数据结构相关选讲

TgopKnight

线性数据结构

- 常见线性数据结构

线性数据结构

- 常见线性数据结构
  - 数组：……

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出
  - 链表：快速插入删除

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出
  - 链表：快速插入删除
  - 分块(这个也算？)

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出
  - 链表：快速插入删除
  - 分块(这个也算？)
- 简单应用：

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出
  - 链表：快速插入删除
  - 分块(这个也算？)
- 简单应用：
  - 数组存储……

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出
  - 链表：快速插入删除
  - 分块(这个也算？)
- 简单应用：
  - 数组存储……
  - 单调队列优化DP

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出
  - 链表：快速插入删除
  - 分块(这个也算？)
- 简单应用：
  - 数组存储……
  - 单调队列优化DP
  - 单调栈及凸包

线性数据结构

- 常见线性数据结构
  - 数组：……
  - 队列：先进先出
  - 栈：先进后出
  - 链表：快速插入删除
  - 分块(这个也算？)
- 简单应用：
  - 数组存储……
  - 单调队列优化DP
  - 单调栈及凸包
  - 邻接表存储图结构

练手题

- 请维护一个序列，按顺序进行如下操作：
  - 1.序列中只有1号元素
  - 2.依次将2~N号元素插入序列，每次操作指定一个元素（先前已经入列），并制定插入左边或右边
  - 3.从中间将M个元素删除

- 操作完毕后，请输出整个序列
- 1<=N, M<=100000

练手题

- 请维护一个序列，按顺序进行如下操作：
  - 1.序列中只有1号元素
  - 2.依次将2~N号元素插入序列，每次操作指定一个元素（先前已经入列），并制定插入左边或右边
  - 3.从中间将M个元素删除
- 操作完毕后，请输出整个序列
- 1<=N, M<=100000
- 链表基础题

POI2010 pilots

- 给定一个长度为N的正整数序列A，请求出一个最长的连续子序列，使得该子序列中任意两数差<=K
- 输出合法的最长连续子序列的长度即可

- 1<=N<=3000000，0<=K<=2*10^9，0<=Ai<=2*10^9

POI2010 pilots

- 注意到以r为右端点的合法最长连续子序列中，左端点l单调不减

POI2010 pilots

- 注意到以r为右端点的合法最长连续子序列中，左端点l单调不减
- 用两个单调队列维护，一个单调递增，一个单调递减

POI2010 pilots

- 注意到以r为右端点的合法最长连续子序列中，左端点l单调不减
- 用两个单调队列维护，一个单调递增，一个单调递减
- 如果两端差值>K，则取位置小的一位删除即可

POI2010 pilots

- 注意到以r为右端点的合法最长连续子序列中，左端点l单调不减
- 用两个单调队列维护，一个单调递增，一个单调递减
- 如果两端差值>K，则取位置小的一位删除即可
- 用cnt记录下左端点

POI2010 pilots

- 注意到以r为右端点的合法最长连续子序列中，左端点l单调不减
- 用两个单调队列维护，一个单调递增，一个单调递减
- 如果两端差值>K，则取位置小的一位删除即可
- 用cnt记录下左端点
- 时间复杂度：O(N)

POI2010 blocks

- 给定一个长度为N的正整数序列A，你可以进行如下操作：
  - 每次选择一个大于K的正整数a[i]，将a[i]减去1，选择a[i-1]或a[i+1]中的一个加上1
- 请问：经过若干次操作之后，最大能够选出多长的一个连续子序列，使得这个子序列的每个数都不小于K
- 本题有M组询问，每次询问一个数K，你需要分别回答

- 1<=N<=1000000，1<=M<=50，1<=Ai<=10^9，1<=K<=10^9

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解
- 将Ai减去K，用sum记录前缀和，区间变成和>=0即为合法

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解
- 将Ai减去K，用sum记录前缀和，区间变成和>=0即为合法
- Fi表示右端点为i的合法区间中最小的左端点

