

《算法设计与分析》期末考试试卷 (A)

$$\begin{cases} x_{n+1} = 2x_n + 15x_{n-1}, n \geq 1 \\ x_0 = 1, x_1 = 1 \end{cases}$$

解: 该关系的特征方程

$$r^2 = 2r + 15$$

其特征根为

$$r_1 = 5, r_2 = -3$$

故通解为

$$x_n = c_1 \cdot 5^n + c_2 \cdot (-3)^n$$

由初值条件

$$x_0 = 1, x_1 = 1$$

得

$$\begin{cases} 1 = c_1 + c_2 \\ 1 = 5c_1 - 3c_2 \end{cases}$$

求得

$$c_1 = 0.5, c_2 = 0.5$$

故

$$x_n = \frac{1}{2} [5^n + (-3)^n]$$

【正确求解得到特征根, 5 分; 根据特征根得到非递归方程式 5 分】

考试 注意 事项	一、学生参加考试须带学生证或学院证明, 未带者不准进入考场。学生必须按照监考教师指定座位就坐。							
	二、书本、参考资料、书包等物品一律放到考场指定位置。							
	三、学生不得另行携带、使用稿纸, 要遵守《北京邮电大学考场规则》, 有考场违纪或作弊行为者, 按相应规定严肃处理。							
	四、学生必须将答题内容做在试题答卷上, 做在草稿纸上一律无效。							
	五、学生的姓名、班级、学号、班内序号等信息由教材中心统一印制。							
考试 课程	算法分析与设计		考试时间		2019 年 6 月 17 日			
题号	一	二	三	四	五	六	七	总分
满分	10	20	20	15	20	15		
得分								
阅卷 教师								

一、(10 分, 每题 5 分) (1) 给出算法分析中符号 O 的定义, 并证明:

$$O(f(n)g(n)) = O(f(n))O(g(n))$$

解:

对于任意 $f_1(n) \in O(f(n))$, 存在正常数 c_1 和自然数 n_1 , 使得对所有 $n \geq n_1$, 有 $f_1(n) \leq c_1 f(n)$ 。

类似地, 对于任意 $g_1(n) \in O(g(n))$, 存在正常数 c_2 和自然数 n_2 , 使得对所有 $n \geq n_2$, 有 $g_1(n) \leq c_2 g(n)$ 。

【评分标准: 依定义给出函数, 得 4 分】

令 $c_3 = c_1 \cdot c_2$, $n_3 = \max\{n_1, n_2\}$, 则对所有的 $n \geq n_3$, 有

【评分标准: 找到值 c_3 和 n , 得 2 分】

$$f_1(n)g_1(n) \leq c_1 f(n) \cdot c_2 g(n) = c_3 f(n)g(n)$$

【评分标准: 得到上面不等式, 得 3 分】

故有关系

$$O(f(n)) \cdot O(g(n)) = O(f(n)g(n))$$

【评分标准: 得到上式得 1 分】

二、(20 分) 快速排序算法是重要的排序算法之一:

(1) 说明算法的实现原理, 分析划分元素的选取对算法性能的影响;

(2) 用 C/C++/Java 语言伪代码描述快速排序算法实现过程, 要求:

i) 给出 partition 过程的代码实现;

ii) 排序算法采用随机选择策略选择划分元素, 给出

RandomizedPartition 过程实现代码;

(3) 证明: 在最好情况下, 快速排序算法的最好时间复杂度为

$T(n)=O(n\lg n)$, 其中 n 为排序元素的个数;

(4) 给定输入序列 {23, 13, 28, 20, 22, 19, 25}, 将序列按照非递增顺序排列, 描述整个排序过程。假设在排序过程中, RandomizedPartition

过程总是选取序列中最后 1 个元素作为划分基准元素

(5) 说明快速排序算法不稳定的原因, 并给出解决不稳定性的方法:

