snippet

```
import sys
# import heapq, collections

sys.setrecursionlimit(10**7)
inf = 10**20
eps = 1.0 / 10**15
mod = 10**9+7

def LI(): return [int(x) for x in sys.stdin.readline().split()]
def LI_(): return [int(x)-1 for x in sys.stdin.readline().split()]
def LF(): return [float(x) for x in sys.stdin.readline().split()]
def LS(): return sys.stdin.readline().split()
def I(): return int(sys.stdin.readline())
def F(): return float(sys.stdin.readline())
def S(): return input()

def solve():

    return -1

def main():

    print(solve())

main()
```

소수 알고리즘

```
import math

def getPrime(number): #number가 소수이면 출력하는 함수
    if number == 2: #2는 소수이므로 바로 출력
        print number,
    for x in range(2,number): #2부터 number-1까지 나누어 본다
        if number % x == 0: #나누어 떨어지면 소수가 아니므로 break
            break
        elif (x > math.sqrt(number)): #sqrt(number)까지 나누어지지 않으면 소수
            print number,
            break
```

```
import math

def getPrime(number):
    primeList = [2] #소수 리스트 생성
    num = 3 #3부터 소수인지 검사한다

    while (num <= number):
        for x in primeList:
            if(num % x == 0): #소수로 나누었을 때 나누어지면 소수아님
                break
            elif(x == primeList[-1]): #소수리스트의 끝원소로도 나누어지지 않으면 소수
                primeList.append(num) #구한 소수를 소수리스트에 추가
                break
        num += 1

    if(number in primeList): #number가 소수리스트에 존재하면 소수인 것이므로
        print number
```

GCD / LCM

```
def gcd(a, b):
    mod = a%b
    while mod > 0:
        a = b
        b = mod
        mod = a%b
    return b
```

```
def lcm(a, b):
    return a*b//gcd(a,b)
```

Merge Sort

```python
# 시간복잡도는 O(n*log n)
def merge_sorted(arr):
    if len(arr)>1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]

        l = merge_sorted(left)
        r = merge_sorted(right)
        return merge(l, r)
    else:
        return arr

def merge(left, right):
    i = 0
    j = 0
    arr = []

    while (i<len(left)) & (j<len(right)):
        if left[i] < right[j]:
            arr.append(left[i])
            i+=1
        else:
            arr.append(right[j])
            j+=1
    #ㅇㅇㅇ
    while (i<len(left)):
        arr.append(left[i])
        i+=1
    #
    while (j<len(right)):
        arr.append(right[j])
        j+=1

    return arr


arr = [3, 5, 1, 2, 9, 6, 4, 5, 7]
print(merge_sorted(arr))
```

Quic Sort

```python
# 평균 시간복잡도는 O(n*log n), 최악의 시간복잡도는 O(n^2)
def quick_sorted(arr):
    if len(arr) > 1:
        pivot = arr[len(arr)-1]
        left, mid, right = [], [], []
        for i in range(len(arr)-1):
            if arr[i] < pivot:
                left.append(arr[i])
            elif arr[i] > pivot:
                right.append(arr[i])
            else:
                mid.append(arr[i])
        mid.append(pivot)
        return quick_sorted(left) + mid + quick_sorted(right)
    else:
        return arr

arr = [3, 5, 1, 2, 9, 6, 4, 7, 5]
print(quick_sorted(arr))
```

계수정렬

```python
# O(n)
def counting_sorted(arr, K):
    c = [0] * K
    sorted_arr = [0] * len(arr)

    for i in arr:
        c[i] += 1

    for i in range(1,K):
        c[i] += c[i-1]

    for i in range(len(arr)):
        sorted_arr[c[arr[i]]-1] = arr[i]
        c[arr[i]] -= 1

    return sorted_arr

arr = [3, 5, 1, 2, 9, 6, 4, 7, 5]
print(counting_sorted(arr, 20))
```

Binary Search

```
def binary_search(arr, value):
    low = 0
    high = len(arr)-1
    while (low <= high):
        mid = (low+high)//2

        if arr[mid] > value:
            high = mid - 1
        elif arr[mid] < value:
            low = mid + 1
        else:
            return mid

    return -1



arr = [1, 5, 7, 10, 25, 32, 79, 80, 125]

print(binary_search(arr, 7))
print(binary_search(arr, 8))
```

