Algorithm-homework2
Greedy Algorithm

孔静-2014K8009929022

October 23, 2016

Contents

1 Greedy Algorithm

a

先经过一次最简单的判断，即 $\sum_{i=1}^{n} d_i$ 是否是偶数。

若是偶数:

$$Opt(D,n) = \begin{cases} 1, n = 0 \\ 0, d_{\max} \geq n \\ Opt(D', n-1), otherwise \end{cases}$$

其中 D 是已经排序好的 $d_1, d_2 ... d_n$ 递增序列。

$$d_i{}' = \begin{cases} 0, i = n \\ d_i - 1, n - d_n \leq i < n \\ d_i, i < n - d_n \end{cases}$$

b

先判断是否已经能判断了，如果最大的度大于等于结点数，那么必然不能成图。然后，将度序列排序成递增序列，将度最大的节点 $d_{\max}$ 即 d_n 删去，同时对剩余度最大的 d_n 个减 1。相当于从图上删去某节点以及与它相连的线。然后重复这个过程。

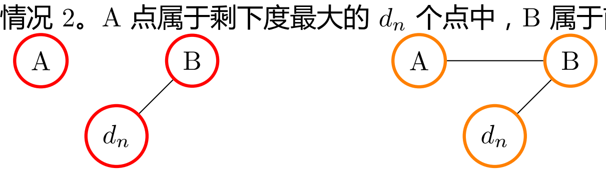
Algorithm 1 Problem1

```
procedure PROBLEM(D,n)
    if add(D)/2! = 0 then
        Opt[n+1] = 0
    elseOpt[n+1] = 1
    end if
    for i = n to 1 do
        quicksort(D,n)                                  ▷ 对前 n 个数，快速排序
        if d(n) >= n || d(1) < 0 || Opt[i+1] == 0 then
            Opt[i] = 0
        else
            Opt[i] = 1
        end if
        for i = n - d(n) to n - 1 do
            d(i) = d(i) - 1
        end for
        n = n - 1
    end for
    return Opt[1]                                       ▷ 如果能构成图，应该输出 1
end procedure
```

c

下面证明为何删去最大度的节点，不影响是否成图。

考虑度最大的节点 n，如果它与剩下度最大的 d_n 个点，有两种情况，一他们相连，那么删去正好。二他们不相连，n 节点还与前 $n-d_n$ 中有相连，下考虑情况 2。A 点属于剩下度最大的 d_n 个点中，B 属于前前 $n-d_n$ 个点。



情况 2 根据 B 和 A 相连不相连又有两种情况：

一（红色），从递增序列来看，A 的度大于等于 B，图中 B 已经少了一个度，所以在剩余 n-3 个点中，必然有一个结点 C 与 A 相连，不与 B 相连。将 AC 连线，Bd_n 连线删去，改为 Ad_n，BC，不影响度数。

二（橙色），将 d_n 与 A 相连，d_n 和 A 均与 B 不连，将 B 插入任意两个和 B 不相连的节点的连线中即可。首先 d_n 已经与 A 不相连，B 的度比 d_n 小，那么至少存在一个与 B 不相连的结点，如果不存在第二个与 B 不相连的结点，说明 B 的度与 d_n 相同，那么也与 A 同，将 B 和 A 结点位置互换即可。

综上这种删去最大度的节点，并删去它所连的线的方法，对是否成图不影响。所以能利用该方法减少结点数，判断是否成图。

d

快排 O(nlogn)，删除节点 O(n)，遍历 O(n)。

综上算法复杂度是 O(n^2logn)。

2 Greedy Algorithm

a

先根据 f_i 从大到小排序。

$$Opt(i,1) = \max\{Opt(i-1,0) + s_i + f_i, Opt(i-1,1)\}$$

b

Algorithm 2 Greedy Algorithm

---

procedure Greedy Algorithm(S,F,n)
  quicksort(S,F) ▷ 根据 f_i 从大到小排序
  for i = 1 to n **do**
    Opt[i][1] = max(Opt[i-1][0] + s[i] + f[i], Opt[i-1][1])
    Opt[i][0] = Opt[i-1][0] + s[i]
  end for
  return $Opt[n]$
end procedure

---

c

如图（红色是 supercomputer 运行时间，橙色是 PC 运行时间，显然应该是 PC 运行时间长的第一条线先运行）

因为不考虑切换任务以及交替任务的时间，所以超级计算机运行的总时间是固定的，即 $\sum_{i=1}^{n} s_i$，所以总共需要的时间，取决于普通计算机的运行结束时间，那当然是需要运行普通计算机最长时间的任务最先开始了。

d

快排 O(nlogn)，遍历 O(n)。
综上算法复杂度是 O(nlogn)。

3 Greedy Algorithm

a

先根据岛屿的 x 轴坐标从小到大排序，并计算以每个岛屿 i 为圆心以 d 为半径的圆与 x 轴的交点，记为 $left_i$, 和 $right_i$。

$$Opt(i) = \begin{cases} Opt(i-1)+1, left_i > l_{Opt(i-1)} \\ Opt(i-1), left_i \le l_{Opt(i-1)} \end{cases}$$

其中 l_i 记录的是雷达坐标，l_1 是 $right_1$，之后是 Opt(i) 雷达数量增加 1，令 $l_{Opt(i)}$

b

Algorithm 3 Greedy Algorithm

```
procedure GREEDY ALGORITHM(X,Y,n)
    quicksort(X,Y,n)                              ▷ 根据 x 坐标从小到大快速排序
    l[1] = right[1]
    Opt[1] = 1
    for i = 2 to n do
        if left[i] > l[Opt[i-1]] then
            Opt[i] = Opt[i-1] + 1
            l[Opt[i]] = right[i]
        else if right[i] < l[Opt[i-1]] then
            Opt[i] = Opt[i-1]
            l[Opt[i]] = right[i]
        else
            Opt[i] = Opt[i-1]
        end if
    end for
    return Opt[n]
end procedure
```

c

从 x 坐标最小的岛屿开始考虑，雷达必覆盖这个岛屿，并且尽可能多的去覆盖其他岛屿，所以雷达建立在这个岛屿对应的 $right_1$，之后一样，$left_i$ 在这个雷达右边的，就表示该雷达覆盖不了，必须新建雷达；如果 $left_i$ 在雷达左边，分两种情况，一是 $right_i$ 在雷达右边，即该雷达能覆盖这个岛屿，二是 $right_i$ 在雷达左变，因为我们排序是按照岛屿横坐标排序的，此情况下，把该雷达移动到 $right_i$ 上即可以覆盖之前的岛屿也能覆盖该岛屿了。

d

快速排序 O(nlogn)，遍历是 O(n)。
综上算法复杂度是 O(nlogn)。

4 Greedy Algorithm

a 对 A，B 排序即可。

b 排序就好，还要算法吗。。。ORZ。。。

c 取对数，$\sum_{i=1}^{n} b_i \times \ln(a_i)$，显然当 a_i 和 b_i 同为第 k 大的数时，该式最大。

d 排序 O(nlogn)，所以算法复杂度是 O(nlogn)。