

“算法分析”期末复习题

1. 选择题

1. 应用Johnson法则的流水作业调度采用的算法是 (D)

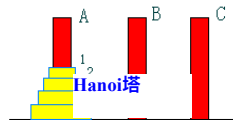
D. 动态规划算法

2. Hanoi塔问题如下图所示。现要求将塔座A上的的所有圆盘移到塔座B上，并仍按同样顺序叠置。移动圆盘时遵守Hanoi塔问题的移动规则。由此设计出解Hanoi塔问题的递归算法正确的为： (B)

```

A. void hanoi(int n, int A, int C, int B)
{
    if (n > 0)
    {
        hanoi(n-1, A, C, B);
        move(n, a, b);
        hanoi(n-1, C, B, A);
    }
}

```



```

B. void hanoi(int n, int A, int B, int C)
{
    if (n > 0)
    {
        hanoi(n-1, A, C, B);
        move(n, a, b);
        hanoi(n-1, C, B, A);
    }
}

```

```

C. void hanoi(int n, int C, int B, int A)
{
    if (n > 0)
    {
        hanoi(n-1, A, C, B);
        move(n, a, b);
        hanoi(n-1, C, B, A);
    }
}

```

```

    }
}

```

```

D. void hanoi(int n, int C, int A, int B)
{
    if (n > 0)
    {
        hanoi(n-1, A, C, B);
        move(n,a,b);
        hanoi(n-1, C, B, A);
    }
}

```

3. 动态规划算法的基本要素为 (C)

- A. 最优子结构性质与贪心选择性质
- B. 重叠子问题性质与贪心选择性质
- C. 最优子结构性质与重叠子问题性质
- D. 预排序与递归调用

4. 算法分析中，记号O表示 (B)，记号 Ω 表示 (A)，记号 Θ 表示 (D)。

- A. 渐进下界
- B. 渐进上界
- C. 非紧上界
- D. 紧渐进界
- E. 非紧下界

5. 以下关于渐进记号的性质是正确的有： (A)

- A.
- B.
- C. $O(f(n)) + O(g(n)) = O(\min\{f(n), g(n)\})$
- D.

6. 能采用贪心算法求最优解的问题，一般具有的重要性质为： (A)

- A. 最优子结构性质与贪心选择性质
- B. 重叠子问题性质与贪心选择性质
- C. 最优子结构性质与重叠子问题性质
- D. 预排序与递归调用

7. 回溯法在问题的解空间树中，按 (D) 策略，从根结点出发搜索解空间树。

A. 广度优先 B. 活结点优先 C. 扩展结点优先 D. 深度优先

8. 分支限界法在问题的解空间树中，按 (A) 策略，从根结点出发搜索解空间树。

A. 广度优先 B. 活结点优先 C. 扩展结点优先 D. 深度优先

9. 程序块 (A) 是回溯法中遍历排列树的算法框架程序。

A.

```
void backtrack (int t)
{
    if (t>n) output(x);
    else
        for (int i=t;i<=n;i++) {
            swap(x[t], x[i]);
            if (legal(t)) backtrack(t+1);
            swap(x[t], x[i]);
        }
}
```

B.

```
void backtrack (int t)
{
    if (t>n) output(x);
    else
        for (int i=0;i<=1;i++) {
            x[t]=i;
            if (legal(t)) backtrack(t+1);
        }
}
```

C.

```
void backtrack (int t)
{
    if (t>n) output(x);
    else
        for (int i=0;i<=1;i++) {
            x[t]=i;
            if (legal(t)) backtrack(t-1);
        }
}
```

D.

```
void backtrack (int t)
{
    if (t>n) output(x);
    else
        for (int i=t;i<=n;i++) {
            swap(x[t], x[i]);
            if (legal(t)) backtrack(t+1);
        }
}
```

10. 回溯法

- B. 满足显约束的 $x[k]$ 值的个数;
- C. 问题的解空间的形式;
- D. 计算上界函数bound的时间;
- E. 满足约束函数和上界函数约束的所有 $x[k]$ 的个数。
- F. 计算约束函数constraint的时间;

11. 常见的两种分支限界法为 (D)

- A. 广度优先分支限界法与深度优先分支限界法;
- B. 队列式 (FIFO) 分支限界法与堆栈式分支限界法;
- C. 排列树法与子集树法;
- D. 队列式 (FIFO) 分支限界法与优先队列式分支限界法;

12. k带图灵机的空间复杂性 $S(n)$ 是指 (B)

- A. k带图灵机处理所有长度为 n 的输入时, 在某条带上所使用过的最大方格数。
- B. k带图灵机处理所有长度为 n 的输入时, 在 k 条带上所使用过的方格数的总和
- C. 。
- C. k带图灵机处理所有长度为 n 的输入时, 在 k 条带上所使用过的平均方格数。
- D. k带图灵机处理所有长度为 n 的输入时, 在某条带上所使用过的最小方格数。

