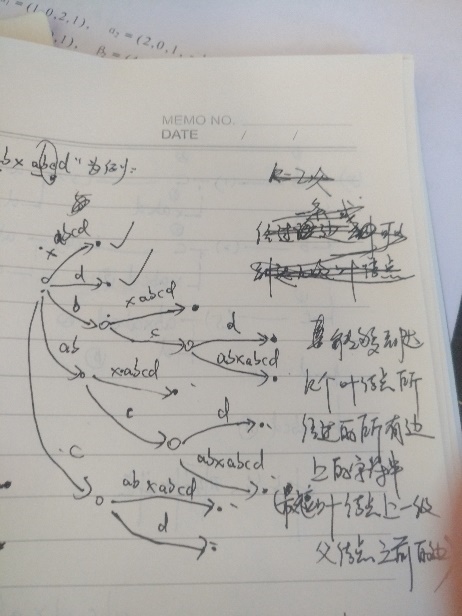
**一、设计求解一下问题算法并给出其时间复杂度**

**1：**

解：首先建立关于字符串S的后缀树T，该过程时间复杂度为O(n)，然后对树中的每个内部节点进行遍历，找到下面叶子节点个数为k的所有内部节点，并记录其对应的子串个数。由于有n个叶子节点的树最多有(n-1)个内部节点，所以该步复杂度为O(n2)，在查找叶子节点个数为k的内部节点时，设一个临时变量，存储最短子串的长度。总时间复杂度为O(n2)。

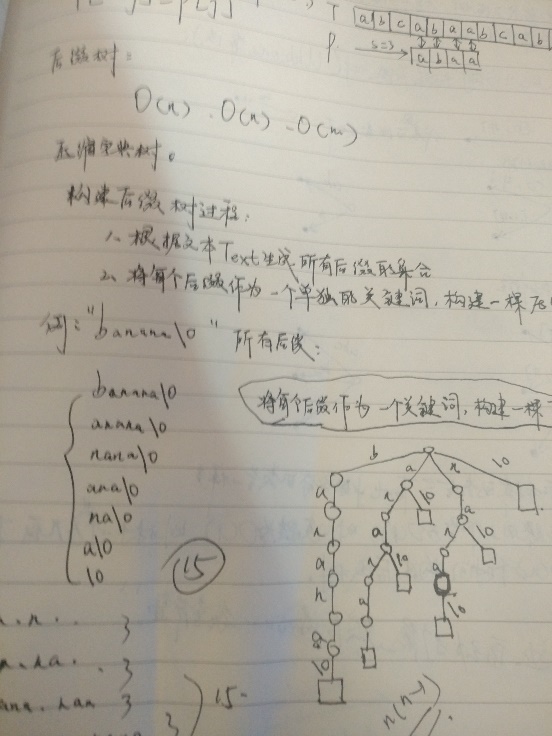
以含有重复字符的字符串“abcabxabcd”为例：



**2：**

解：根据字符串S的内容生成所有后缀的集合。将每个后缀作为一个关键词，构建一颗字典树，时间复杂度为O(n)，然后遍历所有的节点，包括叶节点和非叶节点，时间复杂的为O(n2)，设置一个变量，来存储所有节点的个数，即为子串的个数。总时间复杂度为0(n2)。

以“banana”为例：



图中添加了“\0”，我们可以在这里先忽略不计。

**3：**

解：假定以a[j]结尾的数组序列的最长递增子序列长度为L(j)，则L(j)={ max(L(i))+1, i<j且a[i]<a[j] }。也就是说，我们需要遍历在j之前的所有位置i(从0到j-1)，找出满足条件a[i]<a[j]的L(i)，求出max(L(i))+1即为L(j)的值。最后，我们遍历所有的L(j)（从0到N-1），找出最大值即为最大递增子序列。时间复杂度为O(n^2)。

描述：for(i=1, L[1]=1; i < n; i++)

for(j=0, m=0; j < i; j++)

{

if (a[j]<=a[i]&&m<L[j])

L[i] =max{ L[j]+1，L[i] }

}

**4:**

解：

1. 将可截取选择方案集设为U，其中的子项目为u，令截取的长度为s(u)，则1<=s(u)<=n，而项目u的售价（value）为p(u)，对应于s(u)。将集合U中的项目按照p(u)/s(u)的值从大到小排列，假设结果为u1,u2,…,um，满足p(u1)/s(u1)>=p(u2)/s(u2)>=…>=p(un)/s(un)。

令可能的截取集W为空集，总的售价为P=0；时间复杂度为O(m)。

For i=1 to m do

If s(ui)<=n then

W=W U {ui}，P=P+p{ui}，n=n-s(ui)

Endif

Endfor

For i=1 to m do

If p(ui)>P then

W={ui}, P=p(ui)

Endif

Endfor

Return W

该算法并不是总能给出最优解。

2)

int extend\_bottom\_up\_cut\_mod(int \*p, int n, int \*s)

{

int i, j, q;

int r[n + 1];

r[0] = 0;

for (j = 1; j <= n; ++j) {

q = -1;

for (i = 1; i <= j; ++i) {

if (q < p[i - 1] + r[j - i]) {

q = p[i - 1] + r[j - i];

s[j] = i;

}

}

r[j] = q;

