2402 學期 MMIE05 算法理論基礎期末考

1. 计算复杂度为$O(n\lg n)$的排序算法有哪些？线性时间的排序算法有哪些？

排序算法有快速排序、归并排序；线性时间的排序算法有计数排序、基排序、桶排序。

2. 举例说明最大堆和最小堆的实际应用(给出最大堆实际应用例子 2 个，最小堆实际应用例子 2 个)，为什么可以用于所举的例子中？

最大堆实际应用如下：

（1）优先队列：在优先队列中，每个元素都有一个优先级，优先级最高的元素出队。首先，对队列中的元素进行排序，每个节点的值都大于或等于其子节点的值，因此堆顶元素是最大的元素；其次该队列的出队操作是将当前队列中值最大的元素出队，因此最大堆可以在$O(\lg n)$时间内插入新元素和取出优先级最高的元素即堆顶元素，可以高效处理优先级调度问题。

（2）负载均衡：在服务器运行过程中采用最大堆方式能够跟踪服务器的负载情况。因为最大堆能够快速地在服务器集群中找到负载最轻或最重的服务器（先按照负载情况进行优先级排序），将负载最重的服务器运行任务进行一定程度的转交给负载最轻的服务器运行，可以确保负载均衡的高效性。

最小堆实际应用如下：

（1）Dijkstra 算法：该算法是一种用于解决单源最短路径问题的经典算法，其基本思想是从起始节点开始，创建一个最小堆，将所有节点的路径加入堆中，随后从堆中提取最距离最小的节点以作下一个节点，以此拓展到每个节点至遍历所有节点。在选择最短路径时，采用最小堆可以更高效地通过距离来选取下一个节点，且最小堆允许在$O(\lg n)$时间内调整节点的优先级，当找到更短的路径时，更新节点的距离并调整其在堆中的位置，保证堆性质的维护。

（2）哈夫曼树：是一种用于数据压缩的树形结构，利用最小堆来构建，最终生成哈夫曼编码。该编码构建过程是统计元素出现的频率，首先将每个元素

及其频率作为一个节点；其次将所有节点插入最小堆，构建一个基于频率的最小堆；随后从最小堆中提取频率最小的两个节点，作为左、右子节点；再者创建一个新子节点，其频率为两个子节点频率之和，新节点的左子节点为其中一个提取的节点，右子节点为另一个提取的节点；最后将新节点插入最小堆中。重复上述提取两个频率最小节点—创建新子节点—插入最小堆的操作至堆中只剩一个节点。最后根据上述操作生成的哈夫曼树，从根节点开始，对左子节点分配编码“0”，对右子节点分配编码“1”，递归生成每个字符的哈夫曼编码。在哈夫曼编码中使用最小堆的原因是能够高效地最小频率节点提取和动态调整节点频率，在提取频率最小的节点时能够在最小堆中直接获取；其次允许其允许在$O(\lg n)$时间内插入新节点并调整堆结构，确保每次都能正确地找到最小频率地两个节点进行合并。

3. 具有什么特点的优化问题可以应用动态规划求解？

子问题重叠的优化问题可以应用动态规划求解。因为在问题的求解过程中，出现多次相同计算的子问题，采用动态规划可以记录子问题的结果，从而提高求解效率，避免重复计算。

4. 具有什么特点的优化问题可以应用贪心算法求解？

每个子问题相互独立、每个局部决策保证问题向前推进且不会导致非可行解或无效解、可以通过局部最优选择来达到全局最优结果的优化问题可以应用贪心算法求解。

5. 分治策略、动态规划和贪心算法的区别？

这三者是常见的算法设计策略：

- 分治策略：该思想是最基础的框架，将原问题分解成若干个规模较小但类似于原问题的子问题，递归地求解这些子问题并合并它们的解以建立

原问题的解。

- 动态规划：和分支策略相似，将原问题分解成若干个规模较小的子问题进行求解，最后将子问题的解合并以构建原问题的解。与分支策略的不同是动态规划的子问题会出现重叠的部分，即一个问题求解后可能会再次求解，在求解过程中会将子问题的解一一存储。
- 贪心算法：与分治策略相似，子问题相互独立，但是其基本思想是在每个子问题都做出局部最优的选择，但是这些选择并不会保证原问题能够得到最优解，而是每个局部最优的解。

综上所述，分支策略和贪心算法一般用于子问题相互独立的情况，而动态规划用于子问题有重叠的情况。分支策略会基于全部子问题的解以求得原问题的最优解，而贪心算法的最终解是局部最优解的组合。

