT，包括其上的基本操作：如插入一个结点，寻找一个结点，返回一个结点的最左儿子等（具体情况依据应用自定）。

② 应用 CatalogTree 的 ADT 实现一个完成文件目录系统的 Shell 应用程序。

③ 该 Shell 是一个不断等待用户输入命令的解释程序，根据用户输入的命令完成相关操作，直到退出（quit）。命令名及其含义如上所述。

④ 目录树结构可以保存（save）到文件中，也可从文件中读出（load *.dat）。

⑤ dir 命令的结果应能够区分是子目录和还是文件。

⑥ 应对命令 4)~7)中的 str 区分是绝对路经，还是相对路径。

(4) 实现提示

关键是树上基本操作的实现。

18. 一个基于 XML 的网站生成器

(1) 问题描述

分析一个 XML 文件，并根据分析结果自动生成一个完整的新闻网站。

(2) 课程设计目的

理解栈结构，并能栈结构解决复杂的实际问题。

(3) 基本要求

① XML 文件形如：

```
<newslist>
  <article>
    <header>
      <title>Naked Students Seen on Rice Campus</title>
      <author>Donna B. Cerius</author>
      <section>Metro</section>
      <date>23 Sep 1999</date>
    </header>
    <body>
      ……
    </body>
  </article>
  <article>
    ……
  </artile>
    ……
</newslist>
```

② 结果网站中每个 article 对应一个网页。

③ 结果网站中的主页中有通向各个 article 页面的链接。

④ 主页中的链接需依据 XML 中 article 的 section 进行分类。

⑤ 网站页面的样式自行设计。

(4) 实现提示

需参阅关于 XML 和 HTML 的基本介绍。

19. 简单矢量图形的几何变换

(1) 问题描述

常见的几何图形（二维图形）都是由点、直线和圆组成，而平面上的点和直线都可用矢量进行表示，如果假设几何图形中不包含圆，那么一个无论多么复杂的几何图形都可用矢量表示并存储，且图形上的几何变换都可通过矢量上的变换得以实现。

例如下图：

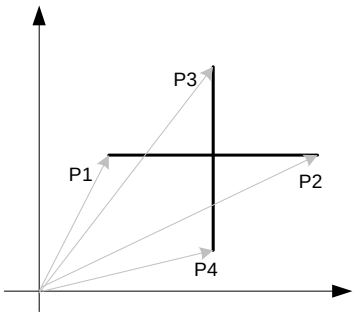


图 6-35 一矢量图形的实例

图中的十字形状就可用矢量 P1, P2-P1 和 P3, P3-P4 表示，当然其中 P1, P2, P3, P4 都是向量，分别表示十字图形的四个顶点。其中 P1 表示直线段的起点，P2-P1 表示该直线的方向和长度。

而该图的平移变换就可表示如下(向右水平平移 5 个单位)：

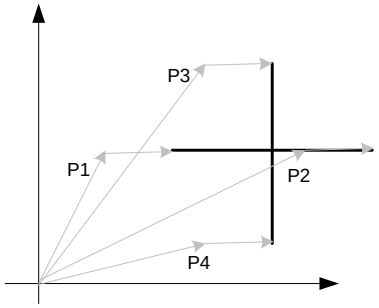


图 6-36 矢量图形向右平移 5 个单位的结果

[P1+H, P2-P1]; [P3+H, P3-P4]其中 H 为矢量 5+0i。

(2) 课程设计目的

能够应用线性数据结构描述简单的矢量图，并能完成简单的几何变换。

(3) 基本要求

① 描述并实现矢量 ADT，包括矢量的表示和存储，以及其上的基本操作：矢量相加；相减；共扼；求模；相乘等。

② 应用矢量实现简单几何图形（由点和直线组成）的表示和存储。同时该几何图形能够保存在文件中（格式自定），并能从文件中读出。

③ 实现矢量几何图形的基本操作，包括平移（水平加垂直）；旋转；依直线镜像；依点镜像；放大；缩小等（可自行扩展）。

④ 提供界面实现用户对图形的操作，包括输入图形，进行变换操作等。

⑤ 每一步操作的结果都能图形化显示。

(4) 实现提示

用二元组表示一个向量，一个几何图形就是一个数组。

20. 扫雷游戏

(1) 问题描述

本题目做一个 N x M 的扫雷游戏，每个方格包含两种状态：关闭（closed）和打开（opened），初始化时每个方格都是关闭的，一个打开的方格也会包含两种状态：一个数字（clue）和一个雷（bomb）。你可以打开（open）一个方格，如果你打开的是一个 bomb，那么就失败；否则就会打开一个数字，该数字是位于[0，8]的一个整数，该数字表示其所有邻居方格（*neighboring squares*）所包含的雷数，应用该信息可以帮助你扫雷。

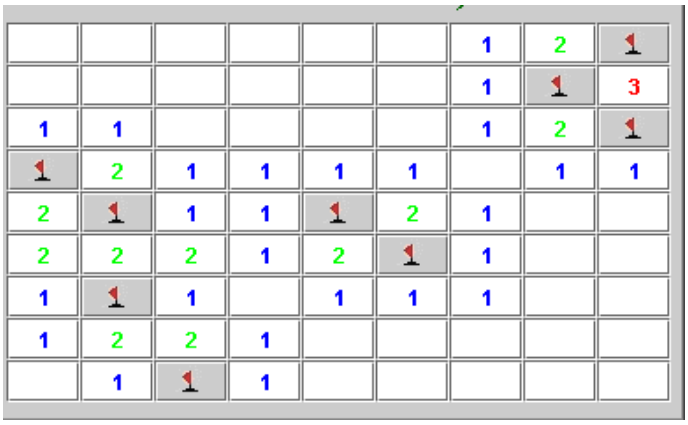


图 6-37 扫雷游戏的图例

具体实现要求的细节：

① 能够打开一个方格，一个已打开的方格不能再关闭。

② 能够标记一个方格，标记方格的含义是对该方格有雷的预测（并不表示真的一定有雷），当一个方格标记后该方格不能被打开，只能执行取消标记的操作，只能在取消后才能打开一个方格。

③ 合理分配各个操作的按键，以及各方格各种状态如何合理显示。

(2) 课程设计目的

掌握线性结构的应用，并体会如何用计算机程序完成一个有趣的游戏。

(3) 基本要求

① 能够给出游戏结果（输、赢、剩余的雷数、用掉的时间按妙计）。

② 游戏界面最好图形化，否则一定要清楚的字符界面。

(4) 实现提示

用二维数组表示 N x M 区间，要仔细考虑如何初始的布置各个雷以及各个方格的状态及变化。

21．实现简单数字图像处理

(1) 问题描述

一幅图像就是一个从位置集到颜色集的变换。考虑二维图像，位置集实际上就是一个矩阵，此时一幅图像实际上就是一个内容为颜色矩阵。如果颜色为 0-255 间的整数，表示该位置的灰度等级，0 为黑色，255 为白色，此时的图像称为灰度图。

而图像的处理就是在该矩阵进行相关计算。一种常见的计算就是通过一点和周围 8 个点的信息共同决定该点的新值：如一点的新值为该点和周围 8 点颜色值之和的均值，这一操作可用下图表示。

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

图 6-38 图像平滑模板

显然这样处理后，图像会变得平滑，因此称为平滑操作。显然将上述操作变为下图时，就成为锐化操作。

-1	-1	-1
-1	9	-1
-1	-1	-1

图 6-39 图像锐化模板

要求实现若干基本的图像处理操作。

(2) 课程设计目的

掌握矩阵上的操作，体会数字图像的处理。

(3) 基本要求

① 熟悉 Windows 下 BMP 文件的格式，能够实现其读写(只考虑灰度图像)。

② 实现图像的平滑和锐化操作，其它处理操作选做。

③ 需用 VC++作为语言。

(4) 实现提示

参阅相关资料。

22. 媒体流调度问题

(1) 问题描述

设一个多媒体播放系统用来播放一个 video segment 集合 V，该集合中的每一个元素 segment S=(B; L; F)，其中 B 为开始时间，L 为媒体播放时间，即媒体长度，F 为该媒体播放被延迟所造成的损失函数（此处可以假定 F 为关于延迟播放时间 D 的线性函数，如就定义 F(S) = S.D，也可以自己设定）。

媒体播放时要求每个 S 的播放不能被中断（不可抢占），只能一个接一个的播放。本课程设计的基本目的是设计给出尽量少损失的调度算法。对于 V={S_1，S_2，…，Sn}，V 上的一个调度就是 V 集合中各元素的一个置换，即重新排列。

本课程设计题目要求给出三个以上的调度方法：此处给出三种，可以也鼓励自己想出新的、更巧妙的调度方法：

① 最早开始时间最早调度

将 V 中的各 S 按其 B 属性从小到大排序，排序结果就是调度结果。

② 最短媒体最先调度

将 V 中的各 S 按其 L 属性从小到大排序，排序结果就是调度结果。

③ 贪心调度

保证每次调度都是此次调度引起最少损失的 Si。

(2) 课程设计目的

应用数据结构解决实际问题，认识贪心法，并用贪心法解决实际问题。

(3) 基本要求

① 随机的产生初始化 V 集合，V 中的各个 S 也是随机产生的，即其中的 B 和 L 随机设定。

② 实现三种以上的调度算法实现调度，并计算比较各算法造成的损失。

③ 完成贪心调度策略设计并实现。

④ 思考能完成最优调度的算法。

(4) 实现提示

阅读贪心法的相关资料。

23. DES 的实现

(1) 问题描述

数据加密标准(DES)作为一个加密算法得到了广泛的应用，也成为了业界的一个标准。它采用多次循环的使用置换、排列等基本加密操作后形成的安全性较高的加密算法，本设计要求依据下面对 DES 的简要描述实现一个 DES 加密程序。

DES 算法框图如下图所示：

图 6-40 DES 算法框图

IP 和 IP^{-1} 是两个基于位的置换算法，且 IP^{-1} 是 IP 的逆置换，其 IP 如下图所示：

```
58, 50, 42, 34, 26, 18, 10, 2,
60, 52, 44, 36, 28, 20, 12, 4,
62, 54, 46, 38, 30, 22, 14, 6,
64, 56, 48, 40, 32, 24, 16, 8,
57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3,
61, 53, 45, 37, 29, 21, 13, 5,
63, 55, 47, 39, 31, 23, 15, 7
```

图 6-41 IP 图

乘积变换是一个连续进行 16 轮的变换，每一轮变换如下图所示：

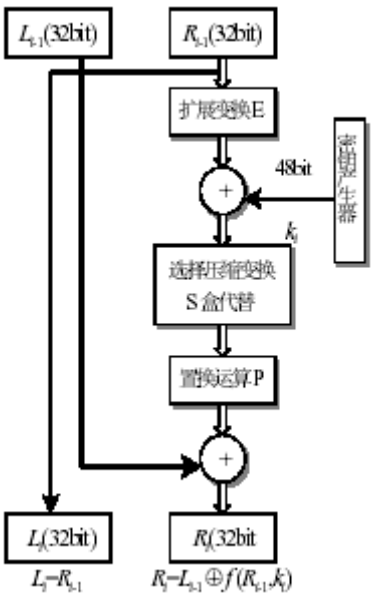


图 6-42 DES 乘积变换图示

该图表示第 i 轮变换，其输入为第 i-1 轮变换的结果(第 0 轮就是明文)，并将该结

果平分成两组 L_{i-1} 和 R_{i-1}(每组 32 位)。E 是一个扩展变换，它将 32 位的 R_{i-1} 扩展变换为 48 位，然后与 48 位的密钥进行按位异或，异或结果进入 S 盒，S 盒操作后的结果在经过 P 置换后与 L_{i-1} 按位异或成为该轮变换的最终结果的 R_i，而 R_{i-1} 直接就成为该轮变换的最终结果的 L_i。

E 将 32 位的 R_{i-1} 扩展为 48 位，其具体变换如下图所示：

```
32 | 01 02 03 04 | 05
04 | 05 06 07 08 | 09
08 | 09 10 11 12 | 13
12 | 13 14 15 16 | 17
16 | 17 18 19 20 | 21
20 | 21 22 23 24 | 25
24 | 25 26 27 28 | 29
28 | 29 30 31 32 | 01
```

图 6-43 扩展变换 E 图示

S 盒会将 48 位的输入重新变为 32 为，所以 P 是一个从 32 位到 32 位的置换，其具体置换规则如下图所示：

```
16 07 20 21 29 12 28 17
01 15 23 26 05 18 31 10
02 08 24 14 32 27 03 09
19 13 30 06 22 11 04 25
```

图 6-44 置换规则 P

DES 算法的核心部分就是 S 盒，其结构如下图所示：

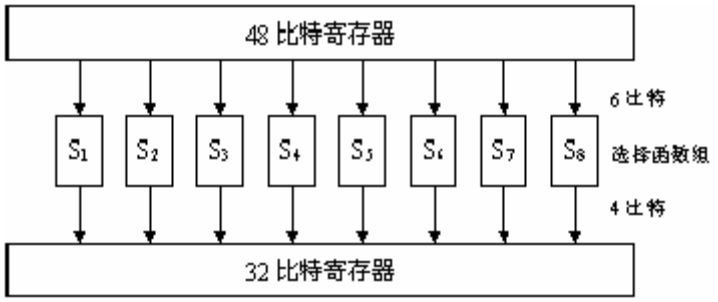


图 6-45 S 盒置换图示

48 位的输入分为 8 组，每组 6 位输入到一个小 S 盒(S(i))中，每个 S(i)将 8 位输入压缩变换为 4 位，8 个 S(i)的输出形成 32 位的 S 盒输出。

每个 S(i)的变换方法为(此处考虑 S(1))：设 S(1)的输入为 $b_1b_2b_3b_4b_5b_6$，用 b_1b_6 对应的 10 进制数作为行号，用 $b_2b_3b_4b_5$ 对应的 10 进制数作为列号查找下表，得到的数对应的二进制就是 S(1)的输出结果。

S_1

行＼列	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

图 6-46 S 盒查表方法

上图只是 S(1)的置换表，S(2~8)可查阅资料获得，如果没有查到，S(2~8)采用和 S(1)一样的置换表就可以。

还有另一方面的问题就是密钥的生成，实际 DES 的加密密钥为 56 位，各处用到的密钥由此密钥通过算法产生。此处我们为了简化，认为各处用到的密钥（都是 48 位）都一样，且是上述 56 位加密密钥的后 48 位（当然你也可以查找并应用密钥生成实际算法）。

上面描述了 DES 算法只是针对 64 位明文的，实际的名文将会很长，就要涉及明文的分割，分割后如何加密，结果如何组合等许多问题。假定方法为 CBC：

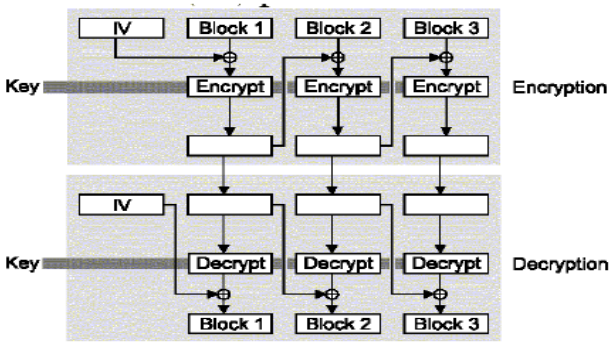


图 6-47 CBC 图示

IV（初始向量）是一个 64 位的伪随机序列。

(2) 课程设计目的

用数据结构构造位操作，并此为基础学习及实现 DES 算法。

(3) 基本要求

① 定义并实现位 ADT，包括其上的基本操作：与、或、非、异或等。

③ 实现 DES 算法。

④ 应用 DES 算法对一文本文件（全是 ASCII 码）加密。

④ 用 ASCII 方法对加密结果进行显示。

(4) 实现提示

将明文分割为位，DES 中用到的大部分是位上的操作。

24. 设计一个动画语言的解释执行器

(1) 问题描述

针对一个简单的基于对象的动画语言，设计一个程序解释并执行用该语言编写的动画程序，最后将该动画演示出来。

该动画语言的语法如下：

<command> = <object defn> | <move-cmd>

<object defn> = (define <var> <draw-list>)

<draw-list> = (<draw-cmd> … <draw-cmd>)

<draw-cmd>= <line-cmd> | <circle-cmd>

<line-cmd> = (line <posn> <posn>)

<circle-cmd> = (circle <posn> *r*)

<posn> = (*x y*)

<move-cmd> = <place-cmd> | <shift-cmd> | <erase-cmd> | <loop-cmd>

<place-cmd> = (place <posn> <var>)

<shift-cmd> = (shift <var> <direction>)

<direction> = left | right | up | down
<erase-cmd>= (erase <var>)
<loop-cmd> = (loop *n* (<move-cmd> … <move-cmd>))

图 6-48 动画语言的文法

用该语言可编写下面的动画程序：

```
(define cross ( (line (0 0) (50 50)) (line (50 0) (0 50)) (circle (25 25) 25) ) )
(place cross (50 50))
(loop 100 ( (shift cross right) (shift cross up) ) )
(erase cross)
```

该动画程序实现了一个内部带叉的圆圈从(50,50)向屏幕右上角(实际上是右下角)移动后消失的动画过程。

(2) 课程设计目的

灵活的应用线性和嵌套结构解决问题。

(3) 基本要求

① 设计并实现该动画语言的解释执行器。

② 写一个人走过街道的动画程序，并用解释器进行解释执行。

③ 将结果演示出来。(用 C 图形库即可)

(4) 实现提示

关键在于将动画程序中的对象提取出来，然后将每一行程序中的动作加在对象上即可。

25. 指令调度算法

(1) 问题描述

许多新的计算机硬件允许多条指令并行执行，前提是如果这些指令互不依赖。指令间的依赖关系可以用指令顺序表示：指令 i 必须在指令 j 之前执行，当且仅当 i 等号左边的变量出现在 j 的等号右边，这一顺序记为 i<j。例如：

```
x=mem[90]+w      (指令 i)
y=x+mem[1]       (指令 j)
```

有 i<j。

编写程序将一段程序中指令间的依赖（顺序）关系用图表示出来，并根据此图编写一个指令调度程序来实现指令的调度执行，要求最大可能的利用程序并行性。

(2) 课程设计目的

学会应用有向图来表示特定结构，并能应用图上的算法解决实际问题。

(3) 基本要求

① 算法的输入为一段程序，该段程序存在文件 ProgIn.txt 中。

输入程序的语法如下图所示：

```
<program> ::= <input spec> <instruction list> <output spec>
<input spec> ::= in <list of variables>
<instruction list> ::= <instruction> | <instruction> <instruction list>
<instruction> ::= variable = <rhs>
<rhs> ::= variable + <rhs> | mem [ integer ] + <rhs> | variable | mem [ integer ]
<output spec> ::= out <list of variables>
<list of variables> ::= variable | variable , <list of variables>
```

图 6-49 输入程序的文法

结果测试时请自己根据上图的语法构造一段示例程序。

② 结果包括两部分：依赖关系图（考虑如何显示）和指令调度结果（用结果序列表示，可并行执行的指令作为序列中的一个单元）。

(4) 实现提示

拓扑排序的变形。

26. 模拟 Sensor network 的工作

(1) 问题描述

Sensor network 是一种新型的网络，其基本结构如下图所示：该网络由两部分组成，Sensor node 集和 Data Collector。Sensor node（可简称为 Sensor）能够完成感知环境数据并将其发往 Data Collector 的功能。Data Collector 完成 Sensor 采集数据的收集，它就是一台带有无线接收功能的计算机。

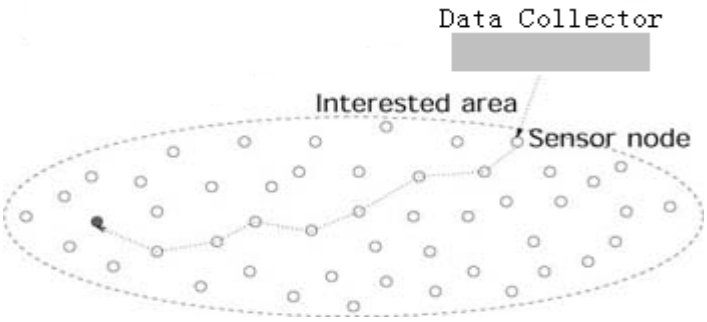


图 6-50 Sensor network 图示

Sensor network 可应用到很多实际领域中，如在战争中将 Sensor 散播在防线的前沿，可以收集敌人的一些情报（如大规模的部队转移等）。Sensor 散播的地方称为 Interested area，Sensor 在这个区域内采集各自所在位置的数据，然后将采集到的数据传送到 Data Collector。各个 Sensor 之间通过无线广播通讯，由于 Sensor 广播能力的限制，它只能和位于自身的一定广播半径内的 Sensor 进行通讯，所以有些 Sensor 就需通过其它 Sensor，经过多次路由后才能到达 Data Collector（如上图）。如何路由由 Sensor 内保存的简单路由表来决定。

Data Collector 的位置就在 Intersted Area 的边缘，且固定不动。

(2) 课程设计目的

应用数据结构知识模拟一个新型网络系统。

(3) 基本要求

① 应用数据结构在单机上模拟 Sensor network 的工作。

② 用 VC++（C 也可以）实现模拟系统，N 个 Sensor 和 1 个 Data Collector，其具体位置随机确定，Interest Area 就是屏幕。N 可配置，缺省为 100。

③ Sensor 进行周期性采集，其采集周期可配置。

④ Sensor 的广播半径固定，也是可配置的参数。

⑤ 路由算法自行选择或设计。

(4) 实现提示

Sensor 集可组织成数组，它采集及收到的数据包用队列存储。具体细节也可参阅有关资料。

27. 模拟贝叶斯网

(1) 问题描述

贝叶斯网是基于贝叶斯条件概率理论建立起来的图论模型，其形式化定义如下：

贝叶斯网(Bayesian Network)是一个有向无环图(DAG)，其中的顶点代表随机变量，

当 A、B 两个变量存在概率依赖关系(非相互独立)时，存在一条有向边连接 AB。同时每个顶点带有一张表用来记录该结点针对其父顶点的条件概率分布情况，B 的父顶点就是存在有向边 AB 时的 A 顶点；如果某顶点没有父顶点，其上记录的就是它的先验概率分布情况。