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解
- 将Ai减去K，用sum记录前缀和，区间变成和>=0即为合法
- Fi表示右端点为i的合法区间中最小的左端点
- Fi=Min(j) (1<=j<=i 且 sum[i]-sum[j-1]>=0)

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解
- 将Ai减去K，用sum记录前缀和，区间变成和>=0即为合法
- Fi表示右端点为i的合法区间中最小的左端点

- Fi=Min(j) (1<=j<=i 且 sum[i]-sum[j-1]>=0)
  - 显然如果存在k<j而且sum[k]<sum[j]，则j为无用决策

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解
- 将Ai减去K，用sum记录前缀和，区间变成和>=0即为合法
- Fi表示右端点为i的合法区间中最小的左端点

- Fi=Min(j) (1<=j<=i 且 sum[i]-sum[j-1]>=0)
  - 显然如果存在k<j而且sum[k]<sum[j]，则j为无用决策

- 单调栈维护决策即可

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解
- 将Ai减去K，用sum记录前缀和，区间变成和>=0即为合法
- Fi表示右端点为i的合法区间中最小的左端点
- Fi=Min(j) (1<=j<=i 且 sum[i]-sum[j-1]>=0)
  - 显然如果存在k<j而且sum[k]<sum[j]，则j为无用决策
- 单调栈维护决策即可
  - 每次计算Fi时二分答案

POI2010 blocks

- 因为操作次数不限，因此若一段区间平均值超过K，即为一组合法解
- 将Ai减去K，用sum记录前缀和，区间变成和>=0即为合法
- Fi表示右端点为i的合法区间中最小的左端点
- Fi=Min(j) (1<=j<=i 且 sum[i]-sum[j-1]>=0)
  - 显然如果存在k<j而且sum[k]<sum[j]，则j为无用决策
- 单调栈维护决策即可
  - 每次计算Fi时二分答案
- 时间复杂度：O(M*N*logN)

POI2010 blocks

- 想过？不可能

POI2010 blocks

- 想过？不可能
- 这种复杂度一定TLE

POI2010 blocks

- 想过？不可能
- 这种复杂度一定TLE

- 事实上，我们并不需要求出所有的Fi，而只需要最大的i-Fi

POI2010 blocks

- 想过？不可能
- 这种复杂度一定TLE

- 事实上，我们并不需要求出所有的Fi，而只需要最大的i-Fi

- 显然如果有i<k且sum[i]<=sum[k]，那么i-Fi事实上也没有用

POI2010 blocks

- 想过？不可能
- 这种复杂度一定TLE
- 事实上，我们并不需要求出所有的Fi，而只需要最大的i-Fi
- 显然如果有i<k且sum[i]<=sum[k]，那么i-Fi事实上也没有用
- 剩下的询问中，sum具有单调性

POI2010 blocks

- 想过？不可能
- 这种复杂度一定TLE

- 事实上，我们并不需要求出所有的Fi，而只需要最大的i-Fi

- 显然如果有i<k且sum[i]<=sum[k]，那么i-Fi事实上也没有用

- 剩下的询问中，sum具有单调性
- 这样就可以使用两个指针完美解决了

POI2010 blocks

- 想过？不可能
- 这种复杂度一定TLE

- 事实上，我们并不需要求出所有的Fi，而只需要最大的i-Fi

- 显然如果有i<k且sum[i]<=sum[k]，那么i-Fi事实上也没有用

- 剩下的询问中，sum具有单调性
- 这样就可以使用两个指针完美解决了
- 时间复杂度：O(M*N)
- 我当然不会告诉你sum要开long long

树形数据结构

- 常见树形数据结构

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆
  - 树状数组：小常数大利器

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆
  - 树状数组：小常数大利器
  - 线段树：区间问题利器

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆
  - 树状数组：小常数大利器
  - 线段树：区间问题利器
  - 平衡树：比上面那个更加利器

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆
  - 树状数组：小常数大利器
  - 线段树：区间问题利器
  - 平衡树：比上面那个更加利器
- 简单应用：

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆
  - 树状数组：小常数大利器
  - 线段树：区间问题利器
  - 平衡树：比上面那个更加利器
- 简单应用：
  - 优先队列

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆
  - 树状数组：小常数大利器
  - 线段树：区间问题利器
  - 平衡树：比上面那个更加利器
- 简单应用：
  - 优先队列
  - 区间的各种问题(O_O)?

