1.

a)贪心策略是若要找零x美分，则每次先找y美分(其中y是不大于x的最大硬币面额)，一直循环直到$x=0$， 由于最小面额是1美分，则这种方式一定能凑成x美分。

证明：①若是$x<5$， 则只能找1美分硬币；

②若是$5\leq x<10$, 按照贪心策略必会找5美分（若是不找5美分，则只能找x个1美分，而5个1美分用1个5美分替代的话，方案会更好）

③若是$10\leq x<25$, 按照贪心策略必会找10美分(若是不找10美分，假设找的5美分大于等于2枚，则2枚5美分可以由1枚10美分替代；若是找1枚5美分，则剩下的肯定是1美分，可以把1枚5美分、5枚1美分用1枚10美分替代；若是只有1美分，则将10枚1美分换成1枚10美分，这样替换方案会更好)

④若是$x\geq 25$, 按照贪心策略必然会找25美分(若是不找25美分，假设10美分大于等于3枚，则将3枚10美分换成1枚25美分和1枚5美分，若是10美分等于2枚，则剩下的5美分和1美分必然可以再凑出5美分，则可以用1枚25美分替代，若10美分数小于2枚，则5美分和1美分必然可以和10美分凑成25美分，用1枚25美分替代，这样方案会更好)

b) 若是面额是$c^0,c^1\dots c^k$,（$c>1$，$k\geq 1$），则找零问题可以看成一个k位的数，①若是找零 c^i面额的硬币m个，则第i位为m; ②对于 $i<k$, 第i位上可以满 c进1位

若是所有的位数能进位就进位，这样得到的数各位置和一定是最小的

而进位则等价到 c枚c^i面值硬币用1枚c^{i+1}替代，上述方法就是a所提到的贪心策略。

c)若是面额为1美分，5美分，6美分

10美分用贪心是1枚6美分，4枚1美分，总共5枚，而两枚5美分是更优的选择

d)可以采取动态规划的做法，$dp[i]$表示i美分用的最少硬币数，若是记k种硬币为$c_1,c_2\dots c_k$（含有1美分），则可以得到以下递推式：

$$dp[i]=\begin{cases}0 & \text{i=0}\\ min(dp[i-c_j])+1 & \text{i}>0, j\leq k, i-c_j\geq 0\end{cases}$$

i从1循环到n，每次循环要有k次计算，所以复杂度是$O(nk)$

2.

活动安排

可以将活动按照开始顺序从小到大先排序，对于某个教室记录最晚的活动结束时间，然后顺次遍历，若是某个活动可以找到某个教室，其最晚活动结束时间小于等于这个活动的开始时间，则可以把这个活动加到这个教室，否则就新找一个教室来进行这个活动，这样贪心选取可以得到最优结果。

若是中间过程不是这样贪心选取的话，对于某个活动，即使某个教室可以加进去，也不加而是新开一个教室，这样的话这个新开的教室的结束时间和之前贪心选取教师的结束时间是一样的，而贪心选取的教室数会少1个，这样的话贪心选取不会比这种情况差，这样可以简单地说明贪心的正确性。

排序的话是 $O(nlogn)$ 的复杂度，每次采用二分查找需要$O(logn)$， 进行n次，总复杂度为$O(nlogn)$

离线缓存

若某个请求的元素在缓存中，则对答案没有影响，若不在缓存中，则答案会加1，要考虑的是不是把这个元素加入缓存。

若是缓存未满，则加进去不会使情况变坏，加进去即可，若是缓存满了，可以采取算法导论提供的方法，去查找所有缓存中的元素出现下一个请求的最近位置a，若是比当前元素的下一个请求的位置b要近，则不换，若是更远，则换掉。

当$a < b$时，贪心策略是不换，因为换掉的话，再到达a位置时又会使答案$+1$，而不换掉的话在到达a位置时答案不会变化，若是(a, b)之间还存在被换掉元素，那么答案就会一直增加，而贪心选取就不会，在到达b位置时换掉可以保证不比a位置换掉差，$b < a$同理可以证明，所以贪心是正确的。

为了标记某个元素出现的下一个位置，我用了一个结构体去记录一个元素出现的pos和$value$，先按照$value$再按照pos进行排序，这样可以保证值一样的元素在一起并按照位置从小到大排序，便于记录下次出现位置，这里需要$O(nlogn)$

然后我用一个set去模拟缓存，记录的是缓存里的元素下一次出现的pos，若是当前pos在set里出现，则表示当前元素在缓存里，要把pos删除并更新新的下一个，若没有，则表示不在，答案$+1$，若是set没满直接加，满了就按照上述贪心判断是否替换，则每一次需要$O(logk)$的时间，一共n次，则为$O(nlogk)$

则总复杂度为$O(nlogn + nlogk)$

双调序列

①由题意，双调序列经过循环移动之后，最多只有一次单调性变化，则未移动前，最多有两次单调性变化，如$[1, 0, 3, 2]$单调性两次变化，可以移动成$[2, 1, 0, 3]$只有一次单调性变化

②算法导论有一个 INITIALIZE_SINGLE_SOURCE(G, S), 即起始点初始化成0, 其他节点都初始化成INF：

```
for(int i = 1; i <= n; i++)
        d[i] = INF;
    d[s] = 0;
```

若是将m条边都relax一遍：

```
for(int i = 1; i <= m; i++)
    d[edge[i].y] = min(d[edge[i].y], d[edge[i].x] + edge[i].w);
```

算法导论的引理24-15：

Let $G = （V, E)$ be a weighted, directed graph with weight function $w : E \to R$, and let $s \in V$ be a source vertex. Consider any shortest path $p = \{v_0, v_1 \dots v_k\}$ from $s = v_0$ to v_k. If G is initialized by INITIALIZE_SINGLE_SOURCE(G, S) and then a sequence of relaxation steps occurs that includes, in order, relaxing the edges $(v_0, v_1), (v_1, v_2) \dots (v_{k-1}, v_k)$, then $V_k.d = \delta(s, v_k)$ after these relaxations and at all times afterward. This property holds no matter what other edge relaxations occur, including relaxations that are intermixed with relaxations of the edges of p.

③由上引理可得，若是将所有的边排序(每一条边长度不相同是互异的), 初始时INITIALIZE_SINGLE_SOURCE(G, S),如果从源点到某一点路径长度一直增加，则按照边长升序来relax一定能获得最小路径，若是先增后减的话就先按照边长升序再按照边长降序relax一定能获得最小路径。

由于双调序列可能先增后减再增，或者先减后增再减，那么最小需要将所有边relax四遍(可以先增后减再增再减)，这样就能覆盖到所有情况了。

初始化复杂度为$O(V)$, 排序复杂度为$O(ElogE)$, 遍历四次复杂度为$O(E)$，则总复杂度为$O(V+ElogE)$

3.

可以采用矩阵的方式记录，若$d[i][j]$= 1, 表示这i可到达j，为0则不可达，若是添加u, v,则对所有$d[x][u]=1, d[v][y]=1$， 可以得到$d[x][y]=1$

这样遍历一遍复杂度是$O(V^2)$

最多可能有$O(V^2)$个节点，若是这样遍历的话$O(V^4)$， 可以优化一下，若是$d[x][v]=1$,则不需要将x进行遍历，矩阵记录也可以动用链表或者$vector$之类，复杂度会降下来，但是不知道怎么才能算到$O(V^3)$