# 做男人不容易系列 Solutions

**Amber**

2 years ago, I trained in Fudan University in Shanghai and attended this contest hold by Lou Tiancheng. The title of this contest is “hard to be a man”and the subtitle is “If you are a man, pass all 8 problems”. I got only 1 problem accepted at that time. Actually, no one can be “a man” except LTC himself at that time. Now there is still only 3 guys who finished all problems except the problemsetter LTC. They are timgreen, geworm, Savior. How I admire them!

Well, I decide to be a man before my 18th birthday. For my foreign friends, I decide to write the English version in the description and the solution with my poor English.

# Problem A (POJ 1737): Connected Graph

[Description]

Find the number of connected labeled undirected graphs with n nodes.

找出有标号的 n 个点的无向连通图的个数.

[Solution]

If we do not considered whether the graph is connected, there exists 2^C(n, 2) graphs in total. Let G(n) = 2^C(n, 2). If we know the number of unconnected ones, we will know the answer indirectly. Let F(n) is the answer. Consider the connected component C that contains node 1 and with the size k, namely |C| = k. Because C is unconnected, k < n. So there exists C(n - 1, k - 1) ways to choose another k - 1 nodes to make up connected component C. So there exists C(n - 1, k - 1) * F(k) different connected component C. Any graphs G(n-k) for remaining nodes are okay. So the answer is:

$$F(n) = G(n) - \sum_{k=1}^{n-1} \binom{n-1}{k-1} F(k)G(n-k)$$

It takes O(n^2) to calculate this formula.

利用**补集转化**思想, 若不考虑连通性,则有 n 个点的任意图有 G(n)=2^C(n, 2)个. 只要求出不连通图的个数就知道答案 F(n). 考虑含结点 1 的连通块 C, |C| = k. 由于不连通,有 k < n 成立. 则有 C(n - 1, k - 1)种方法选择另 k-1 个结点与结点 1 组成 C. 所以有 C(n - 1, k - 1) * F(k)种方法组成 C. 剩下的结点组成一个任意图即可. 答案为:

$$F(n) = G(n) - \sum_{k=1}^{n-1} \binom{n-1}{k-1} F(k)G(n-k)$$

复杂度为 O(n^2).

[Reference]

[1]The information from the On-Line Encyclopedia of Integer Sequences: A001187 Number of connected labeled graphs with n nodes.

[2]杨俊, <<PKU 1737 解题报告>>, IOI2005 国家集训队报告

# Problem B (POJ 1738): An old Stone Game

[Description]

There are n(n<=50000) piles of stones in a line. Each time we can merge two adjoining piles to a new pile until there is only one pile. Minimize the summation of the number of stones in the new pile after merging.

在开始有 n(n<=50000)堆的石头堆排成一行.每次可以合并相邻的两个石头堆成为一个新堆,可以得到这个新堆的石头数的分数.直到剩下一堆.找出一种合并计划,使分数最小,求这个最小分数.

[Solution]

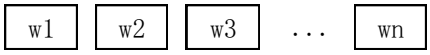
This problem is equivalent to construct an Optimal Alphabetic Binary Search Tree (OABST).As the same As Huffman tree, minimize the summation of the weighted paths lengths of the external nodes(leaves). The difference is that the external nodes in OABST must be in order (alphabetic), namely merge the only adjoining piles each time. It can be solved in O(n^2) by dynamic programming after optimizing by the quadrilateral inequation. But it is not fast enough for this problem. The Hu-Tucker algorithm with the complexity O(nlogn) will be introduced in the following. This algorithm has 3 phases. Because this problem does not need to output the merging method, just to implement the Phase 1 is okay.

**Phase 1, Combination**:

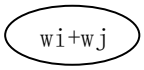
The goal of this phase is to build an optimal binary tree in which the depths of the external nodes are the same as the final OABST.

The **work sequence** is a sequence which consists of the internal or external nodes.

The initial work sequence of nodes consists of all the external nodes in order.




During each iteration of this phase a new working sequence is produced by combining two nodes wi and wj in the previous working sequence into one internal node wi+wj which replaces the leftmost node wi of the two and removing the rightmost one wj from the current sequence.




