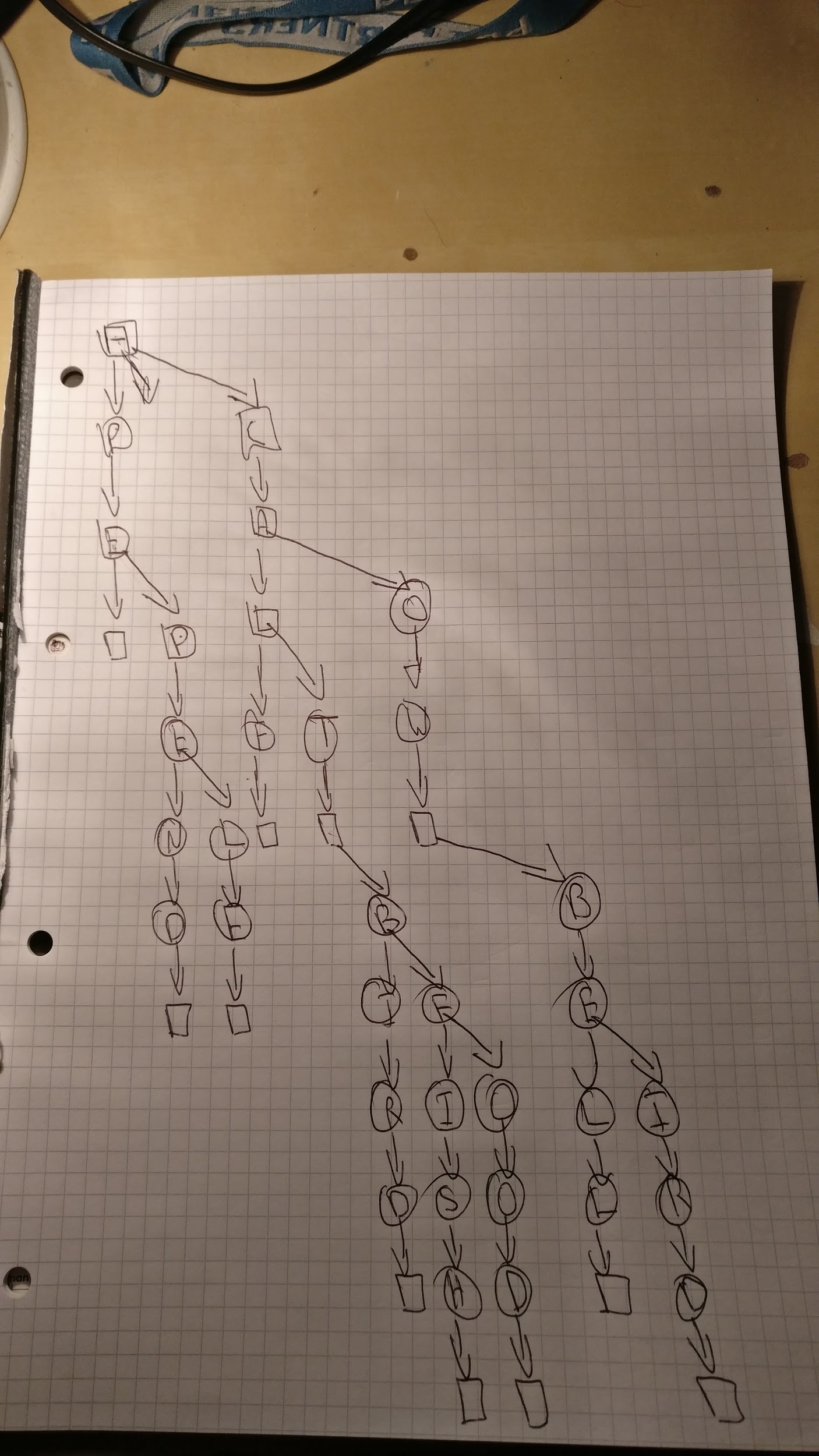
Note: this is a merged SEA and SED paper, but I will only answer the algorithms questions here (questions 3 and 4)

3)

