Q2.

G(V,E): undirected, connected, n vertices, m edge, $w \geq 0$

Given $s, t \in V$; find a path $s \to t$, s.t. max edge weight is minimized. Time: $O(m \cdot \log m)$

Soln: Dijkstra, 用 min heap , see next page

~~1~~. Maintain a table (vertices, max_w, parent) and initialize the max_w of all vertices as positive infinite.

2. update max_w of u, v:

for edge {u,v}. $\underline{O(1)}$.

~~Suppose~~

if max_w(u) < max{weight(u,v), max_w(v)},

do nothing.

☆ else:

max_w(u) = max{weight(u,v), max_w(v)}.

parent(u) = v.

repeat ☆ for max_w(v).

3. Start from ~~edo~~ vertex t. Set max_w(t)=0.

~~4. update max_w for all the neighbors of t and push the nodes neigbors into a set S, the edges into~~

~~5. pop(S) and repeat 4~~

4. ~~Store all the edges as {~~ process the edges that contains t and not visited, push the neigbor of t into a stack S, mark the edge as visited.

5. pop(S) as t and repeat 4 till S is empty

6. Start from s, track down the parents till the parent is t, the path (s, v_1, v_2, …, t) is the result. $T = O(m) = O(m\log m)$

1. maintain a table ...

2. $H \Leftarrow$ empty min_heap., H.push(0, S)

3. for each neighbor u of S.

    parent(u) $\Leftarrow$ S

    H.push(w(u, S), u)

4. while H is not empty:

    w, u $\Leftarrow$ H.pop()

    if is_visited(u):

        continue

    for all adjacent v of u:

        if dist[v] > dist[u] + weight(u, v)

            dist[v] = $\Leftarrow$

            $H \Leftarrow$ (v, dist[v])

Shortest path:

track down the parents

from S.

<2016>

Q1. (a) Search the K-th ~~height~~ minimum value in a BST. time?

height h.

node number n.

~~Step 1~~:

```
Def: Inorder (N, A, i):
    if N == NULL
        return i.
    i = Inorder (N.left, A, i)
    A[i] = N.value
    return Inorder(N.right, A, i+1)
A = empty Array of size n.
Inorder(N, A, 0).
return A[K]
```

$T = O(N)$

~~Q2~~ (b) how to modify the tree structure s.t. you can find K-th min in $O(h)$ time.

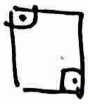
Structure:

```
TreeNode {
    int value;
    int size_of_left_subtree;
    int size_of_right_subtree;
    TreeNode *left, *right; }
```

Algorithm:

```
Def Search (K, N):
    if K == N.size_of_left_subtree +1 :
        return N.value
    elif K < ... :
        Search (K, N.left)
    else:
        Search(K-( ), N.right)
```

<2016>

Q2. Given an $m \times n$ matrix, entries ≥ 0.
find a path from the  top-left to bottom right.
minimize Σ(numbers along the path.)

Soln:

- Store the matrix (M) as a set of edges contains:
  ~~$(M[u,v], M[u,v+1])$ and the weight is M~~

$$\begin{cases} 1. \{(u,v),(u,v+1)\} \text{ and the weight is } M[u,v+1], \forall u \in [0,m], \forall v \in [0,n-1] \\ 2. \{(u,v),(u+1,v)\} \text{ with weight } M[u+1,v], \forall u \in [0,m-1], \forall v \in [0,n] \end{cases}$$

- Maintain a table of [Element, dist, parent]
  element is represent by (u,v). such as $(1,3)$, $(0,7)$. etc.

Def: process ~~the~~ an edge means, for edge $\{\underset{A}{\underline{(u,v)}}, \underset{B}{\underline{(u,v+1)}}\}$.

$$\begin{cases} \text{if } dist(B) < dist(A) + weight(A,B). \\ \quad \text{do nothing} \\ \text{else:} \\ \quad dist(B) = dist(A) + weight(A,B). \\ \quad parent(B) = A \end{cases}$$

- Start from $(0,0)$. set ~~the~~ $dist(0,0) = 0$ and all other dist as ∞.
  push the node N into stack S:
  1. push its descestors into S (nodes with edges points out from the node N)
  2. process the edges pointing out from N
  3. pop(S) and repeat step 1.2, till S is empty.

- Start from (m,n), track down the parent till reach $(0,0)$, we can get the path as result.

<2017>

(a) Given S with n lower-case letters (acaddgeg)

cut S s.t. every substring is a palindrome (aca, dd, geg) 回文.

return the min # of cuts needed.

Time? space?

set i=0, j=n-1, count=0, cut=j, flag=False.

0 1 2 3 4 5 6 7 8
a p e e p e p c e

i=0. # a\e cut=n-2.
a\c, cut=n-3.
a\p, cut=0 e=e→e ≠ p 打破while flag ≠ True.
p=p cut=6.
p p

```
While (i< n-1):
    cut=n-1
    while (i<j):
        I=i  J=j
        ~~cut=j~~
        while A[I]==A[J]:   i=1
            i+=1,   I+=1, J+=-1,
            j+=-1,  if I>=J:
                       flag=True.
                       break.
            if i>=j:
               ~~i=cut~~
               break
        if flag:
            break.
        else:
            J+=-1
            cut=J
    i=cut+1
    count+=1.
return count
```

i=0, j=n-1, count=0, cut=j, flag=False.

```
while (i<n-1): 每次从最后一个开始对比.
    cut=n-1  # 记录cut的位置.
    while (i<j):
        a=i, b=j
        While (A[a]==A[b]):
            a+=1, b+=-1,
            if a>=b:
                Flag=True. #有palindrome
                break
        if flag:
            flag=False
            break
        else:
            j+=-1  # search 下一个.
            cut=j
    i=cut+1
    count+=1
return count
```

Time:

log(n)+...+log(2)
log(n-1)+...+log(2)
log(n-2)+...+log(2)

$= n\log(2)+(n-1)\log(3)+\dots+\log(n) = ?\ O(n^2+\log n)$

$O(n^3)$?