We call the pair of the nodes wi, wj which is combined in each iteration the **Local Minimum Compatible Pairs** (LMCP). In each iteration, LMCP is unique. It satisfies the following 4 rules:

Rule(1): In the current working sequence no external nodes occur between wi and wj.

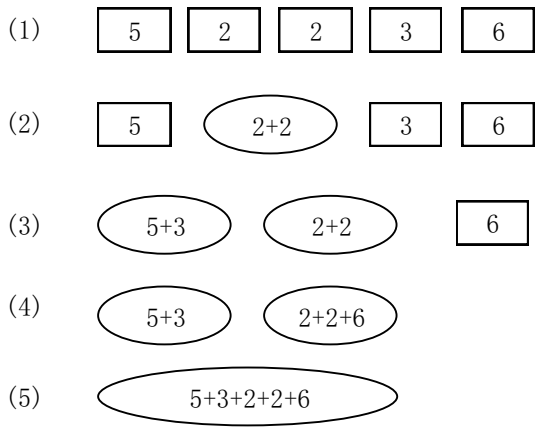
Rule(2): The combined weight w = wi + wj is minimum.

Rule(3): If in case of tie, the subscript i of the leftmost nodes wi needs minimized.

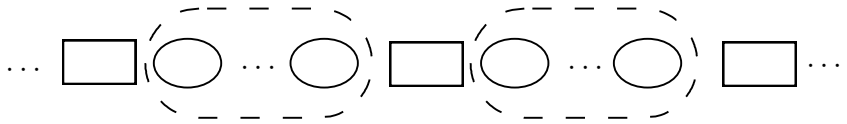
Rule(4): If still in case of tie, the subscript j of the rightmost nodes wj needs minimized.

We use the rectangle to denote an external node and the circle to denote an internal node in the following.

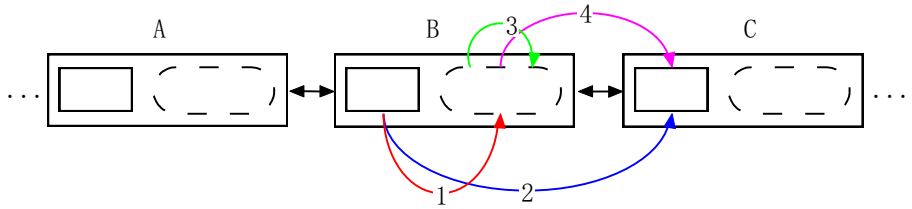
For example, n = 5, w = (5, 2, 2, 3, 6). Notice that in Step (2), LMCP (5, 3) spans the internal node (2+2).




The implementing of this phase is the most complex and the most important also. We can find that **the order of the continuous internal nodes does not matter**. Namely the internal nodes between the two adjoining external nodes are similar to be executed the greedy algorithm of Huffman tree in part. In other words, external nodes divide the work sequence into some segments, the internal nodes between the two adjoining external nodes builds a set of the internal nodes (use the dashed circle to denote).



Because the external node and the set of the internal nodes occurs alternately, we use a **block** (use the big rectangle to denote) to contain the external node and the set of the internal nodes in the right. We use two-directed links to link them up.




List the case of the occurring of the LMCP

Let B be the current block.

Case 1: The LMCP is the external node in B and the minimum internal node in B.

After combining the LMCP, put the new node into the set of internal nodes in A and unite the set of internal nodes in B into A's.

Case 2: The LMCP is the external node in B and the external node in C.

After combining the LMCP, put the new node into the set of internal nodes in A and unite the set of internal nodes in B and C into A's.

Case 3: The LMCP is the minimum and second minimum internal node in B.

After combining the LMCP, put the new node into the set of internal nodes in B.

Case 4: The LMCP is the minimum internal node in B and the external node in C.

After combining the LMCP, put the new node into the set of internal nodes in B and unite the set of internal nodes in C into B's.

The set of internal nodes needs the operation of finding the minimum and second minimum element and uniting. The normal mergable heap such as Leftist Tree and Pair Heap and Fibonacci Heap are okay.