树形数据结构

- 常见树形数据结构
  - 并查集：快速集合合并与查询
  - 二叉堆：查询最值
  - 可并堆：可以合并的二叉堆
  - 树状数组：小常数大利器
  - 线段树：区间问题利器
  - 平衡树：比上面那个更加利器
- 简单应用：
  - 优先队列
  - 区间的各种问题(O_O)?
  - 维护序列
  - ……

NOIP2010 prison

- 给定一个N个点M条边的无向图，请你将N个点分成两个集合，使得两点在同一集合的边权的最大值最小

- 1<=N<=20000，1<=M<=100000

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~而你，却无可奈何~~

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~而你，却无可奈何~~
- 方法一：二分答案

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~两你，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~两你，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去
- 然后只需判断整张图是否可以黑白染色即可

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~而你，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去
- 然后只需判断整张图是否可以黑白染色即可

- 判定一个无向图是否能够黑白染色

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~而你，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去
- 然后只需判断整张图是否可以黑白染色即可

- 判定一个无向图是否能够黑白染色
- 多个连通块显然可以分开考虑

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~而你，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去
- 然后只需判断整张图是否可以黑白染色即可

- 判定一个无向图是否能够黑白染色
- 多个连通块显然可以分开考虑
- 我们首先将一个点染成一种颜色

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~而你，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去
- 然后只需判断整张图是否可以黑白染色即可

- 判定一个无向图是否能够黑白染色
- 多个连通块显然可以分开考虑
- 我们首先将一个点染成一种颜色
- 然后将与之相连的点染成另一种颜色

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~而你，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去
- 然后只需判断整张图是否可以黑白染色即可

- 判定一个无向图是否能够黑白染色
- 多个连通块显然可以分开考虑
- 我们首先将一个点染成一种颜色
- 然后将与之相连的点染成另一种颜色
- 如果某一条边的两个点染成了同一种颜色，则说明不行

NOIP2010 prison

- 这道题我倒是有三种方法AC，~~两俗，却无可奈何~~
- 方法一：二分答案
- 对于所得答案ans，我们将边权<=ans的边删去
- 然后只需判断整张图是否可以黑白染色即可

- 判定一个无向图是否能够黑白染色
- 多个连通块显然可以分开考虑
- 我们首先将一个点染成一种颜色
- 然后将与之相连的点染成另一种颜色
- 如果某一条边的两个点染成了同一种颜色，则说明不行
- 时间复杂度：O((N+M)*logM)

NOIP2010 prison

- 方法二：贪心

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来进来的边，考虑两个点x,y

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来的边，考虑两个点x,y
   - 若x,y都没有分配好，则判断将x与y分在两个集合中边权最大值较小的方案，并按方案将x与y分在对应的集合中

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来进来的边，考虑两个点x,y
    - 若x,y都没有分配好，则判断将x与y分在两个集合中边权最大值较小的方案，并按方案将x与y分在对应的集合中
    - 若x,y中只有一个点配好了，则将另一个点分在另一个集合中

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来进来的边，考虑两个点x,y
   - 若x,y都没有分配好，则判断将x与y分在两个集合中边权最大值较小的方案，并按方案将x与y分在对应的集合中
   - 若x,y中只有一个点配好了，则将另一个点分在另一个集合中
   - 若x,y都分配好了，则判断两点是否在同一集合中

