2011 1.判断 1)所有正 $f(n)$，有 $f(n)+o(f(n))=\Theta(f(n))$ true 2)所有正 $f(n),g(n),h(n)$，若 $f(n)=O(g(n))$ 且 $f(n)=\Omega(h(n))$，则 $g(n)+h(n)=\Omega(f(n))$ true 因为 $f(n)=O(g(n))\Rightarrow g(n)=\Omega(h(n))$ 3)均匀散列下，3 个指定元素散列到同一槽（即 $h(1)=h(2)=h(3)$）的概率为为 $1/m^3$，m 为桶的数目 false 上面的公式只描述了在一个固定的桶中冲突的概率（也就是说桶的数目为 1）。正确答案是 $1/m^2$ 4)n 个元素的整数数组，各元素均属于{-1,0,1}，最坏情况对该数组进行排序的时间为 $O(n)$ true 将所有数加 1 后使用计数排序法 5)该数组为最大堆：[10,3,5,1,4,2] false 元素 3 比它的孩子 4 小，这违背了最大堆的性质 6)如果用插入排序替代计数排序来对每个数字排序，则基数排序得不到正确的输出 false 插入排序是稳定的排序，所以基础排序仍是正确的，但是这个改变会恶化排序运行时间 7)根据有向图 G 构造图 G'：顶点 $u'\in G'$ 代表 G 的强连通分量(SCC)，若在 G 中有一条边从 u' 对应的 SCC 到 v' 对应的 SCC，则在 G' 中存在边 (u',v')。那么 G' 是一个有向非循环图 true 如果在强连通分量图中存在任何循环，那么循环上的所有分量实际上都是一个强连通分量。8)正相关加权图 $G=(V,E,w)$ 和 $G'=(V,E,w')$，它们顶点集 V 和边集 E 相同，对于任意 $e\in E$，有 $w'(e)=w(e)^2$。那么任意 $u,v\in V$，G' 中 u 和 v 的任何最短路径也是 G 中的最短路径 false 假设 G 中有两条路径，一条权重为 2，另一条权重为 3。第一条在较 G' 短，第二条在 G 中较短。 9)背包问题的最优解通常包括产投比 v_i/c_i 最大的对象 i false 贪心选择对于背包问题并不适用。例如，如果最大开销为 2，现有两个项，第一个项开销为 1，价值为 2，第二个项开销为 2，价值为 3，那么将会把第二个项作为最优解。 10)NP 中的每个问题都可以在指数时间内求解 true 2.简答 1)对下列函数升序排序，即找到 g_1,g_2,g_3,g_4 一个排列满足 $g_1=O(g_2)$，$g_2=O(g_3)$，$g_3=O(g_4)$ —— $f_1=(n!)^{1/n}$ $f_2=\log n^n$ $f_3=n^{n^{1/2}}$ $f_4=n\log n\log\log n$ 解：f_1,f_2,f_4,f_3 2)通过给出 *tight* $\Theta-$*notaion* 边界求解下列递归式：i. $T(n)=4T(n/2)+n^2\log n$ ii. $T(n)=8T(n/2)+n\log n$ iii. $T(n)=\sqrt{6006}\cdot T(n/2)+n^{\sqrt{6006}}$ 解：i.主方法的 Case2：$T(n)=n^2\log^2 n$ ii.主方法的 Case1：$T(n)=n^3$ iii.主方法的 Case3：$T(n)=\Theta\left(n^{\sqrt{6006}}\right)$ 3)给出下列算法运行时间递归式 $T(n)=\cdots$ 及其渐进解：i.插入排序 ii.归并排序 iii.二维寻峰 解：i. $T(n)=T(n-1)+O(n)=O(n^2)$ ii. $T(n)=2T(n/2)+\Theta(n)=\Theta(n\lg n)$ iii. $T(n)=T(n/2)+\Theta(n)=\Theta(n)$ 4)在比较模型中是否可以通过 $O(n\lg n)$ 次比较来构造二分查找树？为什么？ 解：不可以，否则我们可以通过在 $o(n\lg n)$ 时间内构造 BST 实现 $o(n\lg n)$ 时间内的排序，然后在 $O(n)$ 的时间内完成树的有序游走。 5)从 0 到 k 的 n 个整数，为了在 $O(1)$ 时间内完成下列查询，应该将这些整数预处理成什么样的数据结构：给定两个整数 a 和 b，有多少个整数落在它们之间？ 解：和计数排序的预处理相同，然后数字的范围为 $C'(b)-C'(a)$ 6)解释为什么基于比较的排序算法是稳定的，不会被多余 1 个的常数因子印象运行时间。 解：标签元素及其在数组中的初始位置最多只会增加两倍 7)对于下列关系，给出一个有效的遍历顺序，如果不存在简要证明为什么。 i. $A(i,j)=F\left(A(i,j-1),A(i-1,j-1),A(i-1,j+1)\right)$

ii. $A(i,j)=F\left(A(\min\{i,j\}-1,\min\{i,j\}-1),A(\max\{i,j\}-1,\max\{i,j\}-1)\right)$

iii. $A(i,j)=F\left(A(i-2,j-2),A(i+2,j+2)\right)$ 解：i.求解 $A(i,j)$，for(i form 0 to n: for(j from 0 to n)) ii.求解 $A(\mathrm{k},\mathrm{k})$，for(k form 0 to n)然后按任意顺序求解剩下的 iii.不可能，因为它是周期的 8)整数数组 $A[1\ldots n]$，整数范围为 1 到 n^2。若数 a 在 A 中至少出现两次，则称数 a 为 heavy hitter。给出一个有效的算法找出 A 中的所有 heavy hitter。 解：基数排序和线性扫描 3.计算 1)在表格中填写序列对其子问题的解，0 代表变化，1 代表插入/删除，3 代表匹配，“ATC”为起始序列，“TCAG”为结束序列，最优的对齐是什么？解：最优的对齐是在两个序列中匹配“TC” 2)画出下列整数的最大堆：{2,3,5,7,11,13,17} 3)用牛顿法计算 $\sqrt[3]{6006}$

-	-	A	T	C
-	0	1	2	3
T	1	2	4	5
C	2	3	5	7
A	3	5	6	8
G	4	6	7	9

i	x_i
0	1
1	$2002.\bar{6}$
2	$2/3x_1+\frac{2002}{x_1^2}$

4.旋转数组 对数组 $A[1\cdots n]$ 进行排序再向右 k 步旋转，给出一个时间为 $O(\log n)$ 的算法，返回 A 中元素 x 的位置

解：

```
SEARCH(A, i, j, x)
    first ← A[i]
    middle ← A[(i + j)/2]
    last ← A[j]
    if x ∈ {first, middle, last}:
      then
            return the corresponding index
      else :
            if (x < first and x > middle) or (x > first and x > middle): [xxx Isn't it equivalent to just x > m
              then
                    return SEARCH(A, (i + j)/2, j, x)
              else :
                    return SEARCH(A, i, (i + j)/2, x)
```

2012 **2. 判断 (a)** 若 $f_1(n)=O(f_2(n))$ 且 $f_2(n)=O(g(n))$，那么 $f_1(n)+f_2(n)=O(g(n))$ True 因为 $f_1(n)\le c_1 f_2(n)\le c_1c_2g(n)$ 且 $f_2(n)\le c_2g(n)$ 所以有 $f_1(n)+f_2(n)\le(c_1c_2+c_2)g(n)=O(g(n))$ **(b)**A=[a1,.....,an]是一维整数数组。定义 A 一个大峰值为元素 $a_i\in A$,其中 $a_i\ge a_j$（$|j-i|\le 2$）。T 或 F：在 A 中找到峰值的时间复杂度为 $O(\log n)$ 吗？True 运行类中原来的峰值查找算法。不是在做分治算法时检测中间元素是否为峰值，而是检查中间 3 个元素是否是大峰值。如果不是，则在包含中间 5 个元素中最大元素的一侧递归。**(c)**使用基数排序，可以对{1，...，n^k}中的 n 个整数进行排序时间复杂度仅为 O（kn）。True.将每个整数表示为基数 n 中的 k 位数，运行时间为 O（n＋kn）＝O（kn）。**(d)**在双重哈希中，我们选择函数 h1 和 h2，并使用散列函数 h（k，i）＝h1（k）＋i＊h2（k）（mod m），其中 m 是素数。在类中，我们研究了 h2（k）∈{1，... ，m-1}。如果对于某些键 k，h2（k）＝0，则将 k 插入哈希表将总是正确地运行（尽管它可能很慢）。False.如果 h2（k）＝0，则探测序列由单个整数重复组成，因此当尝试插入元素时，我们可能无法找到空槽。**(e)**用于乘以两个 n 位数字的 Karatsuba 的算法在时间复杂度为 O（3log2（n））。True 运行时间为 $O(3^{\log_2(n)})=O(n^{\log_2(3)})$.**(f)**假设牛顿法用于求解 f（x）＝0 的方程，其中 f 是具有给定整数系数的多项式。 假设算法产生序列[x0，x1，...]，并且假设对于初始猜测 x0，我们有 f（x0）＝0。然后，对于所有 i＞0， f（xi）＝0。False 在算法过程的任何时间，给定的切线完全可能与 x = 0 处的 x 轴相冲突。**(g)**当在图 G =（V，E）上运行 DFS 时，每个顶点 v∈V 被分配开始时间 v.d 和结束时间 v.f。令 v 和 w 为 V 中的不同顶点。不可能有 v.d＜w.d＜v.f＜w.f。True 这是 parenthesization 引理。**(h)**令 G 是具有正边权重的 DAG。 令 x，y 和 z 在 G 中是不同的顶点，使得 y 在从 x 到 z 的最短路径上，并且假设存在从 x 到 y 的多个最短路径。 然后，有从 x 到 z 的多个最短路径。True 设 P1 和 P2 是从 x 到 y 的两个最短路径，并且令 P3 是从 y 到 z 的最短路径。 然后 P1 P3 和 P2P3 必须都具有相同的长度，因此都是从 x 到 z 的不同的最短路径。**(i)**在 Floyd-Warshall 算法中，我们通过考虑子问题“什么是从 vi 到 vj（使用最多 k 个边）的最短路径”获得用于所有对最短路径问题的 O（V3）算法。False.正确的子问题描述是“从 vi 到 vj 的最短路径是什么，vi，vj 是使用{v1，...，vk}中的中间顶点”(j)令 G 是具有任意（可能为负）边权重的无向图。 假定 P＝NP。确定 G 是否具有负权重周期的决策问题是 NP-hard。False.Bellman-Ford 算法检测多项式时间中的负周期，因此假设 P＝NP，这个问题不是 NP-hard。

2011

5. Taxachusetts 给定公路的权重图 $G=(V,E,w)$，如果路径边的数目大于 10，成本加倍。解释如何减少… 解：Augment to keep track of path lengths between each pair. We also augment so we keep track of the shortest-path weight between each pair using fewer than 10 edges and using greater than 10 edges. **6.Does this path make me look fat** 加权有向图 $G=(V,E,w)$，*fatness* 为路径中权重最大的边，给出算法找出从 u 到 v 的路径中最小的 *fatness* 解：因为 *fatness* 必定是一条边的权重，所以我们将所有边按权重排序，然后使用二分查找。为了测试是否存在 *fatness* 不超过 x 的路径，我们使用广度优先搜索，只遍历权重小于等于 x 的边。如果我们到达 v，那么这条路径存在。如果存在这样一条路径，我们在当前搜索的下部进行递归，判断是否存在更小的符合条件的 *fatness*。如果不存在这样的路径，我们在当前搜索的上部进行递归，来扩张边界。当我们找到两个邻近值，其中一个起作用，一个不起作用，我们便可以得到答案。这花费的时间为 $O((V+E)\lg E)$

7.Indiana Jones and the Temple of Algorithms 设计一个算法将砖分成两堆，使两堆的权重之和尽可能接近 解：只考虑权重和比较小的那一堆砖。目标是使这堆砖的权重尽可能接近 W/2，但不能超过 W/2。我们猜测每一块砖是否属于这堆之中。记子问题为 $P(i,w)$，若一些子集（砖 1~i）使堆的权重 $w\le W/2$，那么为真，否则为假。初始 $P(0,0)$ 为真，对于所有 $w>0$，$P(0,w)$ 为假。记砖 i 的权重为 w_i，用下面的递归式：$P(i,w)=P(i-1,w)\vee P(i-1,w-w_i)$ 如果 $w_i>w$，$P(i-1,w-w_i)$ 为假。如果 $P(i,w)$ 为真，在一个单独的表 $B(i,w)$ 中记录，其中 $P(i-1,w)$ 或 $P(i-1,w-w_i)$ 为真（如果他们都为真，任意选择一个）。在 $P(n,w)$ 中找到最大的 w 值。然后用表 B 回溯确定哪些砖是属于这堆的。这个算法运行时间为 $O(nW)$，因为有 nW 个子问题，每个子问题花费常数时间。