In each iteration, it is needed to find the LMCP. Because the block can be merged, there exists the large number of the inserting and deleting operation. So we use a deletable priority queue to maintain the pairs of the nodes. The relation of the pairs of the nodes is that the weight summation of the pairs as the first key, the block ID as the second key and the number of the case as the third key.

So the maintaining of any case can be done in O(logn).

The total time of this phase is O(nlogn).

**Phase 2, Level Assignment**:

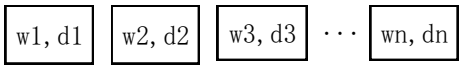
The goal of this phase is to get the depth of all the external nodes in the optimal tree which is constructed in Phase 1. In Phase 1, record the children information of each node (both internal and external nodes). Travel the tree by recursion can get the depths.

The total time of this phase is O(n).

**Phase 3, Recombination**:

The internal nodes in Phase 1 and Phase 2 is not used in this phase. This phase builds an OABST from the initial sequence of external nodes and their depths calculated in Phase 2.

We arrange the external nodes and their depths to the work sequence.




During each iteration of this phase, the pair of adjacent nodes which are at the same maximum depth d are combined to produce a node at depth d - 1 and replace the leftmost one of the two and removing the rightmost one. Exactly, the pair (wi, wj) satisfies:

Rule(1): The nodes wi and wj must be adjoining in the work sequence.

Rule(2): The depths di and dj must be maximized among all the remaining nodes.

Rule(3): If in case of tie, the subscript i of the node wi must be minimized.

According to the rules above, the depths in the pair (wi, wj) must be the same. Actually, only the depths produced by Phase 1, 2 can be executed the Phase 3 correctly. Not any depths assignment can be done.

Use one queue and one stack to implement this algorithm:

1. Put the whole work sequence to the queue in order. The stack is empty.

2. If the 2 elements at the top of stack have equal depths d, then pop the two, push the new nodes at depth d - 1 produced by combining the two, and turn Step 2; else turn Step 3.

3. If the queue is non-empty, pop the head of the queue and push it into the stack; else finish the algorithm.

The total time of this phase is O(n).

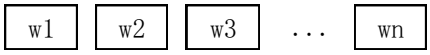
“石子归并”问题等价于构造一棵 Optimal Alphabetic Binary Search Tree (OABST). 和 Huffman 树最优性要求一样, 都是最小化外部结点(叶子)到根的带权路径总长. 区别是 OABST 的外部结点要求有序. 即每次合并要求石子是相邻的. 可以用四边形优化后的动态规划在O(n^2)内解决, 但对于本题来说是不够快的. 下面介绍O(nlogn)的Hu-Tucker算法, 该算法分为三个阶段, 由于本题不要求输出路径, 就是说执行阶段1即可.

**阶段 1, 组合:**

先不考虑有序性，构造一棵最优树，使得所有叶子的深度和最终所求的OABST的叶子的深度一样且最优性一样.

定义**工作序列**是一个可以包含内部或外部结点的序列.

开始时，把所有外部结点按顺序放在工作序列内.




在每次的迭代中,选出两个结点 wi，wj 合并为一个内部结点 w=wi+wj，来替换结点 wi，从工作序列中删去原来的 wj，形成一个新序列.


称每次的迭代被组合的点对(wi，wj)，为**Local Minimum Compatible Pairs**(LMCP). 每次迭代的选择的 LMCP 是唯一的，它满足如下 4 条规则:

规则(1): 在工作序列中，结点 wi 到结点 wj 之间没有外部结点(最重要的规则);

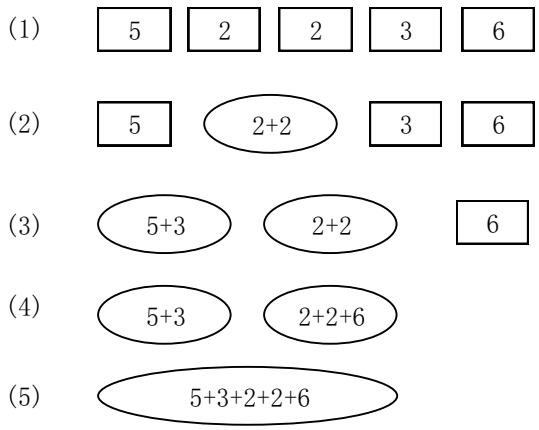
规则(2): 它们的权和 w=wi+wj 要最小;

