#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100005

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

typedef struct Node{

    int city;

    struct Node\* next;

}Node;

Node\*graph[MAX];

bool visited[MAX];

void add(int u,int v){

    Node\* newN=(Node\*)malloc(sizeof(Node));

    newN->city=v;

    newN->next=graph[u];

    graph[u]=newN;

}

int dfs(int u){

    visited[u]=true;

    int size=1;

    for(Node\* cur=graph[u];cur!=NULL;cur=cur->next){

        if(!visited[cur->city]){

            size+=dfs(cur->city);

        }

    }

    return size;

}

/\*

 \* Complete the 'roadsAndLibraries' function below.

 \*

 \* The function is expected to return a LONG\_INTEGER.

 \* The function accepts following parameters:

 \*  1. INTEGER n

 \*  2. INTEGER c\_lib

 \*  3. INTEGER c\_road

 \*  4. 2D\_INTEGER\_ARRAY cities

 \*/

long roadsAndLibraries(int n, int clib, int c\_road, int cities\_rows, int cities[][2]) {

    if(clib<=c\_road){

        return(long)n\* clib;

    }

    for(int i=1;i<=n;i++){

        graph[i]=NULL;

        visited[i]=false;

    }

    for(int i=0;i<cities\_rows;i++){

        int u=cities[i][0],v=cities[i][1];

        add(u,v);

        add(v,u);

    }

    long cost=0;

    for(int i=1;i<=n;i++){

        if(!visited[i]){

            int comp=dfs(i);

            cost+=clib;

            cost+=(long)(comp-1)\*c\_road;

        }

    }

    return cost;

}

int main()

{

    FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

    int q = parse\_int(ltrim(rtrim(readline())));

    for (int q\_itr = 0; q\_itr < q; q\_itr++) {

        char\*\* first\_multiple\_input = split\_string(rtrim(readline()));

        int n = parse\_int(\*(first\_multiple\_input + 0));

        int m = parse\_int(\*(first\_multiple\_input + 1));

        int c\_lib = parse\_int(\*(first\_multiple\_input + 2));

        int c\_road = parse\_int(\*(first\_multiple\_input + 3));

        int\*\* cities = malloc(m \* sizeof(int\*));

        for (int i = 0; i < m; i++) {

            \*(cities + i) = malloc(2 \* (sizeof(int)));

            char\*\* cities\_item\_temp = split\_string(rtrim(readline()));

            for (int j = 0; j < 2; j++) {

                int cities\_item = parse\_int(\*(cities\_item\_temp + j));

                \*(\*(cities + i) + j) = cities\_item;

            }

        }

        long result = roadsAndLibraries(n, c\_lib, c\_road, m, cities);

        fprintf(fptr, "%ld\n", result);

    }

    fclose(fptr);

    return 0;

}

char\* readline() {

    size\_t alloc\_length = 1024;

    size\_t data\_length = 0;

    char\* data = malloc(alloc\_length);

    while (true) {

        char\* cursor = data + data\_length;

        char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

        if (!line) {

            break;

        }

        data\_length += strlen(cursor);

        if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') {

            break;

        }

        alloc\_length <<= 1;

        data = realloc(data, alloc\_length);

        if (!data) {

            data = '\0';

            break;

        }

    }

    if (data[data\_length - 1] == '\n') {

        data[data\_length - 1] = '\0';

        data = realloc(data, data\_length);

        if (!data) {

            data = '\0';

        }

    } else {

        data = realloc(data, data\_length + 1);

        if (!data) {

            data = '\0';

        } else {

            data[data\_length] = '\0';

        }

    }

    return data;

}

char\* ltrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    while (\*str != '\0' && isspace(\*str)) {

        str++;

    }

    return str;

}

char\* rtrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    char\* end = str + strlen(str) - 1;

    while (end >= str && isspace(\*end)) {

        end--;

    }

    \*(end + 1) = '\0';

    return str;

}

char\*\* split\_string(char\* str) {

    char\*\* splits = NULL;

    char\* token = strtok(str, " ");

    int spaces = 0;

    while (token) {

        splits = realloc(splits, sizeof(char\*) \* ++spaces);

        if (!splits) {

            return splits;

        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");

    }

    return splits;

}

int parse\_int(char\* str) {

    char\* endptr;

    int value = strtol(str, &endptr, 10);

    if (endptr == str || \*endptr != '\0') {

        exit(EXIT\_FAILURE);

    }

    return value;

}