CyberChallenge.IT 2023 - Programming
Commented solutions

Contents

1 Problem 1 - "Pretest" [40 points]

1.1 Problem Statement

It's pretest time for the CyberChallenge.IT 2023 edition!

As every year, the pretest is a multiple choice test, with Q questions with 4 possible answers (A, B, C, D) each, of which one and only one is correct.

The organizers want to write a software to automatically grade the tests of the N participating candidates, giving them 1 point for each correct answer and 0 points for each wrong or missing one.

Given the number of questions Q and candidates N, the list of correct answers (in the form of a string with length Q), and N strings representing the answers given by each candidate, compute the number of points scored for each of them.

1.2 Problem Details

Input

The input consists of $N + 2$ lines:

- Line 1: The numbers Q and N, separated by a space.
- Line 2: The list of correct answers, as a string of Q uppercase letters in the set {A, B, C, D}.
- Lines $3, \ldots, N + 2$: a string of Q characters in the set {A, B, C, D, ?}, where ? denotes a missing answer.

Output

The output must contain N lines. In line i, you should output a single integer with the number of points scored by the i-th candidate, in the same order as in the input.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

- **Subtask 1** **[20 points]**: $N = 1$, $1 \leq Q \leq 1000$.
- **Subtask 2** **[20 points]**: $1 \leq N \leq 1000$, $1 \leq Q \leq 1000$.

Examples

INPUT	OUTPUT
10 3 CBBDACCCBC D?ABCD?DDB AB?ACCBAAA B?CCBA?C?D	0 2 1

Explanation

From the first line, we know that there are 10 questions and 3 participants. The answers of the first participant are all different from the correct ones or missing, so their score is 0. The second one only matches on the second and sixth question, so their score is 2. The same goes for the third one, who gave the right answer only for question 8, therefore their score is 1.

1.3 Solution

In this task we need to compare char-by-char the first provided string (the one with the correct answers) with every other one (the ones given by the candidates) and, for each of them, just count the number of occurrences for which they are equal and output it.

1.4 Source Code

C++

```cpp
#include <stdio.h>
#include <iostream>

using namespace std;

int score(string correct, string candidate){
    int res = 0;
    for (size_t i = 0; i < correct.length(); i++)
        res += int(correct[i] == candidate[i]);
    return res;
}

int main(){
    int N, Q;
    string correct;
    cin >> Q >> N >> correct;

    for (int i = 0; i < N; i++){
        string candidate;
        cin >> candidate;
        cout << score(correct, candidate) << '\n';
    }

    return 0;
}
```

Python

```python
#!/bin/env python3

import sys

fin = sys.stdin
fout = sys.stdout

def score(correct, candidate):
    return sum([x == y for x, y in zip(correct, candidate)])

Q, N = map(int, fin.readline().strip().split(" "))
correct = fin.readline().strip()

for i in range(N):
    candidate = fin.readline().strip()
    print(score(correct, candidate), file=fout)
```

2 Problem 2 - "Who's the winner?" [60 points]

2.1 Problem Statement

The local selection of the CyberChallenge.IT program is an individual CTF competition in jeopardy-style.

Each of the M players is faced with N tasks, each of them worth a fixed amount of points $P_1, ..., P_N$. Each task has a correct answer, called the *flag*, that must be submitted by players in order to get the task's points. There are no penalties for wrong submissions and, obviously, a player can solve a task only once (which means that resubmissions of the same flag will not modify the player's score).

At the end of the competition, the player who scored the most points is the winner. In case of a draw, the "submission time" of a player is taken into account, that is, the timestamp of the last submission that modified the player's score. In this case, the player with lower submission time is considered the winner. So, the final scoreboard is computed by number of points in descending order, then submission time in ascending order. Finally, if there are still "draw" situations, the players are ordered by their player id, in ascending order (in particular, this is applied to players with score equal to 0, for which the submission time is not set).

Given the number of players M, tasks N and total submissions S during the competition, the list of tasks with their numerical id, amount of points and the correct flag, and the list of submissions with the associated submitted flag, timestamp and player id, can you determine the final scoreboard?

Note: tasks are numbered from 1 to N, while players are numbered from 1 to M for convenience.
Note: submissions are given in random order.