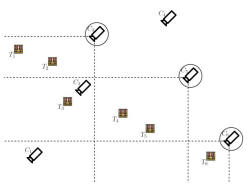
2012　3. 简答

(a)定义 SUPER-HEAP 数据结构具有如下性质：给定 n 个数字键的未排序数组时，支持 O(n)初始化…解释为什么 SUPER-HEAP 不可能存在。解：使用基于比较的算法不可能在 o（n log n）时间内进行排序。 但是如果使用 SUPER-HEAP（通过构造 SUPER-HEAP 然后重复地去除最小元素 n 次）来进行分割，则我们将具有基于 O（n）比较的排序算法，矛盾。

(b)一个使用哈希函数的大小为 m 的哈希表 T，并使用链表进行解决冲突。T 最初为空。四个不同的键顺序插入到 T 中，一个链的大小正好是 3 的概率是什么？ 解：第一个元素可以在表中的任何位置。然后，其他两个元素必须去到同一个槽，这发生在每个元素的概率为 1/m。现在，第四个元素必须去到不同的槽，其发生概率为 m-1 /m。所以概率是：$4(m-1)/m^3$

(c)令 G 是具有正边缘权重的无向图，并且令 s 和 t 是 G 中的两个不同的顶点。令 d 是使用 Dijkstra 算法找到从 s 到 t 的最短路径时松弛的边的总数。令 b 是当使用双向 Dijkstra 算法找到从 s 到 t 的最短路径时松弛的边的总数。总是真的 b≤ d（不考虑 G，s 或 t 的选择）？ 如果是这样，请给出简短的解释。如果没有，请提供反例。解：不正确，比如 G 是在五个顶点[s，u，v，t，w]上的一条路径。**(d)**Peter Looper 声称找到了一个在运行多项式时的归约 f 存在这样的规约 f？ 如果存在，解释为什么我们知道 f 存在。 如果没有，解释为什么 f 不能存在。解：f 不存在。 halting problem 是不可解决的，但是 clique problem 在指数时间内是可解的。使用归约可以允许我们通过求解 clique 实例来测试一个给定的程序是否停止，那么这允许我们解决 halting problem，矛盾。

7.Power to the Network　沿66号公路的2451miles有微波塔，允许信息从一端传输到另一端。用于从66号路的一端向另一端传输信号的功率与之成正比 $P=\sum_{i=1}^{n-1}(x_{i+1}-x_i)^2$ 须设计一个信号塔位置数据结构，以维护塔的位置，并支持以下操作： STL.insert(x)和 STL.getpower()。解：使用 AVL 树，并用 P 的一个整数值来扩充它（换句话说，将一个数字添加到数据结构中，而不是每个节点一个数字）。在将新塔保留在位置 x 时，重新计算值 P 并更新它。 在插入 x_i 时，我们可以花费 O（log n）时间查看 AVL 树中的 x_{i-1} 和 x_{i+1} 的前序和后序更新 P 值，然后加上 $(x_i-x_{i-1})^2+(x_{i+1}-x_i)^2$ 并减去 $(x_{i+1}-x_{i-1})^2$



11.Ganon, the Barbarian 他的 n 个宝藏位于固定位置 T = {T1,…,Tn}，每个摄像机只能面向左下方向安装。对于相机位置 C，将 RLA（C）定义为左侧和上侧 C 的任何宝物 T∈T 的最右侧位置。如果不存在这样的宝物，则定义 $RLA(C)=\phi$。 (a)记下以值{M [Tj]：j <k}和相机安装价格 pi 表示 M [Tk]的递归。解：$M[T_k]=\min_{C\in c:C\,guards\,T_k}\{p_C+M[RLA(C)]\}$ (b)读取下面的伪代码，然后填充缺少的部分 1.设 A 是 AVL 树（最初为空），其中 A 中的每个节点存储在平面中的位置。 使得如果 N 是包含点（x，y）的节点，则 key[N] = ____x____。2. 令 R = C∪T 是包含所有相机和宝藏的集合。3. 以 y 坐标的降序对 R 进行排序（即，从顶部到底部）。4. 对于每个点 a∈R（按照从上到下的顺序），执行以下操作：5. 如果 a 是宝藏位置，请将 a 插入 A. 6. 如果 a 是相机位置 C，则如下计算 RLA（C）：解：对关键字小于或等于 C 的 x 坐标的最右边节点执行 AVL 搜索

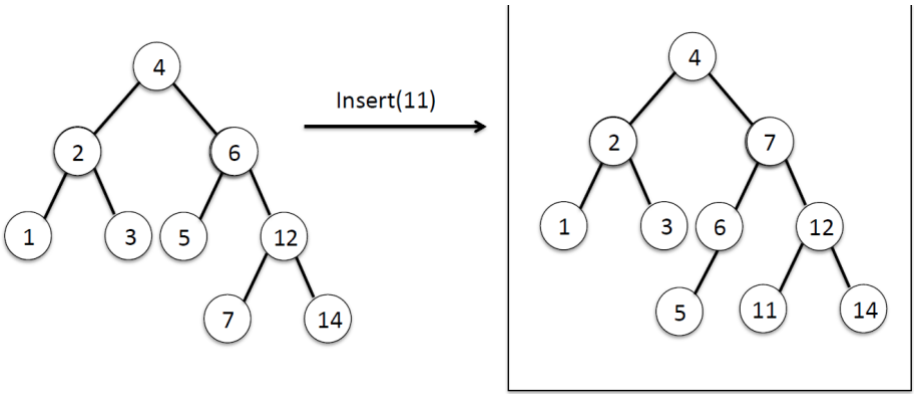
12.Optimizing Santa 圣诞老人希望为孩子分配玩具，以最大化总幸福：$H=\sum_{i=1..n}^{`}\sum_{j=1..n}^{`}x_{ij}h_{ij}$。其中如果孩子 ci 接收到玩具 tj，则 xij = 1，否则 xij = 0。 **(a)**可以使用动态规划来计算最大聚合幸福度 H。令 $H[i,j,k]$ 表示如果从集合{c1，...，k}中至多 k 个孩子（其中 1≤k≤q）从集合{t1，...，tj}分到的玩具的最大聚合幸福度，同时服从无交叉约束。 为 $H[i,j,k]$ 写出递推关系。包括基本案例。

解：
$$H[0,j,k]=H[i,0,k]=H[i,j,0]=0$$
$$H[i,j,k]=\max(H[i-1,j,k],H[i,j-1,k],H[i-1,j-1,k-1]+h_{ij}),\forall i,j,k\geq 1$$

(b)使用大 O 符号填充下面的空格：子问题的总数是 $O(n^2q)$。在每个子问题中评估 H［i，j，k]所需的时间是 $O(1)$。计算 H 所需的总时间是 $O(n^2q)$。13.The Longest Quiz Path in a Tree 设 T =（V，E）是具有非负边强度的无向树（没有回路的连通图）。 设计和分析一种高效算法来确定 T 中最长路径的长度，例如 $\max_{u,v\in V}\delta(u,v)$ 的值，其中 δ（u，v）表示从 u 到 v 的最短路径的长度。对于满标记，需要 O（V）算法。解：首先，选择任意顶点 r∈V 作为 T 的根，并运行 DFS 或 BFS 以将所有边缘向外引导，以便我们现在具有根树。 对于任何给定的路径 P，在该树中存在唯一的“最高”节点，称为 r。（路径从某处开始，沿树向上，然后下降。）在这种情况下，我们说 P 根在 r。
现在，对于每个顶点 v，将 P［v]定义为找到从 v 到以 v 开始的子树中的某个叶的最长路径的子问题。然后我们得到递归：$P[v]=\max_{u}(P[u]+w_{u,v})$ 这是具有 O（V）子问题的 DP，其中每个子问题需要 O（V）时间来评估。然而，注意每个边仅有一个评估，因此计算需要总共 O（E）时间。 因为这是一棵树，｜E｜=｜V｜-1，因此 O（E）= O（V）。接下来，我们要计算 L［v]，最长路径的长度固定在 v。首先，我们声明最长路径必须从一个叶子到另一个叶子。 如果不是，我们可以通过包括邻近非叶的另一边缘来做出更长的路径。因此，为此，对于每个孩子 u，认为量 P［u] + Wu，v。 让 u1 和 u2 是这个值最大的两个孩子。 然后，我们有 $L[v]=P[u_1]+w_{u1,v}+P[u_2]+w_{u2,v}$ 计算所有 L［v]值需要 O（V），与 P［v]的参数相同。最后，我们得到最长路径的长度只是 $\max_{v} L[v]$

2013 1.判断 (a)如果 f(n)=O(g(n))和 g(n)=O(h(n)),那么 f(n)=O(h(n)).对.我们有 f(n)≤ r1·g(n)+c1 且 g(n)≤ r2·h(n)+c2,表示 f(n)≤ r1·(r2·h(n)+c2)+c1=r1·r2·h(n)+c1+r1·c2 且 f(n)=O(h(n)) (b)主定理用来解决复发 T(n)=2T(n/2)+n/log n.错.该方程不满足任何情况。 (c)可以在 O（n）时间内对 0 和 n100 之间的任何 n 个整数进行排序。对,n 的基数排序。 (d)给定 n=2 个排序元素 A[0：n=2]和 n=2 个排序元素 A[n=2：n]在单个数组 A［0：n]（如合并排序）中，合并算法讲座通过移动 A 中的元素同时使用 O(1)额外空间来排序 A.(到位).错，在这里提出的合并算法使用了额外的大小为 O（n）的数组。 (e)二次探测（使用散列/碰撞的开放寻址方案对于常数 c1，c2,函数 h（k,i）= h（k）+ c1i + c2i2）满足均匀性散列假设：每个密钥 k 的探测序列同样可能是任何一位时隙 f0 的置换;{0,1,...,m-1}.错.具有相同散列 h（k）的 k 的所有值最终具有导致二次聚类的相同探测序列。 (f)在一个用杜鹃哈希维护的哈希表中，最坏的情况下运行搜索元素的时间为 O（1）。对.只有两个地方需要被搜索。 **(g)考虑具有开放寻址和线性探测的哈希表.任何从插入和删除到空表中产生的表也可以只是通过插入一个空表来创建。**错 (h)给定有向图 G=(V,E)，我们可以（使用呈现的算法在 6.006）确定 G 是否具有 O(V + E)时间中的有向循环。对.使用 DFS 可以查找返回边。 (i)给定没有负权重的加权有向图=(V,E,w)边缘，我们可以（使用 6.006 中提出的算法）计算最短路径在 O（V + E）时间中从节点 s 到节点 t 的权重。错.因为可能有循环，我们可以使用的最好的算法是 Dijkstra 算法，当实现 Fibonacci 堆时运行时间为 O(V log V+E)。(j)在没有周期和所有边权重的加权图 G=(V,E,w)中负，我们可以运行 Dijkstra 的算法在一些图 GO 找到最大值在 G 中从节点 s 到节点 t 的路径的可能权重。对.在 w 中取消权重。(k)如果你被告知 SAT（布尔可满足性问题）是 NP 完成，并且您发现从 SAT 到 Hamiltonian Path 的多项式时间减少,你可以得出结论，哈密尔顿路径是 NP-hard。对.如果哈密尔顿路径不是 NP-hard，那么多项式时间从 SAT 到哈密尔顿路径的减少意味着 SAT 也不是 NP-hard，这是一个矛盾。(l)给定两个整数 a,b 其中 a <m，我们可以（使用提出的算法在 6.006 中）计算 O（(lg m）lg 3 lg b）位操作中的 ab modm。对.我们可以使用 Karatsuba 和重复平方。(m)给定一个 n 个非零独立整数的数组 A，我们可以计算 O（n）时间一个稳定的符号排序，即 A 的排列 AO，使得所有的负数整数出现在所有正整数之前，并且所有整数具有相同的符号，出现的顺序与它们在 A 中的顺序相同。对.在符号是键的地方使用修改的计数排序。