下图就是一个贝叶斯网的图例：

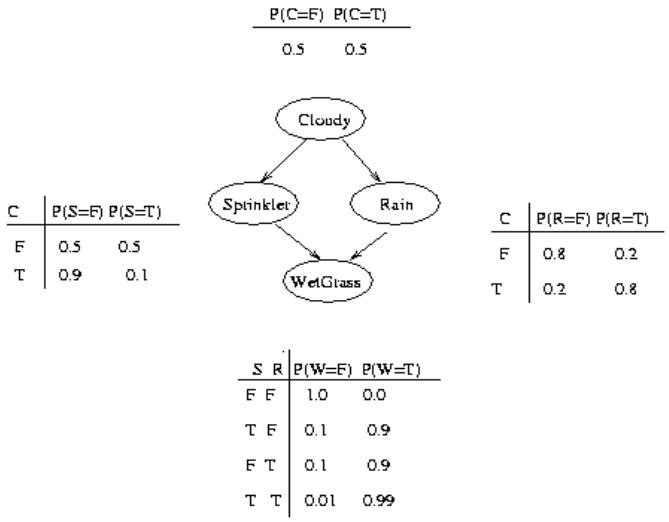


图 6-51 贝叶斯网的图示

该图中的每个随机变量都存在 T(真)和 F(假)两种情况。

计算机领域中的很多问题可以应用贝叶斯网进行解决，用贝叶斯网求解一个问题的基本步骤是：首先是根据问题研究的论域确定描述该问题所需的随机变量集并确定各变量的依赖关系；然后从空贝叶斯网开始按照顶点依赖顺序逐步添加变量来建立贝叶斯网；接下来将实际问题对应到贝叶斯网中，转化为图论问题及概率问题的求解；最后将贝叶斯网上的解映射回原有问题上。

贝叶斯网最常见的应用是用它来实现概率推理（probabilistic inference）。概率推理的应用范围也很广，比如在安全领域中的入侵检测系统，其基本目标就是要计算根据收集到的信息可在多大概率程度上推断出系统存在着入侵行为。我们就依据上面的例子来说明贝叶斯网是如何实现概率推理的。假设我们观测到草是湿的这一事实，此时

$$\Pr(R=1|W=1)=\frac{\Pr(R=1,W=1)}{\Pr(W=1)}=\frac{\sum_{c,s}\Pr(C=c,S=s,R=1,W=1)}{\Pr(W=1)}=0.4581/0.6471=0.708$$

$$\Pr(S=1|W=1)=\frac{\Pr(S=1,W=1)}{\Pr(W=1)}=\frac{\sum_{c,r}\Pr(C=c,S=1,R=r,W=1)}{\Pr(W=1)}=0.2781/0.6471=0.430$$

$$\Pr(W=1)=\sum_{c,r,s}\Pr(C=c,S=s,R=r,W=1)=0.6471$$

有上述计算结果可以看出，此时更有可能是下雨导致了草湿。

(2) 课程设计目的

学习贝叶斯网，应用图的知识实现贝叶斯网，并用该实现解决一个具体问题。

(3) 基本要求

① 设计贝叶斯网结构的 ADT 描述，该 ADT 包含的基本操作有：增加一个顶点(实际上就是一个随机变量)；删除一个顶点（只删出度为 0 的顶点）；列出两个点之间的所有路径；计算若干个变量取得某种状态时的概率值等。此处描述的是

基本操作，可根据需要对其进行扩展。

② 对于增加顶点的操作：在增加变量时，应同时输入该变量对已有变量的依赖关系，包括条件概率值；并假设目前网中已存顶点没有依赖新加顶点的。

③ 计算若干个变量取得某种状态时的概率值的操作形如 P(A=1,B=0,C=1)。

④ 应用图论知识实现上述 ADT。

⑤ 应用贝叶斯网解决下面的问题：在某处行窃(B)和地震(E)都会引起报警(A)，听到报警后，两邻居 Mary(M)和 John(J)都可能呼叫。各种情况的出现概率如下图。

⑥ 根据上面的描述编写算法建立该问题的贝叶斯网。

⑦ 用贝叶斯网概率推理方法回答用户提出的问题：主要包括

1. 如果 J=y；M=y，更有可能是 B 还是 E。
2. 如果 J=y；M=n，更有可能是 B 还是 E。
3. 如果 J=n；M=y，更有可能是 B 还是 E。

$P(B, E, A, J, M)$

B	E	A	J	M	Prob	B	E	A	J	M	Prob
y	y	y	y	y	.00001	n	y	y	y	y	.0002
y	y	y	y	n	.000025	n	y	y	y	n	.0004
y	y	y	n	y	.000025	n	y	y	n	y	.0004
y	y	y	n	n	.00000	n	y	y	n	n	.0002
y	y	n	y	y	.00001	n	y	n	y	y	.0002
y	y	n	y	n	.000015	n	y	n	y	n	.0002
y	y	n	n	y	.000015	n	y	n	n	y	.0002
y	y	n	n	n	.0000	n	y	n	n	n	.0002
y	n	y	y	y	.00001	n	n	y	y	y	.0001
y	n	y	y	n	.000025	n	n	y	y	n	.0002
y	n	y	n	y	.000025	n	n	y	n	y	.0002
y	n	y	n	n	.0000	n	n	y	n	n	.0001
y	n	n	y	y	.00001	n	n	n	y	y	.0001
y	n	n	y	n	.00001	n	n	n	y	n	.0001
y	n	n	n	y	.00001	n	n	n	n	y	.0001
y	n	n	n	n	.00000	n	n	n	n	n	.996

图 6-52 推理数据图

(4) 实现提示

主要是图论上的操作，条件概率知识可参阅有关书籍。

28. 小型文本编辑器的设计

(1) 问题描述

设计一个文本编辑器，使其具有通常编辑器(如 Notepad)所应具备的基本功能。

(2) 课程设计目的

较大规模软件系统的设计与实现，字符串上的基本操作。

(3) 基本要求

① 要求该编辑器在串基本抽象数据型上构建。

② 编辑器应具备如字符串查找，字符串剪切，字符串粘贴，字符串替换，统计字数，统计行数等基本功能。

③ 具备图形化界面(最好用 VC++完成)。

(4) 实现提示

字符串抽象数据型需要为文本编辑器的实现提供足够的支持，所以在此部分需仔细分析、设计；可以行编辑器作为该编辑器的基础。

29. Slide 游戏

(1) 问题描述

编程实现 Slide 游戏，Slide 游戏的棋盘被划分为网格（grid），棋子是一些互不重叠的长方形块（pieces），棋子不能离开棋盘，只能在棋盘表面上水平或垂直移动，不能移

动到其他棋子上面。一系列棋子的移动的目的是从初始状态(格局)变换到最终状态(格局)，如果不存在这样的移动序列，则该游戏以失败告终。如下图所示：

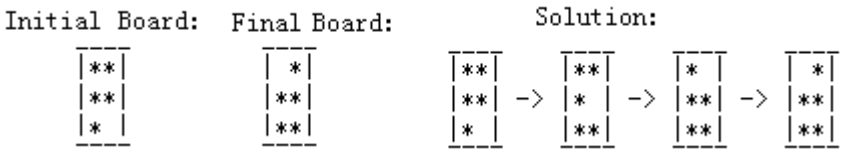


图 6-53 一个简单的 Slide 游戏实例

其中每个 '*' 表示 1x1 的棋子，且' ' 表示一个空位。

可以将上述游戏更加复杂化，引入 1 x 2 或 2 x 2 的棋子，此时的 Slide 游戏如下图所示：

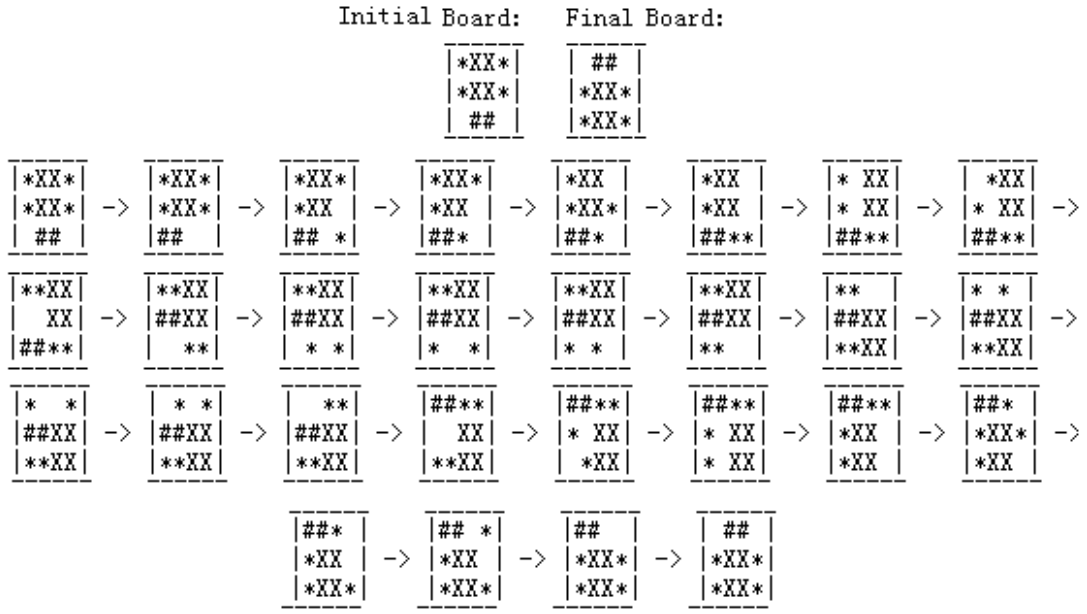


图 6-54 一个较复杂的 Slide 游戏实例

其中'*'是 1x1 的棋子，#是 1x2 的棋子，X 代表 2x2 的棋子。

本课程设计题目的基本目的是找到从初始格局到最终格局的一个最短的棋子移动序列。为找到最短移动序列，可以构建游戏树，对每个格局将其每种可能的下一个格局用树的形式构造出。为避免重复查找已出现的格局，需要记录已出现过的格局，可采用散列方法记录已出现过的格局。

(2) 课程设计目的

如何高效的实现树构造，并能用其解决实际问题。

(3) 基本要求

① 棋盘大小自己随机设定，最少应大于 10 x 10。

② 棋子大小为 1x1 的问题是必做内容，且要求棋盘中只有一个空位，初始格局和目标格局可随机定义。

③ 应该有一个较为形象的输出。

④ 需分析和证明算法的正确性以及时间复杂性。

⑤ 包含其它棋子类型(如 1x2，2x2，等等)的情况是选作问题。

(4) 实现提示

可适当参考人工智能领域中的一些简单内容。

30．后缀树的构造

(1) 问题描述

后缀树是一种数据结构，一个具有 m 个字符的字符串 S 的后缀树 T，就是一个包含一个根节点的有向树，该树恰好带有 m+1 个叶子(包含空字符)，这些叶子被赋予从 0 到 m 的标号。 每一个内部节点，除了根节点以外，都至少有两个子节点，而且每条边都用 S 的一个子串来标识。出自同一节点的任意两条边的标识不会以相同的字符开始。

后缀树的关键特征是：对于任何叶子 i，从根节点到该叶子所经历的边的所有标识串联起来后恰好拼出 S 的从 i 位置开始的后缀，即 S[i,…,m]。

后缀树的图示:

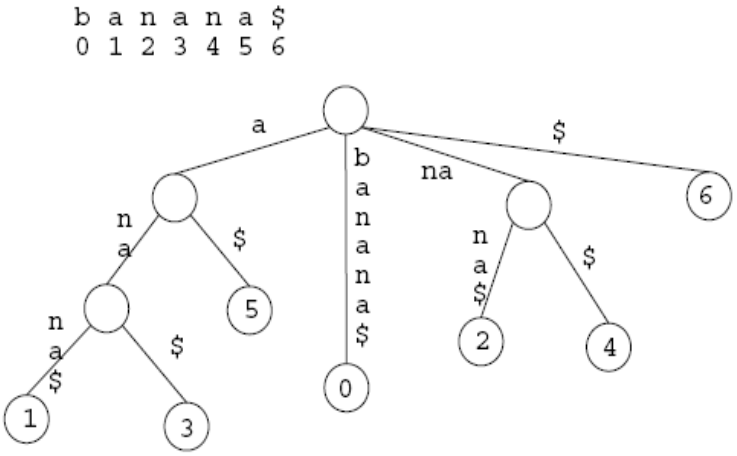


图 6-55 以后缀树图例

(2) 设计目的

认识后缀树的结构，掌握其构造方法，并能个根据其特点解决实际问题。

（3）基本要求

a) 对任意给定的字符串 S，建立其后缀树；

b) 查找一个字符串 S 是否包含子串 T；

c) 统计 S 中出现 T 的次数；

d) 找出 S 中最长的重复子串。所谓重复子串是指出现了两次以上的子串；

e) 分析以上各个算法的时间复杂性。

(4) 实现提示

无。

31．后缀数组的构造

(1) 问题描述

后缀数组是按字典顺序存放一个字符串所有后缀起始位置的一维数组。在后缀数组上可以进行折半查找（A suffix array is an array of all starting positions of suffix of a string in lexicographical order, which allows a binary search.）。对于任意给定的字符串，设计并实现下列算法，并分析各个算法的时间复杂性。

(2) 设计目的

认识后缀数组的结构，掌握其构造方法，并能根据其特点解决实际问题。

(3) 基本要求

a) 对任意给定的字符串 S，建立其后缀数组；

b) 查找一个字符串 S 是否包含子串 T；

c) 统计 S 中出现 T 的位置和次数；

d) 找出 S 中最长的重复子串。所谓重复子串是指出现了两次以上的子串。

（4）实现提示

利用基数分类对所有子串进行排序来构造后缀数组，需要搜索关于后缀数组的文献资料。

32. 模拟理发店并对其进行数据分析

(1) 问题描述

一个理发店是这样的一个系统：理发店内有 m 个理发椅，每个理发椅对应一个理发师，即有 m 个理发师，理发店内有 n 个等候座位。在理发店内添置一个理发椅(同时雇佣一个理发师)的代价为 C_1，在理发店内添置一个等候座位的代价为 C_2，由于物理空间和理发店的经济条件的限制，要求 $mC_1+nC_2 \le C_T$。假定理发店系统的工作过程如下：当一个顾客进入理发店时，如果有空闲的理发师(理发椅)时，顾客就坐在该理发椅上进行理发，顾客的理发时间 T_1 符合指数分布 E_1，顾客的理发费用，即理发店可以获得的收费为 cT_1，其中 c 为每个单位时间的收费；如果没有空闲的理发师(理发椅)时，顾客就看有没有空闲的等候座位，如果有就坐在一个等候座位上，顾客坐在等候座位上的等候时间 T_2 满足指数分布 E_2，如果在该段时间内出现了空闲的理发椅并轮到了自己，顾客就去理发，否则顾客就离去。如果没有等候座位，顾客就站在店里或站在门外等候(这就表示站着的顾客个数是不受限制的)，顾客站着等候的时间 T_3 满足指数分布 E_3，如果在该段时间内有了空闲的等候座位并轮到了自己，顾客就在座位上继续等待，坐在座位上再等的时间 T_4 和用户已经站着等待的时间 T_5 之和 T_4+T_5 满足指数分布 E_4，坐在座位上的顾客的工作方式同前，如果等待的时间用完，用户就离去。用户在理发店的营业时间内按泊松分布 B 到达理发店。

完成上述理发店系统的模拟，在完成该模拟时需要：仿照实际情况对上述四个指数分布 E_1 到 E_4 和泊松分布 B 的基本参数进行设定；仿照实际情况对 c，C_1，C_2 和 C_T 进行设定；设定模拟时间，设定 m 和 n 等参数。

(2) 课程设计目的

学会完成计算机对一些常见的复杂系统进行模拟，学会对一些常见的数学分布进行计算模拟，学会应用线性数据结构（尤其是队列）进行存储和组织信息。

(3) 基本要求

① 实现理发店系统的模拟。

② 根据模拟结果得出不同的 m 和 n 组合获得理发店收益。

③ 模拟系统应该有较为完整地，能有效说明模拟过程的界面（最好是用 VC++实现的类似图形化界面）。

④ 模拟系统的配置参数应该很容易修改(如采用配置文件或在图形界面中用控件输入)。

⑤ 如果可以的话，给出理发店系统更为合理的模拟，即考虑更多的模拟细节。

(4) 实现提示

主要集中在如何应用队列实现一个复杂系统的模拟及分析。

33. 布隆过滤器(Bloom Filter)的实现和应用

(1) 问题描述

布隆过滤器是由巴顿.布隆于一九七零年提出的。它实际上是一个很长的二进制向量和一

系列随机映射函数。假定需要存储一亿个电子邮件地址，首先建立一个十六亿二进制（比特），即两亿字节的向量，然后将这十六亿个二进制全部设置为零。对于每一个电子邮件地址 X，可以用八个不同的散列函数(H1, H2, ..., H8） 产生八个从 1 到十六亿之间中的八个自然数 g1, g2, ...,g8。然后将这八个自然数对应的八个位置的二进制全部设置为一。同样的方法对这一亿个 email 地址都进行处理后，一个针对这些 email 地址的布隆过滤器就建成了。如图 6-56 所示：

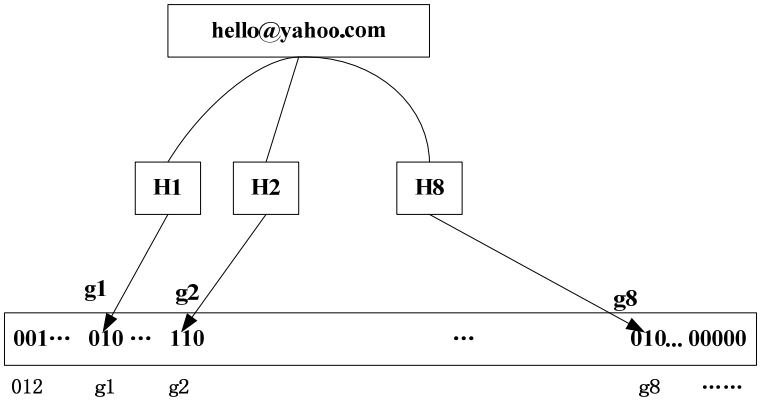


图 6-56 Bloom Filter 的数据结构

现在看看如何用布隆过滤器来实现一个电子邮件地址过滤器，首先将那些放在黑名单上的电子邮件地址放在布隆过滤器中。当检测一个可疑的电子邮件地址 Y 是否在黑名单中，仍用相同的八个随机数产生器（H1, H2, ..., H8）对这个邮件地址产生八个自然数 s1,s2,...,s8，这八个自然数对应的八个二进制位分别是 t1,t2,...,t8。如果 Y 在黑名单中，显然，t1,t2,..,t8 对应的八个二进制一定是一。这样在遇到任何在黑名单中的电子邮件地址，都能准确地发现。

布隆过滤器决不会漏掉任何一个在黑名单中的可疑地址。但是，它有一条不足之处。也就是它有极小的可能将一个不在黑名单中的电子邮件地址判定为在黑名单中，因为有可能某个好的邮件地址正巧对应个八个都被设置成一的二进制位。但这种可能性很小，此处将它称为误识概率。在上面的例子中，误识概率在万分之一以下。

因此布隆过滤器的好处在于快速，省空间。但是有一定的误识别率。常见的补救办法是在建立一个小的白名单，存储那些可能别误判的邮件地址。

(2) 课程设计目的

学习 Bloom Filter 结构，能应用该结构解决一些实际问题。

(3) 基本要求

① 定义 Bloom Filter 结构的 ADT，该 ADT 应支持在 Bloom Filter 中加入一个新的数据，查询数据是否在此过滤器中，并完成该结构的设计和实现。

② 应用 Bloom Filter 结构拼写检查，许多人都对 Word 的拼写检查功能非常了解，当用户拼错一个单词的时候，Word 会自动将这个单词用红线标注出来。Word 的具体工作原理不得而知，但另一个拼写检查器 UNIX spell-checkers 这个软件中就用到了 Bloom Filter。UNIX spell-checkers 将所有的字典单词存成 Bloom Filter 数据结构，而后直接在 Bloom Filter 上进行查询。本课程设计要求针对 C 语言设计和实现上述拼写检查器，即当写了一个正确的关键词，如 int 时，给该词标上颜色，如蓝色。

③ 针对上述 C 语言关键词拼写检查器进行分析，如错误分析，设计散列函数个数分析，运行时间复杂性、空间复杂性的分析。

④ 上述 C 语言关键词拼写检查器最好是在 VC++或 Java 等可视化开发环境下实现。

⑤ 上述 C 语言关键词拼写检查器最好能支持所有的 C++关键词。

(4) 实现提示

Bloom Filter 结构中的散列函数(包括散列函数的个数和散列函数的设计)是本题目中需要深入思考的一个环节。

34. 创建一个 CD 数据库

(1) 问题描述

本设计题目要求学生完成一个小型数据库的创建。

① 完成数据库的组织、存储及其上的操作。

该数据库用来存储歌曲，每首歌曲包含三个属性：歌曲作者、歌曲题目和歌曲创作的年份，即(Artist, Title, year)。数据库中的歌曲被存放在一个数据存储区中，且这些歌曲按照(Artist, Title, year)这三个属性组织成三个有序链表，目的是提高诸如 getByYearRange 等算法的效率。该数据库应该提供如下基本接口：

```
public interface SongDB {
    public void addSong(Song a);
    public void remSong(Song a);

    public Song[] getByArtist(String artist);
    public Song[] getByTitle(String title);
    public Song[] getByYear(int year);

    public Song[] getByArtistRange(String start, String finish);
    public Song[] getByTitleRange(String start, String finish);
    public Song[] getByYearRange(int start, int finish);

    public void printByArtist(PrintWriter out);
    public void printByTitle(PrintWriter out);
    public void printByYear(PrintWriter out);
}。
```

② 在数据库基础上实现 Playlist 接口。

在 CD 数据库基础上，一些应用软件，如 mp3 播放器自己有一个 playlist 结构，playlist 是一个按特定顺序播放的歌曲列表。PlayList 提供如下基本接口：

```
public interface Playlist {
    public void addSongToPlaylist(Song s);
    public void remSongFromPlaylist(Song s);

    public Song getNextSong();   //返回下一首要播放的歌曲

    public int length();
    public boolean isEmpty();

    public void shufflePlaylist();   //搅乱(洗牌)playlist 中的歌曲，自己设计一个搅乱算法
}
```

假定 CD 数据库只支持一个 playlist，playlist 被组织成一个指向歌曲对象“指针”