13. NP类语言在图灵机下的定义为 (D)

- A. $NP = \{L | L \text{ 是一个能在非多项式时间内被一台NDTM所接受的语言} \}$;
- B. $NP = \{L | L \text{ 是一个能在多项式时间内被一台NDTM所接受的语言} \}$;
- C. $NP = \{L | L \text{ 是一个能在多项式时间内被一台DTM所接受的语言} \}$;
- D. $NP = \{L | L \text{ 是一个能在多项式时间内被一台NDTM所接受的语言} \}$;

14. 记号 O 的定义正确的是 (A) 。

- A. $O(g(n)) = \{ f(n) \mid \text{存在正常数} c \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } 0 \leq f(n) \leq cg(n) \}$;
- B. $O(g(n)) = \{ f(n) \mid \text{存在正常数} c \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } 0 \geq cg(n) \leq f(n) \}$;
- C. $O(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } 0 \leq f(n) < cg(n) \}$;
- D. $O(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } 0 \leq cg(n) < f(n) \}$;

15. 记号 Ω 的定义正确的是 (B) 。

- A. $O(g(n)) = \{ f(n) \mid \text{存在正常数} c \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } 0 \leq f(n) \leq cg(n) \}$;
- B. $O(g(n)) = \{ f(n) \mid \text{存在正常数} c \text{ 和 } n_0 \text{ 使得对所有 } n \leq n_0 \text{ 有: } 0 \geq cg(n) \leq f(n) \}$;
- C. $(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0 \text{ 使得对所有 } n \leq n_0 \text{ 有: } 0 \geq f(n) < cg(n) \}$;
- D. $(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } 0 \leq cg(n) < f(n) \}$;

2、填空题

1. 下面程序段的所需要的计算时间为 () 。


```
int MaxSum(int n, int *a, int &besti, int &bestj)
{
    int sum=0;
    for(int i=1;i<=n;i++){
        int thissum=0;
        for(int j=i;j<=n;j++){
            thissum+=a[j];
            if(thissum>sum) {
                sum=thissum;
                besti=i;
                bestj=j;
            }
        }
    }
    return sum;
}
```

2. 有11个待安排的活动，它们具有下表所示的开始时间与结束时间，如果以贪心算法求解这些活动的最优安排（即为活动安排问题：在所给的活动集合中选出最大的相容活动子集合），得到的最大相容活动子集合为活动 ({1, 4, 8, 11}) 。

局部最优解 \rightarrow 整体最优解

- 所谓贪心选择性质是指（所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到）。
- 所谓最优子结构性质是指（问题的最优解包含了其子问题的最优解）。
- 回溯法是指（具有限界函数的深度优先生成法）。限定函数 + DFC
- 用回溯法解题的一个显著特征是在搜索过程中动态产生问题的解空间。在任何时刻，算法只保存从根结点到当前扩展结点的路径。
如果解空间树 中从根结点到叶结点的最长路径的长度为 $h(n)$ ，则回溯法所需的计算空间通常为 $O(h(n))$ ）。
- 回溯法的算法框架按照问题的解空间一般分为（子集树）算法框架与（排列树）算法框架。
- 用回溯法解0/1背包问题时，该问题的解空间结构为（子集树）结构。
- 用回溯法解批处理作业调度问题时，该问题的解空间结构为（排列树）结构。
- 用回溯法解0/1背包问题时，计算结点的上界的函数如下所示，请在空格中填入合适的内容：

```
Typep Knap<Typew, Typep>::Bound(int i)
{// 计算上界
    Typew cleft = c - cw; // 剩余容量
    Typep b = cp;        // 结点的上界
    // 以物品单位重量价值递减序装入物品
    while (i <= n && w[i] <= cleft) {
        cleft = w[i];
        b += p[i];
        i++;
    }
    // 装满背包
    if (i <= n) (b += p[i]/w[i] * cleft);
    return b;
}
```

- 用回溯法解布线问题时，求最优解的主要程序段如下。如果布线区域划分为  的方格阵列，扩展每个结点需 $O(1)$ 的时间， L 为最短布线路径的长度，则算法共耗时（ $O(mn)$ ），构造相应的最短距离需要（ $O(L)$ ）时间。

```
for (int i = 0; i < NumOfNbrs; i++) {
    nbr.row = here.row + offset[i].row;
    nbr.col = here.col + offset[i].col;
    if (grid[nbr.row][nbr.col] == 0) {
        // 该方格未标记
        grid[nbr.row][nbr.col]
            = grid[here.row][here.col] + 1;
        if ((nbr.row == finish.row) &&
            (nbr.col == finish.col)) break; // 完成布线
        Q.Add(nbr);
    }
}
```

12. 用回溯法解图的m着色问题时，使用下面的函数OK检查当前扩展结点的每一个儿子所相应的颜色的可用性，则需耗时（渐进时间上限）（ $O(mn)$ ）。

```
Bool Color::OK(int k)
{
    for(int j=1;j<=n;j++)
        if((a[k][j] == 1)&&(x[j] == x[k])) return false;
    return true;
}
```

13. 旅行售货员问题的解空间树是（排列树）。

6.
7.