(b). Same string S as in (a). find the len of longest alphabetically increasing substring (可以不连续) e.g. for "acabkdgeg". it is "acdge".

encode s into an array alphabetically.

e.g. S = "abc" # array A = [1, 2, 3].

let ~~i=0~~ i=0, max = ~~0~~ [ ] size of A. (n), = zeros. len = [ ] size of A. (n). = zeros. , j = 0, max_len = 0

while i < n-1:
    j = i
    while j < n-1:
        if A[j] > ~~A[i]~~ max[i]:
            max[i] = A[j]
            len[i] += 1.
        j += 1
    ~~i += 1~~
    if len[i] > max_len:
        max_len = len[i]
    i += 1
return max_len.

$T = O(n^2)$

<2018S>.

Q1

(a). Given a binary tree. Given ~~pre~~ two visiting seqs. only (without tree), can you reconstruct the tree uniquely? how?

(1) Pre-order + In-order

(2) Post-order + In-order

(3) Pre + Post.

1) yes.

last element in Pre-order is the root.

find root in In-order seq, left - Root - Right.

then. seperate both the seqs into three ~~co com~~ subseqs

and do the seperation recursively.

(2) yes

similar to above.

(3) yes.

~~left-r right-root~~

~~① left - right - root~~

~~root-right~~

~~② root - l - left - right~~

~~1. find root~~

~~2. find root of left subtree ?~~ } recursively

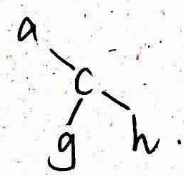
~~3. find root of right subtree~~

find in ②, seperate ①, then seperate

(3) No. e.g.

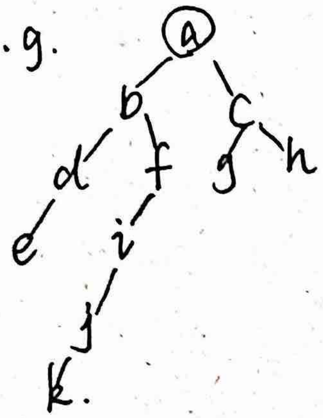
Pre-order : ghca

Post-order : acgh



(b) design an algo to find the longest distance in a binary tree with $T=O(n)$

e.g.



longest: $k \to h$: 7.

Soln:

Step 1: BFS from Node N: (Root) $O(n-1)$: # of edges.

~~push Node into Queue Q~~

(1) push the neighbors of N into Queue Q.

(2). ~~pop~~ dequeue(Q) as N and push N to ~~Array~~ stack A,
repeat (1).

Step 2: BFS from Node N1. $O(n-1)$

N1 ~~is the last element in A~~. = pop(A)

repeat step 1.(1), 1.(2). (push N_1 to stack S)

length S is the result.

$\therefore T=O(n)$.

<2019>

Q1. ~~(a)~~ verify a BST. Time?

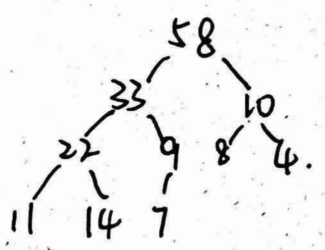
```
int isBST(struct TreeNode *node){

    if node->left != NULL:
        isBST(node->left)
        if node->left.value > node.value
            return False.
    if node->right != NULL:
        isBST(node->right)
        if node->right.value < node.value
            return False.
    return True.
}
```

O(N).

中序遍历+判断递增

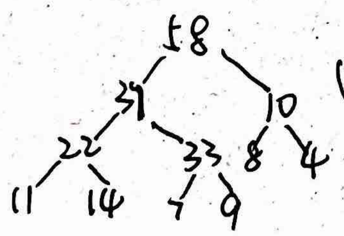
```
def In_order( , , ){        Array A
    if (N==NULL)
        return A;
    In_order(N->left)
    A.push(N->value)
    In_order(N->right);

def isBST(N)
    A = array();
    A = Inorder(N,A);
    for i in range(len(A)):
        if A[i] ≤ A[i+1]:
            i++
        else
            return False
    return True.
```

(b). Given max-heap T_1:



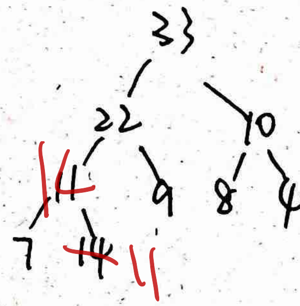
(i) insert 37, what will T_1 be?

Time? Given # of nodes = n.



O(log(n)).

(ii) delete root of T_1



(c) Given hash table T of size 7 and a hash func.

$h(x) = x \% 7$, insert 10, 2, 12, 19, 9, 47 into T. show T.

2	$2 \to 9$
3	10
5	$12 \to 19 \to 47$

Show all the locations examined in order when search for 16 and 47 seperately in T.

16:

$2 \to 9$

47:

$12 \to 19 \to 47$.