(references to song objects)的数组。

③ 将上部分的 playlist 扩展成一个 sortable playlist，该 sortable playlist 支持以下接口：

```
public interface SortablePlaylist extends Playlist {
    public void sortByArtist();
    public void sortByTitle();
    public void sortByYear();
}
```

sortable playlist 中采用的 sort 方法是自底向上的 mergesort 算法。

④ 在上述接口上编写一个小型应用程序，该程序能够测试(演示)上述接口操作，如 SortablePlaylist.sortByYear 是否完成了其预定的功能。同时该应用程序还应完成诸如存入磁盘、从磁盘载入等功能。

(2) 课程设计目的

熟练的实践和掌握基本数据结构(线性表)和其上的一些基本算法，对数据、数据库、数据库上的应用建立初步的直观认识。

(3) 基本要求

① CD 数据库能在内存中实现，同时也应该能够将该数据库存储在磁盘上。

② 同样的，playlist 也应能在内存中实现，并能够存储在磁盘上。

③ 最好是用可视界面说明和验证这些操作的执行效果，即实现上述问题描述中的要求④。

(4) 实现提示

需要注意 data object 和 reference to data object 之间的区别，playlist 中的数据是 reference to data object。

35. 应用不相交集合生成随机迷宫

(1) 问题描述

在本设计题目中，需要使用不相交集合数据结构(disjoint set data structure)来构造一个 N × N 的从左上角到右下角只有一条路径的随机迷宫，然后在这一迷宫上执行深度优先搜索。该设计共包含如下四个部分：

① 不相交集合数据结构的设计和实现

不相交集合即对于任意两个集合 A 和 B，A∩B=∅。不相交集合常可以表示为树，此时两个不相交集合的并的实现很容易，如图 6-57 所示。不相交集合常可用来根据等价关系对集合进行等价划分。

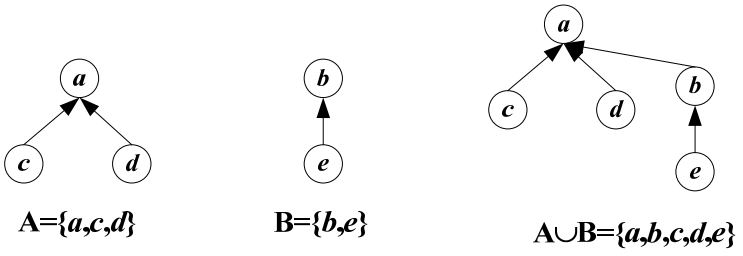
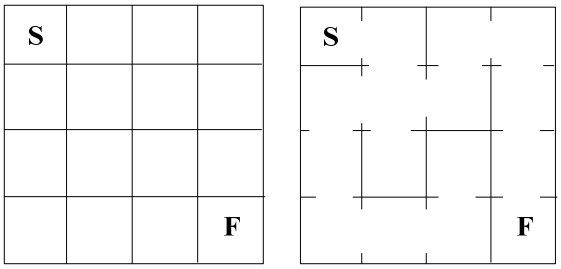


图 6-57 不相交集合的树表示

② 构建随机迷宫

应用不相交集合构建迷宫的算法简要描述如下：给定一个 N × N 的方格(cells)，初始时每个方格的四面都是墙(walls)，如图 6-58(*a*)所示，其中的 S 是迷宫的开始处，F 是迷宫的结束处。N × N 迷宫的 N^2 个方格 0，1，…，N^2-1 初始时每个方格自己成为一个等价类，即{0}，{1}，…，{N^2-1}。生成随机迷宫的方法是随机选择一个内部墙(连接两个相邻方格的墙)，如

果该内部墙关联的两个相邻的方格属于不同的等价类就将该墙除去，在除去该墙的同时将这两个等价类合并。直到所有的方格都在一个等价类中，就完成了随机迷宫的生成。



(*a*) (*b*)

图 6-58 N × N 的迷宫

③ 寻找迷宫路径

迷宫一旦建立后，将迷宫表示为一个无向图：方格作为图中的顶点，如果两个相邻的方格之间没有墙则两个顶点之间有边。为找到从 S 到 F 的一条唯一的路径，在该图上从 S 处开始出发先深搜索，如果搜索到达了 F，则搜索停止。

④ 将迷宫和路径用图形方式画出

用图形方式将上述算法获得的随机迷宫及其上的最短路径画出。用线段来表示迷宫中的墙，用在每个方格中心的点来表示路径。

(2) 课程设计目的

掌握不相交集合结构的实现和使用，掌握图的基本算法，建立集合、等价划分、图、搜索等事物之间如何关联的认识。

(3) 基本要求

① 不相交集合应用数组进行物理存储。

② 在生成随机迷宫时，考虑如何使选取墙的次数尽量少。

③ 在先深搜索找到路径时，要求找到最短的路径，即记录最短路径上的方格，所以先深搜索过程应该是采用栈的非递归实现。此时，在先深搜索结束时，栈中就存放了最短路径上的方格，而不是先深搜索过程访问的所有方格。

④ 迷宫和路径的图形显示的实现方式不限制，如可以选择 VC，TC 或 Java 等。

(4) 实现提示

不相交集合可父链数组进行物理存储，先深搜索采用栈的非递归实现可参阅一些资料，这样的资料很多。

36. 纠错码FEC(Forward Error Correction)

(1) 问题描述

信息在存储和传输过程中常因受到噪音干扰而造成某些随机位的错误。Forward error correction(FEC)在消息包中加入一些冗余数据来检测和纠正一些错误位。FEC 的工作方式，FEC 编码函数 E(x)=y，是一个将 k 位的数据块 x 映射成一个长度为 n 位的码字 y。噪声 noise 将 y 改变成一个错误的和 y 长度相同的 y′。FEC 译码函数 D 将 y′映射成一个 k 位的 x′=D(y′)。成功的 FEC 模式应该有 x= x′。

有许多不同的 FEC 设计方法，此处给出一种较为简单的处理方式。此处选取 k=256(32 字节)，n=288(36 字节)。y=E(x)中的前 32 个字节就是 x，后面的 4 个字节成为错误控制字节，其中的每一位都是数据块中某些位的异或。

在计算错误控制字节时，首先将 256 位数据块变成 16×16 的位矩阵 M，M 中的第一行是数据块中的前两个字节，第二行是紧接其后的两个字节，依次类推。然后，将每行的 16 位进行异或后获得 16 个 row sums，将每列的 16 位进行异或后获得 16 个 column sums。16

个 row sums 形成两个字节 r，16 个 column sums 形成两个字节 c，*E(x)=xrc*。

对于该 FEC 的译码函数 D，D 将从出现错误的 y′中恢复出 x，当然如果 y′中出现的错误太多，则 D 不能从 y′中译码出 x，本题中给出的 FEC 方式只能纠正 1 位错误；可以检测超过一位的错误，此时可以报告未知错误被检测到的信息；另外，某些时候多位错误使得 y′中的错误不能被检测，或者识别成一个出现一位错误的数据块，此时将会恢复出乱码。

该 FEC 的译码函数 D(y′)，其中 y′= x′r′c′，其中 x′是 32 个字节，r′和 c′是两个字节，令 r 和 c 为 x′的正确的 row sums 和 column sums，令 n_r 是 r′和 r 不同位的个数，n_c 是 c′和 c 不同位的个数，根据 n_r 和 n_c 分为三种情况进行处理：

情况 1：如果 $n_r+n_c\leq 1$，定义 D(y′)=x′。

情况 2：如果 $n_r=n_c=1$，令 i_r 为 r 不同位的位置，i_c 为 c 不同位的位置，D(y′)=x″，其中 x″和 x′只有一位不同，即在(i_r，i_c)位置处将 x′的位取反。

情况 3：对于所有其它情况，报告出现了不可纠正的错误。

(2) 课程设计目的

能够设计和实现位操作的数据结构，能应用这种结构解决实际问题，对纠错码建立一定的认识，对随机位错误的处理建立一定的认识。

(3) 基本要求

① 设计和实现一个支持位操作的数据结构 ADT。

② 在此 ADT 基础上实现上述 FEC 编码算法和译码算法。

③ 编写一个能给信息注入错误的程序 noise，该程序以 p 概率对信息中的各个位进行翻转。

④ 针对不同的 p，统计该 FEC 的检错率、纠错率等参数，并实验分析 FEC 的作用。对这些实验结果进行一定的理论分析(主要是概率分析)，来分析这些实验结果的正确性和可信性。

⑤ 实验测试用的信息可以是一段英文文章，其中各字母就用 ASCII 码表示，即每个字母就是一个字节，上述的 FEC 编码处理的 32 个字节就是 4 个字母。

(4) 实现提示

无，不过可以进一步查阅关于 FEC 的其它方法，并和此处给出的处理方法进行对比。

37. LZW 压缩的实践

(1) 问题描述

LZW 压缩算法是由 Lemple-Ziv-Welch 三人共同创造的一种新颖的压缩方法。该算法采用了一种先进的串表压缩法，其基本思想是将每个第一次出现的串放在一个串表中，然后用一个数字来表示串，压缩文件只存贮数字，则不存贮串，从而使图像文件的压缩效率得到较大的提高。有趣的是，不管在压缩还是在解压缩过程中都能正确的建立这个串表，压缩或解压缩完成后，这个串表都可被丢弃，即串表不用放在压缩后的文件中。在建立字符串表时，将把每一个第一次出现的字符串放入串表中，并用一个数字来表示（这个数字与此字符串在串表中的位置有关），并将这个数字存入压缩文件中，如果这个字符串再次出现时，即可用表示它的数字来代替，并将这个数字存入文件中。压缩完成后将串表丢弃。如"print" 字符串，如果在压缩时用 266 表示，只要再次出现，均用 266 表示，并将"print"字符串存入串表中，在解码时遇到数字 266，即可从串表中查出 266 所代表的字符串"print"，在解压缩时，串表可以根据压缩数据重新生成。

LZW 压缩有三个重要的对象：数据流(CharStream)、编码流(CodeStream)和编译表(String Table)。在编码时，数据流是输入对象（图象的光栅数据序列），编码流就是输出对象（经过压缩运算的编码数据）；在解码时，编码流则是输入对象，数据流是输出对象；而编译表是在编码和解码时都须要用借助的对象。

在 LZW 中还有几个重要的概念：字符(Character)：最基础的数据元素，在文本文件中就是一个字节，在光栅数据中就是一个像素的颜色在指定的颜色列表中的索引值；字符串(String)：由几个连续的字符组成；前缀(Prefix)：也是一个字符串，不过通常用在另一个字符的前面，而且它的长度可以为 0；根(Root)：单个长度的字符串；编码(Code)：一个数字，按照固定长度（编码长度）从编码流中取出，编译表的映射值；图案：一个字符串，按不定长度从数据流中读出，映射到编译表条目。

LZW 压缩的原理：提取原始图像数据中的不同图案，基于这些图案创建一个编译表，然后用编译表中的图案索引来替代原始光栅数据中的相应图案，减少原始数据大小。看起来和调色板图像的实现原理差不多，但是应该注意到的是，这里的编译表不是事先创建好的，而是根据原始图像数据动态创建的，解码时还要从已编码的数据中还原出原来的编译表（GIF 文件就是用 LZW 获得的，GIF 文件中不携带编译表信息）。

下面来看一下具体过程：

第一步，初始化一个编译表，假设这个编译表的大小是 12 位的，也就是最多有 4096 个单位，另外假设有 32 个不同的字符（也可以认为图象的每个像素最多有 32 种颜色），表示为 a，b，c，d，e...。初始化编译表：第 0 项为 a，第 1 项为 b，第 2 项为 c...一直到第 31 项，我们把这 32 项就称为根(单个字符)。

开始编译，先定义一个前缀对象 Current Prefix，记为[.c.]，现在它是空的，然后定义一个当前字符串 Current String，标记为[.c.]k，其中[.c.]就为 Current Prefix，k 就为当前读取字符。现在读取数据流的第一个字符，假如为 p，那么 Current String 就等于[.c.]p（由于[.c.]为空，实际上值就等于 p），现在在编译表中查找有没有 Current String 的值，由于 p 就是一个根字符，我们已经初始了 32 个根索引，当然可以找到，把 p 设为 Current Prefix 的值，不做任何事继续读取下一个字符，假设为 q，Current String 就等于[.c.]q（也就是 pq），看看在编译表中有没有该值，当然没有，这时要做下面的事情：将 Current String 的值（也就是 pq）添加到编译表的第 32 项，把 Current Prefix 的值（也就是 p）在编译表中的索引输出到编码流，修改 Current Prefix 为当前读取的字符（也就是 q）。

继续往下读，如果在编译表中可以查找到 Current String 的值([.c.]k)，则把 Current String 的值([.c.]k)赋予 Current Prefix；如果查找不到，则添加 Current String 的值([.c.]k)到编译表，把 Current Prefix 的值([.c.])在编译表中所对应的索引输出到编码流，同时修改 Current Prefix 为 k，这样一直循环下去直到数据流结束。

现在看一个具体的例子，假定有一个字母表 a，b，c，d。有一个输入的字符流 abacaba。现在来初始化编译表：#0=a，#1=b，#2=c，#3=d。现在开始读取第一个字符 a，[.c.]a=a，可以在在编译表中找到，修改[.c.]=a；不做任何事继续读取第二个字符 b，[.c.]b=ab，在编译表中不能找，那么添加[.c.]b 到编译表：#4=ab，同时输出[.c.]（也就是 a）的索引#0 到编码流，修改[.c.]=b；读下一个字符 a，[.c.]a=ba，在编译表中不能找到：添加编译表#5=ba，输出[.c.]的索引#1 到编码流，修改[.c.]=a；读下一个字符 c，[.c.]c=ac，在编译表中不能找到：添加编译表#6=ac，输出[.c.]的索引#0 到编码流，修改[.c.]=c；读下一个字符 a，[.c.]c=ca，在编译表中不能找到：添加编译表#7=ca，输出[.c.]的索引#2 到编码流，修改[.c.]=a；读下一个字符 b，[.c.]b=ab，编译表的#4=ab，修改[.c.]=ab；读取最后一个字符 a，[.c.]a=aba，在编译表中不能找到：添加编译表#8=aba，输出[.c.]的索引#4 到编码流，修改[.c.]=a；好了，现在没有数据了，输出[.c.]的值 a 的索引#0 到编码流，这样最后的输出结果就是：#0#1#0#2#4#0。

现在来看看如何解码数据。数据的解码其实就是数据编码的逆向过程，要从已经编译的数据（编码流）中找出编译表，然后对照编译表还原图像的光栅数据。首先，还是要初始化编译表。如 GIF 文件的图像数据的第一个字节存储的就是 LZW 编码的编码大小（一般等于图像的位数），根据编码大小，初始化编译表的根条目，然后定义一个当前编码 Current

Code，记作[code]，定义一个 Old Code，记作[old]。读取第一个编码到[code]，这是一个根编码，在编译表中可以找到，把该编码所对应的字符输出到数据流，[old]=[code]；读取下一个编码到[code]，这就有两种情况：在编译表中有或没有该编码，先看第一种情况：先输出当前编码[code]所对应的字符串到数据流，然后把[old]所对应的字符（串）当成前缀 prefix [...]，当前编码[code]所对应的字符串的第一个字符当成 k，组合起来当前字符串 Current String 就为[...]k，把[...]k 添加到编译表，修改[old]=[code]，读下一个编码。如果编译表中找不到该编码的情况，回想一下编码情况：如果数据流中有一个 p[...]p[...]pq 这样的字符串，p[...]在编译表中而 p[...]p 不在，编译器将输出 p[...]的索引而添加 p[...]p 到编译表，下一个字符串 p[...]p 就可以在编译表中找到了，而 p[...]pq 不在编译表中，同样将输出 p[...]p 的索引值而添加 p[...]pq 到编译表，这样看来，解码器总比编码器慢一步，当遇到 p[...]p 所对应的索引时，还不知道该索引对应的字符串（在解码器的编译表中还没有该索引，事实上，这个索引将在下一步添加），这时需要用猜测法：现在假设上面的 p[...]所对应的索引值是#58，那么上面的字符串经过编译之后是#58#59，在解码器中读到#59 时，编译表的最大索引只有#58，#59 所对应的字符串就等于#58 所对应的字符串(也就是 p[...])加上这个字符串的第一个字符(也就是 p)，也就是 p[...]p。事实上这种猜测是正确的。

(2) 课程设计目的

学习、掌握 LZW 压缩算法，并进行相应的实践。

(3) 基本要求

① 在一个文本文件上实现 LZW 压缩和解压缩，其中每个字符就是该文本的 8 位 ASCII 码。

② 在实现 LZW 过程中需要仔细考虑如何在编译表中找到匹配或找不到匹配，需要注意匹配算法的时间、空间开销。

③ 应用 LZW 算法实现 256 色灰度 BMP 图像文件的压缩和解压缩。

④ (选做)将一幅 256 色灰度 BMP 图像转化为 GIF 文件。

(4) 实现提示

需要查阅一定数量的相关资料，需要查看 BMP 和 GIF 图像文件的格式。

38. Chord 网络的模拟

(1) 问题描述

Chord 是一种非常经典的 P2P 结构化网络，可以在 Chord 上进行分布式 P2P 文件共享等应用。Chord 网络的基本结构如图 6-59 所示，它是以分布式散列表为基础构建的一种逻辑网络。

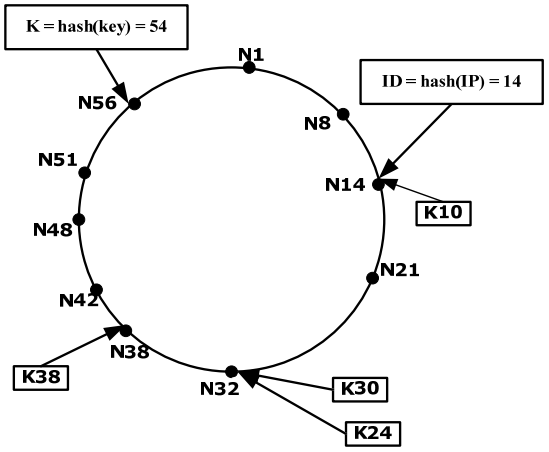


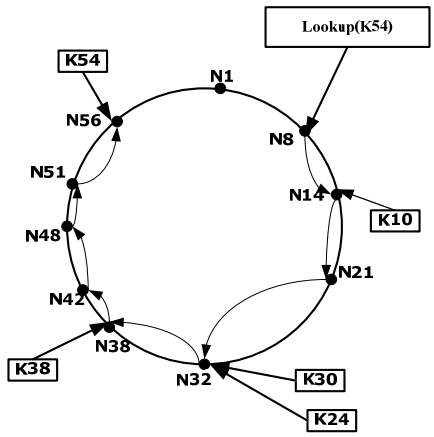
图 6-59 Chord 基本结构

分布式散列表（DHT）实际上是一个由大量结点分布式的共同维护的巨大散列表。散列表被分割成不连续的块，每个结点被分配给一个属于自己的散列块（一个散列值范围），并成为这个散列块的管理者，负责存储散列结果位于该散列块内的信息。DHT 中使用的散列函数通常是加密散列函数(如 MD5，SHA-1 等)，通过这些函数，一个对象的名字（如节点的 ID 或其 IP 地址）或关键词（如文件名）被映射为 128 位或 160 位的散列，如 SHA-1(“202.38.64.1”) =24b92cb1d2b81a47472a93d06af3d85a42e463ea。一个采用 DHT 的系统内，所有计算结点、数据对象等被映射到一个空间内。

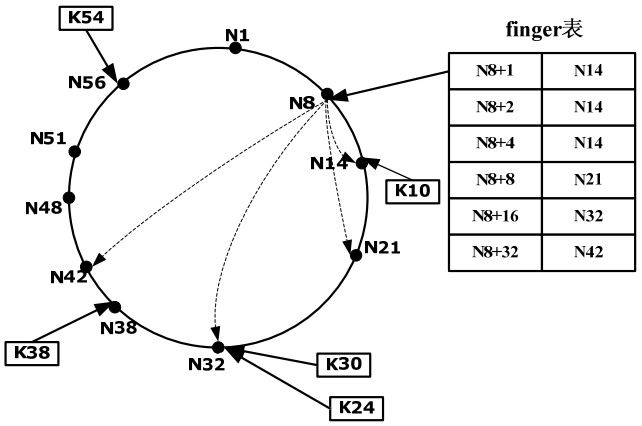
对于图 6-59 中的 Chord 结构，每个计算节点根据其 IP 可以计算出其 ID，如图中的 N14。每个文件根据其关键词可以计算出每个信息的 ID，就是图中的 K，如图中的 K=hash(key)=54，这些 K 被放在 Chord 环中机器节点 ID 大于 K 且最近的 K 的节点，如图中将 K=54 的信息放在 ID=56 的节点 N56 上，将 K=30 和 K=24 的信息放在 ID=32 的节点 N32 上。

Chord 结构主要支持如下操作：①Insert(key, V)，即将关键词 key 对应的信息 V 对存放到节点 ID 大于 K=hash(key)且离 K 最近的节点上，这样的 ID 可用 Successor(K)表示，其中 Successor (K)是从 K 开始顺时针方向距离 K 最近的节点。②Lookup(K)，根据 K 查询相应的 V，主要表现为根据 Chord 中节点的连接(上图中的节点间连线)路由到存放 K 的节点上，即节点 ID= Successor(K)的节点，在该节点上可以找到 K 对应的 V。③Update(K, new_V)：根据 K 更新相应的 V。④Join(NID)：加入一个新节点，NID 是节点的标识，如节点的 IP 地址，在 Chord 中加入一个节点需要建立相应的连接，需要移动信息数据，如上图中新加入一个节点 N26，则需要将 K=24 移动到该节点。⑤Leave()：某个节点主动离开 Chord，在 Chord 中推出一个节点需要修改相应的连接，也需要移动某些信息数据，如上图中退出一个节点 N14，则需要将 K=10 移动到节点 N21。