Infix to prefix

```
#infix = [ '2', '*', '(', '5', '+', '7', ')', '+', '8']
infix = [ '2', '+', '3', '*', '4']
postfix = []
stack = []
operator = ['*', '/', '+', '-']
bracket = ['(', ')']

def is_number(x):
    if x not in operator and x not in bracket:
        return True
    else:
        return False

def pref(x):
    if x is '*' or x is '/':
        return 1
    elif x is '+' or x is '-':
        return 0

for c in infix:
    if is_number(c):
        postfix.append(c)
    elif c in operator:
        p = pref(c)
        while len(stack) > 0:
            top = stack[-1]
            if pref(top) <= p:
                break
            postfix.append(stack.pop())
        stack.append(c)

    elif c == '(':
        stack.append(c)
    elif c == ')':
        while True:
            x = stack.pop()
            if x == '(':
                break
            postfix.append(x)

while len(stack) > 0:
    postfix.append(stack.pop())
```

power O(logN)

```
def power(a, n):
    ret = 1
    while n > 0:
        if n % 2 != 0:
            ret *= a
        a *= a
        n //= 2

    return ret

print(power(5, 5))
print(power(5, 21))
```

최장 공통 부분 수열(Longest Common Subsequence)

```
A = "ACAYKP"
B = "CAPCAK"

lcs = [[0 for i in range(len(A)+1)] for j in range(len(B)+1)]

for i in range(1, len(A)+1):
    for j in range(1, len(B)+1):
        if A[i-1] == B[j-1]:
            lcs[i][j] = lcs[i-1][j-1] + 1
        else:
            lcs[i][j] = max(lcs[i][j-1], lcs[i-1][j])

print(lcs[len(A)][len(B)])
```

가장 긴 증가하는 부분 수열

```
def find_lower(sequence, e):
    for i, item in enumerate(sequence):
        if item >= e:
            return i
    return len(sequence)

def solve(n, sequence):
    LSI = []
    P = []
    for i, item in enumerate(sequence):
        lower_bound = find_lower(LSI, item)
        if len(LSI) == lower_bound:
            LSI.append(item)
        else:
            LSI[lower_bound] = item
        P.append(lower_bound)
    size = len(LSI)
    ans = []
    for i in range(n-1, -1, -1):
        if size < 0:
            break
        if P[i] == size-1:
            ans.insert(0, sequence[i])
            size -= 1
    return len(ans)
```

최소 신장 트리 (Minimum Spanning Tree)

```
parent = {}
rank = {}

# 정점을 독립적인 집합으로 만든다.
def make_set(v):
    parent[v] = v
    rank[v] = 0

# 해당 정점의 최상위 정점을 찾는다.
def find(v):
    if parent[v] != v:
        parent[v] = find(parent[v])

    return parent[v]

# 두 정점을 연결한다.
def union(v, u):
    root1 = find(v)
    root2 = find(u)

    if root1 != root2:
```

```
        # 짧은 트리의 루트가 긴 트리의 루트를 가리키게 만드는 것이 좋다.
        if rank[root1] > rank[root2]:
            parent[root2] = root1
        else:
            parent[root1] = root2

            if rank[root1] == rank[root2]:
                rank[root2] += 1

def kruskal(graph):
    for v in graph['vertices']:
        make_set(v)

    mst = []

    edges = graph['edges']
    edges.sort()

    for edge in edges:
        weight, v, u = edge

        if find(v) != find(u):
            union(v, u)
            mst.append(edge)

    return mst



graph = {
'vertices': ['A', 'B', 'C', 'D', 'E', 'F', 'G'],
'edges': [
    (7, 'A', 'B'),
    (5, 'A', 'D'),
    (7, 'B', 'A'),
    (8, 'B', 'C'),
    (9, 'B', 'D'),
    (7, 'B', 'E'),
    (8, 'C', 'B'),
    (5, 'C', 'E'),
    (5, 'D', 'A'),
    (9, 'D', 'B'),
    (7, 'D', 'E'),
    (6, 'D', 'F'),
    (7, 'E', 'B'),
    (5, 'E', 'C'),
    (15, 'E', 'D'),
    (8, 'E', 'F'),
    (9, 'E', 'G'),
    (6, 'F', 'D'),
    (8, 'F', 'E'),
    (11, 'F', 'G'),
```

```
    (9, 'G', 'E'),
    (11, 'G', 'F'),
    ]
}

print( kruskal(graph) )

# [(5, 'A', 'D'),
#  (5, 'C', 'E'),
#  (6, 'D', 'F'),
#  (7, 'A', 'B'),
#  (7, 'B', 'E'),
#  (9, 'E', 'G')]
```