2. AVL Practice 假设你有如下所示的 AVL 树。 绘制由于将 11 插入树中而产生的 AVL 树。 没有必要展示中间工作：我们将仅查看最终的树配置。解：如图 3. Dynamic Programming vs. Shortest Path in DAG 通过计算子问题依赖 DAG（具有适当的加权边缘）中的最短路径，可以解决讲座中的以下哪些动态程序？ 在每种情况下，“可以”或“不能”。 4. Chaining vs. Linear 教授 Sauron和 Saruman 在如何最好地组织一个哈希表有不同意见，将他们的记录存储在 Middle Earth 的 n 个人身上。计算列表中的每一个中的总渐近运行时间（使用Θ符号）。对于链接，假设插入到长度为k 的链中会花费Θ（k）时间



(a) Fibonacci numbers	can	cannot
(b) text justification	can	cannot
(c) parenthesizing / matrix multiplication	can	cannot
(d) edit distance	can	cannot
(e) knapsack	can	cannot
(f) piano/guitar fingering	can	

Sequence	Chaining	Linear Probing
$1, 2, 3, \ldots, n$	$\Theta(n)$	$\Theta(n)$
$m, 2m, 3m, \ldots, n\,m$	$\Theta(n^2)$	$\Theta(n^2)$
$1, 2, \ldots, \frac{n}{2}, m+1, m+2, \ldots, m+\frac{n}{2}$	$\Theta(n)$	$\Theta(n^2)$
$1, m+1, 2, m+2, \ldots, \frac{n}{2}, m+\frac{n}{2}$	$\Theta(n)$	$\Theta(n^2)$

7.Ternary Search 考虑以下算法用于在有序数组 A [i：k]中搜索元素 x。 不是将数组分成两半，而是在二分之一搜索(a)记下 T（n）的重复，TERNARYS EARCH 在 TERNARY-S 开始时调用的确切次数 EARCH（x，A，0，n）（计算初始调用）解：T(n)=T(n/3)+1 ， T(1)=1

（b）求解 T（n）。解：$T(n) = 1 + \ell = 1 + \log_3 n$

9. Binary Search as a DP Bottoms-Up教授发现了一种新的二叉搜索方法：动态规划！ 要在排序数组 A [0：n]中搜索数字 x，他使用以下动态程序：(a)如果 Bottoms-Up 教授以自下而上的方式实现这个动态程序样式（扩展循环与循环，替换递归调用与表查找），什么会导致渐近运行时间？解： （所描述的自上而实现是：伪代码）子问题/迭代

1. **Subproblems:** For all $0 \le i \le j \le n$, BS(i, j) = the index interval $A[i : j]$.
2. **Guessing:** None.
3. **Recurrence:**

```
BS(i, j)
m = ⌊(i+j)/2⌋.
if j − i ≤ 1:
    return i
elseif x < A[m]:
    return BS(i, m)
else return BS(m, j)
```

4. **Ordering Loop:**

```
for d = 1, 2, ..., n:
    for i = 0, 1, ..., n − d:
        solve BS(i, i + d)
```

5. **Original problem:** BS$(0, n)$

Solution: $\Theta(n^2)$. The described bottom-up implementation is

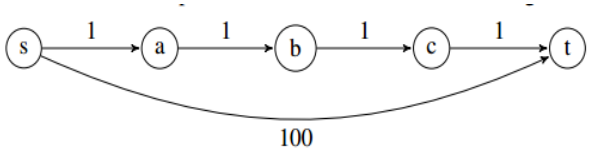
```
for d = 1, 2, ..., n:
    for i = 0, 1, ..., n − d:
        j = i + d
        m = ⌊(i+j)/2⌋.
        if j − i ≤ 1:
            BS(i, j) = i
        elseif x < A[m]:
            BS(i, j) = BS(i, m)
        else BS(i, j) = BS(m, j)
```

This code has

$$\sum_{d=1}^{n}(n-d+1) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \Theta(n^2)$$

（并且遍历所有这些子问题），并且每个子问题花费Θ（1）时间，因此运行时间是Θ（n2） (b)如果Bottoms-Up 教授以自上而下的方式实现这个动态程序，递归样式（将 Recurrence 实现为具有记忆的递归算法），将会是什么会导致渐近运行时间？解：递归算法仅访问 1 + lg n个子问

题。对于二进制搜索：每个子问题只调用一半。事实上，记忆没有做任何事情，因为没有子问题的解决方案需要重新使用。因此，Bottoms-Up 教授应考虑另一种高效二进制的搜索方式：动态编程！ **10. Must-See T.V.**假设给定一个带有非负权重的加权有向图 G＝（V，E，w），并给出一个源节点 s∈V 和一个目的节点 t∈V.此外，有两个红边 e1， e2，每次必须至少遍历一次，尽管顺序并不重要。 给出一个有效的算法来找到从 s 到 t 的包含两个红色边缘的最短路径的权重（如果没有这样的路径，则为∞）答案：这是一个图形转换问题。 调用两个红色边缘 $A=(u_A,v_A)$ 和 $B=(u_B,v_B)$。我们创建一个新的图 G`，包含四个（未修改的）$G:G_\Phi,G_A,G_B$和$G_{AB}$的副本。 G 的下标表示我们已经经过了哪些边。为了强化这个属性，我们将边界从 G0 中的 uA 添加到 GA 中的 vA，将 GB 中的 uA 添加到 GAB 中的 vA，将 G0 中的 uB 添加到 GB 中的 vB，将 GA 中的 uB 添加到 GAB 中的 vB。 这些添加表示分别 G∅,GB 到 GA，GAB 中的的边缘 A 的副本，使得获得具有下标 A 的副本的唯一方式是遵循 A 的副本，并且类似地用于 B.现在，我们可以简单地在 G`上运行 Dijkstra 算法，其中 s'是 G 中的 s 的副本，t'是 GAB 中 t 的副本。 然后，得到的路径必须根据需要经过 A 和 B. 该方法具有等于 Dijsktra 算法的运行时间（即 O（（V＋E）log V）），因此是有效的。**11. Flying Home for Winter Break** 您刚刚获得了代表航空公司航线网络的加权图 G＝（V，E，w）：顶点表示城市，边缘表示航班路线，边缘权重表示票价。(可悲的是，所有的边权重都是正的。)你想用最小总价通过一个有向路径从 s＝波士顿∈V 到 t＝斯普林菲尔德∈V，但你只愿意乘 3 航班。 您的目标是在 O（V＋E）时间内计算从使用最多 3 个边的路径中的 s 到 t 的最小成本路径。(a)你的第一种方法是运行 3 次 Bellman-Ford 迭代。开始距离估计 d [v] ＝∞，除了 d[s]=0; 然后你考虑 E 的所有边缘;给一个反例，其中 d[t]将不包含使用最多 3 个边的从 s 到 t 的最小代价路径的权重。解：顺序很重要！考虑下面的图表，假设我们运行 Bellman-Ford 的 3 次迭代，首先考虑 s→a，然后 a→b，然后 b→c，然后 c→t，最后 s→t。在一次迭代之后，我们将具有 d[t]=4，4 边缘路径的成本 s→a→b→c→t。 经过两次迭代后，我们仍然有 d [t] ＝4。因此，在 Bellman-Ford 的 3 次迭代之后 d[t]≠100，其中 100 是 s→t 的长度，从 s 到 t 的最小成本路径，使用最多 3 个边。



(b)你的第二种方法是运行 BFS 来定位从 s 到达的所有顶点，路径最多为 3 个边，删除所有其他顶点及其入射边，然后在结果子图上运行 Dijkstra。 给出反例，其中 d [t]将不包含使用最多 3 个边的从 s 到 t 的最小代价路径的权重。解：（a）的反例在这里仍然有效。每个顶点是可达的，从 s 到最多 3 个边缘的路径（例如，c 恰好 3 个边缘），所以没有顶点或边缘被去除。 在运行 Dijkstra 在得到的子图，我们将再次具有 d [t] ＝4。(c)给出一个有效的算法来，使用最多 3 个边计算路径从 s 到 t 的最小成本权重。 为了更可靠，你的算法应该运行 O（V＋E）时间。解:我们使用动态规划解决方案。然后我们讨论常见的不正确的方法和他们反例。第一：我们可以使用使用 DP 算法的略微修改的版本，经过最多 3 个边，找到从 s 到 t 的最小成本路径的权重的最短路径。子问题是 δk（s，v），在 k 个边，从 s 到 v 的最短路径的权重。 当计算 δk（s，v）时，我们猜测路径中的最后一条边（u，v）。这导致以下复现：$\delta_k(s,v)=\min_{(u,v)\in E}\{\delta_{k-1}(s,u)+w(u,v)\}$和基本条件：$\delta_0(s,s)=0,\delta_0(s,v)=\infty(v\neq s)$原来的问题是$\delta_3(s,t)$。我们现在分析我们的 DP 算法的运行时间。 给定 δk-1（s，•）时计算 δk（s，v）所花费的时间是 Θ（1 + indegree（v））。 因此，对于每个顶点 v∈V 计算 δk（s，v）所花费的时间是：$\sum_{v\in V}\Theta(1+\text{indegree}(v))=\Theta(V+E)$

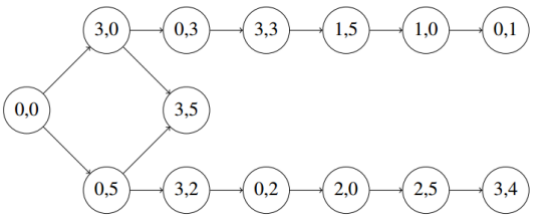
最后，计算 δ1（s，•），δ2（s，•）和 δ3（s，•）所花费的时间，进而我们可以知道 δ3（s，t）是 Θ（3•（V + E））= Θ（V + E）。注意：这相当于运行具有同步边的 Bellman-Ford 的 3 次迭代 - 也就是说，我们只使用前一次迭代的 δ[u]来计算下一个的 δ[v]。

2014 **1. 多选**(a)关于下面的递归式的说法哪些是正确的？i) T(1) =0(1)，T(n) =2T(n=2)+0(n) for n > 1. 结果：T(n)=0(n2);ii) T(1) = 0(1)，T(n) = 8T(n=2)+0(n2). 结果：T(n)=0(n3);iii) T(1) = 0(1),T(n)=7T(n=2)+0(n2). 结果：T(n) = 0(nlog2 7);iv) T(1) = 0(1)，T(n)=8T(n=2) + 0(n3).结果：T(n) = 0(n3 log n)v)T(1)=0(1),T(n) = T(n=2) + 2T(n=4) + 0(n).结果:T(n)=0(n2)答案:234 能够根据主方法判断。有一些例子:(i)是归并排序,(ii)是 brute-force 矩阵乘法,(iii)是 Strassen’s 矩阵乘法.(v)应该是 $O(n)$ (b)下面关于排序算法的描述哪些是正确的？n 是元素个数.1.归并排序最坏情况下的时间复杂度是 Θ（n log n），如果输入已经有序的话，为 Θ（n）。2.插入排序最坏情况下的时间复杂度是 Θ（n^2），如果输入已经有序的话，为 Θ（n）。3.基于大顶堆的堆排序最坏情况下的时间复杂度为 Θ（n log n），如果输入已经降序排列，为 Θ（n）。4.二叉搜索树排序最坏情况下时间复杂度为 Θ（n log n），输入有序则为 Θ（n）。5.基数排序最坏情况下为 Θ（dn） 答案：25.1 错，因为不管输入是否有序，归并步骤都需 Θ(n)时间，共需 log n 趟归并，故整个归并排序需 Θ(n log n)。3 错，因为无论输入是否有序，每次取出最大元素后需要 Θ(log n)去调整，意味着整个算法为 Θ(n log n).