3、证明题

1. 一个分治法将规模为n的问题分成k个规模为 n/m 的子问题去解。设分解阈值 $n_0=1$ ，且adhoc解规模为1的问题耗费1个单位时间。再设将原问题分解为k个子问题以及用merge将k个子问题的解合并为原问题的解需用 $f(n)$ 个单位时间。用 $T(n)$ 表示该分治法解规模为 $|P|=n$ 的问题所需的计算时间，则有：

通过迭代法求得 $T(n)$ 的显式表达式为：
试证明 $T(n)$ 的显式表达式的正确性。

2. 举反例证明0/1背包问题若使用的算法是按照 p_i/w_i 的非递减次序考虑选择的物品，即只要正在被考虑的物品装得进就装入背包，则此方法不一定能得到最优解（此题说明0/1背包问题与背包问题的不同）。

证明：举例如： $p=\{7,4,4\}, w=\{3,2,2\}, c=4$ 时，由于 $7/3$ 最大，若按题目要求的方法，只能取第一个，收益是7。而此实例的最大的收益应该是8，取第2, 3个。

3. 求证： $O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$ 。

证明：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 $g_1(n) \leq c_2 g(n)$ 。

令 $c_3 = \max\{c_1, c_2\}$, $n_3 = \max\{n_1, n_2\}$, $h(n) = \max\{f(n), g(n)\}$ 。

则对所有的 $n \geq n_3$, 有

$$\begin{aligned} f(n) + g(n) &\leq c_1 f(n) + c_2 g(n) \\ &\leq c_3 f(n) + c_3 g(n) \\ &= c_3 (f(n) + g(n)) \\ &\leq c_3 \max\{f(n), g(n)\} \\ &= 2c_3 h(n) = O(\max\{f(n), g(n)\}). \end{aligned}$$

4. 求证最优装载问题具有贪心选择性质。

(最优装载问题: 有一批集装箱要装上一艘载重量为 c 的轮船。其中集装箱 i 的重量为 w_i 。最优装载问题要求确定在装载体积不受限制的情况下, 将尽可能多的集装箱装上轮船。 设集装箱已依其重量从小到大排序, (x_1, x_2, \dots, x_n) 是最

优装载问题的一个最优解。又设 。如果给定的最优装载问题有解, 则有 。)

证明:

4、解答题

1. 机器调度问题。

问题描述: 现在有 n 件任务和无限多台机器, 任务可以在机器上得到处理。每件任务的开始时间为 s_i , 完成时间为 f_i , $s_i < f_i$ 。 $[s_i, f_i]$ 为处理任务 i 的时间范围。两个任务 i, j 重叠指两个任务的时间范围区间有重叠, 而并非指 i, j 的起点或终点重合。例如: 区间 $[1, 4]$ 与区间 $[2, 4]$ 重叠, 而与 $[4, 7]$ 不重叠。一个可行的任务分配是指在分配中没有两件重叠的任务分配给同一台机器。因此, 在可行的分配中每台机器在任何时刻最多只处理一个任务。最优分配是指使用的机器最少的可行分配方案。

问题实例: 若任务占用的时间范围是 $\{[1, 4], [2, 5], [4, 5], [2, 6], [4, 7]\}$, 则按时完成所有任务最少需要几台机器? (提示: 使用贪心算法)

画出工作在对应的机器上的分配情况。

2. 已知非齐次递归方程: , 其中, b, c 是常数,

$g(n)$ 是 n 的某一个函数。则 $f(n)$ 的非递归表达式为: 。

现有 Hanoi 塔问题的递归方程为: , 求 $h(n)$ 的非递归表达式。

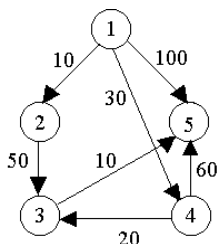
解: 利用给出的关系式, 此时有: $b=2, c=1, g(n)=1$, 从 n 递推到 1, 有:



3. 单源最短路径的求解。


问题的描述：给定带权有向图（如下图所示） $G=(V,E)$ ，其中每条边的权是非负实数。另外，还给定 V 中的一个顶点，称为源。现在要计算从源到所有其它各顶点的最短路长度。这里路的长度是指路上各边权之和。这个问题通常称为单源最短路径问题。