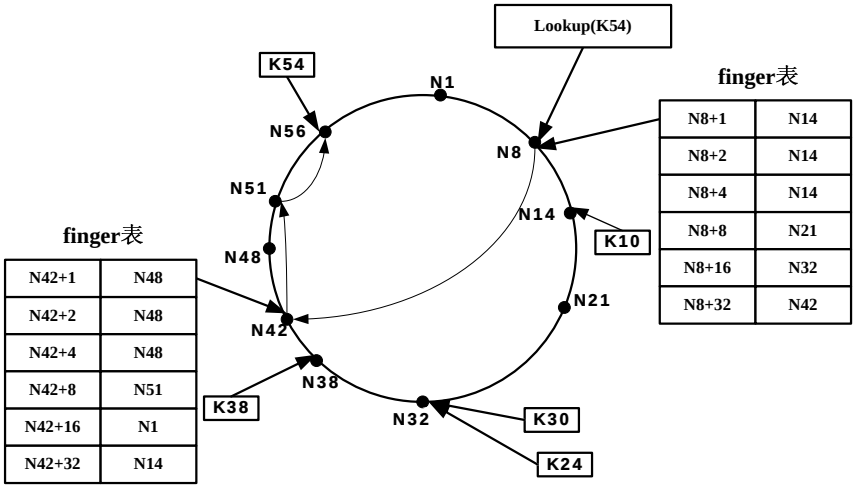
Chord 结构的一个非常重要的特点是如果每个节点仅维护其后继节点 ID、IP 地址等信息，则查询消息通过后继节点指针在圆环上传递，直到查询消息中包含的 K 落在某节点 ID 和它的后继节点 ID 之间，这样的查询速度太慢 O(N)，N 为网络中节点数，如图 6-60(*a*)所示。因此在基本 Chord 上，引入了一些能加快查询的 finger 表，finger 表的结构如图 6-60(*b*)表示，节点为 ID 的 finger 表中存放的是所有的$(ID+2^i)$ mod N≤ID 的 i 从 1 开始且逐个增 1 的 ID=Successor$(ID+2^i)$的那些节点，每个 i 对应了 finger 表中的 1 项。有了 finger 表后，可以加速查询过程，对于路由中的任何一个节点 ID，该节点选择的路由下一跳是 finger 表中满足$(ID+2^i)$ mod N 小于等于 K 且最接近 K 的那个 i 对应的表项，从这个表项中可以找到查询的下一跳，如图 6-60(*c*)所示。



(*a*)



(*b*)



(*c*)

图 6-60 带 finger 表的 Chord 结构

仔细分析可以发现，引入 finger 表后，查询 K 的路由跳数为 O(logN)，相比 O(N)而言有很大的提高。

(2) 课程设计目的

建立对 Chord 结构的认识，能用数据结构知识完成 Chord 结构的模拟。

(3) 基本要求

① 用数据结构知识设计和实现 Chord 网络结构。

② 实现 Chord 结构上的 Insert、Lookup、Update、Leave、Join 五个操作。

③ 构建一个 Chord 的单机模拟应用，其中的数据信息可以自己定义，如可以定义 key 为一个一个的英语单词，而其对应的数据为该单词的英文解释。

④ 应模拟出各个操作的结果，尤其是模拟出 Lookup 的路由过程，模拟过程应该是可视的，即可以看到节点的连接，路由一跳一跳的过程，鼓励使用 VC 实现。

⑤ 用实验结果分析验证引入 finger 表后，查询 K 的路由跳数为 O(logN)的理论结果。

(4) 实现提示

可以查阅 P2P 中关于 Chord 的资料，其中用到的散列函数可以直接使用现有的散列函数，如 SHA-1，可以直接下载使用散列函数的源代码。

39. Treap 结构上的基本操作

(1) 问题描述

如果将 n 个元素插入到一个一棵二元查找树中，所得到的树可能会非常不平衡，从而导致上面的查找时间很长。一种通常的处理办法是首先将这些元素进行随机置换，即先随机排列这些元素，然后再依次插入到二元查找树中，此时的二元查找树往往是平衡的，这种二元查找树称为随机二元查找树。但是这样的处理存在一个问题，即这样的处理方法只适用于固定的元素集合（已经预先知道了所有的元素）。如果没有办法同时得到所有的元素的话，即一次收到一个元素，则没有办法进行处理，此时可以用 treap 来处理这种情况，图 6-61 给出了一个 treap 结构。

Treap 结构中每个节点 x 有两个域，一个是其关键字值 key[x]，一个是优先级数 priority[x](它是一个独立选取的随机数)，上图节点中左边的字符就是 key[x]，右边的整数就是 priority[x]。对于 treap 结构，其关键字遵循二元查找树性质，其优先级遵循最小堆性质，即：如果 v 是 u 的左儿子，则 key[v]<key[u]，如果 v 是 u 的右儿子，则 key[v]>key[u]，如果 v 是 u 的儿子，则 priority[v]>priority[u]。所以这种结构被命名为 treap(tree+heap)。

有了 treap，假设依次插入关键字到一棵 treap 中，在插入一个新节点时：首先给该节点

随机选取一个优先数；然后将该节点按关键字插入到 treap 中的二元查找树中，此时 priority 可能会不满足堆的性质；最后按照 priority 调整 treap，在保证二元查找树性质的同时调整 treap(需要一系列旋转)使之按 priority 满足堆性质。图 6-62 给出了在 treap 结构上进行插入操作的处理过程。

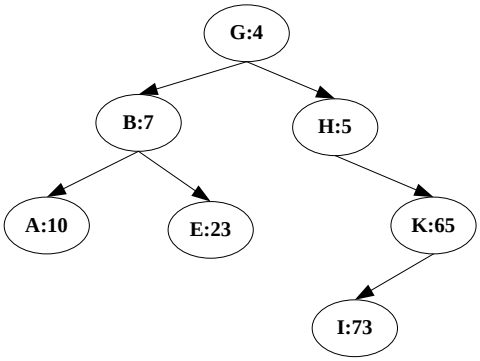
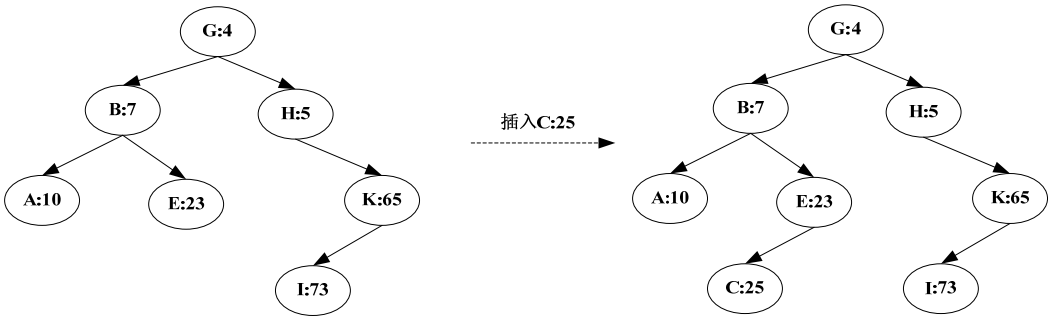
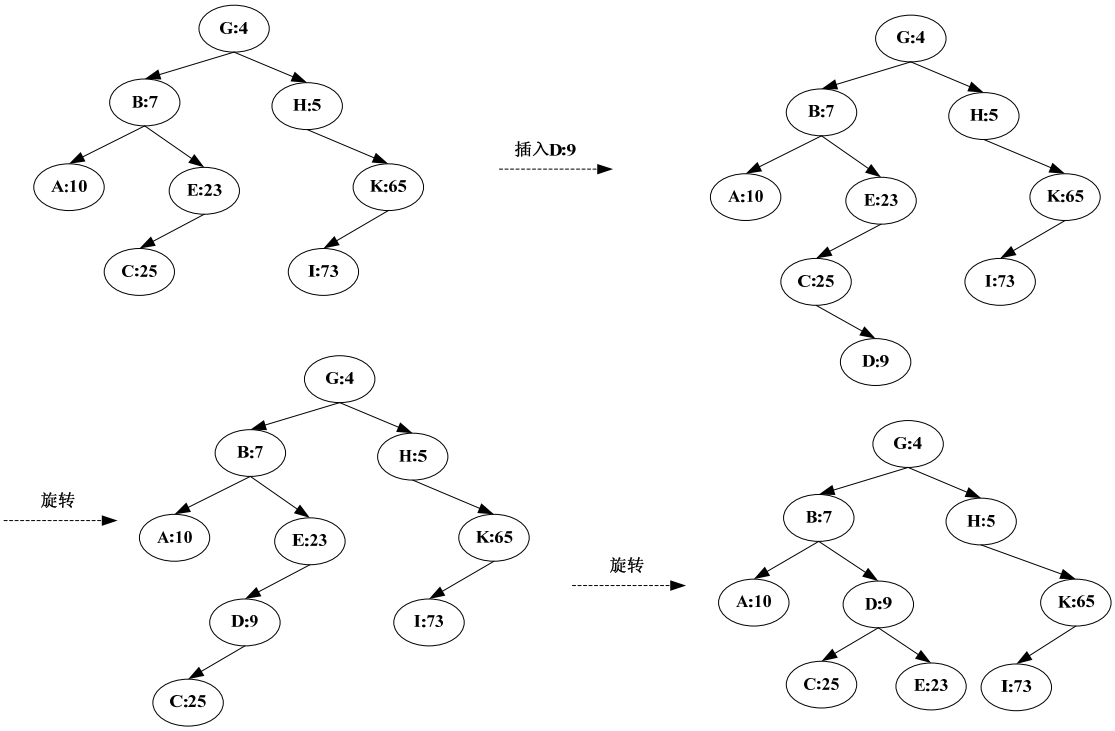


图 6-61 一个 Treap 结构



(*a*)



(*b*)

图 6-62 Treap 上的插入操作

(2) 课程设计目的

认识 treap 结构，对二元查找树和随机二元查找树建立深入的认识，能编程实现二元查找树、堆、treap 等结构以及其上的基本操作。

(3) 基本要求

① 用数据结构知识设计和实现 treap 结构以及上的基本操作：插入和删除。

② 编程实现随机二元查找树。

③ 产生若干个依次插入二元查找树的元素序列(元素个数自己设定)，针对该序列分别实现三种情况：插入到一般二元查找树、插入到 treap、用随机二元查找树处理。分别得出三种情况处理后的结果二元树的高度，进行实际的数据对比，从对比结果中发现一些规律。

④ 应模拟出各个操作的结果，模拟过程应该是可视的，即可以看到依次插入元素后各结构的变化情况。

⑤ 对随机二元树的高度、对 treap 的高度、对 treap 中的插入操作、对 treap 插入操作中旋转次数等指标进行理论分析。

(4) 实现提示

需要设计一个合适的随机排列算法来完成随机二元查找树的实现。

40. 用动态规划方法计算编辑距离并完成自动编辑

(1) 问题描述

对于任意两个字符串 s1，s2 的差别，可以通过其编辑距离来度量。编辑距离就是指让 s1 变成 s2 或将 s2 变成 s1 所需要编辑操作的最小次数。此处定义 3 个基本编辑操作：把某个字符 ch1 变成 ch2；删除某个字符；插入某个字符。如对于 s1=12433 和 s2=1233，则可以通过在 s1 中间删除 4 得到 1233 来与 s2 一致，所以 d(s1，s2)=1(进行了一次删除操作)，也即 s1，s2 的编辑距离为 1。

可以依靠编辑距离的性质来帮助完成编辑距离的计算。当计算两个字符串 s1+ch1, s2+ch2 的编辑距离有这样的性质: ①d(s1，“ ”) = d(“”，s1) = |s1|；d(“ch1”，”ch2”) = ch1 == ch2？0 : 1。②d(s1+ch1，s2+ch2)=min(d(s1,s2)+ch1==ch2？0 : 1，d(s1+ch1,s2)+1，d(s1,s2+ch2)+1)。具有第二个性质的原因是在 s1+ch1 和 s2+ch2 相互转化时可能的操作有：①把 ch1 变成 ch2，此时 d(s1+ch1，s2+ch2)=d(s1,s2)+ch1==ch2？0 : 1。②s1+ch1 删除 ch1 和 s2+ch2 转化，此时 d(s1+ch1，s2+ch2)=(1+d(s1，s2+ch2))。③s2+ch2 删除 ch2 和 s1+ch1 转化，此时 d(s1+ch1，s2+ch2)=(1+d(s1+ch1，s2))。④s2+ch2 后添加 ch1 和 s1+ch1 转化，实际上等同于 s2+ch2 和 s1 转化，此时 d(s1+ch1，s2+ch2)=(1+d(s1，s2+ch2))。⑤s1+ch1 后添加 ch2 和 s2+ch2 转化，实际上等同于 s1+ch1 和 s2 转化，此时 d(s1+ch1，s2+ch2)=(1+d(s2，s1+ch1))。有了这样的性质，可以发现在计算 d(m，n)时用到了 d(m-1，n)，d(m-1，n-1)，d(m，n-1)等，也就是说明该问题具有重叠子问题结构（原问题的递归算法反复的求解同样的子问题，而不产生新的子问题），再由性质②表明的最优子结构（一个问题的最优解包含了子问题的最优解），符合动态规划算法基本要素。因此可以使用动态规划算法来求解该问题。

由此可以得到这样的动态规划算法：设定数组 M[|s1|，|s2|]保存子问题结果，其中 M[i , j] 表示子串 s1(0->i) 与 s2(0->j) 的编辑距离。产生字符之间的编辑距离，E[|s1|, |s2|]，其中 E[i, j] = s[i] = s[j] ? 0 : 1。初始化 M 矩阵，M[0,0] = 0; M[s1i，0] = |s1i|; M[0, s2j] = |s2j|。根据性质②计算出矩阵 M，得到编辑距离：M[i，j]= min(m[i-1,j-1] + E[i, j]，m[i, j-1]+1，m[i-1, j])。

另外，编辑距离也会随着编辑操作而有所变化，如也可以这样来定义编辑：给定两个字符串 x[1..m]和 y[1..n]，编辑的目标是找到一系列的编辑操应用于 x 使它变为 y。用一个中间变量 z 来保存中间结果，算法开始的时候 z 是一个空字符串，算法结束的时候对所有

j=1, 2, …, n 有 zj=yj。用变量 i 标记当前正被处理的字符在 x 中的下标，用 j 标记 z 的当前下标，我们的编辑操作就是作用于 i, j 和 z 上面的。开始时有 i=j=1，要求是在转换过程中一一检查 x 中的每一个字符，也就是说在转换结束的时候必须有 i=m+1。

现定义六种编辑操作：①拷贝：把一个字符从 x 拷贝到 z，即令 zj=xi，之后令 i=i+1, j=j+1。这个操作检查了 xi。②替换：将 x 中的一个字符替换为另外一个，即令 zj=c，之后令 i=i+1, j=j+1。这个操作检查了 xi。③删除：从 x 中删除一个字符，即令 i=i+1，j 不变。这个操作检查了 xi。④插入：向 z 中插入一个字符，即令 zj=c，之后令 j=j+1，i 不变。这个操作没有检查 x 中的任何字符。⑤换位：把 x 中将要处理的下两个字符拷贝到 x 中，但是要颠倒顺序，即令 zj=x[i+1], z[j+1]=xi 之后令 i=i+2, j=j+2。这个操作检查了 xi 和 x[i+1]。⑥截断：删掉 x 中剩下的全部字符，即令 i=m+1。这个操作检查了 x 中当前尚未被检查到的全部字符，这个操作只能作为最后一个操作出现在编辑操作序列中。

例如将源串：algorithm 转换成目标串 altruistic 的一种方法是采用下面的操作序列：

操作	目标串	源串
copy a	a	lgorithm
copy l	al	gorithm
replace g by t	alt	orithm
delete o	alt	rithm
copy r	altr	ithm
insert u	altru	ithm
insert i	altrui	ithm
insert s	altruis	ithm
twiddle it into ti	altruisti	hm
insert c	altruistic	hm
kill hm	altruistic	

上述每一个变换操作都有相应的代价。假定所有编辑操作的代价都是常数并且是已知的，且要求拷贝和替换操作的代价小于等效的删除、插入操作的代价的和。定义给定的编辑操作序列的代价是其中每一个编辑操作的代价的总和，对上面的例子来说，编辑序列的代价是 (3*cost(copy)) +cost(replace) +cost(delete) +(4*cost(insert)) +cost(twiddle) +cost(kill)。显然，需要找到一个从源串到目标串的具有最小代价的编辑操作序列。

(2) 课程设计目的

掌握动态规划方法，能够编程实现。对编辑距离建立一定的认识，并能将某些问题转化为编辑距离问题。

(3) 基本要求

① 实现问题描述中第一种编辑操作(三个操作)编辑距离计算的动态规划算法，并用测试用例验证算法。

② 将问题描述中第二种编辑操作中给出的六个操作都实现，并将其作为串 ADT 的基本操作实现一个串 ADT。

③ 针对问题描述中第二种编辑操作，设计动态规划算法实现其编辑距离的计算，即找到一个代价最小的编辑操作序列，其中各个操作的代价根据实际情况自己设定。

④ 应用③获得的最小代价编辑操作序列实现从源串到目标串的自动编辑。

⑤ 针对问题描述中第二种编辑操作，自行设计可获得自动编辑操作序列(代价不用最小)的一个算法，将其结果编辑操作序列和③获得的最小代价编辑操作序列进行代价上的对比。

⑥ (选做)编辑距离问题是 DNA 序列对齐问题的泛化。可以有很多方式对齐 DNA 序

列，以衡量它们的相似程度。有一种对齐方式是，可以在两个 DNA 序列的任何位置（包括两头）插入空格，但是要求通过这个过程得到的序列 x1 和 y1 长度相同，而且在两个序列的同一位置上不能都是空格。做到这一步之后我们为每一个位置分配一个“得分”，例如位置 j 按照如下规则得分：如果 x1[j]==y1[j]，1 分；如果 x1[i]!=y1[j] 并且两者都不是空格，-1 分；如果 x1[j]或者 y1[j]是空格，-2 分。某种特定对齐结果的得分，是全部位置的得分的总和。例如对序列 x=GATCGGCAT 和 y=CAATGTGAATC，一个可能的对齐如下：

G ATCG GCAT

CAAT GTGAATC

-*+++*+*+-+++*

其中+对应 1 分，-对应-1 分，*对应-2 分。

这个对齐的总分是-1-2+2-2+1-2+1-1+2-2=-1-2+1-2=-4。

在上述编辑距离动态规划算法基础上，编写程序求出两个 DNA 序列的最优对齐，即得分最多的对齐。

(4) 实现提示

需要查阅关于动态规划算法的有关资料。

41. 用线段树进行数据的动态维护

(1) 问题描述

当处理图形的面积、周长等问题的时候，并不需要依赖很深的数学知识，但要提高处理此类问题的效率却又十分困难。这就需要从根本上改变算法的基础——数据结构，不采用线性表结构，这里需要的就是一种特殊的数据结构——线段树。在此类问题中，需要经常处理可以映射在一个坐标轴上的一些固定线段，例如说映射在 X 轴上的线段。由于线段是可以互相覆盖的，有时需要动态地取线段的并，例如取得并区间的总长度，或者并区间的个数等等。一个线段是对应于一个区间的，因此线段树也可以叫做区间树。

线段树是一棵二叉树，树中的每一个结点表示了一个区间 [a,b] 。每一个叶子节点上 a+1=b，这表示了一个初等区间(单元区间)。对于每一个内部结点 b-a>1 ，设根为 [a,b] 的线段树为 T(a,b), 则进一步将此线段树分为左子树 T(a,(a+b)/2), 以及右子树 T((a+b)/2,b), 直到分裂为一个初等区间为止。如图 6-63 给出的就是一棵对应于区间[1，11]的线段树。注意，这只是线段树的基本结构。通常利用线段树的时候需要在每个结点上增加一些特殊的数据域，并且它们是随线段的插入删除进行动态维护的。这因题而异，同时又往往是解题的灵魂。

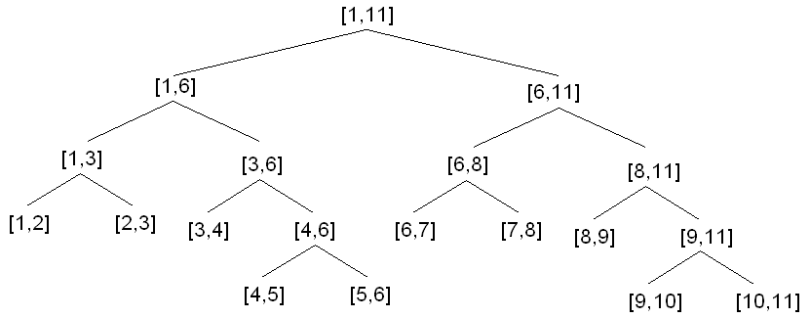


图 6-63 区间[1，11]对应的线段树

图 6-63 是一个抽象的线段树，在实际处理问题时是先将任何一个要处理的线段（区间）[a,b]进行了一定的转换。即在处理问题的时候，首先对各个区间根据其端点进行离散化，即抽取出区间的端点，如有 N 个端点，然后对这些端点进行和自然数 1，2，…，N 对应，然后就可以得出如图 6-64 的线段树。如有 6 个区间[1，10]，[5，100]，[20，40]，[30，60]，

[60，80]，[200，210]。对这些区间进行从小到大排序{1,5,10,20,30,40,60,80,100,200,210}，对应的标号就是{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}。因此图 6-63 对应的实际区间的线段树如图 6-64 所示。

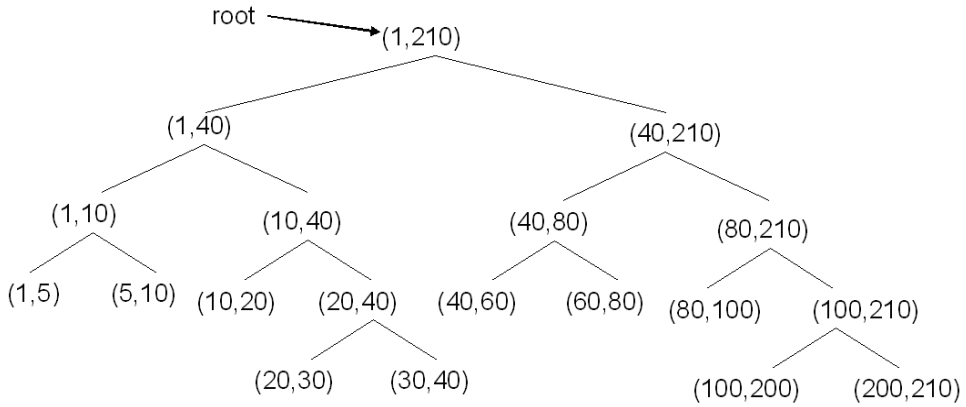


图 6-64 实际区间对应的线段树

线段树节点的数据结构和操作会根据不同的应用来调整，但是它也有基本的结构和操作。线段树节点的基本数据结构：

```
typedef struct node {   int xleft, xright; //节点对应区间的左端点和右端点坐标
                        struct node *left,*right; }Node;
```

