Problem Set 2

This problem set is due **at 11:59pm** on **Thursday, February 19, 2015.**

This assignment, like later assignments, consists of *exercises* and *problems*. Hand in solutions to the problems only. However, we strongly advise that you work out the exercises also, since they will help you learn the course material. You are responsible for the material they cover.

Please turn in each problem solution separately. Each submitted solution should start with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

We will often ask you to "give an algorithm" to solve a problem. Your write-up should take the form of a short essay. Start by defining the problem you are solving and stating what your results are. Then provide:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. A proof (or proof sketch) for the correctness of the algorithm.
3. An analysis of the running time.

We will give full credit only for correct solutions that are described clearly.

Exercise 2-1. Read CLRS, Sections 30.1 and 30.2.

Exercise 2-2. Exercise 30-2.3.

Exercise 2-3. Exercise 30-2.4.

Exercise 2-4. Read CLRS, Chapter 18.

Exercise 2-5. Exercise 18.2-5

Exercise 2-6. Exercise 18.3-2

insert, use another h^* when node is leaf.

Problem 2-1. Pattern Matching [25 points]

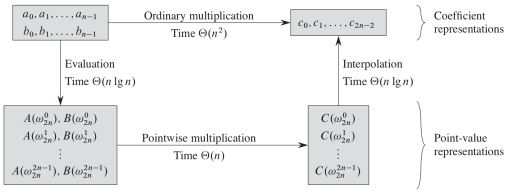
Suppose you are given a ***source string*** $S[0\,..\,n-1]$ of length n, consisting of symbols a and b. Suppose further that you are given a ***pattern string*** $P[0\,..\,m-1]$ of length $m \ll n$, consisting of symbols a, b, and $*$, representing a pattern to be found in string S. The symbol $*$ is a "wild card" symbol, which matches a single symbol, either a or b. The other symbols must match exactly.

The problem is to output a sorted list M of valid "match positions", which are positions j in S such that pattern P matches the substring $S[j\,..\,j+|P|-1]$. For example, if $S = a\,b\,a\,b\,b\,a\,b$ and $P = a\,b\,*$, then the output M should be $[0, 2]$.

2-1. P888 ~ P915

1. C is 2n degree, need expand A, B.

basic idea:



2. choose V_K cleverly.
complex root of unity, $w^n = 1$.
compute $\Rightarrow w$.
$w = e^{2\pi i k/n}$ k for 1, 2, ... n-1.

3. DFT.

4. FFT. (divide and conquer) by even and odd #coeff....

$A(x) = A^{[0]}(x^2) + x \cdot A^{[1]}(x^2)$
↑ even coeff ↑ odd coeff,

$T(n) = 2T(n/2) + \Theta(n)$
$\Rightarrow T(n) = \Theta(n \cdot \lg n)$.

halve them

the key idea.

```
RECURSIVE-FFT(a)
 1  n = a.length                    // n is a power of 2
 2  if n == 1
 3      return a
 4  ω_n = e^{2πi/n}
 5  ω = 1
 6  a^[0] = (a_0, a_2, ..., a_{n-2})
 7  a^[1] = (a_1, a_3, ..., a_{n-1})
 8  y^[0] = RECURSIVE-FFT(a^[0])
 9  y^[1] = RECURSIVE-FFT(a^[1])
10  for k = 0 to n/2 - 1
11      y_k = y_k^[0] + ω y_k^[1]
12      y_{k+(n/2)} = y_k^[0] - ω y_k^[1]
13      ω = ω ω_n
14  return y                        // y is assumed to be a column vector
```

5. come back to coeffecien form. $\Rightarrow$ $a \otimes b = \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(a) \cdot \mathrm{DFT}_{2n}(b))$,

---

2-4. B-Tree.

Why B-tree? lower disk accesss.

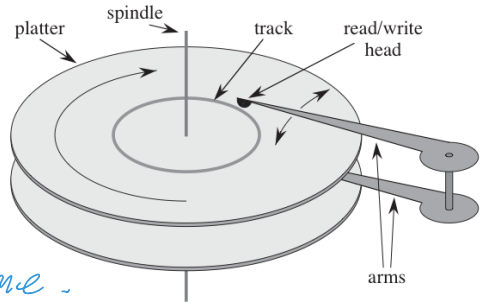
1. access time of disk is about 8~11 ms.

2. amortize the timeing of mechanical movement.

3. time ~ page access time.

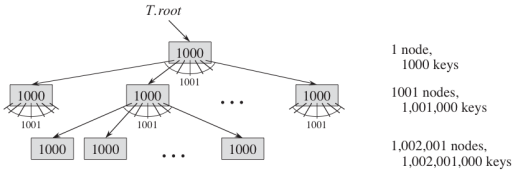
4. limited memory, can hold a few page in menery.