2.2 Problem Details

Input

The input consists of $N + S + 1$ lines:

- Line 1: the numbers M, N and S, separated by a space.
- Lines $2, \ldots, N + 1$: the description of the tasks. Each of these lines will have 3 space-separated values, namely the task id (a unique integer between 1 and N), the correct flag (a string consisting only in uppercase and lowercase letters of length 10), and the number of points associated to the task (a positive integer up to 1000).
- Lines $N + 2, \ldots, N + S + 1$: the description of the submissions. Each of these lines will have 4 space-separated values, namely the player id, the task id for which the submission was made, the flag submitted (a string of length 10) and the timestamp of the submission (a positive integer up to 10^6).

Output

The output must contain M lines, with two space-separated values each. The i-th line must contain the player id of the player in i-th position and the associated number of points.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

- **Subtask 1** **[30 points]**: $1 \leq M \leq 10$, $1 \leq N \leq 10$, $1 \leq S \leq 100$.
- **Subtask 2** **[30 points]**: $1 \leq M \leq 300$, $1 \leq N \leq 300$, $1 \leq S \leq 50000$.

Examples

INPUT	OUTPUT
5 2 6 1 EfnccJSqUyOsYGO 50 2 XWMsVGHynvrEspF 100 2 2 XWMsVGHynvrEspF 7 2 1 EfnccJSqUyOsYGO 4 4 1 EfnccJSqUyOsYGO 10 1 1 EfnccJSqUyOsYGO 5 1 2 bWWinauoIIDfKpz 6 1 1 EfnccJSqUyOsYGO 25	2 150 1 50 4 50 3 0 5 0

Explanation

From the first line, the competition has 5 players, 2 tasks and 6 total submissions. Task 1 is worth 50 points and its flag is `EfnccJSqUyOsYGO`, while task 2 is worth 100 points and its flag is `XWMsVGHynvrEspF`. Let's go through the submissions:

- Player 2 submitted a correct answer to task 2 at time 7. So their total number of points is now 100 and their submission time is 7.
- Player 2 submitted a correct answer to task 1 at time 4. So their score goes up to 150. Notice that, since the submission for task 1 happened before the one for task 2, the submission time is unchanged.
- Player 4 submitted a correct answer to task 1 at time 10. So their total number of points is now 50 and their submission time is 10.
- Player 1 submitted a correct answer to task 1 at time 5. So their total number of points is now 50 and their submission time is 5.
- Player 1 submitted a **wrong** answer to task 2 at time 6. So both their score and penalty time remain unchanged.
- Player 1 submitted a correct answer to task 1 at time 25. However, they already solved that task before, so both their score and penalty time remain unchanged.

At the end, player 2 is the only one with both tasks solved, being so in first position. Players 1 and 4 both solved only one task, so they are both at 50 points, but player 1 is in second place because of submission time (5 for player 1 and 10 for player 4). Both players 3 and 5 did not manage to get any points, so they are ordered by their id, with player 3 finishing in position 4 and player 5 in position 5.

2.3 Solution

This is an implementation task. In order to calculate the final scoreboard, we keep a partial scoreboard and we iterate through all the submissions. For each submission, we check if the submitter has already solved the challenge, otherwise we check if the submission is correct and update the points count accordingly. One important thing when increasing the scores, is to keep also the timestamp updated. At the end, we sort the scoreboard by points, timestamp and then player id.