线段树的基本操作包括插入操作和删除操作。

在节点 p 上插入一个区间[a,b]，insert(a,b,p)要求 p->xleft≤a<b≤ p->xright，而且 a,b 都是离散点的坐标，insert(a,b,p)的处理流程是：

1．mid=p->left->xright；

2．若 a=p->xleft 且 b=p->xright，则执行有关的插入操作；返回；

3．若 b≤mid，则执行 insert(a,b,p->left)；返回； //左边插入，递归执行

4．若 a≥mid，则执行 insert(a,b,p->right)；返回；//右边插入，递归执行

5．执行 insert(a,mid,p->left)；执行 insert(mid,b,p->right)；返回；//两边都递归插入

这是一个递归过程。其中第 1 步执行完后，第 2,3,4,5 步只有一步会被执行。若执行了第 2 步，则称节点 p 为一个插入点，关键工作都是在插入点进行的。若执行了第 3,4 或 5 步，则 p 都不会被称为插入点。在插入点进行的关键工作会在不同的应用中体现为不同的操作，如可以给每个节点上定义一个 count 变量，该变量 count 用来记录覆盖该结点的线段条数，此时每次插入操作使 count 变量增 1。

设 a<b 是两个离散点的坐标，root 是线段树的根节点，称操作 insert(a,b,root)为在线段树上插入区间[a,b]的操作，仍以上面的例子来说明插入操作：

(*a*)设在线段树上插入区间[1,10]，即运行 insert(1,10,root)。其结果是先后递归运行：

```
insert(1,10,(1,210))
insert(1,10,(1,40))
insert(1,10,(1,10))
```

而其中的插入点只有(1,10)这一个节点。

(*b*)在线段树上插入区间[5,100]，即运行 insert(5,100,root)的结果是先后递归运行：

```
insert(5,100,(1,210))
insert(5,40,(1,40))
insert(5,10,(1,10))
insert(5,10,(5,10))          //插入点
insert(10,40,(10,40))        //插入点
```

insert(40,100,(40,210))
insert(40,80,(40,80))　　//插入点
insert(80,100,(80,210))
insert(80,100,(80,100))　//插入点

这次插入出现了(5,10), (10,40), (40,80), (80,100)共 4 个插入点。

一般说来，若有 N 条离散线段，则线段树的叶子节点有 N 个，那么线段树的节点总数≤2N，每次插入操作最多需要 2log N 次递归调用插入过程，也就最多有 2log N 个插入点。

线段树的构造主要是对区间线段的处理，它往往被应用于几何计算问题中。线段树的作用主要体现在可以动态维护一些特征，例如说要得到线段树上线段并集的长度，可以增加一个数据域 p->M：如果 p->C>0，p->M = p->xright - p->xright; p->C=0 且 v 是叶子结点，p->M=0；p->C=0 且 v 是内部结点，p->M=p->left->M+p->right->M。只要每次插入或删除线段区间时，在访问到的结点上更新 M 的值，不妨称之为 UPDATA ，就可以在插入和删除的同时维持好 M 。求整个线段树的并集长度时，只要访问 root->M 的值。这在许多动态维护的题目中是非常有用的，它使得每次操作的维护费用只有 log*n*。类似的，还有求并区间的个数等等。

(2) 课程设计目的

掌握线段树的基本原理，编程实现线段树及其上的基本操作，应用线段树解决实际问题。

(3) 基本要求

① 编程实现线段树的数据结构定义、线段树的初始化操作、线段树的插入和删除操作。

② 应用线段树解决一维线段上的计算几何问题，如求解这些线段的并集的长度，给定一个点，多少线段覆盖这个点等等。

③ 应用线段树解决二维图形上的计算几何问题，如坐标轴的上半平面上有 N 个矩形。每个矩形都有一条边在 x 轴上，记这条边为区间$[A_i,B_i]$，记该矩形的高为 H_i，i=1,…,N。其中 $1\le A_i<B_i\le 10^9$，$1\le i\le N\le 40000$。

信息的输入格式：N

$A_1\ B_1\ H_1$

$A_2\ B_2\ H_2$

…

$A_N\ B_N\ H_N$

要求输出平面上被这些矩形覆盖的部分的面积。

④ 应用线段树解决二维图形上的计算几何问题，平面上有 N 个矩形，矩形的边平行于坐标轴。每个矩形左下角和右上角的坐标是(xl_i,yl_i)和(xu_i,yu_i)，i=1,…,N。其中坐标值的范围是[-10000,10000]， $1\le N\le 5000$。

输入格式：

N

$xl_1\ yl_1\ xu_1\ yu_1$

$xl_2\ yl_2\ xu_2\ yu_2$

…

$xl_N\ yl_N\ xu_N\ yu_N$

要求输出平面上被这些矩形覆盖的部分轮廓的周长。

(4) 实现提示

对于要求③，在节点结构中增加 int height 即节点对应线段被覆盖的高度，初始为 0。并在每个插入点 p 处执行 p->height=max{h,p->height}。所有矩形都处理完以后，就可以求总

面积了。下面的过程 area(h,p)将节点 p 对应的区间上高度≥h 的部分的面积累加到变量 sumarea 中：area(h,p)，1. diff=p->height-h；2. 若 diff>0，则执行 sumarea+=diff*(p->xright - p->xleft)；h=p->height；3. 若 p->left 非空，则执行 area(h,p->left)；area(h,p->right)。

对于要求④，核心是设计一条竖直扫描线，从左到右扫描，每当扫描线固定在某个位置时，计算该扫描线被这些矩形覆盖的部分的长度(不妨称为扫描到的长度)和被这些矩形覆盖的区间的段数(不妨称为扫描到的段数)。例如，当扫描线位于虚线所在位置时，扫描到的段数是 2 段；扫描到的长度自然是这两段长度的和。注意到下面的事实：一是扫描到的长度和段数只有在扫过矩形的竖直边时才会发生变化。二是当扫描线从位置 1 移到位置 2 时：[2*位置 1 扫描到的段数*(位置 2 横坐标-位置 1 横坐标)]就是扫描线从位置 1 到位置 2 扫描到的水平轮廓的长度,[abs(位置 2 扫描到的长度-位置 1 扫描到的长度)]就是扫描线在位置 2 遇到的竖直轮廓的长度。需要说明的是：当扫描线恰好位于一条竖直边上时，其扫描到的长度和段数是有歧义的，定义此时扫描到的长度和段数为扫描线位于竖直边右边任意近位置时扫描到的长度和段数。后面的实际算法是一条竖直边一条竖直边的扫描的，如果有两条竖直边横坐标相同，那么求出的长度和段数与这里定义类似，只是分步得到。由上面的观察，计算轮廓周长 sum 的过程大致如下：(1)初始设 sum=0，scanlength=0，scanseg=0，scanpos=-10000。(2) 找下一条竖直边(x,low,up)=(竖直边的横坐标，下端点的纵坐标，上端点的纵坐标)，计算当前扫描到的长度 newscanlength 和段数 newscanseg。sum+=2*scanseg*(x-scanpos); sum+=abs(newscanlength-scanlength); scanlength=newscanlength; scanseg=newscanseg; scanpos=x; (3) 若还有竖直边未扫描，则转(2)。

42．求第 K 短的最短路径

(1) 问题描述

最短路径问题是图论中的一个经典问题，主要研究成果有Dijkstra、Floyd等优秀算法，Dijkstra算法一直被认为是图论中的好算法。但这两个算法有一个共同的缺陷：这里的最短路径指两点之间最短的那一条路径, 不包括次短、再次短等等路径。实际上，在使用咨询系统或决策支持系统时，希望得到最优的决策参考外，还希望得到次优、再次优等决策参考。这同样反映在最短路径问题上，如一个交通咨询系统，除了希望得到最短路径以外，由于交通堵塞等问题，可能需要获知次短、第K短的最短路径。因此，有必要将最短路径问题予以扩充，能求出第K短最短路径。形式的表述就是想要在图中求出从起点到终点的前k短的路径（最短、第2短、第3短……第k短），并且需要这些路径都是无环的。

常见的较好的求解前 k 短无环路径的算法是 Yen 算法（以发明者名字命名的）。现在简要地描述一下 Yen 算法。设 P_i 为从起点 s 到终点 t 的第 i 短的无环路径。一开始是 P_1，也就是从 s 到 t 的最短路径，可以通过 Dijkstra、Bellman-Ford 或 BFS 等算法轻易地求出。接下来要依次求出 P_2，P_3……P_k。可以将 P_1~P_i 看成一棵树，称为 T_i，它的根节点是 s，所有叶节点都是 t。例如假设 s = 1，t = 5，求出的 P_1~P_5 为，P_1：`1->2->3->5`；P_2：`1->3->2->5`；P_3：`1->3->5`；P_4：`1->4->5`；P_5：`1->4->3->5`。此时的 T_5 就是如图 6-65 所示的一棵树。

定义 dev_i 为 P_i 的偏离点（deviation node），定义为在 T_i 上，P_i 对应的那一分枝中，第 1 个（按从 s 到 t 的顺序）不在 T_{i-1} 上的点（i > 1）。为描述的方便，设 dev_1 为 P_1 的第 2 个点。因此，在图 1 中，dev_1~dev_5 的编号分别为 2、3、5、4 和 3。显而易见，dev_i 至少是 P_i 的第 2 个点。接下来是 Yen 算法的核心部分，即每当求出一个 P_i 时，都可以从 P_i 上发展出若干条候选路径。发展的方法是这样的，对于 P_i 上从 dev_i 的前一个点到 t 的前一个点这一段上的每个点 v，都可以发展出一条候选路径。用 $P_{i\,sv}$ 表示 P_i 上从 s 到 v 的子路径，用 $P_{i\,vt}$ 表示从 v 到 t 的满足下列条件的最短路径：condition 1：设点 v 在 T_i 上对应的点为 v'，则 $P_{i\,vt}$ 上从点 v 出发的那条边不能与 T_i 上从点 v'出发的任何一条边相同。condition 2：$P_{i\,vt}$ 上，除了点 v，其它点都不能出现在 $P_{i\,sv}$ 上。如果找得出 $P_{i\,vt}$，则把 $P_{i\,sv}$ 和 $P_{i\,vt}$ 连起来就组成了一

条候选路径。其中 condition 1 保证了候选路径不与 P_1~P_i 重复；condition 2 保证了候选路径无环。

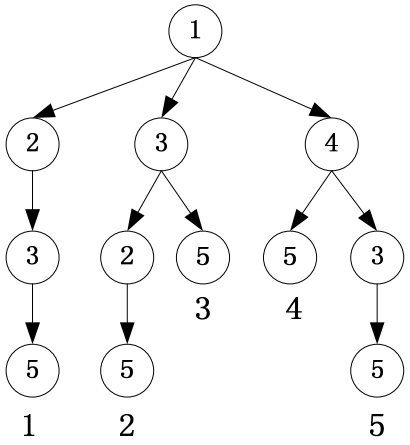


图6-65 路径P_1~P_5对应的树T_5

以图 6-65 中的例子为基础，可以举一个发展候选路径的例子。在求出了 P_5 之后，要在 P_5 上发展候选路径。P_5 的偏离点是 3 号点。因此 v 的范围是{4, 3}。

当 v = 4 时，$P_{i\,sv}$ = 1 -> 4，因此，根据 condition 2，在 $P_{i\,vt}$ 上不能出现 1 号点。找到 P_5 上的 4 号点在 T_5 上对应的那一点，也就是图 6-中位于阴影 3 号点上面的 4 号点，在 T_5 上从它出发的有(4, 5)和(4, 3)这两条边，因此，根据 condition 1，在 $P_{i\,vt}$ 上不能出现这两条边。假设在这样的情况下，求出了从 4 号点到 t 的最短路径为 4 -> 2 -> 5，它就是 $P_{i\,vt}$。此时发展出的候选路径就是 1 -> 4 -> 2 -> 5。

当 v = 3 时，$P_{i\,sv}$ = 1 -> 4 -> 3，因此，根据 condition 2，在 $P_{i\,vt}$ 上不能出现 1 号点和 4 号点。找到 P_5 上的 3 号点在 T_5 上对应的那一点，也就是图 6-66 中阴影的 3 号点，在 T_5 上从它出发的只有(3, 5)这一条边，因此，根据 condition 1，在 $P_{i\,vt}$ 上不能出现边(3, 5)。假设在这样的情况下，我们求出了从 3 号点到 t 的最短路径为 3 -> 2 -> 5，它就是 $P_{i\,vt}$。此时发展出的候选路径就是 1 -> 4 -> 3 -> 2 -> 5。

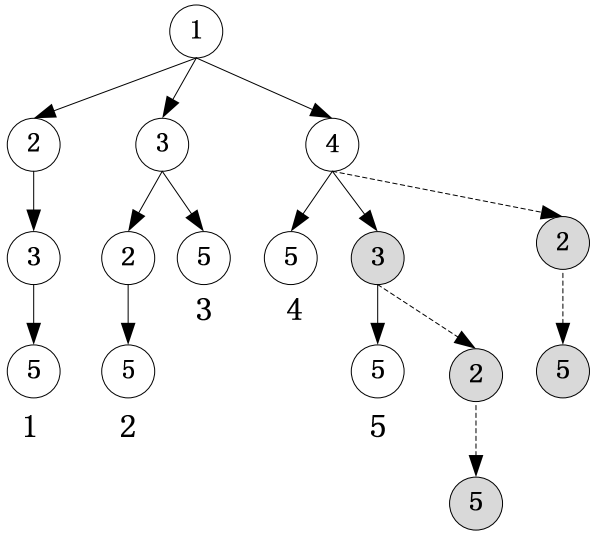


图6-66 扩展路径P_5产生的候选路径

显而易见，在从 P_i 发展出的所有候选路径中，只有当 v 是 dev_i 的前一个点时，条件 1 才有可能阻挡掉 2 条或 2 条以上边。当 v 不是 dev_i 的前一个点时，条件 1 只会阻挡掉 1 条边，那就是本身位于 P_i 上，从 v 出发的那条边。不仅从 P_i，从之前的 P_1~P_{i-1}，我们都发展过若干条候选路径。从候选路径的集合中取出最短的一条，就是 P_{i+1}。把 P_{i+1} 从候选路径的集

合里删掉；然后再从它发展新的候选路径，添加到候选路径的集合里，如此循环，直到求出 P_k 为止。如果在求到 P_k 之前，候选路径的集合就空了，那么说明 P_k 不存在。

(2) 课程设计目的

学习、掌握、编程实现 Yen 算法，知道如何求解第 K 短最短路径。

(3) 基本要求

① 给定一个加权图，编程实现 Yen 算法，依次输出所有的无环最短路径。

② 候选路径集合用堆存储实现，方便快速选取最短的一条。

③ 分析 Yen 算法的时间复杂性。

(4) 实现提示

在 Yen 算法中会调用 Dijkstra 算法，图可用邻接矩阵表示。

43．公交线路上优化路径的查询

(1) 问题描述

最短路径问题是图论中的一个经典问题，其中的 Dijkstra 算法一直被认为是图论中的好算法，但有的时候需要适当的调整 Dijkstra 算法才能完成多种不同的优化路径的查询。

对于某城市的公交线路，乘坐公交的顾客希望在这样的线路上实现各种优化路径的查询。设该城市的公交线路的输入格式为：

线路编号：起始站名(该站坐标)；经过的站点 1 名(该站坐标)；经过的站点 2 名(该站坐标)；……；经过的站点 n 名(该站坐标)；终点站名(该站坐标)。该线路的乘坐价钱。该线路平均经过多少时间来一辆。车速。

例如：63：A(32,45)；B(76,45)；C(76,90)；……；N(100,100)。1 元。5 分钟。1/每分钟。

假定线路的乘坐价钱与乘坐站数无关，假定不考虑公交线路在路上的交通堵塞。

对这样的公交线路，需要在其上进行的优化路径查询包括：任何两个站点之间最便宜的路径；任何两个站点之间最省时间的路径等等。

(2) 课程设计目的

从实际问题中合理定义图模型，掌握 Dijkstra 算法并能用该算法解决一些实际问题。

(3) 基本要求

① 根据上述公交线路的输入格式，定义并建立合适的图模型。

② 针对上述公交线路，能查询获得任何两个站点之间最便宜的路径，即输入站名 S，T 后，可以输出从 S 到 T 的最便宜的路径，输出格式为：线路 x：站名 S，…，站名 M1；换乘线路 x：站名 M1，…，站名 M2；…；换乘线路 x：站名 MK，…，站名 T。共花费 x 元。

③ 针对上述公交线路，能查询获得任何两个站点之间最省时间的路径（不考虑在中间站等下一辆线路的等待时间），即输入站名 S，T 后，可以输出从 S 到 T 的考虑在中间站等下一辆线路的等待时间的最省时间的路径，输出格式为：线路 x：站名 S，…，站名 M1；换乘线路 x：站名 M1，…，站名 M2；…；换乘线路 x：站名 MK，…，站名 T。共花费 x 时间。

④ 针对上述公交线路，能查询获得任何两个站点之间最省时间的路径（要考虑在中间站等下一辆线路的等待时间），即输入站名 S，T 后，可以输出从 S 到 T 的考虑在中间站等下一辆线路的等待时间的最省时间的路径，输出格式为：线路 x：站名 S，…，站名 M1；换乘线路 x：站名 M1，…，站名 M2；…；换乘线路 x：站名 MK，…，站名 T。共花费 x 时间。

(4) 实现提示

需深入考虑，应根据不同的应用目标，即不同的优化查询来建立合适的图模型。

44．聚类分析的初步实践

(1) 问题描述

分类学是人类认识世界的基础科学。聚类分析和判别分析是研究事物分类的基本方法，广泛地应用于自然科学、社会科学、工农业生产的各个领域。聚类分析的基本思想是根据事物本身的特性研究个体分类的方法，原则是同一类中的个体有较大的相似性，不同类中的个体差异很大。

在分类之前首先需要定义分类的根据，根据不同分类的结果也不同，如要想把中国的县分成若干类，就有很多种分类法：可以按照自然条件来分，比如考虑降水、土地、日照、湿度等各方面；也可以考虑收入、教育水准、医疗条件、基础设施等指标。

下面给出一个小的实例，如果想要对 100 个学生进行分类，如果仅仅知道他们的数学成绩，则只好按照数学成绩来分类；这些成绩在直线上形成 100 个点。这样就可以把接近的点放到一类。如果还知道他们的物理成绩，这样数学和物理成绩就形成二维平面上的 100 个点，也可以按照距离远近来分类。三维或者更高维的情况也是类似；只不过三维以上的图形无法直观地画出来而已。因此需要定义数据之间的距离概念，从而可以度量数据之间的远近程度，作为聚类的基础。

在定义数据之间距离概念时，此时需要明确两个概念：一个是点和点之间的距离，一个是类和类之间的距离。点间距离有很多定义方式。最简单的是歐氏距离，还有其他的距离距离。当然还有一些和距离相反但起同样作用的概念，比如相似性等，两点越相似度越大，就相当于距离越短。由一个点组成的类是最基本的类；如果每一类都由一个点组成，那么点间的距离就是类间距离。但是如果某一类包含不止一个点，那么就要确定类间距离，类间距离是基于点间距离定义的：比如两类之间最近点之间的距离可以作为这两类之间的距离，也可以用两类中最远点之间的距离作为这两类之间的距离；当然也可以用各类的中心之间的距离来作为类间距离。在计算时，各种点间距离和类间距离的选择是通过统计软件的选项实现的。不同的选择的结果会不同，但一般不会差太多。

常见的定义点和点之间距离方式，即向量 $x=(x_1,\ldots, x_p)$与 $y=(y_1,\ldots, y_p)$之间的距离或相似系数为：

①欧氏距离(Euclidean)：$\sqrt{\sum_i (x_i - y_i)^2}$

②绝对距离(Block)：$\sum_i |x_i-y_i|$

③Chebychev 距离：$\mathrm{Max}_i |x_i-y_i|$

④夹角余弦(相似度量)：$C_{xy}(1) = \cos\theta_{xy} = \dfrac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}}$

⑤Pearson correlation(相似度量)：$C_{xy}(2) = r_{xy} = \dfrac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$

常见的类 G_p 与类 G_q 之间的距离 D_{pq}($d(x_i,x_j)$表示点 $x_i \in G_p$ 和 $x_j \in G_q$ 之间的距离)。

①最短距离法：$D_{pq} = \min d(x_i, x_j)$

②最长距离法：$D_{pq} = \max d(x_i, x_j)$

③类平均法：$D_{pq} = \dfrac{1}{n_1 n_2} \sum_{x_i \in G_p} \sum_{x_j \in G_q} d(x_i, x_j)$

④重心法：$D_{pq} = \min d(\bar{x}_p, \bar{x}_q)$

有了上面的点间距离和类间距离的概念，就可以介绍聚类的方法了。此处介绍两种常见的聚类算法：

一种是事先要确定分多少类的 k-均值聚类算法（k-means cluster）。假定你说分 3 类，这个方法还进一步要求你事先确定 3 个点为“聚类种子”；也就是说，把这 3 个点作为三类中每一类的基石。然后，根据和这三个点的距离远近，把所有点分成三类。再把这三类的中心（均

值）作为新的基石或种子（原来的“种子”就没用了），重新按照距离分类。如此叠代下去，直到达到停止叠代的要求（比如，各类最后变化不大了，或者叠代次数太多了）。显然，前面的聚类种子的选择并不必太认真，它们很可能最后还会分到同一类中呢。下面是一个数据实例：

饮料编号	热量	咖啡因	钠	价格
1	207.20	3.30	15.50	2.80
2	36.80	5.90	12.90	3.30
3	72.20	7.30	8.20	2.40
4	36.70	.40	10.50	4.00
5	121.70	4.10	9.20	3.50
6	89.10	4.00	10.20	3.30
7	146.70	4.30	9.70	1.80
8	57.60	2.20	13.60	2.10
9	95.90	.00	8.50	1.30
10	199.00	.00	10.60	3.50
11	49.80	8.00	6.30	3.70
12	16.60	4.70	6.30	1.50
13	38.50	3.70	7.70	2.00
14	.00	4.20	13.10	2.20
15	118.80	4.70	7.20	4.10
16	107.00	.00	8.30	4.20