规则(3): 在满足(1)(2)后,下标 i 要最小;

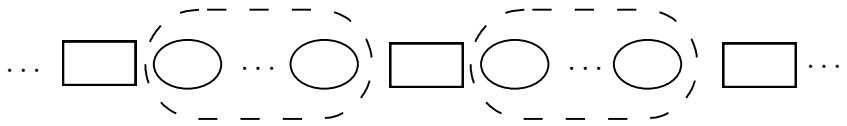
规则(4): 在满足(1)(2)(3)后，下标 j 要最小.

以后统一用方框表示外部结点，用圆圈表示内部结点.

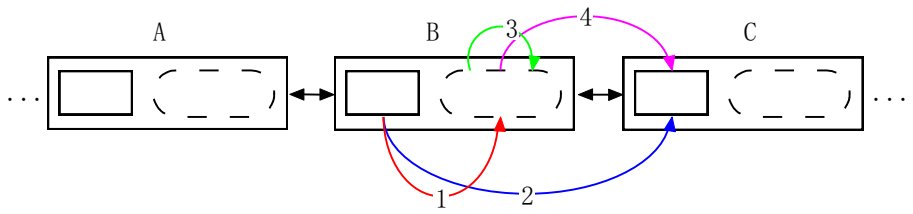
下面举例，n = 5，w = (5，2，2，3，6). 注意: 在第(2)步时，LMCP(5，3)跨越了内部结点(2+2).




本阶段的实现最为复杂，也最为重要，下面介绍实现. 观察算法，发现**连续的内部结点之间的顺序没有意义**. 即两个相邻外部结点间的内部结点，相当于做局部的 Huffman 树的贪心. 可以看成，外部结点将整个工作序列划分为若干的段，两个相邻外部结点间的内部结点，组成一个内部结点集合(用虚圆圈表示内部结点集合).



由于外部结点与内部结点集合交替出现，可以用一个块结构(用外框表示)把外部结点与其右方的内部结点集合包含进来. 再将这些块结构用双向链表串起来.




下面讨论，LMCP 可能出现的情况：

设图中的块结构 B 为当前块结构.

情况(1)：在当前块结构中，外部结点与最小内部结点.

合并 LMCP 后，放入 A 的内部结点集合，同时将 B 的内部结点集合合并到 A 中.

情况(2)：在当前块结构中的外部结点，与右边块结构中的外部结点.

合并 LMCP 后,放入 A 的内部结点集合，同时将 B，C 的内部结点集合合并到 A 中.

情况(3)：在当前块结构的内部结点集合中，最小与次小的内部结点.

合并 LMCP 后,放入 B 的内部结点集合.

情况(4)：在当前块结构中的最小内部结点，与右边块结构中的外部结点.

合并 LMCP 后,放入 B 的内部结点集合，同时将 C 的内部结点集合合并到 B 中.

内部结点集合需要支持找最小和次小元素与合并操作的数据结构来维护. 常见的可并优先队列：左偏树 Leftist Tree，配对堆 Pair Heap，Fibonacci Heap 均可胜任.

每次要找一个全局最小的点对 LMCP，则需要用一个总的优先队列来维护这些点对. 由于块结构的合并,会造成大量合法的点对的改变，这要求总的优先队列支持插入删除操作. 在点对优先级上，点对之间的比较关系为：以点对权值和为第 1 关键字，以当前结构的编号作为第 2 关键字，最后以情况的编号做为第 3 关键字.

这样每种情况的维护都可以在 O(logn)的时间内解决.

整个阶段用时 O(nlogn)

**阶段 2，深度的确定：**

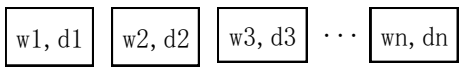
遍历求出执行第 1 阶段后的所有外部结点的在最优树中的深度. 实现时，记录下阶段 1 中每个结点(外部结点和内部结点都有)的儿子，递规遍历一遍，即可求得.