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来进来的边，考虑两个点x,y
    - 若x,y都没有分配好，则判断将x与y分在两个集合中边权最大值较小的方案，并按方案将x与y分在对应的集合中
    - 若x,y中只有一个点配好了，则将另一个点分在另一个集合中
    - 若x,y都分配好了，则判断两点是否在同一集合中
        - 若不在同一集合中，无需处理

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来进来的边，考虑两个点x,y
   - 若x,y都没有分配好，则判断将x与y分在两个集合中边权最大值较小的方案，并按方案将x与y分在对应的集合中
   - 若x,y中只有一个点配好了，则将另一个点分在另一个集合中
   - 若x,y都分配好了，则判断两点是否在同一集合中
     - 若不在同一集合中，无需处理
     - 若在同一集合中，则输出边权

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来进来的边，考虑两个点x,y
   - 若x,y都没有分配好，则判断将x与y分在两个集合中边权最大值较小的方案，并按方案将x与y分在对应的集合中
   - 若x,y中只有一个点配好了，则将另一个点分在另一个集合中
   - 若x,y都分配好了，则判断两点是否在同一集合中
     - 若不在同一集合中，无需处理
     - 若在同一集合中，则输出边权

- 时间复杂度：O(M*logM+M)

NOIP2010 prison

- 方法二：贪心
- 我们按边权从大到小贪心

1. 首先将边权最大的两个点分到两个集合中
2. 对于接下来进来的边，考虑两个点x,y
   - 若x,y都没有分配好，则判断将x与y分在两个集合中边权最大值较小的方案，并按方案将x与y分在对应的集合中
   - 若x,y中只有一个点配好了，则将另一个点分在另一个集合中
   - 若x,y都分配好了，则判断两点是否在同一集合中
     - 若不在同一集合中，无需处理
     - 若在同一集合中，则输出边权

- 时间复杂度：O(M*logM+M)

ERROR!

NOIP2010 prison

- 方法三：贪心Ex

NOIP2010 prison

- 方法三：贪心Ex
- 我们按边权从大到小贪心

NOIP2010 prison

- 方法三：贪心Ex
- 我们按边权从大到小贪心
- 针对每一个点x，构建一个虚拟点x’

NOIP2010 prison

- 方法三：贪心Ex
- 我们按边权从大到小贪心
- 针对每一个点x，构建一个虚拟点x’
- 若x在同一集合内，则x’在另一集合内

NOIP2010 prison

- 方法三：贪心Ex
- 我们按边权从大到小贪心
- 针对每一个点x，构建一个虚拟点x’
- 若x在同一集合内，则x’在另一集合内
- 对于每一条边x,y，若x与y在同一集合内，则输出边权

NOIP2010 prison

- 方法三：贪心Ex
- 我们按边权从大到小贪心
- 针对每一个点x，构建一个虚拟点x’

- 若x在同一集合内，则x’在另一集合内

- 对于每一条边x,y，若x与y在同一集合内，则输出边权
- 否则将x’与y，x与y’合并到同一集合中

NOIP2010 prison

- 方法三：贪心Ex
- 我们按边权从大到小贪心
- 针对每一个点x，构建一个虚拟点x’

- 若x在同一集合内，则x’在另一集合内

- 对于每一条边x,y，若x与y在同一集合内，则输出边权
- 否则将x’与y，x与y’合并到同一集合中

- 时间复杂度：$O(M*logM+M*\alpha(N))$

NOIP2010 prison

- 方法三：贪心Ex
- 我们按边权从大到小贪心
- 针对每一个点x，构建一个虚拟点x’