(1) (5 分) 原理: 假设排序要求为非递减序列

采用分治策略, 针对输入待排序数组 $a[p:r]$, 按照一定规则, 如选取最左端或最右端元素, 从序列中随机选取元素, 选择基准元素 $a[q]$, 将 $a[p:r]$ 划分为三段: $a[p:q-1]$, $a[q]$, $a[q+1:r]$, 使得 $a[p:q-1]$ 中的元素均小于 $a[q]$, $a[q+1:r]$ 中的元素均大于 $a[q]$, 之后对 $a[p:q-1]$ 和 $a[q+1:r]$ 分别递归调用本算法就得到有序的数组。

(2) (7 分) 快速排序过程如下

【答案 1: 以最左端元素 $a[p]$ 为基准/划分元素, 按照排序结果为非递减序列, 划分、排序过程与书、讲义内容保持一致】

void QuickSort (Type a[], int p, int r) (2 分)

```
{
    if (p < r) {
        int q = randomizedPartition(a, p, r);
        QuickSort(a, p, q-1); // 对左半段排序
        QuickSort(a, q+1, r); // 对右半段排序
    }
}
```

对 n 个元素组成的数组 $a[0:n-1]$ 进行排序, 调用 QuickSort (Type a, 0, n-1)

以确定的基准元素 $a[p]$, 对子数组 $a[p:r]$ 进行划分:

int Partition (Type a[], int p, int r) (3 分)

```
{
    int i = p, j = r + 1;
    Type x = a[p]; // *第 1 个元素
    // 将 < x 的元素交换到左边区域
    // 将 > x 的元素交换到右边区域
    while (true) {
        while (a[++i] < x);
        while (a[--j] > x);
        if (i >= j) break;
        Swap(a[i], a[j]);
    }
    a[p] = a[j];
    a[j] = x;
    return j;
}
```

int RandomizedPartition (Type a[], int p, int r) (2 分)

```
{
    int i = Random(p, r); // 随机选 1 个 a[j]
    Swap(a[i], a[p]); // 将 a[i] 交换到 a[] 的第 1 个位置 a[p]
    return Partition(a, p, r);
}
```

【答案 2: 随机选择算法挑选的划分基准元素交换到最左端 $a[p]$, 以最左端元素 $a[p]$ 为基准/划分元素, 但排序结果为非递增序列, 划分过程与书、讲义内容有差异】

void QuickSort (Type a[], int p, int r) (2 分)

```
{
    if (p < r) {
```



```

int q=randomizedPartition(a,p,r);
QuickSort (a,p,q-1); //对左半段排序
QuickSort (a,q+1,r); //对右半段排序
}

```

对 n 个元素组成的数组 $a[0:n-1]$ 进行排序，调用 QuickSort (Type a, 0, $n-1$)

以确定的基准元素 $a[p]$ ，对子数组 $a[p:r]$ 进行划分，但要求小于 $a[p]$ 的元素交换到右边区域，大于 $a[p]$ 的元素交换到左边区域：

int Partition (Type a[], int p, int r) (3 分)

```

{
    int i = p, j = r + 1;
    Type x = a[p]; //第 1 个元素
    // 将 < x 的元素交换到右边区域
    // 将 > x 的元素交换到左边区域
    while (true) {
        while (a[++i] < x);
        while (a[--j] > x);
        if (i >= j) break;
        Swap(a[i], a[j]);
    }
    a[p] = a[j];
    a[j] = x;
    return j;
}

```

int RandomizedPartition (Type a[], int p, int r) (2 分)

```

{
    int i = Random(p,r); //随机选 1 个 a[i]
    Swap(a[i], a[p]); //将 a[i] 交换到 a[] 的第 1 个位置 a[p]
    return Partition (a, p, r);
}