整个阶段用时 O(n)

**阶段 3，重新组合：**

抛弃阶段 1 和阶段 2 的内部结点，重新构造树，得到 OABST.

经过阶段 1 和阶段 2，可得到一个有深度标号的工作序列.




不断地选出两个最大相同深度 d 的相邻结点 wi，wj 合并成一个深度 d-1 的结点,并替换 wi 的位置. 严格地说，选择的点对满足：

规则(1)：wi，wj 在工作序列中要相邻；

规则(2)：di，dj 在所有剩下结点中是最大的；

规则(3)：在满足(1)(2)后,下标 i 要最小.

以上选择的结点，深度必相同即 di = dj. 实际上，通过阶段 1,2 产生的深度标号，才能正确的执行阶段 3，不是任意一种深度标号都可以.

实现上，使用一个队列和一个栈，按如下算法执行即可.

1. 将工作序列按顺序存入队列. 栈清空.

2. 若 2 个栈顶元素有同样的深度标号 d，将它们出栈，合并成一个深度为 d-1 的内部结点，入栈，转

2; 否则, 转 3.

3. 若队列非空, 将队列头入栈, 转 2; 否则, 结束.

整个阶段用时 O(n)

**[Experiment]**

在 Discuss 里有一个 Knuth 的源代码, 不用任何高级数据结构. 我随机生成了 100 个极限数据, Knuth 运行 34.60s, 我运行 27.94s. 但在 POJ 上显示 Knuth 用时 0.250s,我用时 2.062s. 相差甚远. 怀疑 POJ 数据太弱.

**[Reference]**

[1]frkstyc, <<石子归并、OABST 和 Hu-Tucker 算法>>, frkstyc's blog, http://blog.edu.cn/user2/frkstyc/archives/2006/1164187.shtml

[2]Sashka T. Davis, <<Hu-Tucker Algorithm for Building Optimal Alphabetic Binary Search Trees>>, http://www.cs.rit.edu/~std3246/thesis/thesis.html

[3]Donald E. Knuth, <<The Art of Computer Programming>>, Volume 3, Section 6.2.2

**[Thanks]**

This problem jammed me for years. When I got accepted, how happy I am!

Gelin Zhou(cmdButtons) from Changjun Middle School, who introduced the algorithm to me in QQ. That's my first time to know the detail of this O(nlogn) algorithm.

frkstyc, tN from PKU, who helped me for searching some helpful code and thesis.

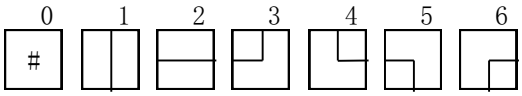
# Problem C (POJ 1739): Tony's Tour

[Description]

Given an n * m gird that has some blocked cells. Find how many paths visit each unblocked cell exactly once from the left low cell to the right low cell in that map.

有个 n 行,m 列的网格,有的格子有障碍物,其他的则可以行走.问有多少条从左下角到右下角的路满足:路经过了所有的可以行走的格子恰好一次.

[Solution]




Because the answer is the large number (e.g. n = m = 8, answer = 8934996), we have to use dynamic programming to solve this problem.

[Reference]

Rujia Liu(SRbGa), <<算法艺术与信息学竞赛>>, p140-142, p184-186

# Problem D (POJ 1740): A New Stone Game

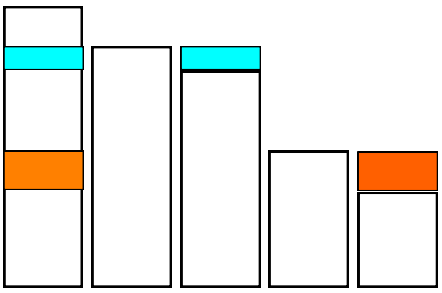
[Description]