2.4 Source Code

C++

```cpp
#include <stdio.h>
#include <iostream>
#include <vector>
```

```cpp
#include <string>
#include <tuple>
#include <map>
#include <algorithm>

using namespace std;

void rank_players(int M, int N, int S, vector<pair<string, int>> tasks,
                  vector<tuple<int, int, string, int>> submissions){
    map<int, pair<int, int>> scores;
    vector<vector<int>> solved(M, vector<int>(N, 0));
    vector<tuple<int, int, int>> scoreboard;

    for(int i = i; i<M; i++){
        scores[i] = make_pair(0, 0);
    }

    for(auto& sub : submissions){
        int player = get<0>(sub);
        int task = get<1>(sub);
        string submitted = get<2>(sub);
        int timestamp = get<3>(sub);

        if (submitted == tasks[task-1].first &&
           (solved[player-1][task-1] == 0 || solved[player-1][task-1] > timestamp)){
            if (solved[player-1][task-1] == 0)
                scores[player-1].first += tasks[task-1].second;
            solved[player-1][task-1] = timestamp;
        }
    }

    for(int i = 0; i<M; i++){
        scores[i].second = *max_element(solved[i].begin(), solved[i].end());
        scoreboard.push_back(make_tuple(scores[i].first, -scores[i].second, -(i+1)));
    }

    sort(scoreboard.begin(), scoreboard.end());
    reverse(scoreboard.begin(), scoreboard.end());

    for(auto& el : scoreboard){
        cout<<(-get<2>(el))<<" "<<get<0>(el)<<endl;
    }

}

int main(){
    vector<pair<string, int>> tasks;
    vector<tuple<int, int, string, int>> submissions;
    int M, N, S;
    cin >> M >> N >> S;

    for(int i = 0; i<N; i++){
        string a;
        int b, x;
        cin >> x >> a >> b;
        tasks.push_back(make_pair(a, b));
    }

    for(int i = 0; i<S; i++){
        int a, b, d;
        string c;
        cin >> a >> b >> c >> d;
```

```
        submissions.push_back(make_tuple(a, b, c, d));
    }

    rank_players(M, N, S, tasks, submissions);

    return 0;
}
```

Python

```
#!/bin/env python3

from collections import defaultdict
import sys

fin = sys.stdin
fout = sys.stdout

def rank(M, N, S, tasks, submissions):
    scores = defaultdict(lambda: [0, 0])
    solved = [[0]*N for _ in range(M)]
    for sub in submissions:
        player, task, submitted, timestamp = sub
        if submitted == tasks[task-1][0] and
           (solved[player-1][task-1] == 0 or solved[player-1][task-1] > timestamp):
            if solved[player-1][task-1] == 0:
                scores[player-1][0] += tasks[task-1][1]
            solved[player-1][task-1] = timestamp
    for player in range(M):
        scores[player][1] = max(solved[player])
    scoreboard = sorted([[scores[i][0], scores[i][1], i+1]
                         for i in range(M)], key=lambda k: (k[0], -k[1], -k[2]), reverse=True)

    for el in scoreboard:
        print(el[2], el[0], file=fout)


M, N, S = map(int, fin.readline().strip().split())
tasks = []
submissions = []

for _ in range(N):
    tid, flag, points = fin.readline().strip().split()
    tasks.append((flag, int(points)))

for _ in range(S):
    player, task, submitted, timestamp = fin.readline().strip().split()
    submissions.append((int(player), int(task), submitted, int(timestamp)))

rank(M, N, S, tasks, submissions)
```

3 Problem 3 - "The national final" [100 points]

3.1 Problem Statement

The day of the national final of CyberChallenge.IT 2023 has arrived. The best six hackers of every venue are ready to battle for the eternal glory.

The organizers' team prepared N vulnerable services for the event, that will run on each team's vulnerable machine. The *checksystem*, hosted by the organizers, has the job of checking that everything works correctly on them in every round of the competition. All the rounds have a fixed duration of T seconds.

The checksystem is a computer program with W *workers*. Every worker can execute at most one task at a time, which means that no more than W tasks can be executed at the same time. Moreover, a task executing on the checksystem can never be interrupted: if it lasts t seconds, a worker will remain busy for exactly that time. When a task finishes, the next task starts immediately, without delays between them.

For every vulnerable service, the organizers wrote a *check* routine that takes a fixed amount of time (possibly different for each service) to be executed as a task on the checksystem.

Given the number of vulnerable service N, the duration (in seconds) of a round T and N positive integers $t_1, ..., t_N$ representing the time needed (in seconds) for each *check* routine to be executed, what is the minimum number of workers W that is necessary to ensure that all the checks can be completed in the round length?

Note: the checksystem executes the tasks in the listed order. In particular, there is a common queue between workers, and when a worker finishes a task it can only start the next task that has not started yet. Moreover, tasks' times can not be splitted between workers.