- 若x在同一集合内，则x’在另一集合内

- 对于每一条边x,y，若x与y在同一集合内，则输出边权
- 否则将x’与y，x与y’合并到同一集合中

- 时间复杂度：O(M*logM+M*α(N))
- 请注意原图为二分图的情况，此时答案为0

Example 1

- 给定一个长为N的数组A，sum[l,r]=$\sum_{i=l}^{r} A[i]$
- 求前k小的sum[l,r]，两者不同当且仅当[l1,r1]，[l2,r2]中l1!=l2或r1!=r2
- 1<=A[i]<=10^9，N,k<=100000

Example 1

- 标准堆的应用题

Example 1

- 标准堆的应用题
  - 注意到A[i]>0，则若l<=r1<r2有sum[l,r1]<sum[l,r2]

Example 1

- 标准堆的应用题
  - 注意到A[i]>0，则若l<=r1<r2有sum[l,r1]<sum[l,r2]
  - 即若sum[l,r1]未输出的情况下sum[l,r2]一定不会输出

Example 1

- 标准堆的应用题
  - 注意到A[i]>0，则若l<=r1<r2有sum[l,r1]<sum[l,r2]
  - 即若sum[l,r1]未输出的情况下sum[l,r2]一定不会输出

- 记F[l]表示以l为左端点，下一个输出的右端点到了F[l]

Example 1

- 标准堆的应用题
  - 注意到A[i]>0，则若l<=r1<r2有sum[l,r1]<sum[l,r2]
  - 即若sum[l,r1]未输出的情况下sum[l,r2]一定不会输出
- 记F[l]表示以l为左端点，下一个输出的右端点到了F[l]
  - 以sum[l,F[l]]为权值建立一个堆，输出堆顶后让F[l]++再进行调堆

Example 1

- 标准堆的应用题
  - 注意到A[i]>0，则若l<=r1<r2有sum[l,r1]<sum[l,r2]
  - 即若sum[l,r1]未输出的情况下sum[l,r2]一定不会输出
- 记F[l]表示以l为左端点，下一个输出的右端点到了F[l]
  - 以sum[l,F[l]]为权值建立一个堆，输出堆顶后让F[l]++再进行调堆
- 单次操作复杂度为O(logN)

可并堆

- 简单介绍一种可并堆的实现

可并堆

- 简单介绍一种可并堆的实现
- 我们将其称之为左偏堆(树)

可并堆

- 简单介绍一种可并堆的实现
- 我们将其称之为左偏堆(树)
- 树如其名，这个堆左边重，右边轻

可并堆

- 简单介绍一种可并堆的实现
- 我们将其称之为左偏堆(树)
- 树如其名，这个堆左边重，右边轻

- 首先我们需要了解一些概念：

可并堆

- 简单介绍一种可并堆的实现
- 我们将其称之为左偏堆(树)
- 树如其名，这个堆左边重，右边轻

- 首先我们需要了解一些概念：
  - 键(权)值key：每个节点的点权，满足堆的性质
  - 距离dist：每个节点距离以自己为根的子树中，叶子节点的最短距离，满足dist[l[i]]>=dist[r[i]]

可并堆

- 简单介绍一种可并堆的实现
- 我们将其称之为左偏堆(树)
- 树如其名，这个堆左边重，右边轻

- 首先我们需要了解一些概念：
  - 键(权)值key：每个节点的点权，满足堆的性质
  - 距离dist：每个节点距离以自己为根的子树中，叶子节点的最短距离，满足dist[l[i]]>=dist[r[i]]

- 整棵树的距离就是根节点的dist

可并堆

- 简单介绍一种可并堆的实现
- 我们将其称之为左偏堆(树)
- 树如其名，这个堆左边重，右边轻

- 首先我们需要了解一些概念：
  - 键(权)值key：每个节点的点权，满足堆的性质
  - 距离dist：每个节点距离以自己为根的子树中，叶子节点的最短距离，满足dist[l[i]]>=dist[r[i]]

- 整棵树的距离就是根节点的dist
- 抽象的看，dist表示的就是以当前点为根的子树最浅的叶子有多深

可并堆

- 几个性质：

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点
  2. 一颗N个节点的左偏树距离最多为log(N+1)-1