Line 3

```
def hanoi_move_count(n):
    dp = {0:0}
    # Loop using O(n)
    for i in range(1, n+1):
        dp[i] = 2*dp[i-1] + 1
    return dp

def find(pegs_given):
    n = sum([len(p) for p in pegs_given])

    origin = [[], [], []]
    _from = min([0, 1, 2], key=lambda x: pegs_given[x][0] if len(pegs_given[x])
> 0 else 21)
    origin[_from] = [i for i in range(1, n+1)]

    hanoi_move = hanoi_move_count(n)
    ret = 0
    # Loop using O(n)
    for i in range(2, n+1):
        next1 = (_from+1)%3
        next2 = (_from+2)%3
        if len(origin[next1]) < len(pegs_given[next1]) and i ==
pegs_given[next1][len(origin[next1])]:
            _to = next1
            _by = next2
        elif len(origin[next2]) < len(pegs_given[next2]) and i ==
pegs_given[next2][len(origin[next2])]:
            _to = next2
            _by = next1
        else:
            continue
        origin[_to] = origin[_to] + [origin[_from][-(n-i+1)]]
        origin[_by] = origin[_by] + origin[_from][-(n-i):]
        origin[_from] = origin[_from][:-(n-i+1)]
        _from = _by
        ret += hanoi_move[n-i] + 1
    return ret
```

Line 4

```
def solve(path, arrived, n):
    for u in range(n):
        visited = [0]*(n)
        if not dfs(u, visited, path, arrived):
            return "X"
    return "O"

def dfs(u, visited, path, arrived):
    if visited[u] == 1:
        return False
    visited[u] = 1
    for v in path[u]:
        if arrived[v] == -1 or dfs(arrived[v], visited, path, arrived):
            arrived[v] = u
            return True
    return False
```

Kakao 4

```
def solution(n, t, m, timetable):
    parsed_time_table = [int(t[:2])*60+int(t[3:]) for t in timetable]
    parsed_time_table.sort()
    people_each_time_zone = [[] for _ in range(n)]

    each_time_zone = 0
    for parsed_time_by_each_person in parsed_time_table:
        while parsed_time_by_each_person > 540 + each_time_zone * t:
            each_time_zone += 1
        if each_time_zone >= n:
            break
        if len(people_each_time_zone[each_time_zone]) >= m:
            each_time_zone += 1
        if each_time_zone >= n:
            break

people_each_time_zone[each_time_zone].append(parsed_time_by_each_person)
    if len(people_each_time_zone[-1]) == m:
        answer = min_to_hhmm(people_each_time_zone[-1][-1]-1)
    else:
        answer = min_to_hhmm(540 + t * (n-1))
    return answer

def min_to_hhmm(_min):
    hh = _min // 60
    mm = _min % 60
    return "{:02d}:{:02d}".format(hh, mm)

# print(solution(2, 10, 2, ["09:00", "09:00", "09:08", "09:09", "09:10"]))
# print(solution(1, 1, 5, ["08:00", "08:01", "08:02", "08:03"]))
# print(solution(2, 10, 2, ["09:10", "09:09", "08:00"]))
print(solution(1, 1, 1, ["23:59"]))
```

Kakao 5

```
def solution(str1, str2):
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    str1 = str1.lower()
    str2 = str2.lower()

    sub_str1_list = []
    for i in range(len(str1)-1):
        if str1[i:i+1] in alphabet and str1[i+1:i+2] in alphabet:
            sub_str1_list.append(str1[i:i+2])
    sub_str2_list = []
    for i in range(len(str2)-1):
        if str2[i:i+1] in alphabet and str2[i+1:i+2] in alphabet:
            sub_str2_list.append(str2[i:i+2])

    sub_str1_count_set = dict()
```

```
    for sub_str1 in sub_str1_list:
        if sub_str1 in sub_str1_count_set:
            sub_str1_count_set[sub_str1] += 1
        else:
            sub_str1_count_set[sub_str1] = 1
    sub_str2_count_set = dict()
    for sub_str2 in sub_str2_list:
        if sub_str2 in sub_str2_count_set:
            sub_str2_count_set[sub_str2] += 1
        else:
            sub_str2_count_set[sub_str2] = 1

    union = dict()
    intersection = dict()
    for sub_str1 in sub_str1_count_set:
        if sub_str1 in sub_str2_count_set:
            union[sub_str1] = max(sub_str1_count_set[sub_str1],
sub_str2_count_set[sub_str1])
            intersection[sub_str1] = min(sub_str1_count_set[sub_str1],
sub_str2_count_set[sub_str1])
        else:
            union[sub_str1] = sub_str1_count_set[sub_str1]
    for sub_str2 in sub_str2_count_set:
        if sub_str2 in sub_str1_count_set:
            continue
        else:
            union[sub_str2] = sub_str2_count_set[sub_str2]
    size_of_union = 0
    for i in union:
        size_of_union += union[i]
    size_of_intersection = 0
    for i in intersection:
        size_of_intersection += intersection[i]
    answer = 0
    if size_of_union > 0:
        answer = int(size_of_intersection / size_of_union * 65536)
    else:
        answer = 65536
    return answer

print(solution("FRANCE", "french"))
# print(solution("aa1+aa2   ", "AAAA12"))
```