5. usually B-tree node is as large as Page size



Def.

* B^+-tree (store all satellite information in the leaves).)



Prop.:

- n keys in node.
- $key_1 \le key_2 \le \cdots \le key_n$.
- mark wether node is a leaf.
- internal node has n+1 pointers to children
- key separate the range of sub-tree.
- all leaves has the same height h.
- $t-1 \le \#key \le 2t-1$. t for minimum degree.
  ↑ full.

$$h \le \log_t \frac{n+1}{2}$$ ← can big.

root in memory.

search.

can use → binary search

$O(th) = O(t \cdot \log_t n)$

```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and k > x.key_i
3      i = i + 1
4  if i ≤ x.n and k == x.key_i
5      return (x, i)
6  elseif x.leaf
7      return NIL
8  else DISK-READ(x.c_i)
9      return B-TREE-SEARCH(x.c_i, k)
```

2. split the child.
promote a child "s"
cut large (T-1) part
of y to z.

happen before insert.

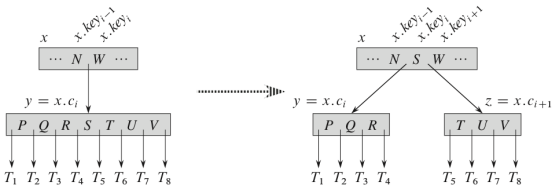


Figure 18.5 Splitting a node with $t = 4$. Node $y = x.c_i$ splits into two nodes, y and z, and the median key S of y moves up into y's parent.

```
B-TREE-SPLIT-CHILD(x, i)
 1  z = ALLOCATE-NODE()
 2  y = x.c_i
 3  z.leaf = y.leaf
 4  z.n = t − 1
 5  for j = 1 to t − 1
 6      z.key_j = y.key_{j+t}
 7  if not y.leaf
 8      for j = 1 to t
 9          z.c_j = y.c_{j+t}
10  y.n = t − 1
11  for j = x.n + 1 downto i + 1
12      x.c_{j+1} = x.c_j
13  x.c_{i+1} = z
14  for j = x.n downto i
15      x.key_{j+1} = x.key_j
16  x.key_i = y.key_t
17  x.n = x.n + 1
18  DISK-WRITE(y)
19  DISK-WRITE(z)
20  DISK-WRITE(x)
```

3. insert.

ensure
no node
is full.

```
B-TREE-INSERT(T, k)
 1  r = T.root
 2  if r.n == 2t − 1
 3      s = ALLOCATE-NODE()
 4      T.root = s
 5      s.leaf = FALSE
 6      s.n = 0
 7      s.c_1 = r
 8      B-TREE-SPLIT-CHILD(s, 1)
 9      B-TREE-INSERT-NONFULL(s, k)
10  else B-TREE-INSERT-NONFULL(r, k)

B-TREE-INSERT-NONFULL(x, k)
 1  i = x.n
 2  if x.leaf
 3      while i ≥ 1 and k < x.key_i
 4          x.key_{i+1} = x.key_i
 5          i = i − 1
 6      x.key_{i+1} = k
 7      x.n = x.n + 1
 8      DISK-WRITE(x)
 9  else while i ≥ 1 and k < x.key_i
10          i = i − 1
11      i = i + 1
12      DISK-READ(x.c_i)
13      if x.c_i.n == 2t − 1
14          B-TREE-SPLIT-CHILD(x, i)
15          if k > x.key_i
16              i = i + 1
17      B-TREE-INSERT-NONFULL(x.c_i, k)
```

(a) [4 points] Describe a straightforward, naïve algorithm to solve the problem. Your algorithm should run in time $O(nm)$.

(b) [12 points] Give an algorithm to solve the problem by reducing it to the problem of polynomial multiplication. Specifically, describe how to convert strings S and P into polynomials such that the product of the polynomials allows you to determine the answer M. Give examples to illustrate your polynomial representation of the inputs and your way of determining outputs from the product, based on the example S and P strings given above.

(c) [3 points] Suppose you combine your solution to Part (b) with an FFT algorithm for polynomial multiplication, as presented in Lecture 3. What is the time complexity of the resulting solution to the string matching problem?

(d) [6 points] Now consider the same problem but with a larger symbol alphabet. Specifically, suppose you are given a representation of a DNA strand as a string $D[0\,..\,n-1]$ of length n, consisting of symbols A, C, G, and T; and you are given a pattern string $P[0\,..\,m-1]$ of length $m \ll n$, consisting of symbols A, C, G, T, and $*$.