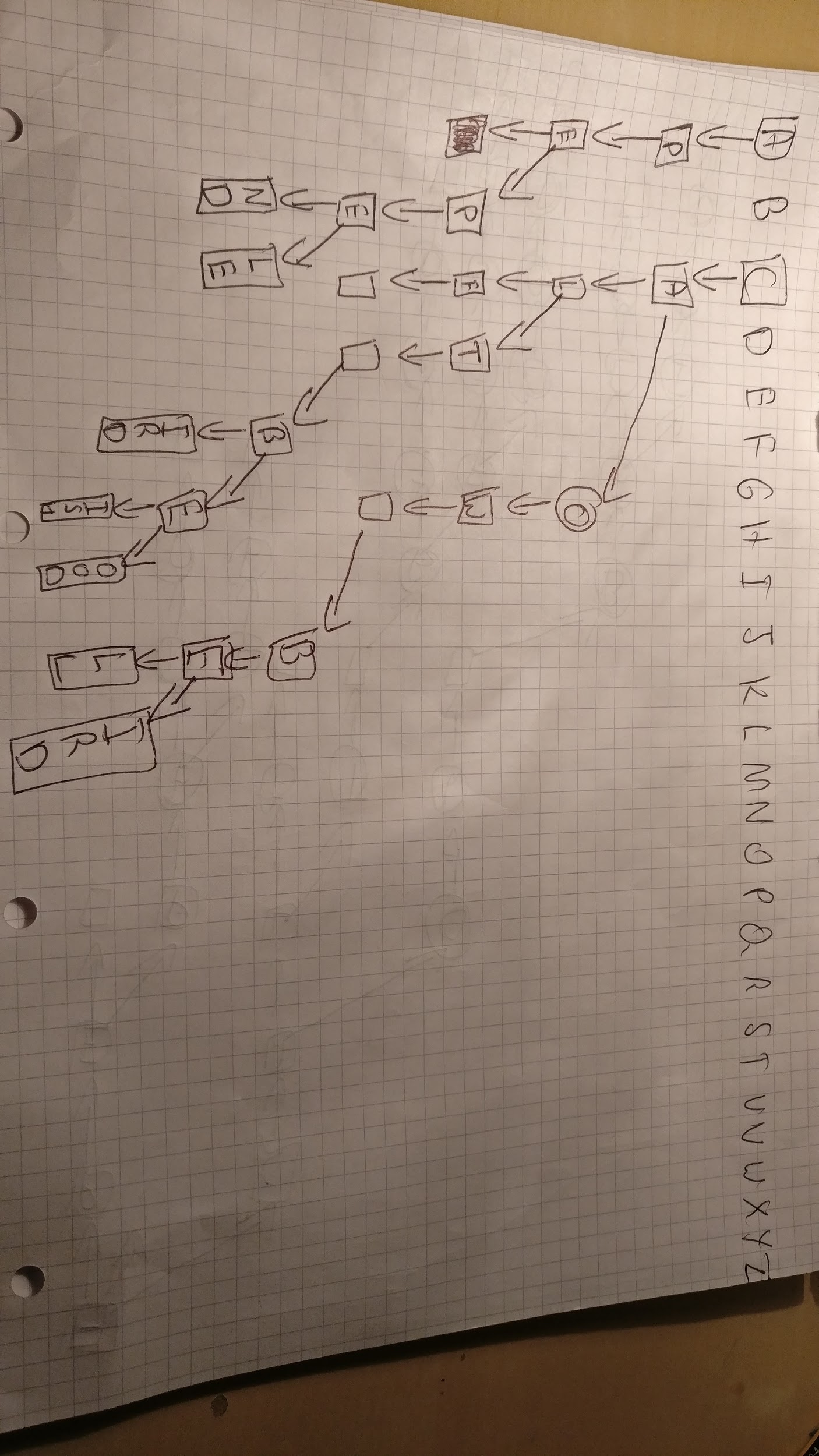
a) This question is **stupid**

(edit: iterate over the first list, adding all of its elements to a hash set. This takes O(n) time. then iterate over the second, checking the hash set (O(1)) for each item. If the item is not in the hash set, add it to the difference skip list (O(1)). This also takes O(n) time)