kakao 6

```
def solution(m, n, board):
    deleted = [list(board[i]) for i in range(m)]
    find = True
    answer = 0
    while find:
        new_board = [list(deleted[i]) for i in range(m)]
        find = False
        for i in range(1, m):
            for j in range(1, n):
                if new_board[i][j] != '-' and new_board[i][j] == new_board[i-1]
[j] and new_board[i][j] == new_board[i][j-1] and new_board[i][j] ==
new_board[i-1][j-1]:
                    deleted[i][j] = '-'
                    deleted[i-1][j] = '-'
                    deleted[i][j-1] = '-'
                    deleted[i-1][j-1] = '-'
                    find = True
        for i in range(m):
            for j in range(n):
                if new_board[i][j] != deleted[i][j]:
                    answer += 1
        for j in range(0, n):
            for i in range(1, m):
                if deleted[i][j] == '-':
                    for k in range(i, 0, -1):
                        temp = deleted[k][j]
                        deleted[k][j] = deleted[k-1][j]
                        deleted[k-1][j] = temp
    return answer

def pprint(two_dimensional_map):
    for each_line in two_dimensional_map:
        print(each_line)
    print()

print(solution(6, 6, ["TTTANT", "RRFACC", "RRRFCC", "TRRRAA", "TTMMMF",
"TMMTTJ"]))
```

Kakao 7

```
def solution(lines):
    parsed_time_table = []
    n = len(lines)
    for line in lines:
        response_completion_time, processing_Time = line[11:].split()
        processing_Time = processing_Time[:-1]
        hh, mm, ss = list(map(int, response_completion_time.replace(".",
"").split(":")))
        parsed_completion_sec = (hh*3600*1000 + mm*60*1000 + ss)
        parsed_start_sec = parsed_completion_sec -
```

```
int(float(processing_Time)*1000) + 1
        parsed_time_table.append((parsed_start_sec, parsed_completion_sec))

    answer = 0
    for i in range(n):
        cur = parsed_time_table[i]
        cnt = 1
        for j in range(n):
            if i == j:
                continue
            compare = parsed_time_table[j]
            if (compare[0] >= cur[1] and compare[0] < cur[1]+1000) or
(compare[1] < cur[1]+1000 and compare[1] >= cur[1]) or (compare[0] < cur[1] and
compare[1] >= cur[1]+1000):
                cnt += 1
        answer = max(answer, cnt)
        cnt = 1
        for j in range(n):
            if i == j:
                continue
            compare = parsed_time_table[j]
            if (compare[0] > cur[0] - 1000 and compare[0] <= cur[0]) or
(compare[1] <= cur[0] and compare[1] > cur[0] - 1000) or (compare[0] <= cur[0]
and compare[1] > cur[0] - 1000):
                cnt += 1
        answer = max(answer, cnt)
    return answer



print(solution([
    "2016-09-15 20:59:57.421 0.351s",
    "2016-09-15 20:59:58.233 1.181s",
    "2016-09-15 20:59:58.299 0.8s",
    "2016-09-15 20:59:58.688 1.041s",
    "2016-09-15 20:59:59.591 1.412s",
    "2016-09-15 21:00:00.464 1.466s",
    "2016-09-15 21:00:00.741 1.581s",
    "2016-09-15 21:00:00.748 2.31s",
    "2016-09-15 21:00:00.966 0.381s",
    "2016-09-15 21:00:02.066 2.62s"
]))
```