6. 广度优先搜索和深度优先搜索的区别？

这两者是基本的图搜索算法，在遍历图或树时采用不同的策略进行求解。广度优先搜索是按照层次节点以寻找最短路径，适用于寻找无权图的最短路径和按层次遍历节点；而深度优先搜索是沿着一条路寻找到最深的节点，适用于深度遍历节点、检测途中的环等任务。

7. Kruskal 算法和 Prim 算法的区别？

这两者是用于求解无向有权图最小生成树问题的检点算法，两者核心思想同样但各自策略和数据结构存在差异。Kruskal 算法的基本策略是从小到大选择权重更小的边，避免子回路形成，数据结构是查并集；Prim 算法是从一个其实节点开始，扩展最小生成树，数据结构是最小堆的优先结构。其次，前者适用于稀疏图，其算法复杂度与边的数量有关；后者适用于稠密图，其算法复杂度与节点的数量有关。

8. Bellman-Ford 适用于求解具有什么特点的最短路径问题？

该算法适用于求解图中存在负权边、需要检测负权回路、单元最短路径的问题上。该算法求解过程中会对图中所有边进行多次松弛操作，每次松弛操作尝试更新从源点到每个节点的最短距离。

9. 使用 Dijkstra 算法是否可以求解存在至少一条边权重为负的最短路径问题？为什么？

不可以。因为该算法是基于贪心策略的求解算法，在选择路径上会依赖于节点之间的距离，先确定的最短路径并不是最短路径，且可能会破坏其贪心选择，已确定的最短路径可能会在后续阶段被更短的路径取代，可能会因此陷入无限循环从而找不到最短路径的解。

10. Johnson 算法的核心是什么？

该算法的核心思想是对所有边权重进行重新添加权重。其通过重新添加权重将原始图的所有负权边转化为非负权边，使得边的权重非负因此可以采取 Dijkstra 等算法进行最短路径计算。

11. Edmonds-Karp 算法的时间复杂度为什么会低于 Ford-Fulkerson 算法。

Edmonds-Karp 算法在 Ford-Fulkerson 算法的基础上进行了改进：在 Ford-Fulkerson 算法的寻找增广路径步骤中采取广度优先搜索以选择增广路径。在搜索增广路径时，采取广度优先搜索可以保证在找到一条从源节点到汇节点最短路径时，所经过的边数最少；因此每次迭代只需要在剩余容量图中找到一条最短路径，相较 Ford-Fulkerson 的随机选择路径所花费的迭代次数更少。因此 Edmonds-Karp 算法的时间复杂度为$O(E|f^*|)$，其中$|f^*|$是最大流量的值；Ford-Fulkerson 算法的时间复杂度为$O(VE^2)$，其中E为节点数量。

12. 证明最大流和最小割是一对对偶问题。

对于任意一个割(S,T)，设从f是一个从源点s到汇点t的流量，即：

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$$

因为s是源点，出发的流量总和减去流入的流量综合就是流量的值$|f|$。对于任意一个割(S,T)，根据流守恒性，只需要考虑流量跨过割的情况：

$$|f| = \sum_{u \in S, v \in T} f(u,v) - \sum_{u \in T, v \in S} f(u,v)$$

由于流量f满足容量约束：

$$\sum_{u \in S, v \in T} f(u,v) \leq \sum_{u \in S, v \in T} c(u,v) = C(S,T)$$

流量f从S到T的总流量不会超过割的容量。

其次，假设已经找到了一个最大流f，构建一个对应的割，使得这个割的容量等于最大流的值。

由于t不在S中，在一个割(S,T)中，从S到T的边都是满的，即流量等于容量；从T到S的边流量为 0，因此割的容量是：

$$C(S,T) = \sum_{u \in S, v \in T} c(u,v) = \sum_{u \in S, v \in T} f(u,v)$$

由于所有从S到T的边都是满的，所以这个容量等于最大流的值$|f|$。

综上所述，最大流量等于最小割的容量，最大流和最小割的问题是一对对偶问题。最大流的最优解可以通过求解最小割问题最优解得到，且这两者的最优解是相等的。

13. 将最短路径问题，最大流问题，最小费用流问题用线性规划进行建模(参考P504、P505)，并对每条约束进行详细解释。

（1） 最短路径问题：

采用线性规划建模：

$$\min d(t)$$

subject to

$$d(v) \leq d(u) + w(u,v)$$
$$d_s = 0$$

其中，目标函数为最小化源点s到汇点t的最短路径长度$d(t)$，$d(v)$表示源点s到汇点v的距离，$w(u,v)$是边(u,v)的权重，对于所有的边$(u,v) \in E$。第一条约束表示源点s到自身的距离为0；第二条保证了每条边(u,v)从s到v的距离不超过从s到v的距离加上u到v的边权重，这确保了路径的最短性。