解法：现采用Dijkstra算法计算从源顶点1到其它顶点间最短路径。请将此过程填入下表中。



4. 请写出用回溯法解装载问题的函数。

装载问题：有一批共 n 个集装箱要装上2艘载重量分别为 c_1 和 c_2 的轮船，其中

集装箱 i 的重量为 w_i ，且 。装载问题要求确定是否有一个合理的装载方案可将这 n 个集装箱装上这2艘轮船。如果有，找出一种装载方案。

解：void **backtrack** (int i)
 {// 搜索第 i 层结点
 if ($i > n$) // 到达叶结点
 更新最优解bestx,bestw;return;
 r := $w[i]$;
 if ($cw + w[i] \leq c$) {// 搜索左子树
 $x[i] = 1$;

```

        cw += w[i];
        backtrack(i + 1);
        cw -= w[i];    }
    if (cw + r > bestw) {
        x[i] = 0; // 搜索右子树
        backtrack(i + 1);    }
    r += w[i];
}

```

5. 用分支限界法解装载问题时，对算法进行了一些改进，下面的程序段给出了改进部分；试说明斜线部分完成什么功能，以及这样做的原因，即采用这样的方式，算法在执行上有什么不同。

```

// 检查左儿子结点
Type wt = Ew + w[i]; // 左儿子结点的重量
if (wt <= c) { // 可行结点
    if (wt > bestw) bestw = wt;
    // 加入活结点队列
    if (i < n) Q.Add(wt);
}
// 检查右儿子结点
if (Ew + r > bestw && i < n)
    Q.Add(Ew); // 可能含最优解
    Q.Delete(Ew); // 取下一扩展结点

```

解答：斜线标识的部分完成的功能为：提前更新bestw值；

这样做可以尽早的进行对右子树的剪枝。具体为：算法Maxloading初始时将bestw设置为0，直到搜索到第一个叶结点时才更新bestw。因此在算法搜索到第一个叶子结点之前，总有bestw=0， $r>0$ 故 $Ew+r>bestw$ 总是成立。也就是说，此时右子树测试不起作用。

为了使上述右子树测试尽早生效，应提前更新bestw。又知算法最终找到的最优值是所求问题的子集树中所有可行结点相应重量的最大值。而结点所相应得重量仅在搜索进入左子树是增加，因此，可以在算法每一次进入左子树时更新bestw的值。

7. 最长公共子序列问题：给定2个序列 $X=\{x_1, x_2, \dots, x_m\}$ 和 $Y=\{y_1, y_2, \dots, y_n\}$ ，找出X和Y的最长公共子序列。

由最长公共子序列问题的最优子结构性性质建立子问题最优值的递归关系。用 $c[i][j]$ 记录序列 X_i 和 Y_j 的最长公共子序列的长度。其中， $X_i=\{x_1, x_2, \dots, x_i\}$ ； $Y_j=\{y_1, y_2, \dots, y_j\}$ 。当 $i=0$ 或 $j=0$ 时，空序列是 X_i 和 Y_j 的最长公共子序列。故此时 $C[i][j]=0$ 。其它情况下，由最优子结构性性质可建立递归关系如下：

在程序中，b[i][j]记录C[i][j]的值是由哪一个子问题的解得到的。

(1) 请填写程序中的空格，以使函数LCSLength完成计算最优值的功能。

```
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;
    for (i = 1; i <= n; i++) c[0][i] = 0;
    for (i = 1; i <= m; i++)
        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; }
        }
}
```

(2) 函数LCS实现根据b的内容打印出Xi和Yj的最长公共子序列。请填写程序中的空格，以使函数LCS完成构造最长公共子序列的功能（请将b[i][j]的取值与（1）中您填写的取值对应，否则视为错误）。

```
void LCS(int i, int j, char *x, int **b)
{
    if (i==0 || j==0) return;
    if (b[i][j]== 1) {
        LCS(i-1, j-1, x, b);
        cout<<x[i];
    }
    else if (b[i][j]== 2) LCS(i-1, j, x, b);
    else LCS(i, j-1, x, b);
}
```

8.对下面的递归算法，写出调用f(4)的执行结果。

```
void f(int k)
{ if (k>0)
    { printf("%d\n ",k);
      f(k-1);
      f(k-1);
    }
}
```

一、填空题 (20)