Kakao 2nd

```
from threading import Timer
import sys
import requests

sys.setrecursionlimit(10**7)
```

```
URL = 'http://api.welcome.kakao.com'

def chunks(l, n):
    for i in range(0, len(l), n):
        yield l[i:i + n]

def get_token():
    loginToken = 'UWrwxkW5ddrqeh4lm1Kz30eqWSYb2Vm76-b47fkqzDO'
    res = requests.get(URL+'/token/'+loginToken)
    token = res.content.decode("utf-8")
    return token

def get_seed(token):
    headers = {'X-Auth-Token': token}
    res = requests.get(URL+'/seed/', headers=headers)
    if not res.ok:
        print("get_seed error")
        # raise NotImplementedError
    else:
        print("get_seed", res.ok)
    split_data = res.content.split(b"\n")
    seed = list(map(lambda x: x.decode('utf-8'), list(filter(lambda x: len(x) >
0, split_data))))
    return seed

def get_document(token, doc_id):
    headers = {'X-Auth-Token': token}
    res = requests.get(URL+doc_id, headers=headers)
    if not res.ok:
        print("get_document error")
        return None, False
        # raise NotImplementedError
    else:
        print("get_document", res.ok)
    return res.json(), True

def extract_feature(token, to_add, add_hash_set):
    headers = {'X-Auth-Token': token}
    to_add = list(filter(lambda x: not x['id'] in add_hash_set, to_add))
    for image in to_add:
        add_hash_set.add(image['id'])

    split_to_add = list(chunks(to_add, 50))
    features = []
    for each_of_split_to_add in split_to_add:
        ids = ",".join(list(map(lambda x: x['id'], each_of_split_to_add)))
        res = requests.get(URL+'/image/feature?id='+ids, headers=headers)
        if not res.ok:
            print("extract_feature error")
            # raise NotImplementedError
            return None, False
```

```
        else:
            print("extract_feature", res.ok)

        features += res.json()['features']
    return features, True

def save_feature(token, features):
    headers = {'X-Auth-Token': token}
    split_features = list(chunks(features, 50))
    for each_of_split_features in split_features:
        data = {'data': each_of_split_features}
        res = requests.post(URL+'/image/feature', headers=headers, json=data)
        if not res.ok:
            print("save_feature error")
            # raise NotImplementedError
        else:
            print("save_feature", res.ok)
    return

def delete_image(token, to_delete, del_hash_set):
    headers = {'X-Auth-Token': token}
    # to_delete = list(filter(lambda x: not x['id'] in del_hash_set,
to_delete))
    # for image in to_delete:
    #     del_hash_set.add(image['id'])

    split_to_delete = list(chunks(to_delete, 50))
    for each_of_split_to_delete in split_to_delete:
        data = {'data': each_of_split_to_delete}
        data['data'] = list(map(lambda x: {'id': x['id']}, data['data']))
        res = requests.delete(URL+'/image/feature', headers=headers, json=data)
        if not res.ok:
            print("delete_image error")
            # raise NotImplementedError
        else:
            print("delete_image", res.ok)
    return

def worker(token, doc_id, add_hash_set, del_hash_set, refresh):
    document, success = get_document(token, doc_id)
    if not success or (doc_id == document['next_url'] and
len(document['images']) == 0):
        if refresh > 100000:
            return
        print("Reload")
        t = Timer(refresh, worker(token, doc_id, add_hash_set, del_hash_set,
refresh*2))
        t.start()
        return
    to_add = list(filter(lambda x: x['type'] == 'add', document['images']))
    to_delete = list(filter(lambda x: x['type'] == 'del', document['images']))
```

```
    delete_image(token, to_delete, del_hash_set)
    features, success = extract_feature(token, to_add, add_hash_set)
    if success:
        save_feature(token, features)
    t = Timer(0.35, worker(token, document['next_url'], add_hash_set,
del_hash_set, refresh))
    t.start()


def main():
    token = get_token()
    seed = get_seed(token)
    for doc_id in seed:
        add_hash_set = set()
        del_hash_set = set()
        worker(token, doc_id, add_hash_set, del_hash_set, 1)

main()
```

BFS DFS

```
def dfs(N, M, V, edges, dfs_visited):
    if dfs_visited[V] == 1:
        return []
    dfs_visited[V] = 1
    ret = [V]
    for dv in edges[V]:
        ret += dfs(N, M, dv, edges, dfs_visited)
    return ret

def bfs(N, M, V, edges):
    from collections import deque
    bfs_visited = [0] * (N+1)
    q = deque()
    q.append(V)
    ret = []
    while q:
        u = q.popleft()
        if bfs_visited[u] == 1:
            continue
        bfs_visited[u] = 1
        ret.append(u)
        for dv in edges[u]:
            q.append(dv)
    return ret

def run():
    import sys
    read = sys.stdin.readline
    N, M, V = map(int, read().split())
    edges = dict([[i, []] for i in range(N+1)])
    for _ in range(M):
        u1, u2 = map(int, read().split())
        edges[u1].append(u2)
        edges[u2].append(u1)
    for each_V in edges:
        edges[each_V].sort()
    dfs_visited = [0] * (N+1)
    print(" ".join(map(str, dfs(N, M, V, edges, dfs_visited))))
    print(" ".join(map(str, bfs(N, M, V, edges))))

run()
```