3.2 Problem Details

Input

The input consists of 2 lines:

- Line 1: the numbers N and T, separated by a space.
- Line 2: N space-separated positive integers representing the length of the checks $t_1, ..., t_N$.

Output

The output must contain a single positive integer W, the minimum number of workers that is necessary to execute all the tasks.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

- **Subtask 1** [**30 points**]: $1 \leq N \leq 100$, $1 \leq T \leq 10^6$, $1 \leq t_1, \ldots, t_N \leq 100000$, $t_1 = t_2 = \cdots = t_N$.
- **Subtask 2** [**40 points**]: $1 \leq N \leq 1000$, $1 \leq T \leq 10^6$, $1 \leq t_1, \ldots, t_N \leq 100000$.
- **Subtask 3** [**30 points**]: $1 \leq N \leq 10^5$, $1 \leq T \leq 10^6$, $1 \leq t_1, \ldots, t_N \leq 100000$.

Examples

INPUT	OUTPUT
6 100 12 49 87 21 11 31	3

Explanation

Three workers are enough. In fact, at the beginning the first 3 tasks are assigned to the 3 workers. Worker 1 is the first to finish at time 12, so it will start the fourth task. Again, the first worker finishes a task at time 33, so it can start the 5th one. Finally, at time 44 the first worker finishes again, starting the 6th and last task. At the end, worker 1 have been busy for 75 seconds, worker 2 for 49 and worker 3 for 87.
Conversely, two workers are not enough, since the total sum of the timings is greater than 200, so it is impossible to divide them between two workers that will work for at most 100 seconds.

3.3 Solution

This is an algorithmic task. The key observation here is that, if W workers are enough, then also $W + 1$ workers are, and similarly, if W workers are not enough, the same goes for $W - 1$. This means that the function $f(W)$ that outputs true if W workers are enough and false otherwise is monotonic, and so we can binary-search through it.
The answer is the minimum value W for which $f(W)$ is true. On the implementation side, we used a priority queue for simplicity to simulate the full process. We simply keep adding jobs when one of them is finished, and we check at the end if we can make it on time.

3.4 Source Code

C++

```cpp
#include <stdio.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

bool simulate(int k, vector<int> t, int N, int T){
    int time = 0;
    priority_queue<int, vector<int>, greater<int>> pq(t.begin(), t.begin() + k);

    int i = k;

    while(i < N){
        time = pq.top();
        pq.pop();
        pq.push(t[i] + time);
        i++;
    }

    int m = 0;
    while(!pq.empty()){
        m = pq.top();
        pq.pop();
    }

```

```cpp
    return m <= T;
}

int main(){
    int N, T;
    cin >> N >> T;

    vector<int> st(N);

    for (int i = 0; i < N; i++)
        cin >> st[i];

    int hi = N+1;
    int lo = 1;

    while(lo < hi){
        int mid = lo + (hi-lo)/2;
        if(simulate(mid, st, N, T))
            hi = mid;
        else
            lo = mid + 1;
    }
    cout << lo << endl;

    return 0;
}
```

Python

```python
import random
import sys
import heapq

def simulate(k, t, N, T):
    time = 0
    pq = []
    for i in range(k):
        heapq.heappush(pq, t[i])
    i = k
    while i < N:
        time = pq[0]
        heapq.heappop(pq)
        heapq.heappush(pq, (t[i]+time))
        i += 1
    return max(pq) <= T

N, T = map(int, input().split())
t = list(map(int, input().split()))

st = t[:]

hi, lo = N+1, 1
while lo < hi:
    mid = lo + (hi-lo)//2
    if simulate(mid, st, N, T):
        hi = mid
    else:
        lo = mid + 1
print(lo)
```

4 Problem 4 - "The team selection" [100 points]

4.1 Problem Statement

It's August 2023. CyberChallenge.IT has come to an end, with the national champions' title assigned with it. So everything is done for this year, right? No! There is one last step: the European Cyber Security Challenge (ECSC). At ECSC, every European nation sends its best M hackers to battle in an intense 2-days long full-immersion of CTFing, with of course Italy being one of them!

In particular, *Team Italy*'s trainers need to choose the best M-people team among N candidates.