- 1.一个算法就是一个有穷规则的集合，其中之规则规定了解决某一特殊类型问题的一系列运算，此外，算法还应具有以下五个重要特性：有穷性、确定性、可行性、输入性、输出性。
- 2.算法的复杂性有时间复杂度和空间复杂度之分，衡量一个算法好坏的标准是时间复杂度。
- 3.某一问题可用动态规划算法求解的显著特征是具有最优子结构性质。
- 4.若序列 $X=\{B,C,A,D,B,C,D\}$ ， $Y=\{A,C,B,A,B,D,C,D\}$ ，请给出序列X和Y的一个最长公共子序列ACBDC。
- 5.用回溯法解问题时，应明确定义问题的解空间，问题的解空间至少应包含问题的一个解。
- 6.动态规划算法的基本思想是将待求问题分解成若干子问题，先求解子问题，然后从这些子问题的解得到原问题的解。
- 7.以深度优先方式系统搜索问题解的算法称为回溯法。
- 8.0-1背包问题的回溯算法所需的计算时间为 $O(2^n)$ ，用动态规划算法所需的计算时间为 $O(n \times \sum_{i=1}^n c_i)$ 。
- 9.动态规划算法的两个基本要素是最优子结构和重叠子问题。
- 10.二分搜索算法是利用分治法实现的算法。

二、综合题 (50分)

- 1.写出设计动态规划算法的主要步骤。
- 2.流水作业调度问题的johnson算法的思想。
- 3.若 $n=4$ ，在机器M1和M2上加工作业i所需的时间分别为 a_i 和 b_i ，且 $(a_1, a_2, a_3, a_4)=(4, 5, 12, 10)$ ， $(b_1, b_2, b_3, b_4)=(8, 2, 15, 9)$ 求4个作业的最优调度方案，并计算最优值。
- 4.使用回溯法解0/1背包问题： $n=3$ ， $C=9$ ， $V=\{6, 10, 3\}$ ， $W=\{3, 4, 4\}$ ，其解空间有长度为3的0-1向量组成，要求用一棵完全二叉树表示其解空间（从根出发，左1右0），并画出其解空间树，计算其最优值及最优解。
- 5.设 $S=\{X_1, X_2, \dots, X_n\}$ 是严格递增的有序集，利用二叉树的结点来存储S中的元素，在表示S的二叉搜索树中搜索一个元素X，返回的结果有两种情形，（1）在二叉搜索树的内结点中找到 $X=X_i$ ，其概率为 b_i 。（2）在二叉搜索树的叶结点中确定 $X \in (X_i, X_{i+1})$ ，其概率为 a_i 。在表示S的二叉搜索树T中，设存储元素 X_i 的结点深度为 C_i ；叶结点 (X_i, X_{i+1}) 的结点深度为 d_i ，则二叉搜索树T的平均路长p为多少？假设二叉搜索树 $T[i][j]=\{X_i, X_{i+1}, \dots, X_j\}$ 最优值为 $m[i][j]$ ， $W[i][j]=a_{i-1}+b_i+\dots+b_j+a_j$ ，则 $m[i][j](1 \leq i \leq j \leq n)$ 递归关系表达式为什么？

6.描述0-1背包问题。

三、简答题 (30分)

1.流水作业调度中，已知有 n 个作业，机器 M_1 和 M_2 上加工作业 i 所需的时间分别为 a_i 和 b_i ，请写出流水作业调度问题的johnson法则中对 a_i 和 b_i 的排序算法。（函数名可写为 $\text{sort}(s,n)$ ）

2.最优二叉搜索树问题的动态规划算法（设函数名 binarysearchtree ）

答案：

一、填空

1. 确定性 有穷性 可行性 0个或多个输入 一个或多个输出

2.时间复杂性 空间复杂性 时间复杂度高低

3.该问题具有最优子结构性质

4.{BACBD}或{CABCD}或{CADCD}

5.一个（最优）解

6.子问题 子问题 子问题

7.回溯法

8. $O(n \cdot 2^n)$ $O(\min\{nc, 2^n\})$

9.最优子结构 重叠子问题

10.动态规划法

二、综合题

1.①问题具有最优子结构性质；②构造最优值的递归关系表达式；③最优值的算法描述；④构造最优解；

2.①令 $N_1 = \{i | a_i < b_i\}$ ， $N_2 = \{i | a_i \geq b_i\}$ ；②将 N_1 中作业按 a_i 的非减序排序得到 N_1' ，将 N_2 中作业按 b_i 的非增序排序得到 N_2' ；③ N_1' 中作业接 N_2' 中作业就构成了满足Johnson法则的最优调度。

3.步骤为： $N_1 = \{1, 3\}$ ， $N_2 = \{2, 4\}$ ；

$N_1' = \{1, 3\}$ ， $N_2' = \{4, 2\}$ ；

最优值为：38

4.解空间为 $\{(0,0,0), (0,1,0), (0,0,1), (1,0,0), (0,1,1), (1,0,1), (1,1,0), (1,1,1)\}$ 。

解空间树为：



该问题的最优值为：16 最优解为：(1, 1, 0)