Dijkstra

```
def packagingRoad(path,n,k):
    import heapq
    MAX_INT = int(1e9)
    visited = [[MAX_INT]*21 for _ in range(n+1)]
    visited[1][0] = 0
    q = []
    heapq.heappush(q,(0,0,1))
    while q:
        d,c,u = heapq.heappop(q)
        if d > visited[u][c]:
            continue
        for v,w in path[u]:
            if d+w < visited[v][c]:
                visited[v][c] = d+w
                heapq.heappush(q,(d+w,c,v))
            if ((c+1) <= k) and (d < visited[v][c+1]):
                visited[v][c+1] = d
                heapq.heappush(q,(d,c+1,v))
    return min(visited[n])

    def run():
        import sys
        read = sys.stdin.readline
        n,m,k = map(int,read().split())
        path = [[] for _ in range(n+1)]
        for _ in range(m):
            u,v,w = map(int,read().split())
            path[u].append((v,w))
            path[v].append((u,w))
        print(packagingRoad(path,n,k))

run()
```

Kth Shortest Path

```
def kthShortestPath(path,n,k):
    import heapq
    MAX_INT = int(1e6)
    visited = [[MAX_INT]*k for _ in range(n+1)]
    visited[1][0] = 0
    q = []
    heapq.heappush(q,(0,1))
    while q:
        d,u = heapq.heappop(q)
        for v,w in path[u]:
            if d+w < visited[v][k-1]:
                visited[v][k-1] = d+w
                visited[v] = sorted(visited[v])
                heapq.heappush(q,(d+w,v))
    solution = [eachShortestPath[k-1] for eachShortestPath in visited][1:]
    for i in range(n):
        if solution[i] == MAX_INT:
            solution[i] = str(-1)
        else:
            solution[i] = str(solution[i])
    return solution

def run():
    import sys
    read = sys.stdin.readline
    n,m,k = map(int,read().split())
    path = [[] for _ in range(n+1)]
    for _ in range(m):
        u,v,w = map(int,read().split())
        path[u].append((v,w))
    print("\n".join(kthShortestPath(path,n,k)))
```

Candy store (dp)

```
import sys
read = sys.stdin.readline

while True:
    n,m = map(float,read().split())
    if n == 0 and m == 0: break
    cent = int(100*m)
    dp = [0]*(cent+1)
    for _ in range(int(n)):
        c,p = map(float,read().split())
        c,p = int(c),int(100*p)
        for i in range(p,cent+1):
            dp[i] = max(dp[i],dp[i-p]+c)
    print(dp[cent])
```

이분 매칭

```
def dfs(u,visited,path,arrived):
    if visited[u] == 1:
        return 0
    visited[u] = 1
    for v in path[u]:
        if arrived[v] == 0 or dfs(arrived[v],visited,path,arrived):
            arrived[v] = u
            return 1
    return 0

def stone(path,arrived,n):
    ret = 0
    for u in range(1,n+1):
        visited = [0]*(n+1)
        if dfs(u,visited,path,arrived):
            ret+=1
    return ret

def run():
    import sys
    read = sys.stdin.readline
    n,k = map(int,read().split())
    path = [[] for _ in range(n+1)]
    arrived = [0]*(k+1)

    for _ in range(k):
        i,j = map(int,read().split())
        path[i].append(j)
    print(stone(path,arrived,n))

run()
```

Longest substring with k unique characters

```
import sys
read = sys.stdin.readline

def longestSubstringWithKUniqueCharacters(string, k):
    ret = ""
    ret_size = 0
    n = len(string)
    _hash = {}
    unique = 0
    lastIdx = 0
    for i in range(n):
        new = string[i]
        if (new not in _hash) or (new in _hash and _hash[new]==0):
            _hash[new] = 1
            unique += 1
        else:
            _hash[new] += 1
        while unique > k:
            last = string[lastIdx]
            if _hash[last] > 1:
                lastIdx += 1
                _hash[last] -= 1
            else:
                lastIdx += 1
                _hash[last] = 0
                unique -= 1
        if unique == k and i+1-lastIdx > ret_size:
            ret = string[lastIdx:i+1]
            ret_size = i+1-lastIdx
    if ret != "":
        return ret_size
    else:
        return -1

def run():
    n = int(read().replace("\n", ""))
    for _ in range(n):
        string= read().replace("\n", "")
        k= int(read().replace("\n", ""))
        print(longestSubstringWithKUniqueCharacters(string, k))

run()
```