```

【答案 3：随机选择算法挑选的划分基准元素交换到最右端 $a[r]$ ，以最右端元素 $a[r]$

为基准/划分元素，但排序结果为非递增序列，划分过程与书、讲义内容有差异】

void QuickSort (Type a[], int p, int r) (2 分)

```

{
    if (p < r) {
        int q = randomizedPartition(a,p,r);
        QuickSort (a,p,q-1); //对左半段排序
        QuickSort (a,q+1,r); //对右半段排序
    }
}

```

对 n 个元素组成的数组 $a[0:n-1]$ 进行排序，调用 QuickSort (Type a, 0, $n-1$)

以确定的基准元素 $a[r]$ ，对子数组 $a[p:r]$ 进行划分，但要求小于 $a[r]$ 的元素交换到右边区域，大于 $a[r]$ 的元素交换到左边区域：

int Partition (Type a[], int p, int r) (3 分)

```

{
    int i = r, j = p - 1;
    Type x = a[r]; //第 1 个元素
    // 将 > x 的元素交换到左边区域
    // 将 < x 的元素交换到右边区域
    while (true) {
        while (a[--i] < x);
        while (a[++j] > x);
        if (i <= j) break;
        Swap(a[i], a[j]);
    }
    a[r] = a[j];
    a[j] = x;
    return j;
}

```

int RandomizedPartition (Type a[], int p, int r) (2 分)

```

{
    int i = Random(p,r); //随机选 1 个 a[i]
    Swap(a[i], a[p]); //将 a[i] 交换到 a[] 的第 1 个位置 a[p]
    return Partition (a, p, r);
}

```


}

(3) (4 分) 在最好情况下, 基准元素恰好选为排序元素的中位数, 划分过程将待排序数组分为长度相等 (或相差为 1) 的 2 个子序列, (2 分)

此时 n 个元素排序的时间复杂度 $T(n)$ 满足: (2 分)

$$T(n) = \begin{cases} O(1), & n \leq 1 \\ 2T(n/2) + O(n), & n > 1 \end{cases}$$

经求解得 $T(n) = O(n \lg n)$

(4) (4 分) 对序列 {23, 13, 28, 20, 22, 19, 25} 的排序过程如下

【答案可以不唯一, 可参照以下样例, 但注意排序结果需要是非递增收】

初始: 23, 13, 28, 20, 22, 19, 25 25 为基准元素

23, 25, 23, 20, 22, 19, 13

左子序列 {28} 长度为 1, 左子序列排序过程结束:

28, 25, 23, 20, 22, 19, 13

处理右子序列, 以 13 为基准元素

23, 25, 23, 20, 22, 19, 13 19 为基准元素

23, 25, 23, 20, 22, 19, 13 22 为基准元素

28, 25, 23, 22, 20, 19, 13

剩余待排子序列 {23} 和 {20} 长度为 1, 排序结束

28, 25, 23, 22, 20, 19, 13

注: 下划线表示当前基准元素, 方框表明排序完毕。

三. (20 分) 最长公共子序列问题: 给定两个序列 $X = \{x_1, x_2, \dots, x_n\}$ 和 $Y = \{y_1, y_2, \dots, y_m\}$, 找出 X 和 Y 的最长公共子序列。例如, 对 $X = (A, B, C, B, D, A, B)$, $Y = (B, D, C, A, B, A)$, 最长公共子序列为 $Z = (B, C, B, A)$ 。试用动态规划法求解最长公共子序列问题, 要求:

(1) 写出最长公共子序列问题的递归方程式:

【评分标准: 给出方程式, 得 5 分】

(2) 基于 C/C++ 语言伪代码描述算法流程, 算法输出包括最长公共子序列长度、最长公共子序列自身;

【评分标准: 长度, 得 3 分; 最长公共子序列, 再给 3 分】

(3) 给出所写算法的时间复杂度分析;