Final Cluster Centers

	Cluster		
	1	2	3
CALORIE	203.10	33.71	107.34
CAFFEINE	1.65	4.16	3.49
SODIUM	13.05	10.06	8.76
PRICE	3.15	2.69	2.94

图 6-67 16 种饮料的各类数据的 k-means cluster 算法的执行结果

执行 k-means cluster 算法后，可得到聚类结果是：第一类为饮料 1、10；第二类为饮料 2、4、8、11、12、13、14；第三类为剩下的饮料 3、5、6、7、9、15、16。

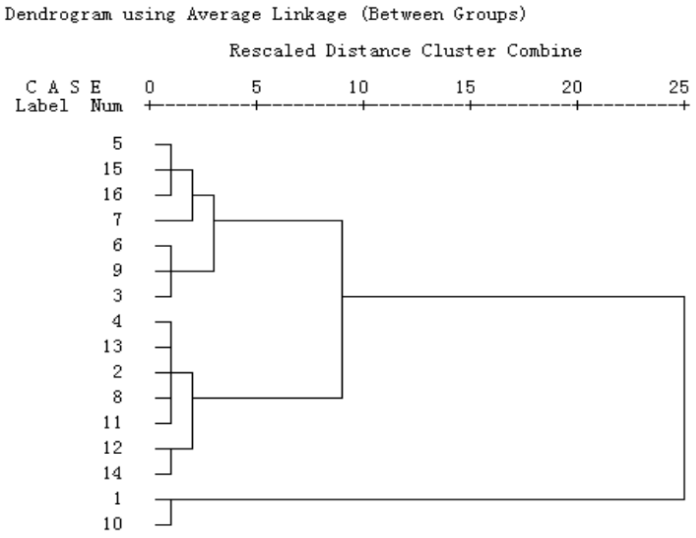


图 6-68 16 种饮料的分层聚类算法的执行结果

另一种是事先不用确定分多少类：分层聚类。在分层聚类或系统聚类（hierarchical

cluster）开始时，有多少点就是多少类。它第一步先把最近的两类（点）合并成一类，然后再把剩下的最近的两类合并成一类；这样下去，每次都少一类，直到最后只有一大类为止。显然，越是后来合并的类，距离就越远。同样以图 6-67 的饮料数据为例，执行分层聚类后，聚类的结果如图 6-68 的树结构。

(2) 课程设计目的

建立聚类分析的认识，掌握并编程实现常见的两类聚类算法，应用数据结构和聚类算法解决实际问题。

(3) 基本要求

① 编程实现 k-means cluster 算法。

② 编程实现分层聚类算法。

③ 上述两个算法中的数据结构和算法自行设计，两个算法中用到的距离定义可以随便选取或自行定义，上述两个算法的输入数据文件可以从网上下载，如使用 SPSS 的样例数据。

④ 熟悉 SPSS 统计分析软件。

(4) 实现提示

本设计的实现可以仿照 SPSS 软件。

45．树搜索之分支限界法

(1) 问题描述

一个8数码问问题是将如图6-69(*a*)所示的初始格局中的小格子中的数字经过若干步移动后（只能水平和垂直移动到一个空白处）形成如图6-69(*b*)所示的初始格局。

2	8	3
1		4
7	6	5

(*a*)

1	2	3
8		4
7	6	5

(*b*)

图6-69 8数码问题

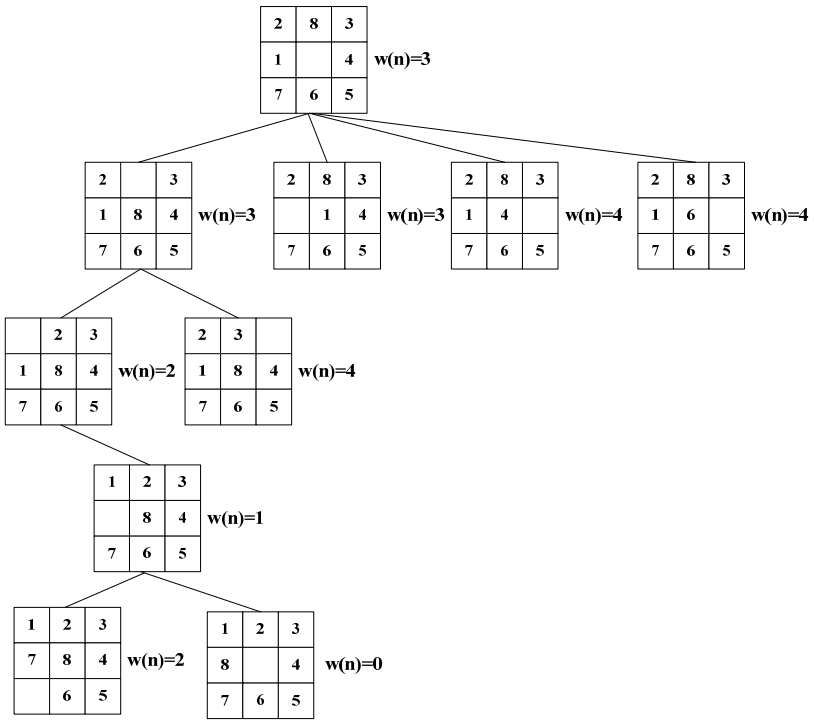


图6-70 用爬山法解决8数码问题

可用树搜索来解决8数码问题，如采用爬山法来解决该问题，爬山法的基本思想是根据一个评价准则贪心的选择一个节点进行所有可能的展开（枚举下一部的所有可能情况），这样处理直到找到解(到达目标格局)。对于上面的8数码问题，定义评价函数w(n)=错误放置的节点个数，显然贪心准则就是选取w(n)最小的节点展开。用爬山法解决上面的8数码问题的过程如图6-70所示。

有一点需要注意的是上面的爬山法给出的是一个解，并不一定是最优解，即移动次数最少的解，在很多情况下需要得到最优解，那么就需要将所有的节点都进行展开，即穷举搜索，此时爬山法没有任何帮助。接下来给出一种分支限界策略，该策略可以较好的解决这些优化问题，其基本思想是通过优化解的界限来修剪可行解从而减少搜索。如对于图6-71所示的最短路径搜索问题，可以用树搜索办法解决。

首先用爬山法对该问题建立一个可行解，结果如图6-72(*a*)-图6-72(*c*)所示。可以求出该可行解对应的路径长度为5，所以在展开其他节点时，如果发现了其他路径已经大于等于5时就不用进一步展开了，如图6-72(d)所示，只有小于该长度的节点采用必要进一步展开，即图6-72(*d*)中的阴影节点，从而修剪了许多分支，提高了算法效率，这就是分支限界法的基本思想。

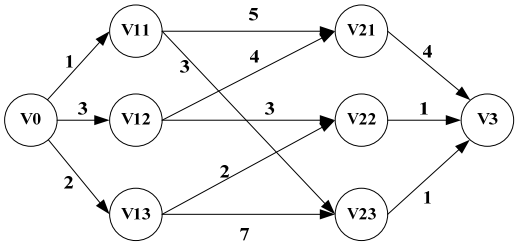


图 6-71 最短路径搜索问题

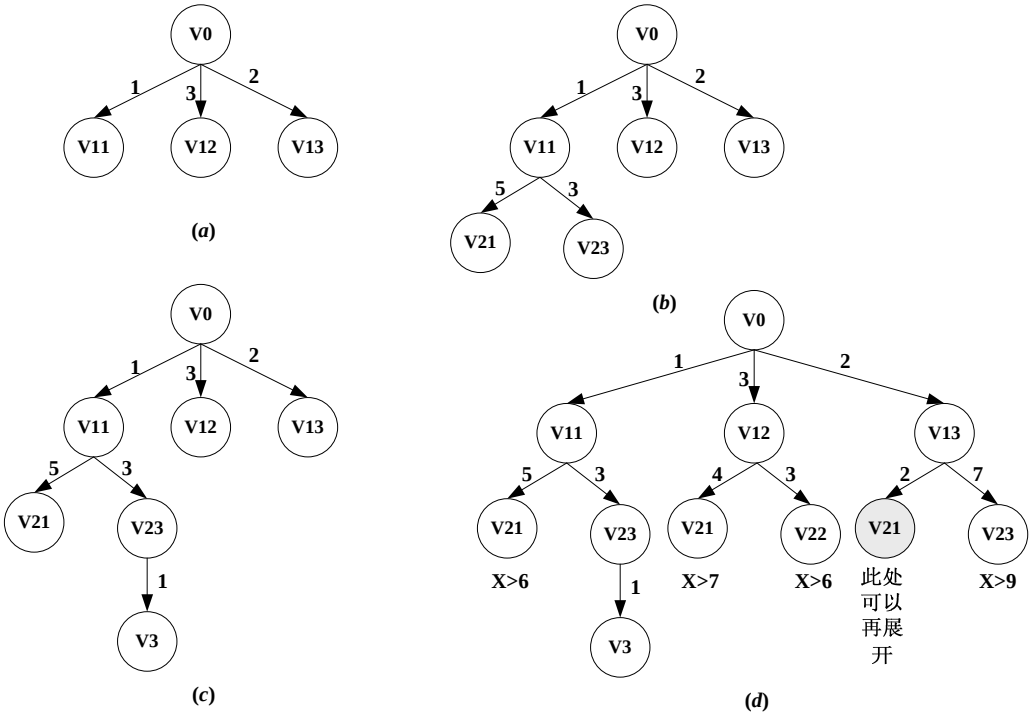


图 6-72 分支界限方法实例

可以用分支限界法解决旅行商问题(traveling saleperson problem 或 TSP)，旅行商问题是

一个在 n 个点的完全有向(或无向)加权图上寻找一个包含所有顶点的代价最小的回路，该问题已被证明是一个 NPC 问题，穷举搜索的时间复杂性是指数函数，可以用分支限界法来避免穷举搜索，减少搜索次数。用分支限界法解决 TSP 问题的方法由两个部分组成。

① 用一种方法划分解空间(从而形成一棵搜索树)。

② 对每一类解预测其下界(需建立一种预测方法)，用爬山法得到最优解的上界(即存在一个代价为次上界的解)，如果某个分支的下界大于该上界，则终止这个分支。

以下面的实例来说明这一过程，设一旅行商问题的代价矩阵(初始情况就是邻接矩阵)为。

	1	2	3	4	5	6	7
1	∞	3	93	13	33	9	57
2	4	∞	77	42	21	16	34
3	45	17	∞	36	16	28	25
4	39	90	80	∞	56	7	91
5	28	46	88	33	∞	25	57
6	3	88	18	46	92	∞	7
7	44	26	33	27	84	39	∞

① 分支限界法需将解分成两组：一组是包含某条特定的弧的解，另一组是不包含该弧的解。

② 按如下方法分析下界：首先注意到从代价矩阵中的任一行和任一列中减去一常数是不会改变最优解的。从该代价矩阵中的每一行分别减去 3，4，16，7，25，3，26。然后再从第 3，4 和 7 列中分别减去 7，1，4 使得代价矩阵中的每一行和每一列都至少包含一个 0。共减去的代价是 3+4+16+7+25+3+26+7+1+4=96，所以该旅行商问题解的代价的下界是 96。而经过这样处理后的代价矩阵为：

	1	2	3	4	5	6	7
1	∞	0	83	9	30	9	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	32	83	66	∞	49	0	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

③ 选取将解分成两组时选取的弧是 4-6，选取 4-6 有三个原因：一是该弧代价为 0，选取包含该弧的解的代价应该较小的代价(贪心法)；另一个原因是当该弧代价为 0 时，如果解不包含该弧时，必定包含一条从 4 出发的弧和一条进入 6 的弧，选取两条代价最小的这样的弧 4-1 和 5-6，其代价为 32 和 0，所以这样解的代价的下界应该为 96+32=128；第三个原因是所有的这样的权值为 0 的弧中，不引入这条弧引起的下界的增加中选 4-6 是最大的，如选弧 3-5 进行划分，则引入的下界的增加为 1+17=18。

④ 选取弧 4-6 后，由于选取了该弧，所以从代价矩阵中删除第 4 行和第 6 列(已经找到了一条边，其他边没有用了)，同时弧 6-4 也没有必要再包含了，即设置 C[6，4]= ∞。此时代价矩阵变为：

	1	2	3	4	5	7
1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5

5	3	21	56	7	∞	28
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

此时发现第 5 行不包含 0，所以对第 5 行减去 3，所以包含弧 4-6 的解的下届应该增 3。

⑤ 用这样的方法继续处理，直到得到一个解，如图 6-73 所示。可得出该解的代价，为 126，所所以所有的下届大于 126 的节点称为终止节点。

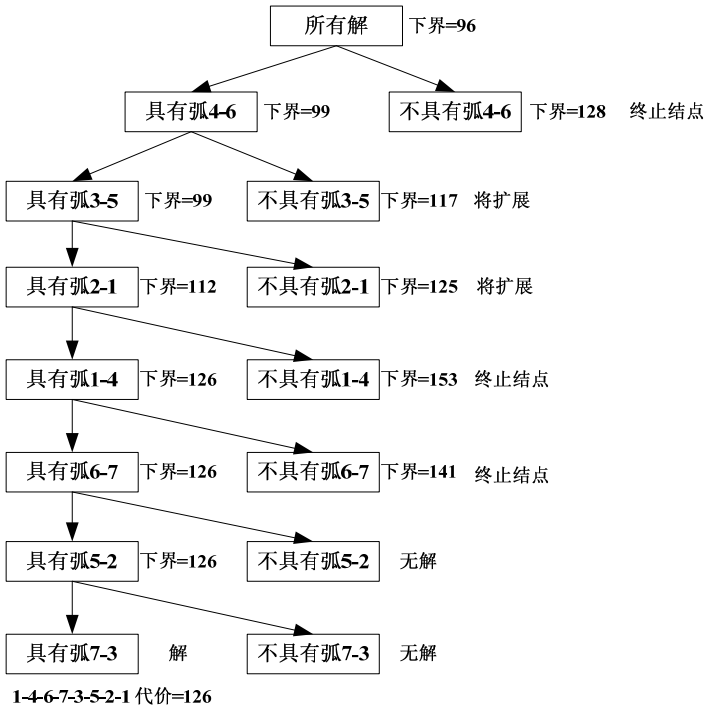


图 6-73 旅行商问题的分支限界解法过程

(2) 课程设计目的

对树搜索建立一定的认识，通过编程实现分支限界方法掌握该方法，并能用该方法解决一些实际的优化问题。

(3) 基本要求

① 针对 8 数码问题应用爬山法和分支限界法分别解决，体会二者的区别。

② 针对 TSP 问题，实现上述的分支限界法。

③ 产生一个 TSP 问题实例(多余 10 个节点)，该实例可以从平面上随机产生 n 个点，权值就是两个点之间的笛卡尔距离，用分支限界法解决该问题，并输出其中间过程。

(4) 实现提示

爬山的过程可用栈来实现。

46．编写 Lex 工具

(1) 问题描述

字母表 ∑ 上的一个语言 L 是 ∑*的子集，其中 $\sum^*=\cup_{n=1}^{\infty}\sum^n$，其中 $\sum^0=\{\varepsilon\}$，是一个空串，$\sum^n=\sum^{n-1}\sum$ 是一个连接操作，字符串连接是把两个字符串拼接在一起的符号串，如 x=dog，y=house，则连接操作 xy=doghouse。

正规表达式是描述单词（如高级程序设计语言中的单词）构成规则的常用方式，定义在字母表 ∑ 上的正规表达式满足下述规则：

① ε 是正规表达式，它表示的语言是{ε}。

② 如果 *a* 是 ∑ 上的符号，那么 *a* 是正规表达式，它表示的语言是{*a*}。

③ 如果 *r* 和 *s* 是正规表达式，分别表示语言 L(*r*)和 L(*s*)，则：

(*a*) (*r*)|(*s*)是正规表达式，表示的语言是 L(*r*) ∪L(*s*)。

(*b*) (*r*)(*s*)是正规表达式，表示的语言是 L(*r*)L(*s*)。

(*c*) (*r*)*是正规表达式，表示的语言是(L(*r*))*。//零个或多个实例

(*d*) $(r)^+$是正规表达式，表示的语言是$(L(r))^+$。//一个或多个实例

有了上述正规表达式，可以定义常见的高级程序语言中的单词，如下面的正规表达式

digit → 0 | 1 | …… | 9

digits → $digit^+$

optional_fraction → (.digits)?　//*r*?是一种缩写形式，表示 *r*|ε

optional_exponent → (E(+|-)?digits)?

num → digits optional_fraction optional_exponent

可以用来定义一个 num，num 可以是整数、实数或科学计数法的计数。

另一方面，在进行单词识别时，单词识别程序的基本思想就是将一个状态转化图变成相应的识别程序。如针对上面的 num 正规表达式中的实数和整数，可以画出如图 6-74 所示的状态转化图。

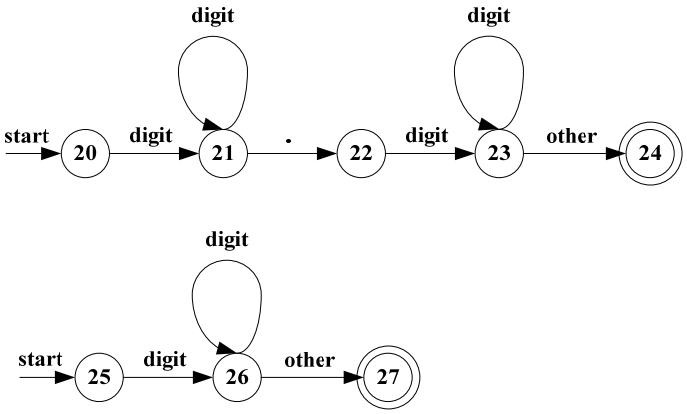


图 6-74 实数和整数的状态转化图

在上面的状态转化图中，圆圈是状态，里面的数字是状态，圆圈之间的有向连线是状态之间的迁移，该线上的字符是执行此次迁移的原因，用 start 标记的是开始状态，两个圆圈的状态是结束状态，显然，从开始状态到结束状态的迁移上面的字符依次列出就是该状态转化图对应的单词。根据上图可以写出如下的基本的词法分析器的 C 代码：

```
while(1)
{
    switch(state) {
      case 20:
            c = nextchar();
            if(isdigit(c)) state = 21;
            else fail();
            break;
        case 21:
            c = nextchar();
            if(isdigit(c)) state = 21;
            else if(c = = '.') state = 22;
            else fail();
```

```
            break;
        case 22:
            c = nextchar();
            if(isdigit(c)) state = 23;
            else fail();
            break;
        case 23:
            c = nextchar();
            if(isdigit(c)) state = 23;
            else state = 24;
            break;
        case 24:
            install_float_num();
            break;
        case 25:
            c = nextchar();
            if(isdigit(c)) state = 26;
            else fail();
            break;
        case 26:
            c = nextchar();
            if(isdigit(c)) state = 26;
            else state = 27;
            break;
        case 27:
            install_int_num();
            break;
    }
}
```

Lex 是一种已经广泛应用于这种高级语言词法分析的工具，Lex 处理的 Lex 语言 lex.l，Lex 语言的主体是正规表达式，Lex 程序根据这些表达式建立状态转化图，并产生词法分析 C 语言程序 lex.yy.c，用 C 语言编译器可以编译 lex.yy.c 程序形成词法分析程序，然后用该程序可以处理 C 语言源程序，输出单词序列。下图就是一个 lex.l 实例。

(2) 课程设计目的

对词法分析的基本原理深入了解，熟悉 Lex 工具，能编程实现状态转化图，对程序的自动生成建立一定的认识。

(3) 基本要求

① 编写一个 Lex 工具，能处理 lex.l 文件，lex.l 文件实例自己给出。

② 要求产生的 lex.yy.c 在处理高级语言程序的单词时，单词的输出格式为：

<IF，—>

<ID, a>

<NUMBER, 10, INT>

<GT, —>

……

```
%{   /*符号常量定义
     LT, LE, EQ, NE, GT, GE,
     IF, THEN, ELSE, ID, NUMBER, RELOP */
%}

/* 正规定义 */
delim          [ \t\n]
ws             {delim}+
letter         [A-Za-z]
digit          [0-9]
id             {letter}({letter}|{digit})*
number         ({digit})+(.({digit})+)?(E(+|-)?({digit})+)?

%%    //定义相应的动作和返回值
{ws}           { }
if             {return(IF);}
then           {return(THEN);}
{id}           {return(ID);}
{number}       {return(NUMBER);return(TYPE);}
"<="           {return(LE);}
">"            {return(GT);}
……
```

图 6-75 一个 Lex 输入实例

(4) 实现提示

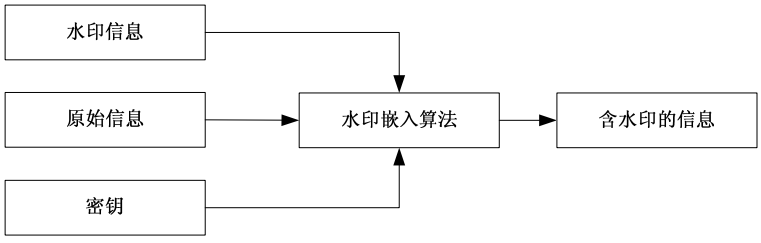
关键在于思考对状态转化图的处理。

47．二值图像数字水印技术的实践

(1) 问题描述

随着计算机通信技术的迅速发展，多媒体存储和传输技术的进步使存储和传输数字化信息成为可能，然而，这也使盗版者能以低廉的成本复制及传播未经授权的数字产品内容。数字水印就是近年来为实现数字产权保护而产生的技术。数字水印是永久镶嵌在其它数据（宿主数据）中具有可鉴别性的数字信号或模式，而且并不影响宿主数据的可用性。作为数字水印技术基本上应当满足下面几个方面的要求：(1)安全性：数字水印的信息应是安全的，难以篡改或伪造；(2)隐蔽性：数字水印应是不可知觉的，而且应不影响被保护数据的正常使用；(3)稳健性：数字水印必须难以被除去，如果只知道部分数字水印信息，那么试图除去或破坏数字水印将导致严重降质或不可用。