Work break

```
def solve(N, words, s):
    words = set(words)
    dp = [[] for _ in range((len(s)+1))]
    dp[0] = [[]]
    for i in range(len(s)+1):
        for j in range(1, 15):
            if i-j >= 0 and len(dp[i-j]) > 0 and s[i-j:i] in words:
                for e_in_dp in dp[i-j]:
                    dp[i].append(e_in_dp + [s[i-j:i]])
    return ("(" + ")(".join(sorted([" ".join(dp[-1][i]) for i, e in
enumerate(dp[-1])])) + ")")

def main():
    T = I()
    for _ in range(T):
        N = I()
        words = LS()
        s = S()
        print(solve(N, words, s))

main()
```

Egg drop

```
def solve(n, k):
    W = [[0] * (k+1) for _ in range(n+1)]
    W[1] = [i for i in range(k+1)]
    for i in range(2, n+1):
        for j in range(1, k+1):
            W[i][j] = 1 + min(max(W[i-1][x-1], W[i][j-x]) for x in range(1,
j+1))
    return W[n][k]

def main():
    T = I()
    for _ in range(T):
        n, k = LI()
        print(solve(n, k))

main()
```

Longest valid parentheses

```
def solve(parenthesis):
    stack = []
    cur = 0
    ret = 0
    for i, e in enumerate(parenthesis):
        if e == '(':
            stack.append(cur)
            cur = 0
        elif e == ')' and len(stack) > 0:
            cur += stack.pop() + 2
            ret = max(ret, cur)
        elif e == ')' and len(stack) == 0:
            cur = 0
    return ret

def main():
    n = I()
    for _ in range(n):
        parenthesis = S()
        print(solve(parenthesis))

main()
```

Shortest Pair

```
def dist(point1, point2):
    return (point1[0]-point2[0]) ** 2 + (point1[1]-point2[1]) ** 2

def recursive_solve(points, start, end):
    if start == end:
        INF = 1e100
        return INF
    mid = (start+end)//2
    d = min(recursive_solve(points, start, mid), recursive_solve(points, mid
+1, end))
    i = mid
    while i <= end and (points[mid][0] - points[i][0])**2 < d:
        i += 1
    j = mid
    while j >= start and (points[mid][0] - points[j][0])**2 < d:
        j -= 1
    center = sorted(points[j+1:i], key=lambda x: x[1])
    for i in range(len(center)-1):
        for j in range(i+1, i+6):
            if j >= len(center):
                break
            if center[i][1] - center[j][1] >= d:
                break
            d = min(d, dist(center[i], center[j]))
    return d

def solve(n, points):
    points.sort(key=lambda x: x[0])
    return recursive_solve(points, 0, n-1)
```

Kakao blind mock 4

```
def get_element_without_null_exception(board, i, j):
    if i < 0 or j < 0:
        return 0
    else:
        return board[i][j]


def solution(board):
    dp = [[0] * len(board[0]) for _ in range(len(board))]
    for i in range(len(board)):
        for j in range(len(board[0])):
            if board[i][j] == 0:
                dp[i][j] = 0
            else:
                dp[i][j] = min(get_element_without_null_exception(dp,i-1,j-1),
get_element_without_null_exception(dp,i,j-1),
get_element_without_null_exception(dp,i-1,j))+1
    max_square_size = 0
    for i in range(len(dp)):
        for j in range(len(dp[0])):
            if dp[i][j]**2 > max_square_size:
                max_square_size = dp[i][j]**2
    return max_square_size

ex_board = \
[[0, 1, 1, 1],
 [0, 1, 1, 1],
 [0, 1, 1, 1],
 [0, 0, 1, 0]]

ex_board2 = \
[[1, 0, 1, 1],
 [1, 1, 1, 1]]
print(solution(ex_board2))
```

Kakao blind mock 7

```
MAX_INT = 1e100
def solution(strs, t):
    set_strs = set(strs)
    dp = [MAX_INT] * (len(t)+1)
    dp[0] = 0
    for j in range(1, len(t)+1):
        dp[j] = MAX_INT
        for i in range(1, 6):
            if (j-i) < 0:
                continue
            if t[j-i:j] in set_strs:
                dp[j] = min(dp[j], dp[j-i]+1)
    ret = dp[len(t)]
    if ret == MAX_INT:
        return -1
    return ret

print(solution(["banan", "n", "a"],"banana")==2)
print(solution(["ba", "na", "n", "a"],"banana")==3)
print(solution(["ba", "na", "n", "b"],"b")==1)
print(solution(["app","ap","p","l","e","ple","pp"],"apple")==2)
print(solution(["ba","an","nan","ban","n"],"banana")== -1)
```