【评分标准: 给出复杂度表达式, 得 4 分】

(4) 利用最长公共子序列求解最长递减子序列, 基于 C/C++ 语言伪代码描述算法流程。算法输出最长递减子序列的长度。

【评分标准: 给出语言描述, 得 3 分; 结合代码, 再给 2 分】

其中, 最长递减子序列问题描述如下: 给定由 n 个整数 a_1, a_2, \dots, a_n 构成的序列, 在这个序列中随意删除一些元素后可得到一个子序列 $a_{i_1}, a_{i_2}, \dots, a_{i_k}$, 其中 $1 \leq i_1 < i_2 < \dots < i_k \leq n$, 则称序列 $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ 为原序列的一个递减子序列, 长度最长的递减子序列即为原序列的最长递减子序列。例如, 序列 {1, 7, 2, 3, 6, 5}, 可以得到递减子序列 {7, 6, 5}, 以及 {6, 5} 等。第一个递减子序列明显长度要长于第二个递减子序列。

答案

(1) 根据最优子结构性质, 建立子问题最优值的递归关系。

$c[i][j]$: 序列 $X(i)$ 和 $Y(j)$ 的最长公共子序列的长度, $X(i) = \{x_1, x_2, \dots, x_i\}$; $Y(j) = \{y_1, y_2, \dots, y_j\}$ 。

当 $i=0$ 或 $j=0$ 时, 即其中一个序列为空, 则空序列是 X 和 Y 的最长公共子序列, 故此时 $c[i][j]=0$ 。

其它情况下, 由最优子结构性质可建立递归关系如下:

$$c[i][j] = \begin{cases} 0 & i=0, \text{ 或 } j=0 \\ c[i-1][j-1]+1 & i, j > 0; x_i = y_j \\ \max\{c[i][j-1], c[i-1][j]\} & i, j > 0; x_i \neq y_j \end{cases}$$

(2) 长度:

void LCSLength(int m, int n, char *x, char *y, int **c, int **b)

```
{
    int i, j;
    for (i = 1; i <= m; i++) c[i][0] = 0; /* 初始化, Y[j] 为空时
    for (i = 1; i <= n; i++) c[0][i] = 0; /* 初始化, X[i] 为空时
```



```

for (i = 1; i <= m; i++)
    /*两重循环，自下而上，
    计算子问题 {X(i),
    Y(j)}

    for (j = 1; j <= n; j++) {
        if (x[i] == y[j]) {
            c[i][j] = c[i-1][j-1] + 1;
            b[i][j] = 1;
        }
        else if (c[i-1][j] >= c[i][j-1]) {
            c[i][j] = c[i-1][j]; b[i][j] = 2;
        }
        else { c[i][j] = c[i][j-1];
            b[i][j] = 3;
        }
    }
}

```

最长公共子序列

```

void LCS(int i, int j, char *x, int **b)
{
    if (i == 0 || j == 0) return;
    if (b[i][j] != -1) { LCS(0, 1, x, b); cout << x[i]; }
    /*第1种情况下，X(i)和Y(j)的最长公共子序列由 X(i-1)和
    Y(j-1)的解 LCS(i-1, j-1, x, b)，加上位于最后的 x[i]
    组成
    else if (b[i][j] == 2) LCS(i-1, j, x, b);
    else LCS(i, j-1, x, b);
    /*其它2种情况下，原问题解
    等于子问题
    解
}

```

(3) 算法时间复杂度
算法耗时: $O(mn)$

(4)
利用最长公共子序列求解最长递减子序列

由于最长递减子序列问题是求解一个给定序列的最长递减子序列，因此要用最长公共子序列问题来求解最长递减子序列问题，首先要构造出第二个序列，即让原序列和哪个序列求最长公共子序列才能得到原序列的最长递减子序列。显然，只需将原序列降