4 错，因为往二叉搜索树中有序插入 n 个元素需要 $\Theta(n^2)$ 而不是 $\Theta(n \log n)$：这是最坏情况，非平衡树。如果用平衡二叉树，最坏情况确实需 $\Theta(n \log n)$ 时间建造树，但无论输入是否有序，都是 $\Theta(n \log n)$。(c)在二叉搜索树上，下面的说法是否正确？n 是节点数，h 是树高。1.在 $O(\log n)$ 时间查找中间值。2.在 $O(n)$ 时间，返回一个所有关键字的有序序列。3.在 $O(h)$ 时间，查找所有关键字的均值。4.在 $O(h)$ 时间，查找第 n/4 大的关键字。5.在 $O(n)$ 时间，查找所有在 a 和 b 之间的关键字，a 和 b 是随机的。答案：245。1 错，因为需要 $O(h)$，而 h 不一定是 log n，普遍来说，查找第 k 大的元素都需要 $O(h)$ 时间，所 4 是对的。2 和 5 对，因为可以枚举树的元素在 $O(n)$ 时间，在这个过程中可以找到 a 和 b 之间的元素。3 错，因为计算均值需要查找每一项，所以需要 $\Omega(n)$ 时间。(d)下面关于平衡二叉树（AVL）的说法哪些正确？1.中值一定出现在顶端节点。2.任一节点的关键字一定比它的子节点的关键字大。3.能够在 $O(n)$ 时间内产生树中的关键字有序。4.在一次插入之后重新平衡能够在 $O(1)$ 时间完成。5.任一关键字 k，能够在 $O(\log n)$ 时间内找到大于 k 的最小值。答案：35。 1 和 2 明显错误。2 是大顶堆的性质。4 错，因为，尽管我们仅仅执行一次单一的旋转，但是我们需要一直查找到根节点发现非平衡因子，因此需要 $O(\log n)$ 时间。5 对，因为查找 k 时，将可以找到 k 的前驱或后继节点，若果我们在前驱节点停止，我们能够查找到后继节点在 $O(\log n)$ 时间内。(e)一个用链式处理冲突的哈希表。哈希表有 m 个位置 0,。。。，m-1。

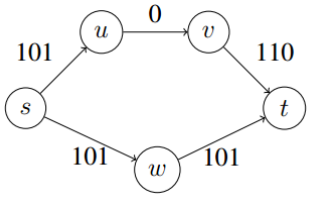
现在相继有 n 个插入操作，插入的关键字分别为 $k1; k2; : : : ; kn$，接下来有一次查询操作。下面那些是对的？1.假如有一半插入表中 0 位置，一半插入表中 1 位置。查询操作要随机查一个关键字，这个关键字可能可能在表中任一位置，那么期望的查询操作能够在 $O(n/m + 1)$ 时间内完成。2.假如插入的关键字散列到表中随机位置。查询操作与 1 中一样，那么期望的查询操作能够在 $O(n/m + 1)$ 时间内完成。3.假如有一半插入表中 0 位置，一半插入表中 1 位置。现在一直查询操作成功了，即要查询的关键字在表中，要查询的关键字可能是表中的任意一个。那么期望的查询操作能够在 $O(n/m + 1)$ 时间内完成。4.假如插入的关键字散列到表中随机位置。查询操作与 3 中一样，那么期望的查询操作能够在 $O(n/m + 1)$ 时间内完成。5.现在假设插入满足简单均匀哈希假设，查询操作与 3 中一样，那么期望的查询操作能够在 $O(n/m + 1)$ 时间内完成。答案：125。12 对，因为对于随机哈希位置来说，期望的花费时间是列表的平均长度，即 n/m(无论关键字怎么分布).3 错，应为 $\Theta(n)$，5 包含 3 的情况，也应为 $\Theta(n)$。(f)下面关于牛顿方法哪些正确？1.牛顿方法对于平方根总是收敛于一个积极地初始猜测。2.牛顿方法对于平方根收敛的充分接近最初的猜测。3.牛顿方法对于倒数总是收敛。4.牛顿方法对于倒数总是收敛的充分接近最初的猜测。5.牛顿方法对于倒数总是平方收敛充分接近初始猜测。答案：1245 对。(g)下面对于无向图的深度优先搜索产生的结构说法哪些一定正确？1.所有边都是树的边或者 back edges，并且没有交叉边。2.图中能够互相到达的两个节点 u 和 v 一定在同一个树中。3.如果 G 是连通图，那么深度优先搜索（DFS）森林就是一个树。4.如果 u 是 DFS 森林一些树的根节点，并且 u 从 G 中断开，那么 G 的 DFS 树有超过一个孩子。5.如果 u 是 DFS 森林一些树的内部节点，并且 u 从 G 中断开，then no child of u in its DFS tree has a back edge to a proper ancestor of u。答案：1234 对。(h)下面关于拓扑排序哪些正确？1.拓扑排序排序一个长度为 n 的数组，时间为 $O(n)$。2.拓扑排序排序任一图中的节点，如果有从 u 到 v 的路径，那么排序结果 u 在 v 的前面。3.DAG 的拓扑排序能够用 DFS 实现，复杂度为 $O(V + E)$，基于在 DFS 时第一个节点。4.拓扑排序能够被用来作为一个子程序去发现在权重 DAG 中的最短路径，复杂度为 $O(V + E)$；特别的，这个时间不取决于边上权重的数量级，并且边上的权重可以是负值。5.当拓扑排序被用来发现在权重 DAG 中的最短路径时，没有放松步骤是必要的。答案：4 对。1 错，因为拓扑排序只能应用于图，不能用于数组排序。2 错，因为拓扑排序只能用于有向无环图（DAG），不能是任一图。3 错，因为我们需要用的时间是 DFS 处理完每一个节点的时间。4 对，5 错。(i)减少允许将一个问题 Q 用作子程序来帮助在解决另一个问题 P.要测量 Q 提供的帮助的数量，我们假设 Q 的答案是免费的，除了生产所需的任何时间子程序的输入和处理输出。 我们分析剩余时间使用 Q 来求解 P 的算法，并将其与最佳已知时间进行比较以求解 P 从头开始。 也就是说，我们考虑求解 P 相对于 Q 的成本并进行比较对于自己解决 P 的成本。以下哪项代表有帮助的减少，也就是说，它们说明了情况其中 Q 减少解出 P 所需的已知时间？1.Q 是布尔公式的满足性问题.P 是旅行推销员的问题 2.Q 是具有非负权重的加权有向图的单源最短路径问题。 P 是旅行推销员的问题。3.Q 是旅行推销员问题。 P 是布尔公式满足性问题 4.Q 是在具有非负权重的连接加权有向图中找到从节点 s 到节点 t 的最短路径的问题。 P 是相同的问题，但是对于连接的加权无向图。5.Q 是从给定的整数数组生成任何二叉搜索树的问题。 P 是排序数组的问题答案：1345.1 和 3 是对的，因为这两个问题都是 NP 完成的.4 是对的，因为我们可以在 $O(|V| + |E|)$ 时间中创建等效有向图，而用于该问题的最快已知算法需要 O（|V |log |V j|+ |E |）时间.5 是对的，因为如果给定二叉排序树，我们可以在 O（n）时间中产生排序好的数组，而所有已知的排序算法采用（n）时间。

问题 3：假设你给了有一个 3 加仑的水壶和一个 5 加仑的水壶，你想在 5 加仑水壶里最终得到 4 加仑。水壶开始为空。

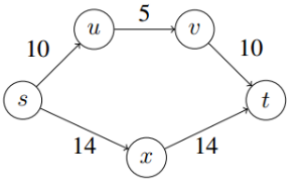


你被允许做的是：1. 倒空一个水壶 2. 装满一个水壶将一个水壶的水倒入另一个水壶，直到第一个水壶是空的或第二个水壶满了

给出将产生所需结果的最短移动序列。 将系统的“状态”表示为数字对（a, b），其中 0≤a≤3 和 0≤b≤5，表示两个水槽中的水量。起始状态为(0,0)。展示您的工作。答案：广度优先搜索以下图所示的方式从状态（0,0）开始发现节点：垂直排列的节点与（0,0）的距离相等。 注意，这个图片在图中没有显示从后面的节点返回到较近的节点的任何边缘，因为没有最短路径可以使用这样的节点。这个图表显示到解决方案的距离是 6.（BFS 之后会发现另外两个节点，但是我们可以停止在这一点，因为我们已经找到了解决方案，任何后来发现的节点会更远）。**问题 4：**假设我们有一个有向图 G 在边上有（可能是负的）整数权重。 我们有兴趣确定从特定源节点 s 到特定目标节点 t 的最短路径。 如果没有从 s 到 t 的路径，则计算的距离应该是∞，而如果存在任意的负长度的路径，则计算的距离应该是-∞。这里是一个建议的算法来解决这个问题。 令-w 是 G 的任何边上的最大负权重。构造具有与 G 相同的节点和边的不同权重的新图 H：H 中的每个边的权重仅是其在 G 中的权重加 w。 因此，H 中的所有边的权重是非负的。 然后在 H 上运行 Dijkstra 的最短路径算法，找到 H 中从 s 到 t 的最短路径，并返回该路径。这个算法对于解决原始问题是否正确？ 或者给出非正式的正确性论证，或者给出一个小的反例（最多 5 个节点）。答案：这个算法是不正确的。看下面的例子：可以看出最短路径是 s,u,v,t，然而，当把每个权重加 100 后，图变成下面这样：很明显，原来长的路径现在变短了，这是不对的。这个转换失败的原因是添加到每个路径的权重与路径中的边的数量成比例。 因此，这可以使得一些较长的路径变得较短，仅仅因为它们具有较少的边（即使这些权重具有更多的总权重）.



问题 5:在演讲中，我们讨论了使用 Dijkstra 找到有向图 G 中从某个节点 s 到某个其他节点 t 的最短路径。 该方法包括从 s 开始的 Dijkstra 算法和从 t 开始的“反向”图（具有反向边缘）上的 Dijkstra 算法的交替步骤，直到第一次单个节点 x 出现在两组处理的节点 S s 和 St.(a)绘制一个小图形，显示我们不一定通过简单地连接已经找到的从 s 到 x 的最短路径和已经找到的从 x 到 t 的最短路径来获得最短路径。答案：我们可以看到 x 距离 s 和 t 的距离为 14，x 将是出现在 Ss ∩St 中的第一个节点。在那一点上，我们将有 Ss = {s, u,x}，因为 u 比 x 更接近 s，并且 St = {t,v,x}，因为 v 比 x 更接近 t。而最短路径是 s，u，v，t 是 25.



(b)解释为什么一定存在从 s 到 t 的使用一些边（u,v）的最短路径，其中 u 是在 Ss 中，v 是在 St 中，也就是说，u 已经从 s 中找到，v 已经从 t 中找到。 最短路径由已经找到的从 s 到 u，边（u,v）和已经找到的从 v 到 t 的最短路径的最短路径组成。答案：我们让 s,v1,…v_k,t 是最短路径，j 是 $v_1 \cdots v_t$ 的权重，其小于 s~x 的权重，而 $v_1 \cdots v_t, v_{t+1}$ 的权重大于或等于 s~x 的权重。可以看到 v_{t+1} 必须比 x 更接近于 t，否则从 s 到 t 通过 x 的路径的权重将小 于通过 v_{t+1} 的路径的权重，其被假设为最短路径。因为 v_{t+1} 比 x 更接近于 t，所以我们必须有 $v_{t+1} \in$ St.。因此，我们有 $v_t \in$Ss，$v_{t+1} \in$ St，它们之间的边（v_t，v_{t+1}）。(c)描述一个简单的算法来确定从 s 到 t 的最短路径，假设我们已经从两端执行了 Dijkstra 并且找到了第一个公共节点 x。 你的算法应该有一个合理的时间复杂度不差于 O（E）答案：经过每个 v∈V，计算 $ds[v] + dt[v]$，其中 d *分别是 Dijkstra 从 s 和 t 计算的距离，并返回计算的最小值。如我们在前面的答案中看到的，节点 v_{t+1} 与在 Ss 中的节点 v_t 相邻，因此，Dijkstra 已经在（v_t，v_{t+1}）处放松，这意味着 $ds[vt+1]$ 将是正确的。此外，由于我们有 $vt+1 \in St$，我们知道 $dt[vt+1]$ 是正确的。因此，$ds[vt+1]+ dt[vt+1]$ 是真实的最短路径距离。 并且由于 d 值总是下限，这对于任何其他节点不会大于相同的总和，因此返回值将是这个和，这是真正的最短路径权重。问题8：这个问题涉及 使用开放寻址策略来实现支持插入，删除和搜索操作的哈希表 T. 散列函数为 h：$\{0,\ldots,m-1\} \to \{0,\ldots,m-1\}$，将全集 U 的键值和索引映射到表中的位置。表中的条目可以是 U 中的键值，“空”指示符 nil 或“已删除”指示符 del。要使这些操作组合正常工作，您的算法应保留以下不变量：对于任何的关键字 k 和索引 i，如果 k 出现在表中，位置为 h(k,i)，那么不存在 $i' < i$ 使得位置 $h(k,i')$ 包含 nil。(a)通过填充四个空格，为查询操作完成以下伪代码，并使用不变量来非常简短地解释为什么如果这个键在表中就能保证查找到该键值。答案：我们知道这将找到键值，因为我们继续执行循环，除非我们看到“nil”，在这种情况下，不变式告诉我们 k 不在表中。Skip,return”not here”,skip,”not here”(b)填空完成下面插入操作的伪代码，并且简单的解释为什么这个操作保留了不变量。假设这个操作是插入关键字 k，k 还不在表中。答案：保留不变量原因:1 对于 k 本身:我们只跳过被键占用的位置，没有 nil 个位置。2 对于任何其他键 k':在插入之前不变量是真的，我们不会伪造它，因为我们不改变任何位置的内容为 nil。Skip, T[j]:=k;return; ,T[j]:=k;return;(c)填空完成下面删除操作的伪代码，并且简单的解释为什么这个操作保留了不变量。假设这个操作是删除关键字 k，k 在表中。答案：保留不变量原因：对于 k 本身：在这个操作后它不在表中，所以这个声称是真的。对于另一个 k'：和(b)中一样，我们不会将任何位置的内容更改为 nil。T[j]=del;return,skip,return,skip