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点
  2. 一颗N个节点的左偏树距离最多为log(N+1)-1
  3. dist[x]=dist[r[x]]+1

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点
  2. 一颗N个节点的左偏树距离最多为log(N+1)-1
  3. dist[x]=dist[r[x]]+1

- 左偏树的操作：

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点
  2. 一颗N个节点的左偏树距离最多为log(N+1)-1
  3. dist[x]=dist[r[x]]+1

- 左偏树的操作：
  1. 合并Merge(x,y)：核心操作

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点
  2. 一颗N个节点的左偏树距离最多为log(N+1)-1
  3. dist[x]=dist[r[x]]+1

- 左偏树的操作：
  1. 合并Merge(x,y)：核心操作
  2. 插入Insert(x,y)：将原堆与一个节点合并即可

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点
  2. 一颗N个节点的左偏树距离最多为log(N+1)-1
  3. dist[x]=dist[r[x]]+1

- 左偏树的操作：
  1. 合并Merge(x,y)：核心操作
  2. 插入Insert(x,y)：将原堆与一个节点合并即可
  3. 删除Delete(x)：将x的两颗子树合并，并用得到的新根节点替代x

可并堆

- 几个性质：
  1. 若一棵左偏树的距离为k，则这棵左偏树至少有2^(k+1)-1个节点
  2. 一颗N个节点的左偏树距离最多为log(N+1)-1
  3. dist[x]=dist[r[x]]+1

- 左偏树的操作：
  1. 合并Merge(x,y)：核心操作
  2. 插入Insert(x,y)：将原堆与一个节点合并即可
  3. 删除Delete(x)：将x的两颗子树合并，并用得到的新根节点替代x

- 如何精妙的实现核心操作？

可并堆

```
Function  Merge(A, B)
    If  A = NULL  Then  return B
    If  B = NULL  Then  return A
    If  key(B) < key(A)  Then  swap(A, B)
    right(A) ← Merge(right(A), B)
    If  dist(right(A)) > dist(left(A))  Then
        swap(left(A), right(A))
    If  right(A) = NULL  Then  dist(A) ← 0
    Else  dist(A) ← dist(right(A)) + 1
    return A
End  Function
```

可并堆

```
Function Merge(A, B)
    If A = NULL Then return B
    If B = NULL Then return A
    If key(B) < key(A) Then swap(A, B)
    right(A) ← Merge(right(A), B)
    If dist(right(A)) > dist(left(A)) Then
        swap(left(A), right(A))
    If right(A) = NULL Then dist(A) ← 0
    Else dist(A) ← dist(right(A)) + 1
    return A
End Function
```

- 时间复杂度?

可并堆

```
Function Merge(A, B)
    If A = NULL Then return B
    If B = NULL Then return A
    If key(B) < key(A) Then swap(A, B)
    right(A) ← Merge(right(A), B)
    If dist(right(A)) > dist(left(A)) Then
        swap(left(A), right(A))
    If right(A) = NULL Then dist(A) ← 0
    Else dist(A) ← dist(right(A)) + 1
    return A
End Function
```

- 时间复杂度?
- 合并：O(logN1+logN2)
- 插入与删除：近似O(logN)

Example 2

- 给定一个长为N的数列A，求其中有多少对逆序对
  - 逆序对定义：一对i,j满足i<j且A[i]>A[j]

Example 2

- 给定一个长为N的数列A，求其中有多少对逆序对
  - 逆序对定义：一对i,j满足i<j且A[i]>A[j]

- 归并排序/线段树

Example 2

- 给定一个长为N的数列A，求其中有多少对逆序对
  - 逆序对定义：一对i,j满足i<j且A[i]>A[j]

- 归并排序/线段树

- 给定一个长为N的数列A，求其中有多少对本质不同的逆序对
  - 本质不同的逆序对定义：(l1,r1)和(l2,r2)两对逆序对中A[l1]!=A[l2]或A[r1]!=A[r2]