序排序，得到降序序列，再让原序列和降序序列求一下最长公共子序列，则这个最长公共子序列即为原序列的一个最长递减子序列。要注意的是，在最长公共子序列算法中，两个参数数组都是字符数组，在求解最长递减子序列问题时，需先将两个参数数组改为整型数组。算法如下：

```

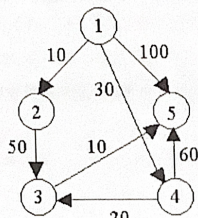
void LIS (int a[], int x[], int m)
//该算法求解 x[1: m]这个长度为 m 的整型数组的最长递减子序列长度
{
    for (int i=1; i<=m; i++) //输入数组 x
        scanf ("%d", &x[i]);

    for (int i=1; i<=m; i++) //数组 a 为数组 x 的副本
        a[i]=x[i];

    mergesort (a, 1, m); //对数组 a 进行递减归并排序
    LCSLength (x, a, m, m); //求数组 x 与数组 a 的最长公共子序列长度
}

```


四. (15 分) 如下图所示, 应用 Dijkstra 算法计算从源顶点 1 到其它顶点间最短路径的过程。



(1) 说明 Dijkstra 求解该问题的贪心策略, 并给出求解过程;

【评分标准: 得出以下表格, 得 5 分】

(2) 证明该贪心算法的正确性, 即该贪心算法能够获得最优解;

【评分标准: 给出贪心性质及最优子结构, 得 6 分】

(3) 分析所写算法的时间复杂性。

【评分标准: 给出复杂度表达式, 得 4 分】

答案:

(1) 每次从 $V-S$ 中取出具有(只经过 S 中顶点)的最短特殊路径长度 $\text{dist}[u]$ 的顶点 u , 将 u 添加到 S 中, 同时对数组 dist 作必要的修改

迭代	S	u	$\text{dist}[2]$	$\text{dist}[3]$	$\text{dist}[4]$	$\text{dist}[5]$
初始	{1}		10	$+\infty$	30	100
1	{1, 2}	2	10	60	30	100
2	{1, 2, 4}	4	10	50	30	90
3	{1, 2, 4, 3}	3	10	50	30	60
4	{1, 2, 4, 3, 5}	5	10	50	30	60

(2) 先证明贪心性质

贪心选择策略:

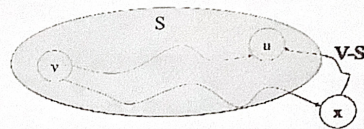
在每步迭代时, 从 $V-S$ 中选择具有最短特殊路径 $\text{dist}[u]$ 的顶点 u , 加入 S

贪心策略正确性: 需证明 对任意顶点 u ,

从 v 开始、经过 G 中任意顶点到达 u 的全局最短路径的长度 $d(v, u)$

= 从 v 开始、只经过 S 中顶点到达 u 的最短路径的长度 $\text{dist}(u)$

即: 不存在另一条 v 到 u 的最短路径, 该路径上某些节点 x 不在 $V-S$ 中, 且该路径的长度 $d(v, u) < \text{dist}[u]$.



假设:

(1) 在迭代求解过程中, 顶点 u 是遇到的第 1 个满足: $d(v, u) < \text{dist}[u]$, 即 $d(v, u) \neq \text{dist}[u]$ 的顶点

(2) 从 v 到 u 的全局最短路径上, 第 1 个属于 $V-S$ 的顶点为 x 首先, 因为 u 是第一个满足全局最短路径不完全在 S 集合中的顶点, 即

$$d(v, u) < \text{dist}[u]$$

, 而 x 是在 u 之前遇到的顶点, x 的最短路径完全在 S 中, 因此:

$$\text{dist}[x] = d(v, x) \leq d(v, u)$$

对 v 到 u 的全局最短路径, 有

$$d(v, x) + \text{distance}(x, u) = d(v, u) < \text{dist}[u]$$

由于 $\text{distance}(x, u) > 0$, 因此

$$\text{dist}[x] = d(v, x) \leq d(v, u) < \text{dist}[u], \text{ 即}$$

$$\text{dist}[x] < \text{dist}[u]$$

但是根据路径构造方法, 在下图所示情况下, u, x 都在集合 S 之外, 即 u, x 都属于 $V-S$, 但不被选中时, 并没有选 x , 根据扩展 S 的原则——选 dist 最小的顶点加入 S , 说明此时:

$$\text{dist}[u] \leq \text{dist}[x]$$

这与 前面推出的

$$\text{dist}[x] < \text{dist}[u]$$

相矛盾

从而证明了贪心性质

再证明最优子结构

对顶点 u , 考察将 u 加到 S 之前和之后, $\text{dist}[u]$ 的变化, 添加 u 之前的 S 称为老 S , 加入 u 之后的 S 称为新 S 。