（2） 最大流问题：

采用线性规划建模：

$$\max \sum_{(s,v)\in E} f(s,v)$$

subject to

$$0 \leq f(u,v) \leq c(u,v)$$
$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w)$$
$$\sum_{(u,s)\in E} f(u,s) = 0$$

其中，目标函数为最大化源点s到汇点t的最大流$\sum_{(s,v)\in E} f(s,v)$，$c(u,v)$是边(u,v)的容量。第一个约束表示每条边的流量不超过其容量；第二条表示每个中间节点，流入的总流量等于流出的总流量，保持流的守恒性；第三条确保没有流进入源点s。

（3） 最小费用流问题：

采用线性规划建模：

$$\max \sum_{(u,v)\in E} c(u,v) f(u,v)$$

subject to

$$0 \le f(u,v) \le u(u,v)$$

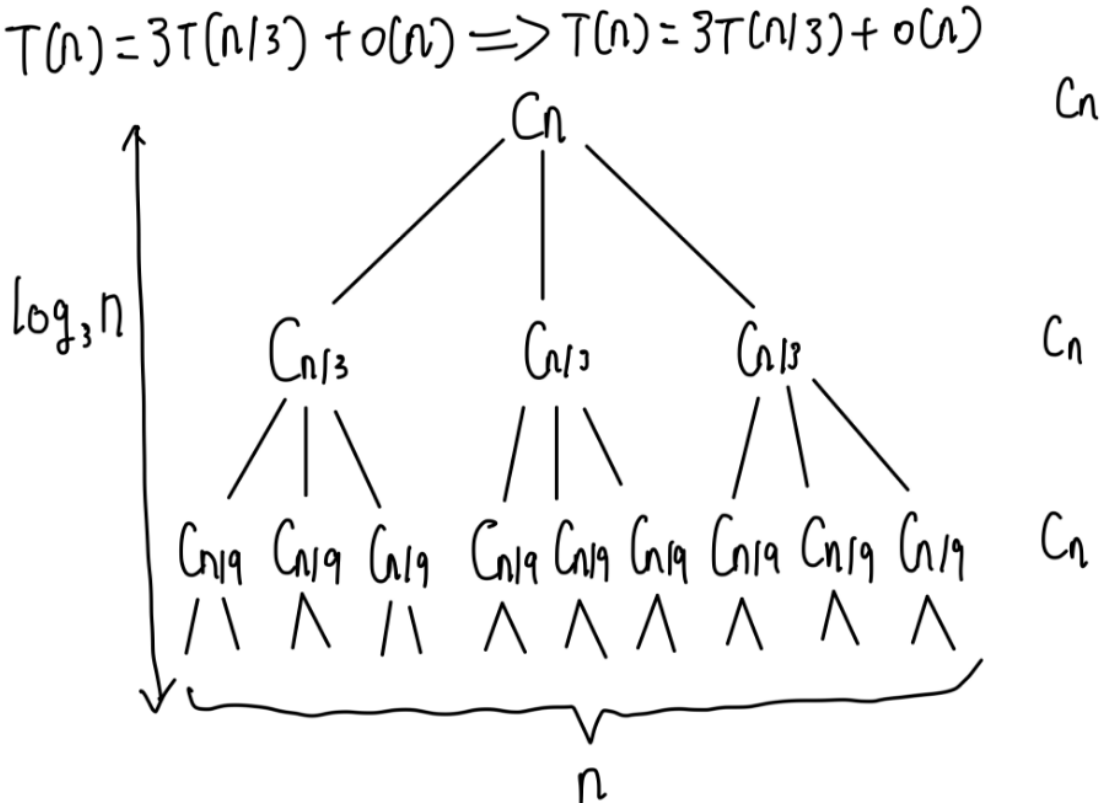
$$\sum_{(u,v)\in E} f(u,v) - \sum_{(v,w)\in E} f(v,w) = 0$$

$$\sum_{(s,v)\in E} f(s,v) = d$$

$$\sum_{(v,t)\in E} f(v,t) = d$$

其中，目标函数为最小化源点s到汇点t的流量，使得总流量最小费用。第一条约束表示每条边的流量不超过其容量；第二条表示对于每个中间节点，流入的总流量等于流出的总流量，保持流的守恒性；第三条表示源点s流出的总流量等于需求量d；第四条表示汇点t流入的总流量等于需求量d。