Example 2

- 给定一个长为N的数列A，求其中有多少对逆序对
  - 逆序对定义：一对i,j满足i<j且A[i]>A[j]
- 归并排序/线段树
- 给定一个长为N的数列A，求其中有多少对本质不同的逆序对
  - 本质不同的逆序对定义：(l1,r1)和(l2,r2)两对逆序对中A[l1]!=A[l2]或A[r1]!=A[r2]
- 只需从左往右插入A[i]并检查有多少种数字比它小，若这个数字插入过一次就把上一次的答案删了重算。

Example 2

- 给定一个长为N的数列A，求其中有多少对逆序对
  - 逆序对定义：一对i,j满足i<j且A[i]>A[j]

- 归并排序/线段树

- 给定一个长为N的数列A，求其中有多少对本质不同的逆序对
  - 本质不同的逆序对定义：(l1,r1)和(l2,r2)两对逆序对中A[l1]!=A[l2]或A[r1]!=A[r2]

- 只需从左往右插入A[i]并检查有多少种数字比它小，若这个数字插入过一次就把上一次的答案删了重算。
  - 使用线段树可以解决

Physics

- 请维护一颗树，支持更换父亲，查询深度，查询子树大小
- 点数和操作次数均<=100000

Physics

- 请维护一颗树，支持更换父亲，查询深度，查询子树大小
- 点数和操作次数均<=100000

- 将子树大小看作权值，每次断开相当于修改一条链上的权值，连接相当于修改另一条链上的权值，我们可以使用Link-Cut Tree解决

Physics

- 请维护一颗树，支持更换父亲，查询深度，查询子树大小
- 点数和操作次数均<=100000

- 将子树大小看作权值，每次断开相当于修改一条链上的权值，连接相当于修改另一条链上的权值，我们可以使用Link-Cut Tree解决
  - 在NOIP考场上写Eular-Tour Tree是不现实的

Physics

- 请维护一颗树，支持更换父亲，查询深度，查询子树大小
- 点数和操作次数均<=100000

- 将子树大小看作权值，每次断开相当于修改一条链上的权值，连接相当于修改另一条链上的权值，我们可以使用Link-Cut Tree解决
  - 在NOIP考场上写Eular-Tour Tree是不现实的
- 我们可以直接维护DFS序，即每个点的DFS进入时间和离开时间

Physics

- 请维护一颗树，支持更换父亲，查询深度，查询子树大小
- 点数和操作次数均<=100000

- 将子树大小看作权值，每次断开相当于修改一条链上的权值，连接相当于修改另一条链上的权值，我们可以使用Link-Cut Tree解决
  - 在NOIP考场上写Eular-Tour Tree是不现实的
- 我们可以直接维护DFS序，即每个点的DFS进入时间和离开时间
  - 对于X，深度即为进入时间在X之前，离开时间在X之后的点的个数
  - 子树大小即为进入时间在X之后，离开时间在X之前的点的个数
  - 更换父亲直接将进入时间在X之后，离开时间在X之前这段区间移位即可

Physics

- 请维护一颗树，支持更换父亲，查询深度，查询子树大小
- 点数和操作次数均<=100000

- 将子树大小看作权值，每次断开相当于修改一条链上的权值，连接相当于修改另一条链上的权值，我们可以使用Link-Cut Tree解决
  - 在NOIP考场上写Eular-Tour Tree是不现实的
- 我们可以直接维护DFS序，即每个点的DFS进入时间和离开时间
  - 对于X，深度即为进入时间在X之前，离开时间在X之后的点的个数
  - 子树大小即为进入时间在X之后，离开时间在X之前的点的个数
  - 更换父亲直接将进入时间在X之后，离开时间在X之前这段区间移位即可
- 以上操作均可用splay维护

讲完了

- 预祝大家NOIP取得好成绩~