5.二叉树T的平均路长 $P =$ $+$

$$m[i][j]=W[i][j]+\min\{m[i][k]+m[k+1][j]\} \quad (1 \leq i \leq j \leq n, m[i][i]=0)$$

$m[i][j]=0 \quad (i>j)$

6.已知一个背包的容量为C，有n件物品，物品i的重量为 W_i ，价值为 V_i ，求应如何选择装入背包中的物品，使得装入背包中物品的总价值最大。

三、简答题

1.

```
void sort(flowjope s[],int n)
{
    int i,k,j,l;
    for(i=1;i<=n-1;i++)//-----选择排序
    {
        k=i;
        while(k<=n&& s[k].tag!=0) k++;
        if(k>n) break;//-----没有ai，跳出
        else
        {
            for(j=k+1;j<=n;j++)
                if(s[j].tag==0)
                    if(s[k].a>s[j].a) k=j;
            swap(s[i].index,s[k].index);
            swap(s[i].tag,s[k].tag);
        }
    }
    l=i;//-----记下当前第一个bi的下标
    for(i=l;i<=n-1;i++)
    {
        k=i;
        for(j=k+1;j<=n;j++)
            if(s[k].b<s[j].b) k=j;
        swap(s[i].index,s[k].index); //-----只移动index和tag
        swap(s[i].tag,s[k].tag);
    }
}
```

2.

```
void binarysearchtree(int a[],int b[],int n,int **m,int **s,int **w)
{
    int i,j,k,t,l;
    for(i=1;i<=n+1;i++)
    {
        w[i][i-1]=a[i-1];
        m[i][i-1]=0;
    }
    for(l=0;l<=n-1;l++)//----l是下标j-i的差
        for(i=1;i<=n-l;i++)
        {
```

```

j=i+1;
w[i][j]=w[i][j-1]+a[j]+b[j];
m[i][j]=m[i][i-1]+m[i+1][j]+w[i][j];
s[i][j]=i;
for(k=i+1;k<=j;k++)
{
    t=m[i][k-1]+m[k+1][j]+w[i][j];
    if(t<m[i][j])
    {
        m[i][j]=t;
        s[i][j]=k;
    }
}
}
}

```

1、填空题（本题15分，每小题1分）

- 1、算法就是一组有穷的___，它们规定了解决某一特定类型问题的___。
- 2、在进行问题的计算复杂性分析之前，首先必须建立求解问题所用的计算模型。3个基本计算模型是___、___、___。
- 3、算法的复杂性是___的度量，是评价算法优劣的重要依据。
- 4、计算机的资源最重要的是___和___资源。因而，算法的复杂性有___和___之分。
- 5、 $f(n) = 6 \times 2^n + n^2$ ， $f(n)$ 的渐进性态 $f(n) = O(\underline{\hspace{1cm}})$
- 6、贪心算法总是做出在当前看来___的选择。也就是说贪心算法并不从整体最优考虑，它所做出的选择只是在某种意义上的___。
- 7、许多可以用贪心算法求解的问题一般具有2个重要的性质：___性质和___性质。

二、简答题（本题25分，每小题5分）

- 1、简单描述分治法的基本思想。

2、简述动态规划方法所运用的最优化原理。

3、何谓最优子结构性质？

4、简单描述回溯法基本思想。

5、何谓P、NP、NPC问题

三、算法填空（本题20分，每小题5分）

1、n后问题回溯算法

(1)用二维数组A[N][N]存储皇后位置,若第i行第j列放有皇后,则A[i][j]为非0值,否则值为0。

(2)分别用一维数组M[N]、L[2*N-1]、R[2*N-1]表示竖列、左斜线、右斜线是否放有棋子,有则值为1,否则值为0。

for(j=0;j<N;j++)

if(1) /*安全检查*/

{ A[i][j]=i+1; /*放皇后*/

2 ;

if(i==N-1) 输出结果;

else 3 ; /*试探下一行*/

4 ; /*去皇后*/

5 ; ;

}

2、数塔问题。有形如下图所示的数塔，从顶部出发，在每一结点可以选择向左走或是向右走，一起走到底层，要求找出一条路径，使路径上的值最大。

for(r=n-2;r>=0;r--) //自底向上递归计算

for(c=0; 1 ;c++)

if(t[r+1][c]>t[r+1][c+1]) 2 ;

else 3 ;

3、Hanoi算法

Hanoi(n,a,b,c)

if (n==1) 1 ;

else

{ 2 ;

3 ;

Hanoi(n-1,b, a, c);

}

4、Dijkstra算法求单源最短路径

d[u]:s到u的距离 p[u]:记录前一节点信息

Init-single-source(G,s)

for each vertex v∈V[G]

do { d[v]=∞; 1 }

d[s]=0

Relax(u,v,w)

if d[v]>d[u]+w(u,v)

then { d[v]=d[u]+w[u,v];