2016 问题 1PD 下面哪一个渐进关系是正确的(a) if $f(n)=4n^2+5n+4 then f(n)=O(n^3)$ 对: $f(n)=O(n^3)$, 也是 O（n^3）$O(n^4)$ 或 n 任意更高值 (b) if $f_1(n)=5n+\log n$ 和 $f_2(n)=2n+9\sqrt{n}$, 那么 $f_1(n)\cdot f_2(n)=\Theta(n^2)$ 对 $f_1(n)\cdot f_2(n)=10n^2+45n\sqrt{n}+2n\log n+9\sqrt{n}\log n=\Theta(n^2)$ (c) if $f_1(n)=\Theta(n)$ 和 $f_2(n)=\Theta(n)$ 那么 $\sqrt{f_1(n)\cdot f_2(n)}=\Theta(n)$ 对： $f_1(n)\cdot f_2(n)=\Theta(n^2) and \sqrt{f_1(n)\cdot f_2(n)}=\Theta(n)$ (d) if $f(n)=n+1$ 那么 $n^{f(n)}=O(n^n)$ 错: $n^{f(n)}=n^{n+1}=n\cdot n^n=n\cdot\Theta(n^n)$ 问题 2PD （a） T(1)=0(1)， T(n)=3T(n/3)+0(n2) 对:这是 Master Theorem 的情况 3,在这里面顶级占主导地位.解:T(n)=0(n2) (b) T(1)=0(1),T(n)=8T(n/2)+0(n3).解:T(n) = $\Theta(n^3)$ 错:根据 Master Theorem 的情况 2，在这里所有等级分布类似(c) T(1)=0(1),T(n)=9T(n/3)+0(n)for n>1.解:T(n) = $\Theta(n^3)$ 错:根据 Master Theorem 的情况 1，在这里最低等级主导. (d) T(1)=0(1),T(n)=2T(n/3)+3T(n/9)+0(n3).解:T(n)=$\Theta(n^3\log n)$ 错:这个情况是顶级主导。问题 3 PD(当最大值堆 H 被看做为一颗树时，哪一个是真的状态？不需要理由.(a)树满足 AVL 平衡特性，意思是，对于在 H 中的每一个节点 u，u 的左子集的高度和 u 的右子集的高度最多相差 1。答:正确。最大值堆被作为数组保存起来，但被看最一颗几乎完整的二叉树。只有树的底层可能不完整，但是任意节点的两个子集的最大高度差是 1.(b)如果我们用 INSERT 操作添加一个新 key 到树中，在被移到它的合适位置之前，这个 key 最初是作为叶子被添加的。答:正确。一个新 key 是作为树的第（n+1）个元素添加进去的（这就是一个叶子），然后被提升（向上逆流）到它的最终位置。(c)给 n 个元素建立一个新堆，需要。。。时间。答:错误。第 4 课展示了分而治之的方法来为 n 个元素创造一个堆，花费直线时间。。（d）如果我们提高了在 H 中的一个单独节点 u 的 key 的值，那么 max-heapify 会在时间。。内为 n 个元素的堆恢复堆属性。答:错误。key 的提升有潜力去给最大值堆创造一个障碍，这个障碍被提升 key 来修复（“往上流”，和在 increase-Key 操作中一样）。这和“往下流”的 MAX-HEAPIFY 有点不同。问题 4 判断题 (a)为了找到 e^4 的值，我们可以运行牛顿方法在函数 $f(x)=\ln x-4$ 上，用一个合适的初始值 $x^{(0)}$。答:正确。牛顿方法可以被用来找到函数的根。 f 的根是 $f(x)=0$ 时 x 的值。令 $f(x)=0=\ln x-4,\ln x=4,x=e^4$ 因此，如果我们通过牛顿方法，从适当的初始值中找到一个 x 的值令 f(x)=0，那么这个 x 的值等于 e^4 （或十分接近）。(b)牛顿方法保证收敛，无论从哪个初始值 $x^{(0)}$ 开始。答:错.牛顿方法只在初始值足够接近函数的根的时候收敛，但是不保证从每一个初始值出发都收敛.(c)应用于函数的梯度下降算法 $f(x)=(x+1)^2(x-1)(x-2)=x^4-x^3-3x^2+x+2$ 保证收敛到函数的一个全局最小值。答:错.f(x)是一个四次多项式，而且至少有一个局部最大值(在它的根 x1=-1 和 x=1 中).所以，这个最大值是一个临界点，如果这个梯度下降算法从这个值出发，那么它会停留在那里(因此也不会收敛到任意最小值).另外，注意到多项式 f(x)的表达式告诉我们，它有两个一个双根 x1=-1，和两个根 x2=1 和 x3=2。因为领先系数为 1，x1 是一个局部最小值令 f(x1)=0，另一个最小值在根 x2 和 x3 之间而且绝对为负。因此，x1 是另一个零界点，且不是一个全局最小值。作为一个结果，收敛到全局最小值仅仅发生在特殊的初始值和步骤尺寸.(d)每个有一个单独的全局最小值的函数是凸形曲线.答:错.一个函数有多个极小值，这里面只有一个是全局的。在这个情况下，它有一个单独的全局最小值，但不是凸形的。一个这样的函数的例子就是 c 部分中的函数 f.5. Sorting Prefixes 假设我们给出一个有

n 个不同元素，且对于$0 \le k \le \frac{n}{2}$的整数 k 来说，是 k-prefix 无序的序列 A[1...n]。就是说，A 的长度 s 后缀，在这里 s=n-k，已经被排序在非减秩序中（长度 k 前缀的元素可以是任意的，也是在任意顺序里的）。例如，A=[3,1,5,2,4,6]是一个 3 前缀的长度为 6 的无序序列.(a)提供一个最坏情况边界的渐进估计（就 n 和 k 而言），在插入排序作用在 k-prefix 无序序列上的比较的数量上。答：可以通过插入排序将 K 前缀无序序列的顺序分成两个阶段。在第一个阶段中，k 长度前缀被排序，在这个前缀中的元素会在任意顺序中出现，在最差情况下，这会花费Θ（k^2)的对比量.(最差情况相当于在不增次序中给所有的 k 元素排序).在第二个阶段中 s 长度后缀逐步被加入到排序好的前缀，每个元素向前移动到正确的位置，通过一个比较交换操作。观察到当后缀开始被排序到不减次序时，后缀中没有元素会和后缀中的其他元素交换。因此后缀中的每一个元素在到达正确位置之前需要最多 k 次比较交换操作。在最坏的情况下，例如，如果所有的初始 k 长度前缀的元素都比 s 长度后缀的元素要大，那么比较交换操作的数量为 k 每次，在这个阶段中会有Θ（$s \cdot k$)次比较。综上，最差情况下的对比数量会是：

Θ（k^2)$+\Theta(s \cdot k) = \Theta(k^2) + \Theta((n-k) \cdot k) = \Theta(k^2 + nk) = \Theta(n \cdot k)$ 在这里我们认为$k \le \frac{n}{2}$。一个 k 前缀无序序列，引起了最差情况的例子：A=[n,n-1,...,n-k+1,1,2,...,n-k](b)描述一个给 k 前缀无序序列排序的基于比较的算法，这个序列关于它造成的比较的数量是有效的。提供使用这个算法得到的对比数量的一个渐进最坏估计。答.考虑一个算法，在这个算法中前缀中的每个元素都是已获得的，都通过二分法被插入到已经排序好的前缀中。这个插入的总量是 k，每一个值都在 s 后缀中使用二分法。（最初，s’=s=n-k，但是每次插入一个新元素，它都在增长）显而易见，一旦算法终止，我们有 s’=n，例如整个序列 A 被排序。而且，k 次插入中的每一次都需要最多O（$\log s'$)$= O(\log n)$次对比。这给出了对比数量的一个最差边界O（$k \log n$)。(c)给出一个渐进下界，为任何基于对比排序的算法来排序任何 k 前缀无序序列 A 时需要的比较次数。答：我们使用一个决策树分析来估计需要的比较的次数来计算结果：一个不减已排序序列。决策时的叶子表示问题的所有可能结果，例如，输入序列的所有能产生一个正确输出的可能排列。树的中间节点表示已有的比较。比较提供了一个二进制决定，因此这棵树是二进制的。对这颗树的树叶数量的下界来说，我们需要计算将一些 k 前缀无序序列转换成一个不减顺序序列的排列数量。利用对称性，这个次数和我们从一个已知不减顺序序列中选择一个 k 元素的子集然后将他们作为一些任意顺序的一个前缀的次数是相等的。注意到，有整整$\binom{n}{k}$种方法来选择这些子集，$k!$中方法来整理 k 元素的选择。结果是，这种排列的总的数量是$[\binom{n}{k} \cdot k!]$ 现在，每个对于 k 前缀无序序列的正确排序算法需要表示成一颗这样的二进制决策树。而且，我们知道这样一棵树的高度表示在最坏情况下这个算法产生的比较次数。此外，因为这棵树是二进制的，它的高度有下界，下界是它的叶子数量的对数。因此，在最坏情况下任何算法需要的比较次数最少是

$\log(\binom{n}{k} \cdot k!) = \log\binom{n}{k} + \log k! = \log \Theta(n^k) + \Theta(k \log k) = \Theta(k \log n + k \log k) = \Theta(k \log n)$ 渐进匹配我们在 b 中获得的边界。6.Currency Market Disequilibrium 由于市场的无能，有时有一些短暂的可能来在交易市场中获得巨大的利益。例如，想象下面的交换率：以 100 美元开始，可以交换 10000 烟，用所有的烟购买欧元，然后再用所有的欧元重新购买美元。在购买循环最后所有的现金会是$100 • R1,2 • R2,3 • R3,1=$125。答：考虑构造一个加权的有向图 G（V,E），其顶点是货币 1,2，...，n。在该图中，对于每对货币（i,j），我们有边（i,j）和另一边（j,i），其权重是相应汇率的（负）对数。也就是说，对于每对货币（i,j），$w_{i,j} = -\log R_{i,j}$ 和 $w_{j,i} = -\log R_{j,i}$。观察到在这个设置中，，对于 G 中的任何周期 i1,...,im，其总权重刚好是

$$-\log R_{i1,i2} - \log R_{i2,i3} - \ldots - \log R_{i_{m-1},i_m} - \log R_{i_m,i1} = -\log(R_{i1,i1} \cdot R_{i2,i3} \cdot \ldots \cdot R_{i_{m-1},i_m} \cdot R_{i_m,i1})$$

。因此当且仅当其权重为负时，周期 i1,i2,...,im,i1 对应于可兑换货币交易的序列。（注意，如果我们采取更自然的方法，并使每个边缘的权重对应于相应汇率的对数，而不是它们的否定，那么当且仅当其权重为正时有盈利，然而这会更难检查我们在类中学习的算法。）因此，我们的任务归结为检查在图 G 中是否存在负权重周期包含定点 1。从类中我们知道，这可以通过运行 Bellman-Ford 算法来检查。算法在时间$O(V \cdot E)$中运行，其为$O(n^3)$，因为我们的顶点数量是 n，边数量在这里是$n(n-1)/2 = \Theta(n^2)$。