数字水印技术是通过一定的算法将一些标志性信息直接嵌到多媒体内容中，水印的嵌入和提取方法如图 6-76 所示：



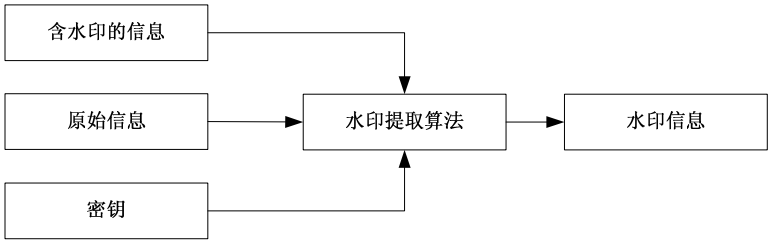


图 6-76 数字水印的嵌入和提取

数字水印的实现可以分为空间域数字水印和变换域数字水印两大类，较早的数字水印算法都是空间域上的，通过改变某些象素的灰度将要隐蔽的信息嵌入其中，将数字水印直接加载在数据上。空间域方法具有算法简单、速度快、容易实现的优点。特别是它几乎可以无损的恢复载体图象和水印信息，可细分为如下几种方法：最低有效位法，该方法就是利用原始数据的最低几位来隐蔽信息的，具体取多少位以人的听觉或视觉系统无法察觉为原则。Patchwork 方法及纹理映射编码方法，该方法是通过任意选择 N 对图象点，增加一点亮度的同时，降低相应另一点的亮度值来加载数字水印。文档结构微调方法，在通用文档图象（postcript）中隐藏特定二进制信息的技术，主要是通过垂直移动行距，水平调整字距，调整文字特性等来完成编码。

基于变换域的技术可以嵌入大量比特的数据而不会导致不可察觉的缺陷，往往通过改变频域的一些系数的值，采用类似扩频图象的技术来隐藏数字水印信息。这类技术一般基于常用的图象变换，如离散余弦变换（DCT）、小波变换（WT）、付氏变换（FT 或 FFT）以及哈达马变换（Hadamard Traansform）等。频域方法具有如下优点：（1）在频域中嵌入的水印的信号能量可以分布到所有的象素上，有利于保证水印的不可见性；（2）在频域中可以利用人类视觉系统的某些特性，可以更方便、更有效的进行水印的编码。不过，频域变换和反变换过程中是有损的，同时其运算量也很大，对一些精确或快速应用的场合不太适合。目前常用的方法有平面隐藏法和基于 DCT 或 DFT 的系数隐藏法。其中基于分块的 DCT 是最常用的变 换之一，现在所采用的静止图像压缩标准 JPEG 也是基于分块 DCT 的。

下面概要描述一种简单的二值图像的数字水印算法，本设计的基本目标就是该水印算法的编程实现。

二值图像又称为单色图像或黑白图像，一般用 1 或 0 表示黑色或白色像素的颜色值。二值图像数字水印的常见方法是：一、使用一个特定图像区域中的黑色像素个数来编码水印信息。把一个二值图像分解成一系列图像块 B_i，分别令 $P_0(B_i)$和 $P_1(B_i)$代表黑白像素在该块图像中所占的百分比。基本做法是判断如果 $P_1(B_i)>50\%$，则在该块中隐藏一个 1，如果 $P_0(B_i)>50\%$，则在该块中隐藏一个 0。隐藏是需要修改图像块中的一些像素的颜色。该修改是在那些邻近像素有相反的颜色的像素中进行的。二、将二值图像进行分块，使用一个与图像块同样大小的密钥二值图像块，该密钥图像块和每一格图像块按像素进行“与”运算，根据“与”的结果确定是否在该块中隐藏数据，并隐藏咋样的数据。但这两种简单处理方法会导致图像质量下降，如会在一个全白的图像块中插入一个黑点，如也应避免在图像中的细线(一个像素宽)、直线边的中间像素、孤立像素等区域隐藏像素。所以二值图像的数字水印嵌入算法的关键是隐藏点的选取。

二值图像的数据隐藏应该着眼于图像黑(或白)色区域的边界上。下面给出一种筛选隐藏点的方法，该方法的基本思想是在满足下面条件的位置上不隐藏数据：

该像素既是其所在区域的左边界，又是其右边界（实际上是一条直线）；

该像素既是其所在区域的上边界，又是其下边界；

该像素只是左边界、右边界、上边界、下边界 4 种边界情况中的一种；

该像素的周围 8 个像素中与该像素同色的所有像素都是既是左边界或右边界，同时又

是其上边界和下边界(实际上是一个小点)。此处给出一个算法，设二值图像为 I，其像素矩阵为 $A_{M\times N}$，数字水印为 W，是一个长度为 L 的二进制比特流 $W=\{w_1, w_2, \ldots, w_L\}$。设 A(i:j, s:t)为矩阵 A 的从第 i 行到第 j 行，从第 s 列到第 t 列的子矩阵。数据隐藏算法如下：

第 1 步：计算图像边界。

$A_L = A(0: (M-1), 0: (N-2)) - A(0: (M-1), 1: (N-1))$

$A_R = A(0: (M-1), 1: (N-1)) - A(0: (M-1), 0: (N-2))$

$A_T = A(0: (M-2), 0: (N-1)) - A(1: (M-1), 0: (N-1))$

$A_B = A(1: (M-1), 0: (N-1)) - A(0: (M-2), 0: (N-1))$

将 A_L，A_R，A_T，A_B 中值为-1 的元素置为 0，得到 B_L，B_R，B_T，B_B。则 B_L，B_R，B_T，B_B 为图像 I 的左、右、上、下边界矩阵。

第 2 步：选择隐藏点。

$B = B_L + B_R + B_T + B_B$

$B_{LR} = B_L + B_R$

$B_{TB} = B_T + B_B$

矩阵 B_{LR} 和 B_{TB} 的元素值可能为 0，1，2。值为 2 表示该像素同时为左右或上下边界。矩阵 B 中元素的可能值为 0，1，2，3，4，值为 3 和 4 的一定是左右或上下边界，值为 2 的像素可能会同时为左右边界或上下边界，此时需根据 B_{LR} 和 B_{TB} 的值加以区分。

计算 $B' = (b_{ij}')$，其中 $b_{ij}' = 0$，如果 $b_{ij}=3,4$；$b_{ij}' = 0$，如果 $b_{ij}=2$ 且 $b_{ij}^{LR}=2$ 或 $b_{ij}^{TB}=2$；$b_{ij}' = 0$，如果 i=0 或 j=0；$b_{ij}' = 1$，如果 $b_{ij}=2$ 且 $b_{ij}^{LR}\neq 2$ 或 $b_{ij}^{TB}\neq 2$。

再计算 $B'' = (b_{ij}'')$，其中 $b_{ij}''=0$，如果 $b_{ij}'=0$；$b_{ij}''=0$，如果 $b_{ij}'=1$ 且 $\sum_{u=i-1}^{i+1}\sum_{v=j-1}^{j+1}a_{uv}=\sum_{u=i-1}^{i+1}\sum_{v=j-1}^{j+1}b'_{uv}$；$b_{ij}''=1$，其他情况。

第 3 步：水印数据处理。为确保水印数据的安全，输入一个密钥 K，根据该密钥产生一个长度为 L 的伪随机比特流 $R=\{r_1,r_2,\ldots,r_L\}$（R 的产生方法是应用一个高级语言提供的伪随机函数，如 C 语言的 rand 函数，将 K 作为该函数的种子产生随机数，可产生 0-1 的随机数，如果该数小于等于 0.5 就输出 0，否则输出 1。也可以用其他方法获得。）将 W 与 R 按位异或就得到了 $W'=\{r_1',r_2',\ldots,r_L'\}$。

第 4 步：数据隐藏。设计一个遍历图像矩阵 A 诸元素的遍历算法，根据 B″和 W′计算获得 A′。$A' = (a_{i,j}')$，$a_{i,j}' = a_{i,j}$，如果 $b_{i,j}'' = 0$；$a_{i,j}' = w_l'$，如果 $b_{i,j}'' = 1$，其中 l=M×i+j。A′对应的就是含有水印信息的新图像 I_W。

第 5 步：水印数据的提取。第 1 步，第 2 步同上，在 A(图像 I)上可以计算出矩阵 B″，此时可以获得隐藏在 A′(图像 I_W)中的数据 S，注意需按照同样的遍历方法，然后根据 K 获得同样的伪随机序列 R，R 和 S 按位异或后就是数字水印。

(2) 课程设计目的

对数字水印技术建立一定的认识，能建立位矩阵、位向量等 ADT，并能用这些 ADT 完成上述二值图像数字水印的嵌入和抽取。

(3) 基本要求

① 设计并实现位矩阵、位向量等 ADT，要求支持+、-、与、或、异或等基本操作。

② 针对二值图像实现上述水印嵌入和提取算法。

③ 针对二值图像实现实现另一个水印基本做法：判断如果 $P_1(B_i)>50\%$，则在该块中隐藏一个 1，如果 $P_0(B_i)>50\%$，则在该块中隐藏一个 0。也可以自行设计一个嵌入方式。

④ 针对一幅二值图像(要求是 BMP 文件格式)，分别用上面的两种方法嵌入水印，查看嵌入水印后的图像 I_W 的区别，水印信息可以设定为一段文字。

(4) 实现提示

可以查阅关于数字水印方面的相关资料。

48．基于图像插值的图像水印算法实践

(1) 问题描述

数字水印技术是近年来为实现数字产权保护而产生的技术，能有效的控制未经授权数字产品内容的复制及传播。数字水印是永久镶嵌在其它数据（宿主数据）中具有可鉴别性的数字信号或模式，而且并不影响宿主数据的可用性。上题给出的是二值图像的数字水印嵌入技术，由于这种方法的使用面较窄，不能适用于其他类型的图象。本题目给出的是一种能用在常见的灰度图像上的数字水印嵌入算法，一种被称为基于图像插值的图像水印嵌入算法，本设计的基本目标就是该水印算法的编程实现。

设 OC 和 OS 分别是原始载体图像和原始秘密图像，为描述的方便，此处设定 OC={oc(i, j)，1≤i,j≤N}，其中 oc(i, j)为图像 OC 第 i 行和第 j 列的像素的灰度值，OS={os(i, j)，1≤i,j≤N′}，其中 os(i, j)为图像 OS 第 i 行和第 j 列的像素的灰度值，要求 N′=N/2。基于图像插值的图像水印算法分为如下几步进行。

第 1 步：随机置换。为了确保秘密信息不被泄漏，需要首先对原始秘密图像进行伪随机置换。该随机置换是在密钥 K(*l*)的控制下进行的，K(*l*)定义为从 1 到 N'^2 的随机置换。K(*l*)这个随机置换是在用户输入的嵌入(提取)密钥 key 的基础上产生的，此处给出一种简单的产生方法：根据输入的密钥 key，可以产生一个长度为 N'^2 的伪随机数序列 R={r_1,r_2,…,r_L…}，每个 r_i 满足 $1\le r_i\le N'^2$，其中的 R 可以直接应用一个高级语言提供的伪随机函数，如 C 语言的 rand 函数，将 key 作为该函数的种子产生随机数，可产生 1 到 N'^2 的随机数。初始情况设 K(i)=i，最终的 K(*l*)可以通过如下方式置换获得，for i=1 to N'^2，计算 j = R[i]，交换 K(i)和 K(j)。

在 K(*l*)这个从 1 到 N'^2 的随机置换上(实际上 K 就是一个大小为 N'^2 的数组)，对秘密图像的随机置换定义为：os′(i, j) = os(i′+1, j′+1)，i′ = K[(i-1)×N′ + j] mod N′，j′ = K[(i-1)×N′ + j] /N′。

第 2 步：将图像 OC 和 OS′分别分为大小为 8×8 和 4×4 的图像块，分别记为 $BOC_{m,n}$ 和 $BOS'_{m,n}$，其中 1≤m, n≤M，M=N/8。

第 3 步：对于每一对图像块 $BOC_{m,n}$ 和 $BOS'_{m,n}$，假设像素 cp_1^t，cp_2^t，cp_3^t，cp_4^t(t=0,1,2,3)的值为 0。使用图像插值算法计算这些像素的新值，记为 Vcp_1^t, Vcp_2^t, Vcp_3^t, Vcp_4^t (t=0,1,2,3)。这些隐藏点的选取如图 6-77 所示。

图像插值的主要目的是确定图像中的一些未知像素值，只用于受损图像的恢复、图像的方法等处理操作。用于数字水印加载的插值方法可将图像插值看作是数学上的边界问题，所有已知的像素均作为边界条件加以利用。计算所得的位置像素值受到拉普拉斯方程的限制：$(\partial^2/\partial x^2 + \partial^2/\partial y^2)\,\phi(x, y) = 0$，即是该方程的解。该方程的唯一解满足最大最小法则：在插值区域不会产生最大值或最小值。可使用下面的公式来计算该方程唯一解的近似值：$4I_{m,n} - I_{m-1,n} - I_{m+1,n} - I_{m,n-1} - I_{m,n+1} = 0$，其中 $I_{m,n} = \phi(m\Delta, n\Delta)$，Δ 为步长，m 和 n 为坐标值。应用该公式计算所有的未知点，其中 $I_{m,n}$ 必须是未知点，其他各点可以为未知点，也可以是已知点。设 U 为所有需要计算的未知点的坐标值，则可以用连续迭代来近似计算这些未知点：$I_{m,n}^k = I_{m,n}^{k-1} - \omega(4I_{m,n}^{k-1} - I_{m-1,n}^{k} - I_{m+1,n}^{k-1} - I_{m,n-1}^{k} - I_{m,n+1}^{k-1})/4$，$\forall(m,n)\in U$。其中 $I_{m,n}^k$ 是第 k 次迭代之后得到的值，ω 是一个满足 $1<\omega<2$ 的迭代参数，并要求所有的未知点 $I_{m,n}$ 是按照字典顺序处理的，这样才能逐步迭代出结果。由于所有的未知点要受到周围各点的限制，所以上面的迭代公式应改为：$I_{m,n}^k = I_{m,n}^{k-1} - \omega(4I_{m,n}^{k-1} - I_{\max(m-1,0),n}^{k} - I_{\min(m+1,mmax),n}^{k-1} - I_{m,\max(n-1,0)}^{k} - I_{m,\min(n+1,nmax)}^{k-1})/4$，$\forall(m,n)\in U$，该公式记为公式(1)。

因此图像的插值算法为：

① 确定边界条件。即选定哪些像素未知，哪些像素为已知点。

② 给未知像素编号。即选定一个遍历所有未知像素的顺序。

③ 初始化未知像素的值。

④ 用公式(1)计算未知像素的值。

⑤ 重复步骤④直到 $\max_{(m,n)\in U} \| I_{m,n}^{k} - I_{m,n}^{k-1} \| \le T$，T 是设定的阈值。

⑥ 将最后得到的像素值量化为整数。

第 4 步：使用下面的公式里用像素 sp_1^t，sp_2^t，sp_3^t，sp_4^t(t=0,1,2,3)的值来计算像素 cp_1^t，cp_2^t，cp_3^t，cp_4^t(t=0,1,2,3)的新值。$Vcp_t^t = Vcp_s^t (1+\partial(s, t)Vsp_s^t)$。其中 Vcp_s^t 和 Vsp_s^t 为像素 cp_s^t 和 sp_s^t 的像素值，系数∂(s, t)(也是一个控制算法的参数)用来调整原图象和新图像的差异。得到了新的图像块 $BOC_{m,n}'$。

第 5 步：将所有的图像块 $BOC_{m,n}'$组合成一个新的图象，NC，它就是隐藏秘密图像的载体图象。

sp_1^0	sp_1^1	sp_1^2	sp_1^3
sp_2^0	sp_2^1	sp_2^2	sp_2^3
sp_3^0	sp_3^1	sp_3^2	sp_3^3
sp_4^0	sp_4^1	sp_4^2	sp_4^3

(*a*) $BOS_{m,n}'$的图像块

	cp_1^0	cp_1^1			cp_2^0	cp_2^1	
	cp_1^2	cp_1^3			cp_2^2	cp_2^3	
	cp_3^0	cp_3^1			cp_4^0	cp_4^1	
	cp_3^2	cp_3^3			cp_4^2	cp_4^3	

(*b*) $BOC_{m,n}$ 的图像块

图 6-77 隐藏点筛选算法

接下来看如何将隐藏图像恢复出来，实际上隐藏图像就是嵌入图像的逆过程。设 VI 是收到的载体图像，恢复隐藏图像的处理过程是：

第 1 步：将 VI 分割成大小 8×8 的图像块 $BVI_{m,n}$，$1 \le m,n \le M$，M=N/8。设像素 vp′，是与隐藏算法中像素 cp′相同位置的像素，记其灰度值为 Vvp′。使用图像插值算法计算其新值，记为 $NVvp_s^t$(s=1,2,3,4；t=0,1,2,3)。

第 2 步：使用下式计算隐藏的秘密数据 Nsp_s^t。

$Nsp_s^t = (Vvp_s^t/NVvp_s^t - 1) / \partial(s, t)$。将所得到的图像块重组为一幅新图像，NS′，该图象是秘密图像随机置换后的结果。

第 3 步：使用密钥 K(*l*)可以计算 NS(i′+1, j′+1) = NS′(i, j)，就得到了隐藏的水印图像。

(2) 课程设计目的

对灰度图像上的数字水印算法建立认识，对图象插值算法的思想、原理和实现建立认识，能编程实现基于图象插值的灰度图像数字水印嵌入和提取算法。

(3) 基本要求

① 编程实现上述基于图像插值的数字水印嵌入和提取算法。

② 针对一个 BMP 灰度图像，应用上述算法实现在其中嵌入另一个 BMP 灰度图象。

(4) 实现提示

拉普拉斯方程可以查阅相关资料，关于公式(1)的正确性可以查阅线性方程组的迭代近似求解。

49．剪枝搜索策略解决 1-中心问题

(1) 问题描述

1 中心问题(1-center problem)定义为：给定 n 个平面点的集合，问题要求找到一个覆盖所有 n 个点的最小圆，如图 6-78 所示。实际上该问题的关键就是圆心的确定，因为一旦确定圆心后就可以找到离该圆心最远的点，从而可以确定半径，就可以画出该圆了。用穷举法解决该问题费时费力，所以可以用其他的优化算法来解决该问题。剪枝搜索策略(prune-and-search)世界解决该问题的一个不错的方法。下面介绍一下剪枝搜索策略的基本思想。

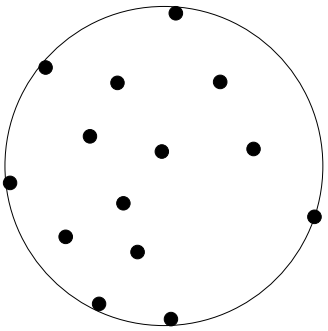


图 6-77 1-中心问题实例

剪枝搜索策略通常由几次迭代组成，每次迭代中都会剪除输入数据的一部分(如剪掉的部分占总体的比例为 f)，当然必须要求这些输入数据不影响问题的最终解，然后递归调用同样的算法来处理剩余的数据，从而递归解决该问题。由于每次迭代会剪除输入数据，p 次迭代后输入数据的规模就会变得很小，可以在常数时间内解决。设每次迭代执行的剪枝时间为 $O(n^k)$，此时剪枝搜索算法的最坏时间复杂性为：

$T(n) = T((1-f)n)+O(n^k) \le T((1-f)n) + cn^k \le T((1-f)^2n) + cn^k + c(1-f)^k n^k$

$\le c' + cn^k + c(1-f)^k n^k + c(1-f)^{2k} n^k + \ldots + c(1-f)^{pk} n^k$

$= c' + cn^k[\ 1+ (1-f)^k + (1-f)^{2k} + \ldots + (1-f)^{pk}\ ]$

$n \to \infty$时，$T(n) = O(n^k)$。这表明剪枝搜索算法的时间复杂度和每次迭代(即剪枝)的时间复杂度是一样的，而通常就是线性时间，即 $T(n) = O(n)$，因此剪枝搜索算法是一个效率非常高的优化搜索算法。

现在来看如何应用剪枝搜索策略解决 1-中心问题，假设现在有如图 6-78 所示的点分布，并假定已知最优解圆心分布的区域(这是剪枝搜索的一个关键点)——图中的阴影区域，其中 L_{12} 和 L_{34} 分别是线段 p_1p_2 和线段 p_3p_4 的垂直平分线。因为 L_{12} 和阴影区域没有相交，所以 p_1 和 p_2 两点中必有其一离最优解圆心近，而另一点远，显然这一近的点可以被删掉，因为它不影响结果，在本图中就是 p_1。

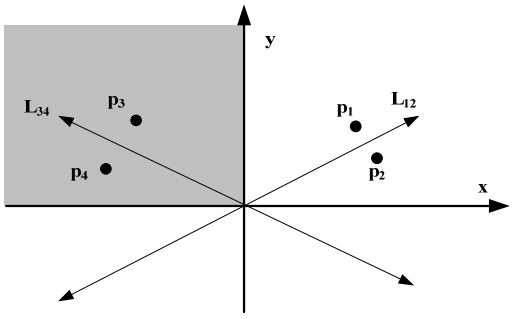


图 6-78 1-中心问题可能的一种点剪除的情况

由上面给出的简单实例可以看出，剪枝算法的关键在于知道最优解的圆心位于哪一部分，哪些点比其他点离圆心近，从而将这些点剪除。

此处首先给出一个受约束的 1-中心问题算法，其中的约束是指圆心必须位于一条直线

上，不是一般性，假定该直线为 y=y′，该算法将被通用 1-中心问题解法调用。该受约束的 1-中心问题算法如下：

算法：解决受约束的 1-中心问题算法

输入：n 个点和一条直线 y= y′

输出：在直线 y= y′上的满足最小覆盖的圆心

第 1 步：如果 n 小于等于 2，用几何方法直接解决；

第 2 步：构造偶数个不相交(交集为空)点对(p_1, p_2)，(p_3, p_4)，…，(p_{n-1}, p_n)。如果有奇数个点，则令最后点对(p_n, p_1)。

第 3 步：对每个点对(p_i, p_{i+1})，求出在直线 y= y′上的点 $x_{i,i+1}$，使得 $d(p_i, x_{i,i+1}) = d(p_{i+1}, x_{i,i+1})$。

第 4 步：求出这$\lceil n/2 \rfloor$个 $x_{i,i+1}$ 的中点，记为 x_m。

第 5 步：对所有的 i，计算 p_i 和 x_m 的距离。另 p_j 为离 x_m 的最远点，x_j 表示 p_j 在直线 y= y′上的投影。如果 x_j 在 x_m 的左方，则最优解 x*必位于 x_m 的左方；反之，如果 x_j 在 x_m 的右方，则最优解 x*必位于 x_m 的右方。