14. 用递归树方法求解递归式：$T(n) = 3T(n/3) + O(n)$（画出递归树）。

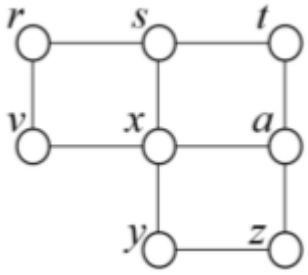


假设递归树的层数为k，则$n\left(\frac{1}{3}\right)^k = 1$，解得$k = \log_3 n$。由于每一层的工作量是$O(n)$，树的高度为$\log_3 n$层，即递归树的总工作量为每层工作量乘层数：

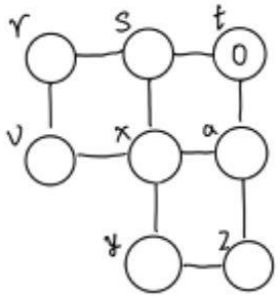
总工作量 $= O(n) \cdot k = O(n) \cdot \log_3 n = O(n \cdot \log n)$

即最终求得$T(n) = 3T(n/3) + O(n)$的时间复杂度为$O(n \cdot \log n)$。

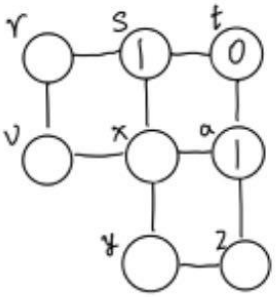
15. 画图分析广度优先搜索的推进过程，起点为t点。



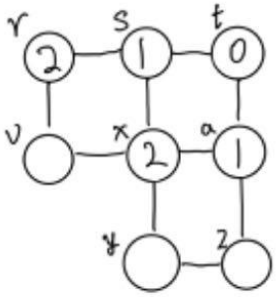
a)



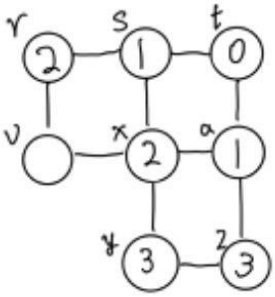
b)



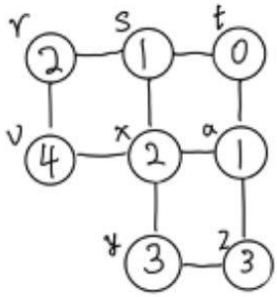
c)



d)



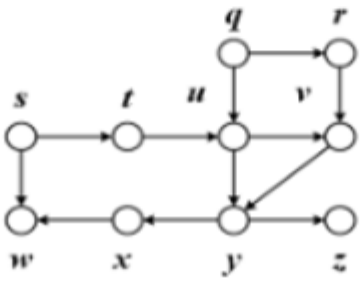
e)



16. 给出算法给出 TOPOLOGICAL-SORT 运行于图上时所生成的结点次序，并画图分析深度优先搜索的推进过程（请给出每个结点的发现时间和完成时间，并给出每条边的分类）。假定深度优先搜索算法 DFS 的第 5~7 行的 for 循环是以字母表顺序依次处理每个结点，并假定每条邻接链表皆以字母表顺序对里面的结点进行排序。

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
1   time = time + 1
2   u.d = time
3   u.color = GRAY
4   for each v ∈ G: Adj[u]
5       if v.color == WHITE
6           v.π = u
7           DFS-VISIT(G, v)
8   u.color = BLACK
9   time = time + 1
10  u.f = time
```



图如下所示，其中每个结点中斜杠左边为发现时间，右边为完成时间。

a)

q
r
1/
s
t
u
v
w
x
y
z

b)

q
r
1/
2/
s
t
u
3/
G
v
4/
w
x
5/
y
6/
z

c)

q
r
1/
2/
s
t
u
3/
G
v
4/
8/9
w
7/10
x
5/
y
6/
z

d)

q
r
1/
2/
s
t
u
3/13
G
v
4/14
8/9
w
7/10
x
5/12
y
6/11
z

e)

q
r
1/
2/
s
t
u
3/13
G
v
4/14
8/9
w
7/10
x
5/12
y
6/11
z

f)

q
r
1/
2/15
s
t
u
3/13
G
v
4/14
7/8
w
6/9
x
5/12
y
10/11
z

g)

q
r
1/
2/15
s
t
u
3/13
G
v
4/14
16/
7/8
w
6/9
x
5/12
y
10/11
z

h)

1/20
2/15
s
t
u
3/13
G
v
4/14
16/9
17/18
7/8
w
6/9
x
5/12
y
10/11
z