于所有的顶点 v，我们只需要从中运行一个 BFS，在从其中去狭窄的走廊 E^之后获得的 G 的子图中的顶点 G，这需要花费 O（V+E）时间。现在，为了避免拦截，我们需要确保我们从不在时间步长 g_v 以后或更晚访问顶点 v。所以，要检查是否有从来没有经过节点 v 的路径或者在时间 g_v 以后出现在 G 中的安全路径，我们需要运行从节点 s 出发以后的 BFS 算法的修改版本。在这个修改版本中，每当我们发现 BFS 树的级别 l 的顶点 v，且 $l \geq g_v$，我们就可以立刻回溯并且删除这个顶点和所有由这个顶点发射出去的边缘。然后继续运行这个算法，（注意如果我们在由该修改的 BFS 创建的树的第 1 级发现顶点 v，这意味着在 v 长度小于 l 时没有“安全”路径）。

显然，此修改版本的 BFS 仍然在 O（V+E）时运行，此外，如果它找到了一个合适的 s~t 路径，我们就可以把这个路径 P 作为安全路径返回。否则就说不存在安全路径。(c)假设接下来，由于你掌握着 6006 皇冠，你拥有把自己传送到相邻房间的力量，从而横穿相应的边缘。显然，只要存在一个从 s~t 的路径，避免了食人魔起始出发的地点 g，只要多次使用这个传送能力，你总是可以找到 6006 皇冠，同时还能避免被食人魔拦截。（我们这里假设食人魔有反射能力，即不可能透过它传送）然而，经验告诉你这样的能力只有非常需要的情况下才能使用一次。设计一个计算最小数 ℓ^* 的 O（V+E）的算法，使用逃避食人魔所需的力量。（你应该在这里忽略 b）中狭窄走廊的存在）答案：注意到我们需要使用传送力 ℓ，所有的这些使用 ℓ 的都发生在时间步长 0 之前。此外，最初使用的远程传输可以使我们能够有效的将我们的起始位置移动到 G 中的 s 距离范围内的任何位置（我们需要删除 g 确保我们不会传送到食人魔身边），让我们用 S_e 来代表所有这些集合的可能起始顶点 V。根据上面的注释，我们的任务就是确定 ℓ 的最小值，其实集合 S_e 包含一个顶点 s’如果我们从它开始（并没有使用我们的传送）出发，我们就可以保证自己在到达藏宝室之前不会被食人魔拦截。注意，由于没有狭窄走廊，我们在 a）部分进行的分析使我们得出结论：当且仅当在 G 中的 s’~t 的距离小于顶点 g 到 t 的距离 D 的时候，我们可以成功的避开食人魔。现在只需要计算这个最小值就足够了：1. 计算从顶点 s 到图 G 中所有的顶点 v 的距离 v’，同时移除顶点 g。2. 计算图 G 中所有顶点 v 到顶点 t 的所有距离 dv，接着：3. return $\ell^* = \min_{s':\, d_{s'} < D} \ell_{s'}$. 注意到如果 $\ell^* = +\infty$ 就意味着在 G 中不存在 s~t 的路径通向 g。不难看出，计算所有的 $\ell_v s$ 和所有的 $d_v s$，g~t 的距离 D 都可以在运行三次 BFS 算法之后来得到。并且可以在 O（V）时间中执行最终值的最小化。所以整个算法的时间复杂度就是 O（V+E）。

上面这部分接2016最后

10.Winning a Game of Strategy 考虑一个单人游戏，其中在桌子上有 r 个红色芯片和 b 个蓝色芯片，r = R 和 b = B。在每次移动中，您必须删除一个红色芯片或一个蓝色芯片。 在每次结束你的移动以后你都需要添加一个数字一个数字 f（r,b）到运行总和（它最初为零），其中 r 和 b 是每次移动完成后分别留在桌子上的红色和蓝色芯片的数量，f 是您预先知道的某个函数。 游戏继续直到没有芯片留在桌子上。 您的目标是在游戏结束时最大化运行总和的值。(a)试给出一个算法以实现如下功能：对于一个给定的函数 f 和桌上的红蓝芯片的数量R，B，实现步数总合的最大值，并求此算法的运行时间多少？答案：我们使用动态规划的办法，我们让 F（r，b）表示从剩余的红色芯片和蓝色芯片的位置开始可以实现的总和的最大值。现在我们使用方程：

$$F(r,b)=\begin{cases} 0 & r,b=0 \\ -\infty & r<0 \ \ or \ \ b<0 \\ \max\{F(r-1,b)+f(r-1,b), F(r,b-1)+f(r,b-1)\} & else \end{cases}$$

在最后一种情况下，max 函数的第一项表示如果去除红色芯片可以实现的最大值，方程的第二项表示如果去除蓝色芯片可以实现的最大值，注意到我们的最终解决方案将是 F（R，B）。子问题的数量是 R.B. 同样在以上循环中，只有在解决每个子问题的时候，工作量是恒定的（假设需要解决的子问题已经完成计算）另外，输出最终答案的时间只需要 $O(1)$，所以，整体运行时间为 $O(R \cdot B)$。(b)一个新的规则加入到当前游戏中，即如果你在每次移动结束以后的得到的红色和蓝色芯片的乘积为 6006 时你将自动消掉这一项。你将如何改进你的算法以实现当前的新规则？答案：我们用 f’（r，b）就足

以替换答案（a）中的所有 f（r，b）,f’（r，b）的表达式如下：

$$f'(r,b)=\begin{cases}-\infty & r\cdot b=6006\\ f(r,b) & else\end{cases}$$

(c)让我们现在舍弃 b）中的新规则，并考虑这个游戏的双玩家版本，其中你和对手轮流选择。 在轮到他/她的时候，对手可以选择删除红色或蓝色芯片，我们将每次移动以后仍在桌面上的红蓝色芯片的数目添加到步数运行总和的值 f（r，b）中。 然而，对手选择他或她的移动方式以使最终的步数总和尽可能小。描述对于给定函数 f 和初始红蓝芯片的数量计算的算法，你要保证得到的最终值是最大的，这样你将至少能够实现在双人游戏中与实现最优方案的对手对抗。 这个算法的运行时间是多少？ 假设你先移动，R + B 是偶数。答案：我们使用一个非常类似 a）情况中的动态规划算法。这一次假设对手选择的是最小化最终结果的移动方式，而玩家选的是一种最优方案。循环体如下：

$$F(r,\mathrm{b})=\begin{cases}0 & r,b=0\\ -\infty & r<0\ \ or\ \ b<0\\ \max\begin{Bmatrix}best\_resp(r-1,b)+f(r-1,\mathrm{b}),\\ best\_resp(r,b-1)+f(r,b-1)\end{Bmatrix} & else\end{cases}$$

在上面给出的情况中（同时也是最有趣的情况），max 对应于决定我们是否应该移除红色芯片还是蓝色芯片，best_resp（r，b）决定了如果我们从 f（r，b）开始并且对手先移动的情况下可以实现的最大值的总和。然而，随着对手移动的最优化，我们可以很清楚的表示在子问题为 F（r，b）的情况下如何表达 best-resp（r，b）。具体来说，当对手在移动红色或蓝色芯片这两种选择中选一种我们可以实现的将最大求和值最小化的方案。我们用下式描述：

$$best\_resp(r,b)=\min\{F(r-1,b)+f(r-1),F(r,b-1)+f(r,b-1)\}$$

其中第一个选项对应去除红色芯片的对手，而另一个去除蓝色芯片，将上述对 best_resp(r,b)实现的递归带入到我们的 F（r，b）循环中，我们就得到仅仅根据子问题 F（r，b）就如愿得到了一个循环。同样，最后的答案只是 F（R，B）要限制此算法的时间，需注意子问题 R.B 的数量.同样，在每次循环判断中，在基于现有的子问题来解决新的子问题所设计到的工作量全部是恒定的。（我们仅仅需要去评估最多两种选项，而其中每个选项最少实现两件事）。所以，整体运行时间仍然是 $O(R\cdot B)$。11.Retrieving the 6.006 Crown 你开始在一个迷宫中寻找一个传说中的珍宝-6006 皇冠（一旦你有皇冠在手，它所拥有的算法力量将会使你轻松的干掉食人魔）你需要找到通过迷宫到达皇冠所在地的最优路径，同时确保避开食人恶魔所在的路径。

迷宫可以被建模成无方向图 G=（V，E），其中顶点表示入口，宝藏的位置为 t，食人魔的起始位置为 g（与 s 和 t 不同）。你从入口 s 开始，在每一次时间步长中，你可以遍历对应当前所在顶点的每一个迷宫边缘，类似的，食人魔从顶点 g 开始，并且在每个时间步长中，可以穿过一个入射边缘或保持当前状态。(a)设计一个 O（V+E）时间算法，给出 G：返回 s~t，寻求一个在 G 只中的路径 P，如果你沿着这条路走你将肯定不会被食人魔抓住。也就是说，不管食人魔从 g 开始以什么顺序移动，他都不会在你走完之前出现，也不会在你走到终点的时候同时出现（包括最后一个顶点 t）。或者断定在G 中不存在这样的路径 P。答案：假设 P 是在 G 中的一个 s~t 的通向皇冠的路径，观察到如果食人魔能够在路径 P 的某个顶点 v 处拦截我们，然后通过 v~t 这个路径，他能够在我们到达藏有皇冠的屋子不久以后追上我们，在我们只是随着路径 p 并且没有任何拦截的情况下（回想一下食人魔以和我们相同的速度移动，并且可以在我们穿过的完全相同的边缘穿过去。）

所以魔鬼可以拦截你的条件就是：当且仅当他比你先到达藏宝室。（这里的如果遵从这样一个事实，即如果他比我们先到达，他就可以在顶点 t 拦截我们）。但是如果想检查食人魔是否可以以不晚于我们的时间 t 到达，我们只需要在节点 G 运行两次 BFS 算法（一次从 s 出发，一次从 g 出发）比较从 s 到 t 和从 g 到 t 的最短路径距离。如果前者小于相应的最短路径 s~t,则返回一个 P 的安全值，否则就不存在这样的安全路径。显然这个算法需要在 O（V+E）时间中工作。(b)假设食人魔体积太大无法通过相对狭窄的走廊，也就是说，图 G 具有标记为窄边缘的一些子集 E^,这些地方食人魔遍历不到（但是你却可以）。试提供一个 O（V+E）时间算法，它解决了 a）部分定义的任务，同时考虑到狭窄边缘的存在。答案：一旦狭窄走廊存在，从 a）中的解决方案我们可以得出结论，我们可以专注于评估食人魔在我们之前到达藏宝室的能力，这不是长久有效的。相反，让我们计算每一个当食人魔可以到达那个顶点 v 的最早的时间步长 g_v。要计算 g_v，对

2008

2 判断。a)一个算法的运行时间满足递归 P (n) = 1024P (n/2) + O(n^{100})渐近快于一个算法的运行时间满足递归 E(n) = 2E(n−1024)+ O(1)。**True**.第一个递归的结果是 n 的多项式，而第二个递归的结果是 n 的指数。 **b)**运行时间满足递归式 A(n) = 4A (n/2) + O(1)的算法比运行时间满足递归式 B(n) = 2B (n/4) + O(1)的算法渐近地快。 **False**. 考虑 A(n)和 B(n)的递归树，很容易看出 A 的树具有更小的高度(log4(n) vs. log2(n))和更小的分支因子。 **c)**只有排序的元素是在范围{0,1，…， c n}的整数，其中 c = O(1)，基数排序才能在线性时间内工作。 **False**.如果要排序的元素是在范围{0,1，…， n^d}内的整数，其中 d 为常数，基数排序也可以在线性时间内工作。 **d)**给定一个无向图，可以在 O(V + E)时间内测试它是否为树。树是没有任何环的连通图。**True**.使用 DFS 或 BFS 都会产生 O(V + E)的运行时间。 **e)**Bellman-Ford 算法适用于单源最短路径问题的实例，该实例不具有负权向循环，但如果存在负权向循环，它不会检测到其存在。**False**.Bellman-Ford 在其输入图中检测负权有向环。 **f)**任意有向无环图 G = (V, E)的拓扑排序可在线性时间内计算。**True**.通过按照 DFS 遍历图产生的退出时间的相反顺序列出节点，可以获得拓扑排序。DFS 还可以用于检测图中是否存在循环(在这种情况下，没有有效的拓扑排序)。DFS 的运行时间为 O(V + E)。 **g)**我们知道一种在 O(V + E)时间内检测任意有向图中负权环的算法。**False**.本课程给出的最佳解是 Bellman-Ford 算法，其运行时间为 O(V E)。 **h)**我们知道一个求解任意图上无负权的单源最短路径问题的算法，它在 O(V + E)时间内有效。**False**.本课程提出的最佳解决方案是带 Fibonacci 堆的 Dijkstra，其运行时间为 O(V log V + E)。 **i)**要删除最小堆中的第 i 个节点，可以将最后一个节点与第 i 个节点交换，然后在第 i 个节点上执行 min-heapify，然后将堆大小缩小到比原始大小小 1。**False**.最后一个节点可能小于第 i 个节点的父节点;min-heapify 无法解决这个问题。 **j)**推广 Karatsuba 的分治算法，通过将每个乘数分解为 3 个部分，进行 5 次乘法，提高了渐近运行时间。**False**.Karatsuba 的运行时间是 T (n) = 3T (n/2)+O(n) = O($n^{\log_2 3}$)。广义算法的运行时间为 T (n) = 5T (n/3) + O(n) = O($n^{\log_3 5}$)。