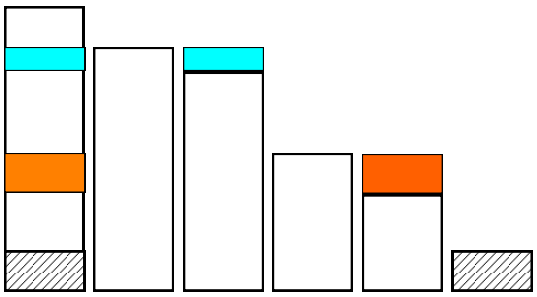
Given n piles of stones (n<=10), two players play in turn. Each time a player can choose a pile, and removed at least one stone from this pile and can move any number of stones from this pile to any other piles (can move to multiple piles). The one who can not operate loses. Who will win?

[Solution]

It's easy to find the balanced state is that all piles can divided into pairs, which two piles in one pair have the same stone number. If one player do some operation in one pile A, on the next turn the other player can do the same operation in the pile that is corresponding to A, namely A and B is in one pair. In other words, the second player can always imitate the thing the first player did in the last step. Because the game ends in limited step, the balanced state is a lose state. The following show how to use only one step to convert any unbalanced state to a balanced state.

Suppose no two piles have the same stone, because they are already balanced. Sort all piles. If the number of piles is odd number, then use the highest pile to fill up the lower pile in the pair right to make the same height with the higher one in the pair left for each pair. The stone for filling in highest pile is disjoint, so it is enough for filling and removing one stone at least.(Figure Left) If the number of piles is even number, then use highest pile to fill up the lower pile in each pair like it is in odd number's case. At last, remove stones in the highest pile to reach the height of the lowest pile and make the lowest pile and the highest pile a pair.(Figure Right)





[Thanks]

Discussion in Fudan University after this contest

# Problem E (POJ 1741): Tree

[Description]

Given a weighted tree with n nodes (n<=10000). Find the number of pair (u, v) that the distance between u and v is not more than k.

给出一个有 n 个节点的树, 一个整数 k. 定义 dist(u, v)为节点 u, v 间的距离. 求这棵树中有多少节点对(u, v)满足 dist(u, v)<=k.

[Solution]

Divide and Conquer.

Each iteration, we should choose an edge (u, v) and divide the tree into two parts disjoined by the edge. Due to avoid from degenerating, that partition edge should be chosen to divide two parts as equally as possible. Then we should merge two parts and count the valid pairs between them. It can be implemented by two sorted list that denotes the distances between u and the posterities of u and the distances between u and the posterities of v respectively. And like merge sort, use two scan line l, r in two list and maintain the property d(u, l) + d(v, r) <= k.

[Thanks]

Yuqian Li(Spaceflyer), from Zhejiang Province, who gave me codes for reference

I discussed with Yangmu, from Fuzhou No.3 Middle School and Chenxue, from Yali Middle School.

# Problem F (POJ 1742): Coins

[Description]

Given n kinds of coins whose value is Wi and whose number is Ci. Find how many k (1<=k<=m) can be reached by combining coins.

有面值为 Ai 的硬币 Ci 个,有 n 种硬币.问价值在 1 到 m 的整数范围内, 有多少个价值能通过现有硬币的组合达到.

[Solution]

It is a classical knapsack problem. The algorithm is as following.

```
For each i in Coins
    For each j in Values
        If F[i - 1, j] is true, M[i, j] = Ci. Else M[i, j] = -1
    For each j
        If M[i, j + Ai] = -1 and M[i, j] > 0
            F[i, j + Ai] = true
            M[i, j + Ai] = M[i, j] - 1
```

It runs in O(nm).

# Problem G (POJ 1743): Musical Theme

[Description]

Given the integer sequence A with length n (n <= 20000). Find the longest consecutive subsequence B in A so that the length of B is not less than 5 at least, there exists the same length consecutive subsequence C in A, the subsequence B and C is non-overlapping, and {Ci - Bi} is a constant sequence.

给出长度为n(1<=n<=20000)的数列A.在这个数列中找出一个最长的至少含有5个数的子数列B, 满足在A中还存在一个同样长度子数列C, 子数列B与子数列C没有交叠部分, 且{Ci - Bi}为常数数列.