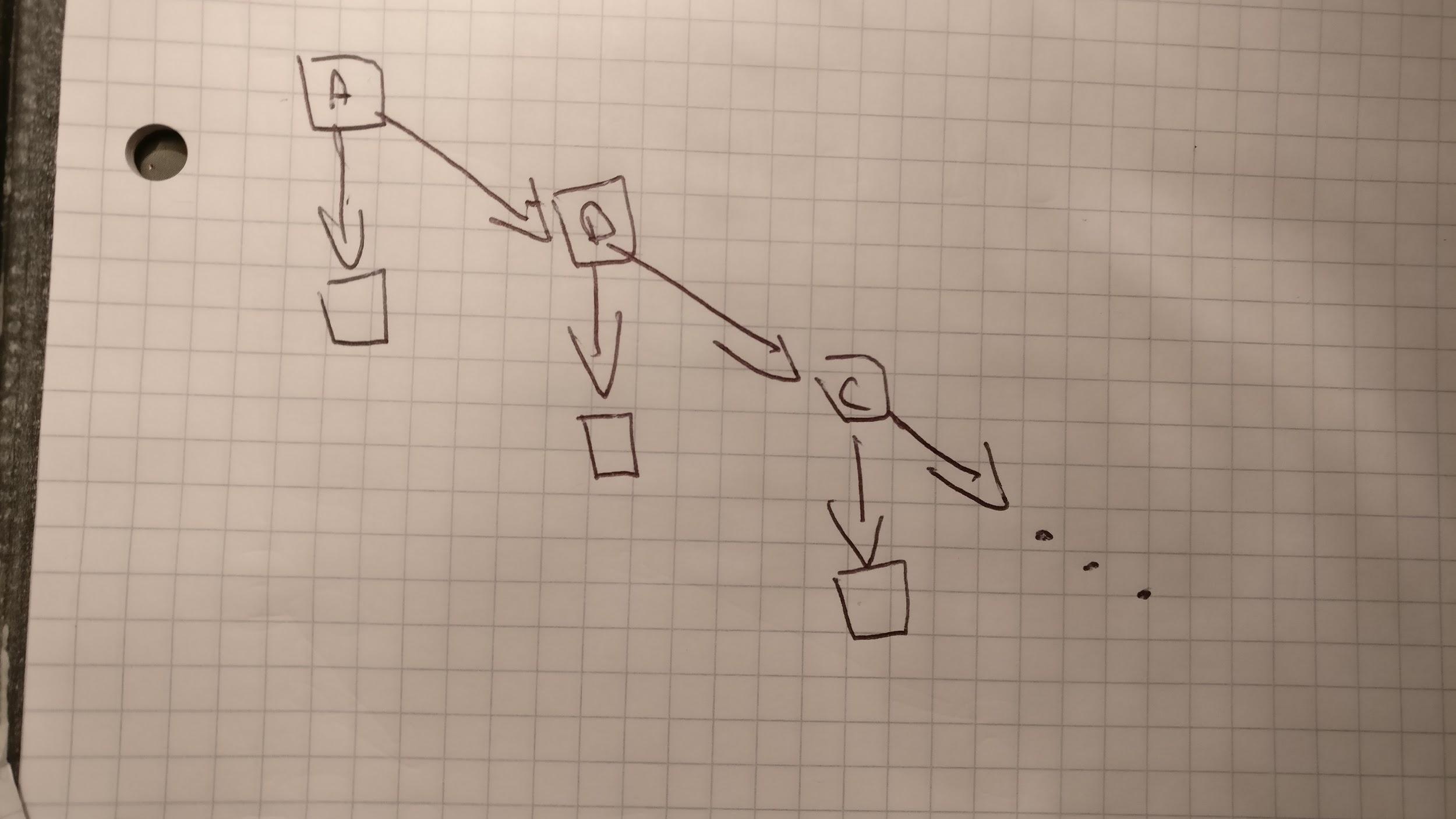
b)

i)

ii)



iii) The worst-case complexity of search in a basic TST is O(), i.e. the number of strings in the alphabet. This would occur when every single string is different, such that no strings share the same prefix. Take an alphabet with strings, “a”, “b”, “c”... etc. Then the TST will look like this:



This is a perfectly unbalanced TST and thus the search operation can be thought of as performing a linear search on a linked list, i.e. in the worst case we have to inspect every node.

4)

a)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Iteration | 1 | 2 | 3 | 4 | 5 |  |  |
| Edge Included | AB | BC | CG | DG | GH | GF | AE |
| Total Cost | 1 | 3 | 5 | 6 | 7 | 8 | 12 |

b)

gcd(a, b):

if a == b:

return a

if a == 1 || b == 1:

return 1

If b mod 2 == 0:

if a mod 2 == 0:

return 2 \* gcd(a / 2, b / 2) # a and b are even

return gcd(a, b / 2) # a is odd, b is even

if a mod 2 == 0:

return gcd(a/2, b) # a is even, b is odd, switch order

return gcd((a - b) / 2, b) # a and b are odd

c)

k-heads(n, k, p):

let heads[1..n, 1..k] be a new array

heads[0, 0] = 1

For i = 1 to n:

heads[i, 0] = heads[i-1, 0] \* (1-p[i])

for j = 1 to k:

# Could initialise to heads[1..n] to avoid this

heads[0, j] = 0

for m = 1 to n:

for q = 1 to k:

heads[m, q] = p[m] \* heads[m - 1, q - 1] + (1 - p[m]) \* heads[m - 1, q]

return heads[n, k]

Two nested for-loops from 1 to n and then 1 to k mean a running time of O(nk) (assuming addition and multiplication is constant)

So total running time = O(nk)

<https://math.stackexchange.com/a/1802543>

d)

k\_heads(n, k, p):

If n == 0 && k != 0:

return 0

if k == 0:

x = 1

for i = 1 to n:

x = (1- p[i]) \* x

return x

if n mod 2 != 0:

return k\_heads(n - 1, k, p) \* k\_heads(1, 0, p)

if k mod 2 != 0:

return k\_heads(n - 1, k - 1, p) \* k\_heads(1, 1, p)

x = k\_heads(n / 2, k / 2, p)

return x \* x

def k\_heads(u,l,k,P):

if u < l or l < 0 :

return 0

if l == u :

return P[l]

else:

return sum(

map(range(0,u-1/2),

\i -> k\_heads(u,u-l/2,i,p) \*

k\_heads(u-l/2,l,k-i,p)))

Assume that n and k are even. The running time T(n) of k\_heads is therefore

T(n, k) = T(n / 2, k / 2)