2

}

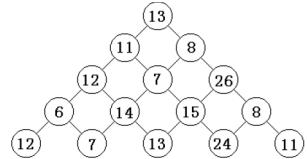
dijkstra(G,w,s)

1. Init-single-source(G,s)

2. S=∅

3. Q=V[G]

4.while Q≠∅



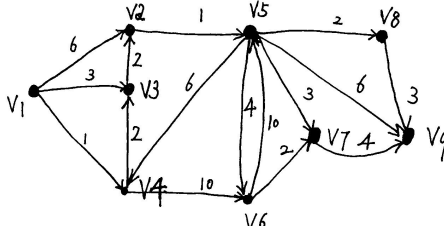

```

do u=min(Q)
S=S∪{u}
for each vertex 3
do 4

```

四、算法理解题 (本题10分)

根据优先队列式分支限界法, 求下图中从 v_1 点到 v_9 点的单源最短路径, 请画出来得最优解的解空间树。要求中间被舍弃的结点用 \times 标记, 获得中间解的结点用单圆圈 \circ 框起, 最优解用双圆圈 \odot 框起。



五、算法理解题 (本题5分)

设有 $n=2^k$ 个运动员要进行循环赛, 现设计一个满足以下要求的比赛日程表:

- ①每个选手必须与其他 $n-1$ 名选手比赛各一次;
- ②每个选手一天至多只能赛一次;
- ③循环赛要在最短时间内完成。

- (1) 如果 $n=2^k$, 循环赛最少需要进行几天;
- (2) 当 $n=2^3=8$ 时, 请画出循环赛日程表。

六、算法设计题 (本题15分)

分别用贪心算法、动态规划法、回溯法设计0-1背包问题。要求: 说明所使用的算法策略; 写出算法实现的主要步骤; 分析算法的时间。

七、算法设计题 (本题10分)

通过键盘输入一个高精度的正整数 n (n 的有效位数 ≤ 240), 去掉其中任意 s 个数字后, 剩下的数字按原左右次序将组成一个新的正整数。编程对给定的 n 和 s , 寻找一种方案, 使得剩下的数字组成的新数最小。

【样例输入】

178543

$S=4$

【样例输出】

13

一、填空题 (本题15分, 每小题1分)

1. 规则 一系列运算
2. 随机存取机 RAM(Random Access Machine); 随机存取存储程序机 RASP(Random Access Stored Program Machine); 图灵机(Turing Machine)
3. 算法效率
4. 时间、空间、时间复杂度、空间复杂度
5. 2^n
6. 最好 局部最优选择
7. 贪心选择 最优子结构

二、简答题 (本题25分, 每小题5分)

6. 分治法的基本思想是将一个规模为 n 的问题分解为 k 个规模较小的子问题,

这些子问题互相独立且与原问题相同；对这 k 个子问题分别求解。如果子问题的规模仍然不够小，则再划分为 k 个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止；将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。

- 7、“**最优化原理**”用数学化的语言来描述：假设为了解决某一优化问题，需要依次作出 n 个决策 D_1, D_2, \dots, D_n ，如若这个决策序列是最优的，对于任何一个整数 $k, 1 < k < n$ ，不论前面 k 个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以以后的决策 $D_{k+1}, D_{k+2}, \dots, D_n$ 也是最优的。
- 8、某个问题的最优解包含着其子问题的最优解。这种性质称为**最优子结构性**。
- 9、**回溯法的基本思想**是在一棵含有问题全部可能解的状态空间树上进行深度优先搜索，解为叶子结点。搜索过程中，每到达一个结点时，则判断该结点为根的子树是否含有问题的解，如果可以确定该子树中不含有问题的解，则放弃对该子树的搜索，退回到上层父结点，继续下一步深度优先搜索过程。在回溯法中，并不是先构造出整棵状态空间树，再进行搜索，而是在搜索过程，逐步构造出状态空间树，即边搜索，边构造。
- 10、P(Polynomial)问题：也即是多项式复杂程度的问题。
NP就是Non-deterministic Polynomial的问题，也即是多项式复杂程度的非确定性问题。
NPC(NP Complete)问题，这种问题只有把解域里面的所有可能都穷举了之后才能得出答案，这样的问题是NP里面最难的问题，这种问题就是NPC问题。

三、算法填空（本题20分，每小题5分）

1、n后问题回溯算法

- (1) $!M[j] \& \& !L[i+j] \& \& !R[i-j+N]$
- (2) $M[j]=L[i+j]=R[i-j+N]=1;$
- (3) $\text{try}(i+1, M, L, R, A)$
- (4) $A[i][j]=0$
- (5) $M[j]=L[i+j]=R[i-j+N]=0$