The problem is, again, to output a sorted list M of valid "match positions", which are positions j in D such that pattern P matches the substring $D[j\,..\,j+|P|-1]$. For example, if $D = A\,C\,G\,A\,C\,C\,A\,T$ and $P = A\,C * A$, then the output M should be $[0, 3]$.

Based on your solutions to Parts (b) and (c), give an efficient algorithm for this setting. Illustrate your algorithm on the example above.

Problem 2-2. Combining B-trees [25 points]

Consider a new B-tree operation COMBINE(T_1, T_2, k). This operation takes as input two B-trees T_1 and T_2 with the same minimum degree parameter t, plus a new key k that does not appear in either T_1 or T_2. We assume that all the keys in T_1 are strictly smaller than k and all the keys in T_2 are strictly larger than k. The COMBINE operation produces a new B-tree T, with the same minimum degree t, whose keys are those in T_1, those in T_2, plus k. In the process, it destroys the original trees T_1 and T_2.

In this problem, you will design an algorithm to implement the COMBINE operation. Your algorithm should run in time $O(|h_1 - h_2| + 1)$, where h_1 and h_2 are the heights of trees T_1 and T_2 respectively. In analyzing the costs, you should regard t as a constant.

(a) [5 points] First consider the special case of the problem in which h_1 is assumed to be equal to h_2. Give an algorithm to combine the trees that runs in constant time.

(b) [5 points] Consider another special case, in which h_1 is assumed to be exactly equal to $h_2 + 1$. Give a constant-time algorithm to combine the trees.

(c) [5 points] Now consider the more general case in which h_1 and h_2 are arbitrary. Because the algorithm must work in such a small amount of time, and must work for arbitrary heights, a first step is to develop a new kind of ***augmented B-tree*** data

2-1. a). for i in len(s):

check S[i, i+p-1] match.

if (match, add to list

return list.

b). represent a by 1, b by -1, * by 0.

$$f_c(x) = 1 - x + x^2 - x^3 - x^4 + x^5 - x^6$$

$$f_p(x) = -x + x^2.$$

$P[i] = 0$ or $S[j+i] \cdot P[i] = 1 \Rightarrow$

$$\sum_{j=0}^{m-1} S[j+i] \cdot P[i] = k.$$

c). $n \cdot \lg n$.

D). Encode A as aa, C as a b, G as ba

T as bb, and * as **.

⇒ use sultion 2, find M.

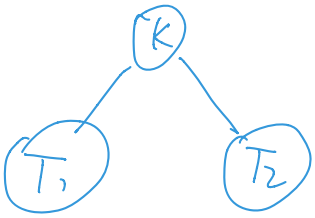
only use even number of M ⇒ M'

⇒ $O(2n + 2n \lg 2n) = O(n \lg n)$

structure in which each node x always carries information about the height of the subtree below x. Describe how to augment the common B-tree insertion and deletion operations to maintain this information, while still maintaining the asymptotic time complexity of all operations.

(d) [10 points] Now give an algorithm for combining two B-trees T_1 and T_2, in the general case where h_1 and h_2 are arbitrary. Your algorithm should run in time $O(|h_1 - h_2| + 1)$.

a). $h_1 = h_2$.



if T_1.root $< n-1$

if T_2.root + T_1.root $< 2n-2$

merge T_1.root, k, T_2.root

else. borrow from T_2.

if T_2.root $< n-1$.

answer is merge first, if R.n $\geq 2t-1$, then split Node.

b). $h_1 = h_2 + 1$.

if T_2.root.n $< n-1$

⋮

if T.root $\geq 2n-2$.

spl ;

if > 2n-1

split and

promote median,

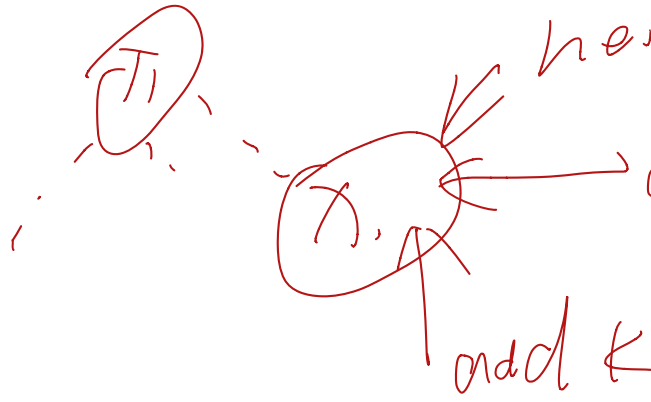
then split root, if needed.

c). split add "1" height for promote Node.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015

D. $|h_1 - h_2| \subset 2$. use (a,) or (b).

if $h_1 > h_2 + 1$. (symmetrically).



$(t+1 \sim 4t-1)$.

split a c b. introduced.