第 6 步：如果 $x^* < x_m$（图 6-79 就是这种情况）

对每个 $x_{i,i+1} > x_m$，如果 p_i 比 p_{i+1} 离 x_m 近，那么剪去点 p_i，否则剪去点 p_{i+1}。

如果 $x^* > x_m$（图 6-79 就是这种情况）

对每个 $x_{i,i+1} < x_m$，如果 p_i 比 p_{i+1} 离 x_m 近，那么剪去点 p_i，否则剪去点 p_{i+1}。

第 7 步：转向第 1 步。

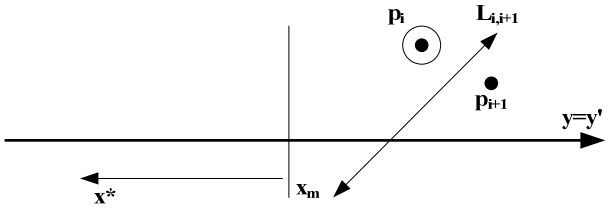
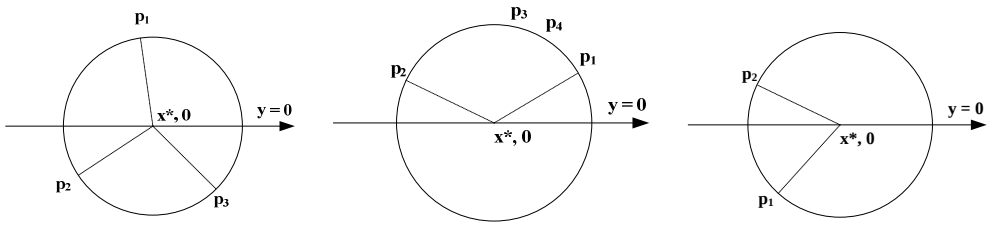


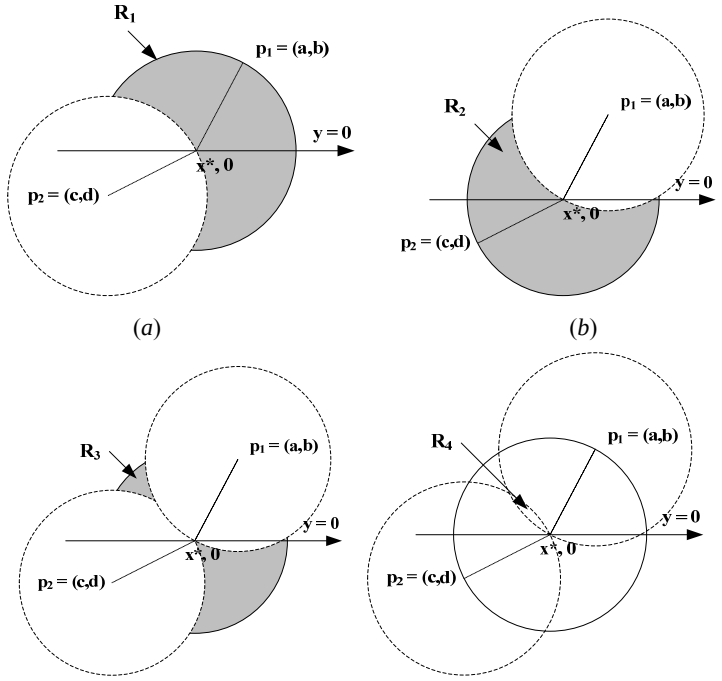
图 6-79 有约束 1-中心问题的点剪除

在此基础上可以考虑这样一个问题：如果已经求得在直线 y=0 上的有约束 1-中心问题的解 x*，现假定(x_s, y_s)为包括所有点的最优解圆心（无约束），则可以根据 x*的信息求出一些关于 x_s 和 y_s 的信息，即可以确定 $y_s>0$，$y_s<0$ 或 $y_s=0$。用同样的方法可以确定 $x_s>0$，$x_s<0$ 或 $x_s=0$。确定的基本方法是：令 I 表示距离点$(x^*, 0)$最远点组成的集合，则此时有两种情况。情况 1：I 包含一个点，记为 p。此时 p 的 x 坐标必定等于 x*，否则 x*可以沿着 y=0 向 p 移动，此时最优解就不是 x*，而是移动后的点。所以此时的 y_s 必定在靠近 p 的一侧，这是由于 p 是唯一的最远点，所以必有 y_s 与 p 的 y 坐标正负号相同。情况 2：I 中包含多个点，显然这些点一定在同一个圆上，且该圆的圆心为$(x^*, 0)$，所以可以找到覆盖 I 中所有点的最小弧，令该弧的两个端点为 p_1 和 p_2，如果该弧的度数大于等于 180 度，如图 6-80 中的(*a*)，则 $y_s=0$。现说明原因，由于包含某点集的最小圆一定是由这些点中的两个或三个确定的，显然，两个点确定最小圆，这两个点的连线就是该圆的直径，三个点确定最小圆，当且仅当这三个点形成的是锐角三角形，显然这正是 p_1p_2 弧大于等于 180 度的结果，也就是说此时的最小圆(圆心为$(x^*, 0)$)已经是最优的，进而可以推断 $y_s=0$。如果 p_1p_2 弧小于 180 度，如图 6-80 中的(*b*)，此时必定有 p_1 和 p_2 的 x 坐标必定在 x*的两侧，否则如图 6-80(*c*)所示，则 x*可向 p_1 和 p_2 方向移动，约束 1-中心问题的最优解就不是 x*了。因此可以假定 $p_1=(a, b)$，$p_2=(c, d)$，不是一般性假定 $a>x^*$，$b>0$，$c<x^*$，$d<0$。可以画出如图 6-81 所示的四种情况，显然最优解(x_s, y_s)一定在区域 R_4 中，因为在区域 R_1、R_2、R_3 中都有最优解圆心和 p_1 或 p_2 的距离大于约束最优解的半径 r，这是不应该的。所以 y_s 与$(b+d)/2=(y_1+y_2)/2$ 同号。



(*a*) (*b*) (*c*)

图 6-80 I 中包含多于 1 点的情况



(*a*) (*b*)

(*c*) (*d*)

图 6-81 度数小于 180 度时 y_s 的方向

因此就可以给出下面的程序来确定 y_s 的方向。

程序：确定 y_s 的方向

输入：点集合 S，直线 y=y*，(x*, y*)是对于 S，在 y=y*上的有约束的 1-中心问题解。

输出：$y_s>y^*$，$y_s<y^*$，或 $y_s=y^*$。其中(x_s, y_s)为 S 的 1-中心问题解。

第 1 步：找到距离(x*,y*)最远点的集合 I。

第 2 步：情况 1。I 中只包含 1 个点 $p=(x_p, y_p)$。

如果 $y_p>y^*$，输出“$y_s>y^*$”并退出；

如果 $y_p<y^*$，输出“$y_s<y^*$”并退出。

情况 2。I 中包含多于 1 个点。在 I 中找到形成覆盖 I 中所有点的最小弧的两个端点 $p_1=(x_1, y_1)$和 $p_2=(x_2, y_2)$。

情况 2.1。p_1 和 p_2 形成的弧大于等于 180 度，输入 $y_s = y^*$并退出。

情况 2.2。p_1 和 p_2 形成的弧小于 180 度，令 $y_c= (y_1+y_2)/2$，

如果 $y_c>y^*$，输出“$y_s>y^*$”并退出；

如果 $y_c<y^*$，输出“$y_s<y^*$”并退出。

现在可以给出在上面两个算法的基础上如何求解无约束的 1-中心问题：对于任意两个点对(p_1, p_2)和(p_3, p_4)，分别画出线段 p_1p_2 和 p_3p_4 的垂直平分线 L_{12} 和 L_{34}，令 $_{12}$ 和 L_{34} 的交点

为 p，旋转 x 轴使 L_{34} 的斜率为负，使 L_{12} 的斜率为正，将坐标原点移动至 p，形成如图 6-78 所示的情况，此时应用约束 1-中心问题的求解算法得到约束为 y=0 的解，然后调用确定 y_s 的方向来确定 y_s 的方向，对于图 6-78 有 y_s 向上。在应用约束 1-中心问题的求解算法得到约束为 x=0 的解，然后调用确定 x_s 的方向来确定 x_s 的方向，对于图 6-78 有 x_s 向左。所以有 1-中心问题的解在图 6-78 中的阴影上，可以剪除点 p_1，缩小输入。

总结上述思想可以得到解决 1-中心问题的剪枝搜索算法，算法如下：

算法：1-中心问题的剪枝搜索算法

输入：n 个点的集合 $S=\{p_1, p_2, \dots, p_n\}$。

输出：包括 S 中所有点的最小圆。

第 1 步：如果 S 包含不多于 16 个点，用穷举方法解决。

第 2 步：形成不相交(集合不相交)点对，(p_1, p_2)，(p_3, p_4)，…，(p_{n-1}, p_n)。对每个点对(p_i, p_{i+1})找到线段 p_ip_{i+1} 的垂直平分线，记为 $L_{i/2}$，计算其斜率，将 L_k 的斜率记为 s_k。

第 3 步：计算 s_k 的中点 s_m，旋转坐标系使 x 坐标轴与 $y=s_mx$ 重合，令 L_k 中斜率为正(负)组成的集合为 $I^+(I^-)$。

第 4 步：构建不相交(集合不相交)的直线对(L_{i+}, L_{i-})，$L_{i+}\in I^+$，$L_{i-}\in I^-$，找到每对直线的交点，记为(a_i, b_i)。

第 5 步：找出 b_i 的中点，记为 y*。对 S 应用有约束的 1-中心问题子程序求得其在 y=y* 上的圆心，令其为(x′, y*)。

第 6 步：以 S 和(x′, y*)作为参数，应用求 y_s 方向程序。

如果 $y_s = y^*$，输出(x′, y*)为最优解，退出。

否则输出 $y_s > y^*$或 $y_s < y^*$。

第 7 步：找出 a_i 的中点，记为 x*。对 S 应用有约束的 1-中心问题子程序求得其在 x=x* 上的圆心，令其为(x*, y′)。

第 8 步：以 S 和(x*, y′)作为参数，应用求 x_s 方向程序。

如果 $x_s = x^*$，输出(x*, y′)为最优解，退出。

否则输出 $x_s > x^*$或 $x_s < x^*$。

第 9 步：情况 1。$x_s > x^*$且 $y_s > y^*$

找到所有 $a_i < x^*$且 $b_i < y^*$的(a_i, b_i)，令(a_i, b_i)为(L_{i+}, L_{i-})的交点，L_{i-}为 p_j 和 p_k 的垂直平分线。如果 $p_j(p_k)$比 $p_k(p_j)$离(x*, y*)更近，则剪除 $p_j(p_k)$。

情况 2。$x_s < x^*$且 $y_s > y^*$

找到所有 $a_i > x^*$且 $b_i < y^*$的(a_i, b_i)，令(a_i, b_i)为(L_{i+}, L_{i-})的交点，L_{i+}为 p_j 和 p_k 的垂直平分线。如果 $p_j(p_k)$比 $p_k(p_j)$离(x*, y*)更近，则剪除 $p_j(p_k)$。

情况 3。$x_s < x^*$且 $y_s < y^*$

找到所有 $a_i > x^*$且 $b_i > y^*$的(a_i, b_i)，令(a_i, b_i)为(L_{i+}, L_{i-})的交点，L_{i-}为 p_j 和 p_k 的垂直平分线。如果 $p_j(p_k)$比 $p_k(p_j)$离(x*, y*)更近，则剪除 $p_j(p_k)$。

情况 4。$x_s > x^*$且 $y_s < y^*$

找到所有 $a_i > x^*$且 $b_i < y^*$的(a_i, b_i)，令(a_i, b_i)为(L_{i+}, L_{i-})的交点，L_{i+}为 p_j 和 p_k 的垂直平分线。如果 $p_j(p_k)$比 $p_k(p_j)$离(x*, y*)更近，则剪除 $p_j(p_k)$。

第 10 步：设 S 为剩余点，转向步骤 1。

(2) 课程设计目的

对剪枝搜索算法建立一定的认识，编程实现上述算法，对计算几何和 1-中心问题进行实践。

(3) 基本要求

① 编程实现上述 1-中心问题剪枝搜索算法。

② 点在平面上随机放置，最好给出解的图形表示，以方便验证结果的正确性。

③ 自己设计一个针对 1-中心问题的穷举算法。

④ 在一台机器上对上面的两个算法进行实验对比，点数分别为 10，30，50，100，150，200，250，300。

⑤ 分析 1-中心问题剪枝搜索算法的时间复杂度和 1-中心问题的穷举算法的时间复杂度。

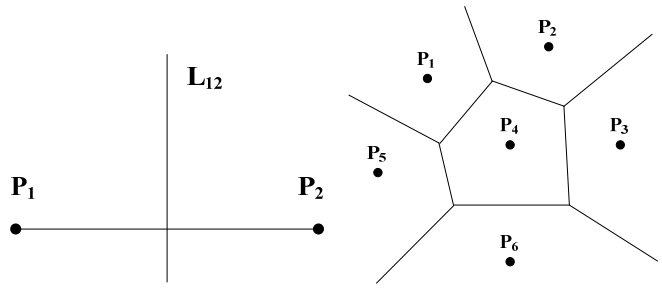
(4) 实现提示

无。

50．Voronoi 图及其简单应用

(1) 问题描述

Voronoi(一位著名的俄罗斯数学家)图是一种非常有趣的数据结构，该数据结构可以用于表示有关平面点最近邻点的有关信息。图 6-82(*a*)给出的是两个点的 Voronoi 图，实际上就是这两个点的垂直平分线 L_{12}。此时对于任意的点 X，不用计算 X 和 P_1 和 P_2 的距离，只要看 X 位于 L_{12} 的哪一边(将 X 的坐标代入 L_{12} 就能判断)就可以了。图 6-82(*b*)给出的是 6 个点的 Voronoi 图。



(*a*) (*b*)

图 6-82 Voronoi 图实例

Voronoi 图可以用分治算法实现。

算法：构造 Voronoi 图的分治算法

输入：n 个平面点集合 S

输出：集合 S 的 Voronoi 图

第 1 步：如果集合 S 只有 1 个点，则返回。

第 2 步：找出垂直于 x 轴的中线 L，L 将 S 划分为相同大小的集合 S_L 和 S_R，S_L 在 L 的左边，S_R 在 L 的右边。

第 3 步：递归的构造 S_L 和 S_R 的 Voronoi 图，分别用 $VD(S_L)$和 $VD(S_R)$表示各自的 Voronoi 图。

第 4 步：构造分段直线 HP，删除 HP 右边的 $VD(S_R)$所有的线段，删除 HP 左边的 $VD(S_L)$所有的线段，剩下的图就是 S 的 Voronoi 图。

图 6-83 给出了一个该分治算法的实例。

该分治算法的核心是将两个 Voronoi 图合并成一个 Voronoi 图的算法，现给出该算法。

算法：将两个 Voronoi 图合并成一个 Voronoi 图的算法

输入：$VD(S_L)$和 $VD(S_R)$

输出：VD(S)

第 1 步：找出 S_L 的凸包 $Hull(S_L)$，S_R 的凸包 $Hull(S_R)$。

第 2 步：找出将 $Hull(S_L)$和 $Hull(S_R)$合并成一个凸包的线段 P_aP_b 和 P_cP_d (P_a 和 P_c 属于 S_L，P_b 和 P_d 属于 S_R)，假如 P_aP_b 在 P_cP_d 的上面，令 x=a，y=b，SG= P_xP_y，HP=∅。

第 3 步：找出 SG 的垂直平分线，用 BS 表示。令 HP = HP ∪ BS。如果 SG = P_cP_d，转

向第 5 步，否则转向第 4 步。

第 4 步：令 BS 首先与来自 VD(S_L)或 VD(S_R)的射线相交，该射线一定是相对于某 z 的 P_xP_z 或 P_zP_y 的垂直平分线，如果该射线是 P_zP_y 的垂直平分线，则令 SG= P_xP_z，否则令 SG= P_zP_y。

第 5 步：删除 HP 右边的 VD(S_R)所有的边，删除 HP 左边的 VD(S_L)所有的边，最终的图就是 S 的 Voronoi 图。

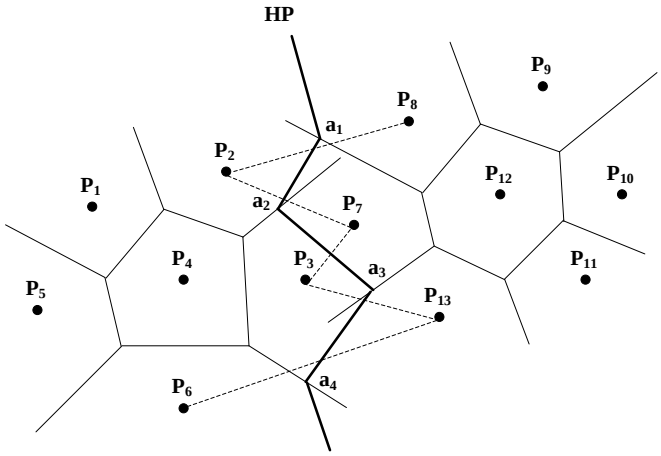


图 6-83 Voronoi 图分治算法的实例

显然在该算法中，有两个算法需要详细描述，一是将 Hull(S_L)和 Hull(S_R)合并成一个凸包的两条线段；另一个是找出 S 的凸包 Hull(S)。对于如何找出两个凸包合并成一个凸包的线段，来看如图 6-84 的实例，显然从一个凸包内的任一个点向另一个凸包的所有点连线，将这些线的方向排序，显然向上倾斜和向下倾斜度最大的两个顶点就是两个凸包合并成一个凸包的线段的在一个凸包中的两个端点，同样可以找到另一个凸包中对应的两个端点，对应的连接这两对端点就能得到将两个凸包合并成一个凸包的线段。

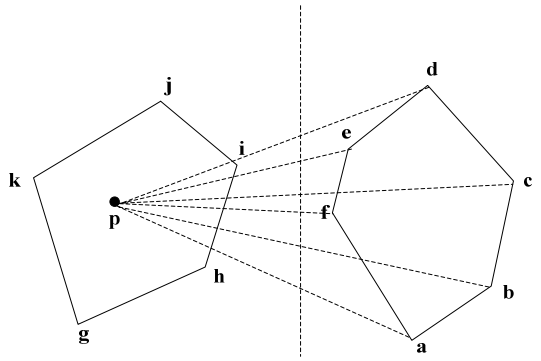


图 6-84 合并两个凸包

第二个需要回答的问题是如何找出 S 的凸包 Hull(S)。可以用 Voronoi 图的信息来求该凸包，如图 6-85 所示：即检查所有 Voronoi 边直到找到一条射线，设 P_i 是此射线左边的一点，那么 P_i 就是凸包的一个顶点，重复这一过程直到找到所有的射线，找出所有的凸包顶点，就获得了凸包。

下面来看一个 Voronoi 图的应用，解决欧几里得近邻搜索问题：已知平面上的 n 个点 P_1，P_2，…，P_n，和一个检测点 P，找出和 P 最近(欧几里得距离)的 P_i。一个非常简单的方法是依次求出 P 和每个点的距离，然后求出最小的距离，其时间复杂性是 O(n)。可以用 Voronoi 图来提高该搜索的时间。

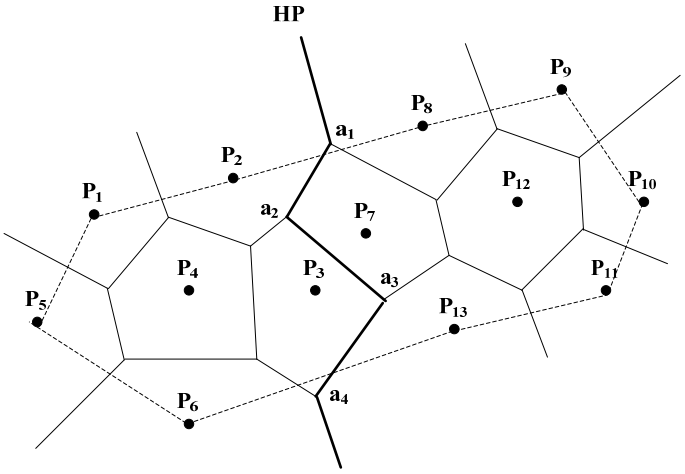


图 6-85 根据 Voronoi 图求凸包

下面给出应用 Voronoi 图解决欧几里得近邻搜索问题的基本想法：首先将所有的 Voronoi 顶点按照其 y 值排序，然后在每个 Voronoi 顶点处画一条水平线，这些线将整个空间分割成多个片隙。在每个片隙内可以排序 Voronoi 边，根据这些边可以很容易判断该片隙中的各个区域，因此可以根据 P 和这些 Voronoi 边的关系来判断 P 位于哪个区域，从而判断 P 和哪个点最近（由于已经排序，所以可以用二分查找）。如图 6-86 所示。

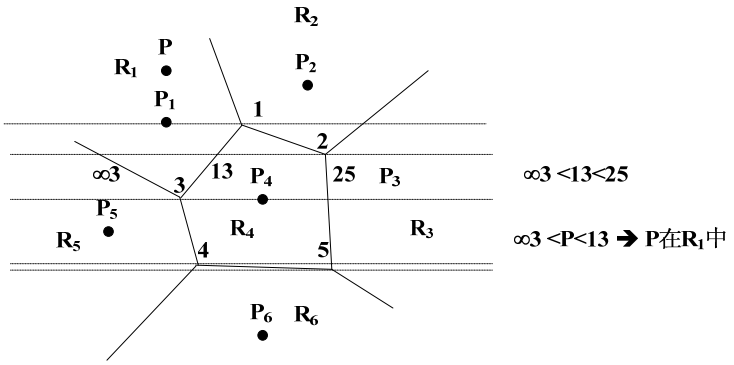


图 6-86 应用 Voronoi 图解决欧几里得近邻搜索问题

(2) 课程设计目的

认识 Voronoi 图，能够编程实现 Voronoi 图及其上的简单应用。

(3) 基本要求

① 编程实现上述求解 Voronoi 图的分治算法。

② 编程实现应用 Voronoi 图解决欧几里得近邻搜索问题的算法。

③ 平面点在平面上随机放置，最好给出解的图形表示，以方便验证结果的正确性。

④ 在一台机器上对应用 Voronoi 图解决欧几里得近邻搜索问题的算法和求出所有距离后找最近的两个算法，并对这两个算法进行实验对比，点数分别为 100，300，500，1000，10000，5000。

⑤ 分析求解 Voronoi 图的分治算法时间复杂度和应用 Voronoi 图解决欧几里得近邻搜索问题的算法的时间复杂度。

(4) 实现提示

需仔细思考 Voronoi 图的存储结构。