6 Maintaining Medians 我们的赌博策略是在所有选项中押注于中位数。就是，如果有 n 个不同的选项，它们的排序顺序是 c[1] <c[2]<…<c[n]，那么我们将押注选项 c[⌊(n+1)/2⌋]。随着时间的流逝，出现了新的选择，每一次都要对当前的选择进行排序，并在中位数上下注，所以需要构建一个数据结构，实现跟踪中位数的功能。具体来说，这个数据结构存储了 n 个选项，当前的中位数 m，以及两个 AVL 树 S 和 T，其中 S 存储小于 m 的所有选项，T 存储大于 m 的所有选项。**a)**解释如何在数据结构中添加一个新选项 C_{new}，恢复不变量(1)m 是当前所有选项的中位数;(2)S 存储小于 m 的所有选项;(3)T 存储大于 m 的所有选项。分析算法的运行时间。**解：**存储 AVL 树的大小|S|和|T|，并在添加元素或者删除元素时更新它们。要保持||S|−|T||<=1（左右子树的高度差的绝对值不超过 1）。如果 C_{new}<m 并且|S|<=|T|，则在 S 树中插入 C_{new}，不改变 m。如果 C_{new}<m，|S|=|T|+1，则在 S 树中插入 C_{new}，在 T 树中插入 m。S 树中最大的元素是新的中位数，因此从 S 树中删除最大的元素并将该值赋给 m。(其中 C_{new}>m 时的操作是对称的情况)。每次 AVL 插入和删除需要 O(lgn)时间，所以总时间为 O(lgn)。**b)**解释如何从数据结构中删除现有的选项 C_{old}，并恢复上面的不变量(1-3)。分析算法的运行时间。**解：**如果 C_{old}<m，则从 S 树中删除 C_{old}，|S|的值减小。如果||S|−|T||<=1，则不处理。如果|S|=|T|−2，则需要修改中位数。将 m 插入到 S 树中，然后提取 T 树的最小元素并将其存储在 m 中。每次 AVL 插入和删除需要 O(lgn)时间，所以总时间为 O(lgn)。

8 d-max-heap a)描述如何在数组 A[1…n]中表示一个 d-max-heap。特别是对于存储在任意位置 A[i]的 d-max-heap 的内部(非叶)节点，A 中的哪些位置保存着它的子节点?**解：**在数组中使用层次顺序遍历给出的节点进行存储。例如，根节点是节点 0，子节点是节点 1，2，3 和 4。对于非叶节点 i，它的子节点的位置是 d·i−d+2,d·i−d+1...d·i,d·i+1。**b)**将堆的高度定义为从根节点到叶节点的最长路径上的节点数。用 n 和 d 表示，有 n 个元素的 d-max-heap 的高度是多少?**解：**高为 h 的完全 d 叉树至多有 d^h-1 个结点，至少有 d^h 个结点，高为 h-1 的满 d 叉树有 d^{h-1}-1 个结点：d^{h-1}≤n<d^h ---> h-1≤$\log_d n$<h ---> h=⌈$\log_d n$⌉

12 Dance Dance Evolution 正在为即将到来的舞蹈比赛进行培训，并决定将应用动态规划的方法，以找到播放每首歌曲的最佳策略。问题的简化版本可以如下建模。这首歌指定了你必须做出的一系列“动作”。每一步都是集合 B={↑,↓,←,→}你必须用一只脚按压。歌曲的一个例子如左图。你有两只

脚。在任何时候，每只脚都在四个按钮中的一个上；因此，您的脚的当前状态可以由有序对(L，R)指定，其中 L∈B 表示左脚下方的按钮，R∈B 表示右脚下方的按钮。一次一只脚：当你移动 M∈B 在歌曲中，你必须将一只脚放在按钮上，转换到状态(M，R)或(L，M)。请注意，每次移动只能更改一只脚。如果你的脚已经踩在正确的按钮上，那么你不需要换脚（尽管你可以换另一只脚）。禁止状态：你也会得到一个禁止状态 F 的列表，你永远不会被允许进入这些状态。F 可能包括双脚都在同一个正方形上的状态，或者你最终面朝后的状态。你的目标是开发一种多项式时间算法，在给定 n 首移动歌曲 M1,M2,…,Mn 的情况下，该算法可以找到初始状态(L0,R0)和有效的转换序列(Li,Ri)→(Li+1,Ri+1)∉F，对于 0≤i<n，其中 Mi+1∈{Li+1,Ri+1}和 Li=Li+1 或 Ri=Ri+1

a)明确说明将用于解决此问题的一组子问题。**解：**子问题是这样一个问题，即是否可以“清除”状态 1…i，并以脚在配置(j，k)中结束，其中 j，k∈ B。为了简化下面的数学递归，yes 的答案是 1，no 的答案是 0。**b)**写一个将一般子问题的解与较小子问题的解决方案联系起来的递归。**解：**使用 DP[i][j][k]存储上述子问题的答案：

$$\mathcal{DP}[i][j][k] = \begin{cases} 1 \text{ if } & i = 0 \wedge (j,k) \notin \mathcal{F} \\ 0 \text{ if } & (j,k) \in \mathcal{F} \vee M_i \notin \{j,k\} \\ \max\limits_{u,v \in B, u=j \vee v=k} \mathcal{DP}[i-1][u][v] \text{ otherwise} & \end{cases}$$

简而言之，我们可以从任何不在 F 中的脚配置开始。任何移动序列都不能包含 F 中的配置，而声称“清除”移动 1…i 的序列必须以(i，R)或(L，i)结尾。一旦这些手续被排除在外，我们试图通过使用我们的答案来“清除”步骤 1…i−1，然后最多移动一只脚。注意，max 作为逻辑函数∨（或），给定布尔值的表示。**c)**分析算法的运行时间，包括子问题的数量和每个子问题花费的时间。**解：**设 b=|B|（已知 b=4，但为了分析起见将其保留为变量）。所以有 n×b×b=nb^2 个子问题。在最坏的情况下，计算子问题的答案需要在 O(b)之前解决的子问题上取最大值（左脚移动 b-1 次，右脚移动 b-1 次，1 种情况下无需移动）所以总的运行时间是 O(nb^3)，但是 b=O(1)，所以总的时间是 O(n) **d)** 根据子问题的解决方案写出原始问题的解决方法。**解：**解决方法是 A=max$_{j,k\in B}$DP[n][j][k]。移动序列可以通过使用父指针来恢复，父指针可以记住每个最大值的 argmax。

2009

1 判断 a) 有一种算法可以在 O (n)时间内从一个未排序的列表中构建一个二叉查找树 **解：**错。如果顺序遍历可以在 O (n)时间内从 BST 创建排序列表，那么这种算法的存在将违反基于比较的排序的 Ω (n lg n)下界

b) 有一种算法可以在 O (n)时间内从未排序的列表构建二叉堆 **解：**对。标准的 BUILD-HEAP 算法在 O (n)时间内运行。

c) 为了解决无负权边图的 SSSP 问题，需要至少松弛两次边 **解：**错。Dijkstra 的算法解决 SSSP 问题，最多松弛一次每个边。

d)在一个只有正边权的连通有向图上，Bellman-Ford 的运行速度和 Dijkstra 一样快。**解：**错。Bellman-Ford 运行时间是 Θ (VE)不管边权是多少。Dijkstra 运行时间是 Θ (E + V lg V)。因为图是连通的，所以 Θ (VE) = Ω (V^2) ，这明显比 Dijkstra 差。

e)Givens 旋转需要 O (1)时间。 **解：**错。如果矩阵是稀疏的，它可能需要 O (1)的时间，但一般来说，对于一个 n 列的矩阵，Givens 旋转需要 O (n)。

f)在最坏的情况下合并排序在 O (n^2)时间内运行 **解：**对。合并排序在 O (n lg n)时间内运行即 O (n^2)

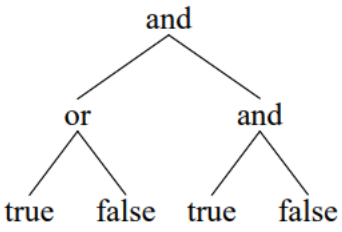
g) 存在一个稳定的合并排序实现 **解：**对。如果合并的两个列表中的项相等，则选择“左”半部分中的项(原始数组中第一个出现的那一半)

h) AVL 树 T 包含 n 个完全不同的整数。对于给定的整数 k，存在一个 Θ (lgn)算法来求 T 中的元素 x，使 | k-x | 最小。**解：**对。插入 k，然后找到 k 的先前者和后继者，返回与 k 的差值较小的那个。这三个步骤都需要 Θ (lgn)时间。

5 Local Minimum 给定一个数组，数组中每个值各不相同。现需要给出一个算法高效找出数组中的一个极小值。极小值即该数小于左邻居以及右邻居。若该数位于左右两端，则仅需小于一个邻居即可。**解：**使用分而治之的算法。让 m = n/2，并检查值 A [m](即数组中间的元素)。情况 1: A [m-1] <

A [m]。然后数组的左半部分必须包含一个局部最小值，因此在左半部分递归。我们可以通过矛盾来证明这一点: 假设 A [i]不是每个 0≤ i < m 的局部极小值。那么 A [m-1]不是局部极小，这意味着 A [m-2] < A [m-1]。同样，A [m-3] < A [m-2]。按照这种方式继续，我们得到 A [0] < A [1]。但是 A [0] 是一个局部极小值，与我们最初的假设相反。情况 2: A [m + 1] > A [m]。然后数组的右半部分必须包含一个局部最小值，所以在右半部分递归。这与情况 1 是对称的。情况 3: A [m-1] > A [m]和 A [m + 1] < A [m]。那么 A [m]是一个局部极小值，所以返回它。运行时间递归为 T (n) = T (n/2) + Θ (1) ，得到 T (n) = Θ (log n)

7 The Cake Is a Lie! 在 Aperture Bakeries，每个蛋糕都带有一个二值布尔值树,以指示是否可用。树中的每个叶子都有一个 true 或 false 值。剩下的每个节点都恰好有两个子节点，并被标记为 and 或 or; 该值是递归地将运算符应用于子节点值的结果。下面的树就是一个例子,如果树的根的计算结果是假的,就像上面的那个.....可以修改一棵树使之成真:要改变一棵树，你唯一能做的就是把假叶子变成真叶子，或者把假叶子变成真叶子。每更换一片叶子要花 1 美元。你不能改变操作符或树的结构。描述一种确定有 n 个节点的蛋糕树最小代价的有效算法,并分析其运行时间。



解：我们可以在 0(n)时间内通过从叶子开始并向上计算每个节点的真值。使用拓扑排序或 BFS 可以在 0(n)时间内计算出精确的顺序。如果根为真，则蛋糕是免费的，因此返回$0。否则，我们可以使用动态规划来确定这个成本。为每个节点增加一个字段 C，使节点为真的最小代价。每个节点都对应于在该节点上确定 C 的子问题。基本情况由叶节点组成。对于每个真叶子,将 C 设置为$0。对于每个 f 逆叶,将 C 设置为$1。递归如下:对于所有和节点:将 C 设置为子节点 C 值的和。对于所有或节点:将 C 设置为子节点 C 值中较小的一个。在为所有节点计算 C 之后,以最初计算真值的相同顺序,返回根节点的 C 值。由于有 O(n)个子问题,且每个子问题的代价是常数,所以总运行时间为 O(n)。