[Solution]

Because {Ci - Bi} should a constant sequence, let S[i] = A[i] - A[i - 1]. Then the problem is equivalent to find the non-overlapping longest repeated sequence in S.

Calculate the sorted Suffix Array (SA) of the sequence S by doubling algorithm within O(nlogn) time. And then get the Height array. Here Height[i] denotes the longest common prefix (LCP) of SA[i - 1] and SA[i].

If suppose the length of the longest repeated sequence is not more than the limit L and there exists the solution, then for any limit L'<L, there must exist the solution. It denotes the continuity of the solutions. So use Binary Search to search for the answer L.

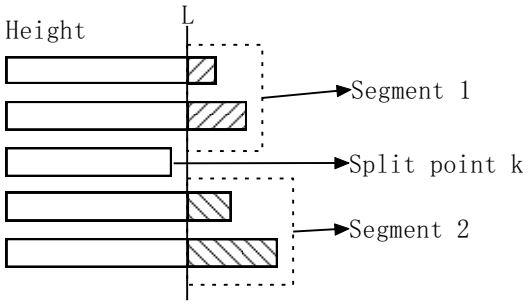
There exists the theorem LCP(SA[i], SA[j]) = RMQ(Height[i+1..j]). So if there exists k that Height[k] < L, then LCP(SA[i], SA[j]) < L for all i, j that i < k < j. Namely, the two suffixes whose common length is L at least must not span the valley k whose Height is less than L. So we can get many consecutive **segments** in SA array after removing all valley k that Height[k] < L. If in the certain consecutive segment there exists i, j that SA[i] + L < SA[j], then there exists two non-overlapping subsequences with the common prefix with length L at least. For implementing, just record the minimum and the maximum of SA in each consecutive segment.

因为{Ci - Bi}为常数列, 所以令S[i] = A[i] - A[i-1]. 问题就转化为在序列S中, 找最长的不相交重复子串.

用倍增算法在O(nlogn)时间内, 计算将序列S的所有后缀排序后的后缀数组SA. 之后, 求出高度数组Height. 这里Height[i]表示SA[i - 1]与SA[i]的最长公共前缀(LCP).

若在假设重复子串的长度最多为L的限制下有解, 则对于任意一个比L小的限制L'<L, 也一定有解. 这就说明存在解的连续性, 这样就可以用二分查找答案长度L.

给出一个关于LCP的定理LCP(SA[i], SA[j]) = RMQ(Height[i+1..j]). 由此, 若存在k, 满足Height[k] < L, 则对于所有i, j 满足i < k < j, 有LCP(SA[i], SA[j]) < L. 即公共长度至少为L的两个后缀, 不会跨过一个小于L的Height低谷k, 所以我们可以得到一些由这些低谷划分开的连续的**段**.




在某段内，若存在 i，j 满足 SA[i]+L<SA[j]，则存在一个长度至少为 L 的 2 个相同不交迭子串．实现时只要记录在每段内，最大和最小的 SA 值即可．

**[Reference]**

[1]许智磊，<<后缀数组>>，IOI2004 国家集训队论文

# Problem H (POJ 1744): Elevator Stopping Plan

[Description]

There is only one elevator in the building. The elevator costs 4s to go up a floor. If the elevator stops on some floor except the first and the last floor, it takes 10s. It takes person 20s to go up or down a floor. Given Fi (1<=i<=n) that is the number of the floor the i-th person needs to reach. Make a plan for reducing the last time that all persons get the floors they want. (n<=30000).

有一栋楼, 里面只有一架电梯. 电梯上一层需 4s,电梯在一层停下需等 10s(1 层和最后一层的等待不计时),人走上或走下一层需 20s.给出 Fi(1<=i<=n)层有人要到达, 求一个安排计划,使最后到达自己目的地的人的用时最短. (n<=30000)

[Solution]

Use Binary Search to search for the answer L. And try to stop the elevator as high as possible greedily each time. It’s like to set the service stations.

[Thanks]

Linxi Xie(198808xc), from Fuzhou No.1 Middle School

Discussion in Fudan University after this contest