}

return r[n];

}

static void print\_cut\_mod\_solution(int n, int \*s)

{

while (n) {

printf("%d ", s[n]);

n = n - s[n];

}

}

第一个循环计算的是来选择长度，长度为j；第二个循环是计算长度为j时的最优解。过程是这样的：当j为1时，也就是说此时长度为1，这时最优解q=max（-1，p[0]+r[0])，此时q=p[0],即长度为1英寸的钢条的售价；当j为2时，此时长度为2，这时最优解就要在i=1和i=2之间进行选择，而i=1时的值在第一次循环中已计算出来保存在r数组中，i=2时等于是不切割。所以每次循环计算到j时，依赖的j-1对应的最优解已经计算出来，不需要再重复计算，还保存最优解对应的第一段钢条的切割长度sj

时间复杂度为O(n2)。

**二、证明以下问题的NP完全性**

**1：**

**证：**

限制成为X3C 问题，令|C|=3K，并且对于任意c∈C, |c|=3，则问题转化成寻找C中的三元素子集的集合，覆盖C中所有的元素。

**2：**

已知问题：HC -> 目标问题：两点之间Hamilton通路

基本单元：顶点u -> V中两点a, b

**证：**

把hc转化成这个问题。给出一个hc的实例图g=（v，e），任意选取一个顶点∈v，用两个顶点a和b代替g中的u，并将这两个顶点链接到u所链接的顶点。让我们调用结果图g。我们认为原始的图G有一个hc，如果并且仅当得到的图G有一个a和b之间的hp。如果g有一个a和b之间的hp，将两个顶点a和b压缩成一个顶点u给我们一个g中的hc。相反，如果g有一个hc，将u分成两个顶点a和b，将g中的hc从a改为g中的hp。

**3：**

证：我们证明3sat设置分裂。对于由变量集U={u1、u2，…、un}和子句集C={c1,c2,…,cm}组成的任意3SAT实例，我们的相应SetSpliting实例由S和C组成，构造如下。

有限集S={u1,u2,...,un, 1, 2,...,n}∪{a}

子集C’= {{ui, i|ui∈U} ∪ {cj ∪ {a}|cj∈C}

假设任务t : U → {T,F} 满足任何cj ∈ C.我们可以将S拆分为：

S1 = {ui | t(ui)=T} ∪{i | t(ui)=F}

和S 2 = {ui | t(ui)=F} ∪ {i | t(ui)=T} U {a}，我们声称S1和S2分割了C‘的每个子集，它们将{ui,i} 分割，因为t是一个可行的真值分配，并且分割了{cj U {a}}，因为cj U {a} ⊊S1，如果cj U {a} ⊆S2，那么t就不满足cj。

**六、近似算法**

**1.**

**1）**证明：从划分（Partition）问题图灵归约到优化形式的划分问题。

假设S[A,B，k]是用来解决优化形式划分问题的一个子算法（程序），这里参数A={a1, a2, a3, …, an}. B=Σa∈A a, k即为题目条件中的k。

使用实例A={a1, a2, a3, …, an}解决划分问题的相应算法如下：

1．If Σa∈Aa is odd then return “no”

else set b= (Σa∈Aa )/k.

2. using the assumed subroutine S.

call S[A,B,2].

如果对于优化形式的划分问题来说S是多项式时间算法，那么划分（Partition）问题也是多项式时间算法。

因此，优化形式的划分问题是NP-hard问题。

**2）**假设该问题存在近似比小于3/2的近似算法，则近似算法将n个整数划分为k=A(T)/OPT(T)<3/2时每组成员之和相同。即A(T)<3\*OPT(T)/2。

若OPT(T)=2，则A(T)只能等于2，因此在这种情况下划分问题有解。

若OPT(T)>2，则A(T)最少为3，因此在这种情况下划分问题无解。

容易判断，上述判断过程可以在多项式时间内完成。

因此如果优化形式的划分问题存在近似比小于3/2的多项式时间近似算法，则由以上结论及NP问题的定义，划分问题存在多项式时间的算法。即P=NP，与P!=NP矛盾，因此若P!=NP，则优化形式的划分问题不存在近似比小于3/2的近似算法。

**2.**

**证：**该算法是一个贪心算法。

第1步将超出约束B的项从U中清除；

第2步将U中的项目按value/size的比值从大到小排序，设排序结果为u1, u2,…, un，满足v(u1)/s(u1)>=v(u2)/s(u2)>=…>=…v(un)/s(un)；

第3步设所求子集W初始为空集，value之和V为0；

第4步for循环，按照从大到小的顺序将项目存放到W中，直到有一个放不进去，并且value之和V为所有放进去后的v(ui)之和；

第5步for循环，将第4步中获得的value之和V（即贪心思想获取的value和）同第i个项目ui的value值相比，如果后者较大，那么选择项目ui，否则选择项目u1,u2,…,ui-1；

令ui的value值为maxvalue，第4步获取的V为greedy；

近似算法A（W）=max(greedy, maxvalue)，又最优算法OPT(W)<=greedy + maxvalue = 2\*max(greedy, maxvalue)=2\*A(W)，所以OPT(W)/A(W)<=2，所以算法是对以上KNAPSACK问题的一个性能比为2的近似算法。