要求: u 加到 S 中后, $\text{dist}[u]$ 不增加。

对另外 1 个节点 i , 考察 u 的加入对 $\text{dist}[i]$ 的影响:

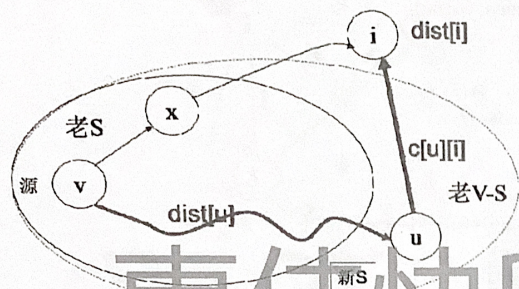
1) 假设添加 u 后, 出现 1 条从 v 到 i 的新路, 该路径先由 v 经过老 S 中

的顶点到达 u ，再从 u 经过一条直接边到达 i

该路径的最短长度 $= \text{dist}[u] + c[u][i]$

如果 $\text{dist}[u] + c[u][i] < \text{dist}[i]$ ，则算法用 $\text{dist}[u] + c[u][i]$ 替代 $\text{dist}[i]$ ，得到新的 $\text{dist}[i]$ 。否则， $\text{dist}[i]$ 不更新。

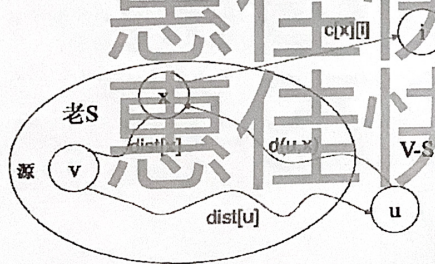
(3) 复杂性分析



2) 如果新路径如下图所示，先经过 u ，再回到 S 中的 x ，由 x 直接到达 i 。

x 处于老的 S 中，故 $\text{dist}[x]$ 已经是最短路径。 x 比 u 先加入 S ，因此

$$\text{dist}[x] \leq \text{dist}[u] + c(u, x)$$



此时，从原 v 到 i 的最短路径 $\text{dist}[i]$ 小于路径 (v, u, x, i) 的长度，因此算法更新 $\text{dist}[i]$ 时不需要考虑该路径， u 的加入对 $\text{dist}[i]$ 无影响。因此，无论算法中 $\text{dist}[u]$ 的值是否变化，它总是关于当前顶点集合 S 的到顶点 u 的最短路径。

也就是说：对于加入 u 之前、之后的新老 S 所对应的 2 个子问题，算法执行过程保证了 $\text{dist}[u]$ 始终是 u 的最优解

五. (20 分) 4 皇后问题。在 4×4 的棋盘上摆放四个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上。