8 I Am Locutus of Borg You will Respond To Ky Questions 到达行星顶点 T 时，你和海军上尉被博格人俘虏……顶点 T 的停车场是一个 n×n 矩阵 a,已经有一些飞船停放在停车场。对于 0≤i,j<n，如果有船舶占据位置(i, j)，则 A[i][j]=0，否则为 1。博格人想要找到最大的方形停车位来停敢博格立方体。也就是说。找到最大的 k,使 a 中存在一个 k 的平方，包含所有的 1，不包含 0。在示例图中,解决方案是 3，如突出显示的框 3 所示。描述一个有效的算法，它可以找到 a 中最大的广场停车位的大小。分析算法的运行时间。

	0	1	2	3	4
0	0	1	1	1	0
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	0	0	0
4	1	1	1	0	1

解：基本情况是沿停车场顶部和左侧的位置。在每个位置，你可以容纳一个 1×1 平方的停车位，如果且仅当空间是无人占用的。因此,对于基本情况(i=0 或 j=0)，我们有 s[i, j]=A[i][j]。现在来看一般情况。同样，如果 A[i][j]=O，那么 S[i, j]=0，所以我们只考虑 A[i][j]=1 的情况。当且仅当每个位置(i-1，j)、(i,j-1)、(i-1,j-1)有三个大小为 x-1 的(重叠的)车位时，在(i,j-1)有一个位于(i,j)右下角的大小为 x 的车位。因此，最大可能的停车位是:

$S[i,j] = 1 + \min(\, S[i-1,j]\,,\, S[i,j-1]\,,\, S[i-1,j-1]\,)$ 期望的答案是 $\max_{i,j}$ s[i,j],需要 O(n^2)来计算。有 n^2 个子问题。每一个都需要 O(1)时间来解决，时间为 O(n^2)。添加到 O(n^2)以提取所需的答案。得到的总运行时间是 O(n^2)，这显然是最优的，因为必须检查所有数据。

2010 2 Storing Partial Maxima 学生 Mike Velli 想建立一个网站，用户可以输入历史上的时间间隔,网站将返回在这段时间内发生的最激动人心的体育赛事。正式地说，假设 Mike 有一个按时间顺序排列的 n 个体育赛事列表，其相关的整数兴奋因子为 e1,…, en。为了简单起见，你可以假设 n 是 2 的幂。一个用户的查询将由一对(i，j)和 1≤i ≤j≤n，并且站点应该返回 max(ei,ei+1，…ej)。.....这样对于任何用户查询，服务器都可以检索两个预先计算的值，并取这两个值的最大值来返回最终的答案。给出一个只需要预先计算 O(nlog n)个值的方法。**解：**我们得到了这个列表 el,.. . ,en.对于每个 1≤i ≤n/2,存储 max(ei , ei+1, ..., en/2)，对于每一个 n/2< j≤n 存储 max ($e_{n/2+1}$，…ej)。在两个链表上分别递归 e1,…, $c_{n/2}$ 和 $e_{n/2+1}$,…, en。当列表大小变为 1 时停止递归。 如果用户的查询是(i,j)且 i≤n/2 且 j>n/2。然后我们可以返回 max(max (ei,ei+1,…e/2) , max ($e_{n/2+1}$，…ej)).如果 i,j 都≤n/2 或者 i，j>n/2,然后递归地找到答案。 设 S(n)为长度为 n 的列表中存储的值的数量。根据构造，S(n) = O(n) +2S(n/2)

因此，S(n) = O(nlog n)

4. Closest pair 我们感兴趣的是找到平面上最近的一对点,直线距离上的最近点(也称为曼哈顿距离或 L1 距离)。平面上两点 p1 和 p2 的直线距离定义为……**a)**热身-提供一种有效的方法，在线上[0,1]区间的 n 个点中找到最接近的一对。充分的信用将给予最有效的算法与正确的分析。**解：**对 n 个数字(k 个数字表示)使用基数排序。然后遍历已排序的列表,计算每个点与下一个点之间的距离,在整个过程中保持最小值。基数排序需要 O(k×n)，距离计算 n -1 倍 O(k)，所以总体线性时间 O(n)，因为 k 是固定的。**b)**平面的情况-一个分而治之的方法,在正方形[0,1]×[0,1]中的 n 点。这里有一个可能的策略。将这组点分为两组，大小约为一半:在 x 坐标中位数的右边，和在左边。递归地，在右边找到最近的一对点，在左边找到最近的一对点，δ_r 和 δ_l 为对应的距离。最接近的一对要么是这两个选项中的最小值，要么对应于一个点在中间值的右边，另一个点在左边的一对。已知 δ_r 和 δ_l，应该有一个有效的方法来求后者。解释并写出该方法的运行时间的完整递归式;最后用它的总体运行时间来总结。**解：**寻找穿过中位 x 坐标的最近的一对点，可以限制在宽度+/- δ 附近的垂直线上，其中 $\delta=\min\{\delta_r, \delta_l\}$。通过将属于这个条带的点按照它们的 y 坐标排序，我们可以自下而上的遍历所有这些点并且对于每个检查点到它们上面的点的距离都在一个垂直的 δ 范围内，在最坏的情况下，所有的 n 个点都在这个条带上，所以我们最终得到一个运行时间为 T(n)=2T(n/2)+0(n),所以 T(n) =O(n log n)的递归式。

5. APSP Algorithm for Sparse Graphs 设 G=(V, E)是一个有权有向图它的一些权值为负。设 n = |V|, m=|E|，假设 G 是强连通的，即对于任意顶点 u 和 v，G 中都有一条从 u 到 v 的路径。我们想要解决 G 中的全对最短路径问题(APSP). .... **a)**固定某个顶点 V。考虑顶点势 $\lambda_t(u)= \delta(u, t)$，其中 δ (u, t)是 u 到 t 的最短路径。给出计算所有对于所有 $u\in V$ 的 $\lambda_t(u)$的算法。并分析运行时间。**解：**O(|V||E|)时间的 Bellman-Ford 算法

b) 证明了(a)部分的势 $\lambda_t(u)$是可行的，即使原来的一些权是负的。**解：**如图

Consider $w^*(u,v)$ for any $u, v \in V$. By definition

$$w^*(u,v) := w(u,v) - \lambda(u) + \lambda(v) = w(u,v) - \delta(u,t) + \delta(v,t).$$

Since $\delta(u,v) \leq w(u,v)$, we have that

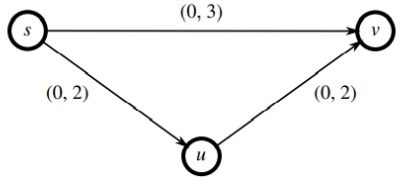
$$w^*(u,v) = w(u,v) - \delta(u,t) + \delta(v,t) \geq \delta(u,v) - \delta(u,t) + \delta(v,t).$$

But, by triangle inequality we have that $\delta(u,t) \leq \delta(u,v) + \delta(v,t)$, thus

$$w^*(u,v) \geq \delta(u,v) - \delta(u,t) + \delta(v,t) \geq 0,$$

as desired.

c)展示你如何使用(a)和(b)部分的顶点势在 $O(mn + n^2\log n)$时间内解决 APSP，包括计算顶点势所花费的时间。**解：**选择 G 中的任意顶点 t,在转置图 G 上运行 Bellman-Ford 算法，或者计算每个 $u \in V$ 的所有距离 δ(u, t)，或者检测出一个负长度的循环。如果在转置图中检测到这样一个循环，那么它也存在于原始图中，所以只需报告它的存在并终止。否则，定义顶点势 λ(u) := δ(u, t)，看得到的减重 $w^*(u, v) := w(u, v)-\lambda(u)+\lambda(v)$。由于 G 是强连通的(我们现在知道它没有负长度的环)，所有的距离 δ (u, t)都是有限的，因此所有 λ(u)也是有限的。我们知道这意味着 λ 是一个可行势，即对于每个 $u, v \in V$， $w^*(u, v)$ u 是非负的。由于简化权值 w^*是非负的，可以从每个节点 $u \in V$ 在 G 上运行 Dijkstra 算法，计算关于 w^*的所有顶点到顶点的距离$\{\delta^*(u, v)\}_{u,v}\in V$。接下来，通过对所有 $u \in V$ 计算 $\delta(u, v) = \delta^*(u, v) - \lambda(v) + \lambda(u)$提取 G 中顶点到顶点的真实距离，并输出它们。上述算法的运行时间主要由 BellmanFord 算法执行一次 O(mn)时间和采用 Fibonacci 堆实现的 Dijkstra 算法执行 n 次 $O(n(m + n\log n))= O(mn + n^2\log n)$时间。因此总运行时间为 $O(mn + n^2\log n)$，满足要求。



7 Traveling on a Budget 阿瑟 · 登特有 500 美元和 1000 小时的时

间从马萨诸塞州的剑桥到加利福尼亚州的伯克利。他有一张表示为有向图 G = (V，E)的美国地图。图的顶点代表城镇，如果有公共交通工具连接两个城镇，则从 A 城镇到 B 城镇有一个有向边 e = (A，B)。此外，边被标记为一对(m_e，t_e)，表示从 A 到 B 的运输成本 $m_e \in \{0,1，..\}$，以及从 A 到 B 所花费的时间 $t_e \in \{0,1，..\}$，以小时为单位。 Arthur 有兴趣找到一条从剑桥到伯克利不超过 500 美元，不超过 1000 小时的路径，同时也最小化目标 $5M^2 + 2T^2$，其中 M 是以美元为单位的旅行成本，T 是以小时为单位的旅行持续时间。他在寻找一个运行在时间多项式中的算法。**a)** 由于缺乏算法知识，他放弃了预算和时间限制的想法。至少他认为他可以有效地找到最小化目标 $5M^2 + 2T^2$ 的路径。他试图修改 Dijkstra 的算法如下: 如果一个边 e 被标记为(m_e，t_e)，他给它赋予一个权重 $w_e = 5m_e^2 + 2t_e^2$，然后在得到的加权有向图上运行 Dijkstra 的算法。证明 Arthur 的算法可能返回不正确的结果，即返回一条不会使目标 $5M^2 + 2T^2$ 最小化的路径。**解：**由边(s，v)构成的从 s 到 v 的路径有 $5M^2 + 2T^2=18$ 和$\Sigma 5m_e^2+2t_e^2=18$；从 s 到 v 由边(s，u)和(u，v)组成的路径有 $5M^2 + 2T^2=32$ 和$\Sigma 5m_e^2+2t_e^2=16$。因此，Arthur 的算法将错误地返回路径“(s，u)，(u，u))”而不是(s，v)。

b) 现在提供一个算法来解决亚瑟在 | E | 和 | V | 中的时间多项式的原始问题。您的算法应该找到最小化目标 $5M^2 + 2T^2$ 的路径，同时遵守约束 $M \leq 500$ 和 $T \leq 1000$。请准确地描述您的算法，并证明其正确性和运行时间。[提示 1: 使用动态规划][提示 2: 对于每个城镇 A，整数值 $m \leq 500$ 和 $t \leq 1000$，要么有一条从剑桥到 A 的路径需要成本 m 和时间 t，要么没有。] **解：** 我们在每个顶点 v 上构造一个 501 × 15 的矩阵 M_v，如果有一条从剑桥节点到 v 的路径，总成本 i 和总持续时间 j，则 M_v 的(i，j)条目为 1，否则为 0。$M_{Cambridge}$ 在入口(0,0)处，设置为 1，其他地方设置为 0。

$$M_v(i,j) = \begin{cases} 1 & v = \text{Cambridge},\ i = 0,\ j = 0 \\ \max_{u \in V \text{ with } (u,v) \in E} \left(M_u(i - m_{(u,v)}, j - t_{(u,v)})\right) & \text{otherwise} \end{cases}$$

在填写完所有 M 后，使用一个备注函数回答原来的问题，在矩阵中的(i，j)位置为 1 则说明在该位置的 $M_{Berkeley}$ 满足条件，最小化了 $5i^2+2j^2$。如果在 $M_{Berkeley}$ 中没有位置为 1，那么我们返回的是从剑桥到伯克利的路径不符合要求。