자두나무

```
def get_item_without_null_exception(i, k, p, dp):
    if i < 0 or k < 0 or p < 0:
        return 0
    else:
        return dp[i][k][p]

def solve(T, W, order):
    import heapq
    q = []
    heapq.heappush(q, (W, 1 if 1 == order[0] else 0, 0, 1))
    heapq.heappush(q, (W, 1 if 2 == order[0] else 0, 0, 2))
    dp = [[[0]*2 for _ in range(W+1)] for _ in range(T)]

    for i in range(0, T):
        for k in range(W+1):
            if order[i] == 1:
                dp[i][k][0] = max(get_item_without_null_exception(i-1, k, 0,
dp), get_item_without_null_exception(i-1, k-1, 1, dp)) + 1
                dp[i][k][1] = max(get_item_without_null_exception(i-1, k, 1,
dp), get_item_without_null_exception(i-1, k-1, 0, dp))
            else:
                dp[i][k][0] = max(get_item_without_null_exception(i-1, k, 0,
dp), get_item_without_null_exception(i-1, k-1, 1, dp))
                dp[i][k][1] = max(get_item_without_null_exception(i-1, k, 1,
dp), get_item_without_null_exception(i-1, k-1, 0, dp)) + 1
    return max(max(dp[T-1], key=lambda x: max(x)))
```

Coin

```
def solve(n, k, values):
    cache = [0]*(k+1)
    cache[0] = 1
    for value in values:
        for i in range(k+1):
            if i >= value:
                cache[i] += cache[i-value]
    return cache[k]
```

전생했더니 슬라임 연구자였던 건에 대하여 (Hard)

```
import heapq

def solve(N, slimes):
    if N == 1:
        return 1
    ret = 1
    heap_slimes = []
    for slime in slimes:
        heapq.heappush(heap_slimes, (slime, slime))
    for _ in range(N-1):
        first_min = heapq.heappop(heap_slimes)
        second_min = heapq.heappop(heap_slimes)
        # first_min = slimes.pop(slimes.index(min(slimes)))
        # second_min = slimes.pop(slimes.index(min(slimes)))
        new_slime = first_min[1] * second_min[1]
        heapq.heappush(heap_slimes, (new_slime, new_slime))
        slimes.append(new_slime)
        ret *= new_slime
    return ret % 1000000007
```

퇴근시간

```
INF = 1e100
def solve(distance, late, N, S, E):
    import heapq
    times = [INF] * (N+1)
    times[1] = 0
    q = []
    heapq.heappush(q, (0, 1))
    while q:
        from_time, from_idx = heapq.heappop(q)
        for to_idx, from_to_distance in distance[from_idx]:
            rush_hour = late[from_idx][to_idx] and from_time < E and from_time
+ from_to_distance > S
            to_time = from_time

            if not rush_hour:
                to_time += from_to_distance

            if rush_hour and from_time < S and from_to_distance - (S -
from_time) < (E-S) / 2:
                to_time += (S-from_time) + (from_to_distance - (S - from_time))
* 2
            elif rush_hour and from_time < S and from_to_distance - (S -
from_time) >= (E-S) / 2:
                to_time += (S-from_time) + (E - S) + (from_to_distance - (S -
from_time) - (E-S) / 2)
            elif rush_hour and from_time >= S and from_to_distance < (E-
from_time) / 2:
                to_time += from_to_distance * 2
```

```
            elif rush_hour and from_time >= S and from_to_distance >= (E-
from_time) / 2:
                to_time += (E-from_time) + (from_to_distance - (E - from_time)
/ 2)

            if times[to_idx] > to_time:
                times[to_idx] = to_time
                heapq.heappush(q, (times[to_idx], to_idx))
    ret = max(times[1:])
    if int(ret) == ret:
        return int(ret)
    return ret


def run():
    import sys
    read = sys.stdin.readline
    N, M, S, E = map(int, read().split())
    distance = dict([[i, []] for i in range(N+1)])
    late = [[0]*(N+1) for _ in range(N+1)]
    for _ in range(M):
        A, B, L, t1, t2 = map(int, read().split())
        distance[A].append((B,L))
        distance[B].append((A,L))
        late[A][B] = t1
        late[B][A] = t2
    print(solve(distance, late, N, S, E))

run()
```