(1) 请基于回溯法设计本问题的解向量；

【评分标准：给出解向量表达式，得 5 分】

(3) 给出搜索的剪枝策略，并画出解空间树；

【评分标准：剪枝策略，得 2 分；画出解空间树，再给 3 分】

(4) 写出基于 C/C++ 的算法伪代码；

【评分标准：给出逻辑合理的伪代码，得 5 分】

(5) 分析所写算法的时间复杂性。

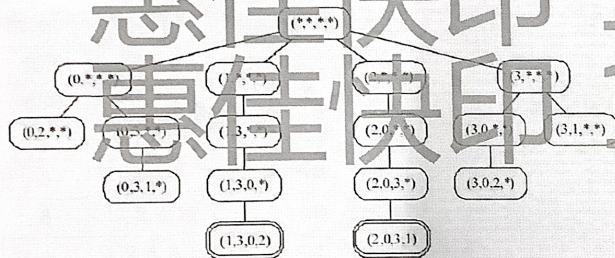
【评分标准：给出伪代码对应的时间复杂度表达式，得 5 分】

答案如下

(1) 给棋盘上的行和列从 1 到 4 编号，同时也给皇后从 1 到 4 编号。由于每一个皇后应放在不同的行上，不失一般性，假设皇后 1 放在第 i 行上，因此 4 皇后问题可以表示成 4 元组 (x_1, x_2, \dots, x_4) ，其中 $x_i (i=1, 2, \dots, 4)$ 表示皇后 1 所放置的列号。

(2) 约束条件 1：当 $i \neq j$ 时， $x_i \neq x_j$ (式 1)

约束条件 2： $|i - j| \neq |x_i - x_j|$ (式 2)



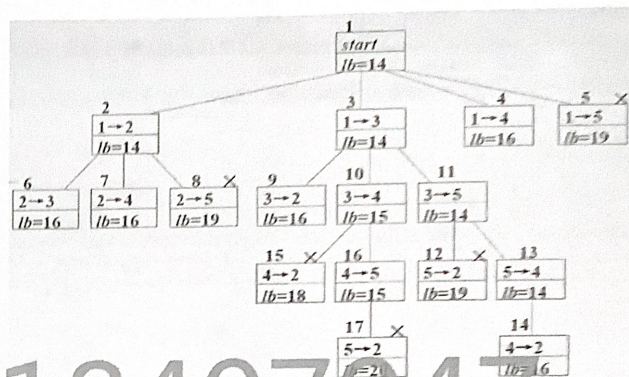
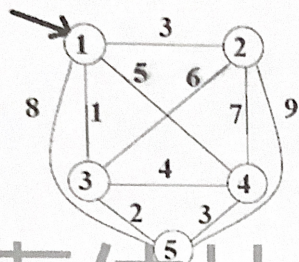
(3) 代码

```
#include<iostream>
#include<math.h>
using namespace std;
int n=8;
int total=0;
```

```
int *c=new int(n);
bool is_ok(int row){
    for(int j=0;j!=row;j++){
        if(c[row]==c[j]) || row-c[row]==j-c[j] ||
        row+c[row]==j+c[j])
            return false;
    }
    return true;
}
void queen(int row){
    if(row==n)
        total++;
    else
        for(int col=0;col!=n;col++){
            c[row]=col;
            if(is_ok(row))
                queen(row+1);
        }
}
int main(){
    queen(0);
    cout<<total;
    return 1;
}
```

(4) 时间复杂性

六、(15 分) 旅行售货员问题。某售货员要到若干城市去推销商品, 已知各城市之间的路程(如下无向图所示)。他要选定一条从驻地(城市 1)出发, 经过每个城市一次, 最后回到驻地的路线, 使总的路程(或总旅费)最小。



(1) 求解基于优先队列的分支限界法的最短回路及其长度。

【评分标准: 给出回路路径及长度值, 得 5 分】

(2) 给出界限函数的设计以及求解过程中所采用的剪枝策略。

【评分标准: 给出界限函数, 得 3 分; 给出剪枝策略, 得 2 分】

(3) 画出搜索过程中生成的解空间树, 说明发生剪枝的节点, 以及树中的各个叶节点。画出剪枝对应的路径长度。

【评分标准: 画出解空间树, 得 3 分; 标注正确长度值, 再给 2 分】

(1) 1-3-5-4-2-1 长度为 16

(2) 利用贪心算法得到上界值 1 (路径 1-3-4-2-1) (也可以是其他值)

(3) $lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)

$lb = (\sum_{i=1}^{k-1} 2c_{ij} + \sum_{j \in U} c_{ij}) / 2$ (也可以是最小元素)