2、数塔问题。

- (1) $c \leq r$
- (2) $t[r][c] += t[r+1][c]$
- (3) $t[r][c] += t[r+1][c+1]$

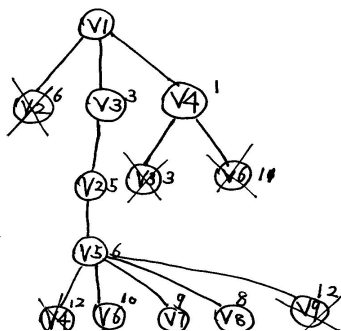
3、Hanoi算法

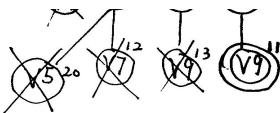
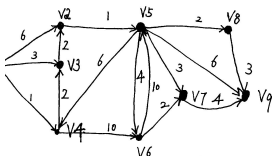
- (1) $\text{move}(a, c)$
- (2) $\text{Hanoi}(n-1, a, c, b)$
- (3) $\text{Move}(a, c)$

4、(1) $p[v]=\text{NIL}$

- (2) $p[v]=u$
- (3) $v \in \text{adj}[u]$
- (4) $\text{Relax}(u, v, w)$

四、算法理解题（本题10分）





五、(1) 8天 (2分) ;

(2) 当 $n=2^3=8$ 时, 循环赛日程表 (3分) 。

六、算法设计题 (本题15分)

(1) 贪心算法 $O(n \log(n))$

> 首先计算每种物品单位重量的价值 V_i/W_i , 然后, 依贪心选择策略, 将尽可能多的单位重量价值最高的物品装入背包。若将这种物品全部装入背包后, 背包内的物品总重量未超过 C , 则选择单位重量价值次高的物品并尽可能多地装入背包。依此策略一直地进行下去, 直到背包装满为止。

> 具体算法可描述如下:

```
void Knapsack(int n,float M,float v[],float w[],float x[])
{Sort(n,v,w);
 int i;
 for (i=1;i<=n;i++) x[i]=0;
 float c=M;
 for (i=1;i<=n;i++)
 {if (w[i]>c) break;
  x[i]=1;
  c-=w[i];
 }
 if (i<=n) x[i]=c/w[i];
 }
```

(2) 动态规划法 $O(nc)$

$m(i, j)$ 是背包容量为 j , 可选择物品为 $i, i+1, \dots, n$ 时0-1背包问题的最优值。由0-1背包问题的最优子结构性质, 可以建立计算 $m(i, j)$ 的递归式如下。



```
void KnapSack(int v[],int w[],int c,int n,int m[11])
{int jMax=min(w[n]-1,c);
for (j=0;j<=jMax;j++) /*m(n,j)=0 0=<j<w[n]*/
m[n][j]=0;
for (j=w[n];j<=c;j++) /*m(n,j)=v[n] j>=w[n]*/
m[n][j]=v[n];
for (i=n-1;i>1;i--)
{ int jMax=min(w[i]-1,c);
for (j=0;j<=jMax;j++) /*m(i,j)=m(i+1,j) 0=<j<w[i]*/
m[i][j]=m[i+1][j];
for (j=w[i];j<=c;j++)/*m(n,j)=v[n] j>=w[n]*/
m[i][j]=max(m[i+1][j],m[i+1][j-w[i]]+v[i]);
}
m[1][c]=m[2][c];
if(c>=w[1])
m[1][c]=max(m[1][c],m[2][c-w[1]]+v[1]);
}
```

(3) 回溯法 $O(2^n)$

cp:当前重量 cp:当前价值 bestp: 当前最优值

void backtrack(int i)

//回溯法 i初值1

```
{ if(i > n) //到达叶结点
{ bestp = cp; return; }
if(cw + w[i] <= c) //搜索左子树
{ cw += w[i];
cp += p[i];
backtrack(i+1);
cw -= w[i];
cp -= p[i];
}
if(Bound(i+1)>bestp)
//搜索右子树
backtrack(i+1);
}
```

七、算法设计题 (本题10分)

为了尽可能地逼近目标，我们选取的贪心策略为：每一步总是选择一个剩下的数最小的数字删去，即按高位到低位的顺序搜索，若各位数字递增，则

删除最后一个数字，否则删除第一个递减区间的首字符。然后回到串首，按上述规则再删除下一个数字。重复以上过程s次，剩下的数字串便是问题的解了。

具体算法如下：

输入s, n;

while (s > 0)

{ i=1; //从串首开始找

while (i < length(n)) && (n[i]<n[i+1])

{i++;}

delete(n,i,1); //删除字符串n的第i个字符

s--;

}

while (length(n)>1)&& (n[1]!='0')

delete(n,1,1); //删去串首可能产生的无用零

输出n;