Every candidate is described by a unique id (from 1 to N) and a set of S skills, each with an associated *skill name* (in the form of a 3-characters long string, with only uppercase letters) and a non-negative integer *skill score* (from 1 to 100), stating how strong a candidate is in that specific skill.

The trainers studied a lot the competition format, and established an optimal set of (possibly repeated) skills that the team should satisfy to guarantee the best possible performance. Each of the 10 selected participants will be assigned one of these skills as their *role* inside the team.

The global score of the team is the sum of the skill scores of its members in the role they got assigned to. Other skills of each members do not count in the global score.

Your task is to determine the maximum possible global score for Team Italy, given the list of candidates.

Note: it is possible to assign a player to a role that is not listed in their skills. In that case, that player will count as 0 in the global score.

4.2 Problem Details

Input

Each test contains multiple testcases. The first line contains the number of test cases T. Description of the test cases follows.

The input consists of $2 + N(S + 1)$ lines:

- Line 1: the numbers N, M and S, separated by a space.
- Line 2: the optimal set of skills required by the trainers, as a list of M space-separated skill names.
- Lines $3, \ldots, 2 + N(S + 1)$: every group of $S + 1$ lines is formatted as follows:
  - Line 1: the unique id of the candidate.
  - Lines $2, \ldots, S + 1$: one skill name and the corresponding skill score, separated by a space.

Output

The output must contain T positive integers, one for each line, that represent the maximum global score that can be achieved with the available candidates for each testcase.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all its testcases.

- **Subtask 1** [**30 points**]: $10 \leq N \leq 100$, $M = 10$, $S = 1$, $T = 1$.
- **Subtask 2** [**30 points**]: $10 \leq N \leq 100$, $M = 10$, $1 \leq S \leq 10$, $1 \leq T \leq 10$.
- **Subtask 3** [**40 points**]: $12 \leq N \leq 1000$, $M = 12$, $1 \leq S \leq 10$, $1 \leq T \leq 10$.

Examples

INPUT	OUTPUT
1 14 10 1 CRY MOB FOR MOB MOB WEB MSC WEB CRY WEB 1 WEB 98 2 FOR 14 3 MSC 82 4 MSC 9 5 OSI 90 6 FOR 52 7 MSC 95 8 NET 85 9 REV 46 10 CRY 16 11 MOB 32 12 PWN 41 13 CRY 59 14 CRY 34	370

Explanation

The example asks for 2 crypto players (`CRY`, pun intended), 3 mobile players (`MOB`), one forensics (`FOR`), 3 web (`WEB`) and 1 misc (`MSC`). In this case, we can simply pick the top players for each category to fill the team. Since there are only one mobile and one web players in the candidates, we fill the other two spots with random remaining players, counting as 0 in the global score.

4.3 Solution

This is an algorithmic task. The first thing to notice is that M is very small. This suggests that we can iterate through all subsets of M-sized sets. The strategy that we propose is a bitmask-based dynamic programming one then.

We denote by `dp[i][m]` the score of the best possible team if only players numbered $1, 2, ..., i$ are used, and only roles that are represented by 1 in the binary expansion of m are assigned. For example, if $m = 5$, we have that (considering $M = 10$) m in binary is written as 0000000101, so `dp[i][5]` considers the team using only the last and second to last role. It is clear that m will range from 0 to $2^M - 1$.

Now the only remaining thing is to populate the `dp` array iteratively, starting from `dp[0][0] = 0` and iterating through i, from 0 to N, and m, from 0 to $2^M - 1$.

The value of `dp[i][m]` can be the same of `dp[i-1][m]`, if the i-th player is not worth to put in any role, or can be the result of substituting the player in a specific role, if this increases the score. The final result will be of course `dp[N][`$2^M - 1$`]`.

4.4 Source Code

C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int T;
    cin >> T;

    while(T--){
        int N,M,S;
        cin >> N >> M >> S;

        vector<string> skills;
        vector<unordered_map<string, int>> players;

        for(int i = 0; i < M; i++){
            string x;
            cin >> x;
            skills.push_back(x);
        }

        for(int i = 0; i < N; i++){
            unordered_map<string, int> player;
            int id;
            cin >> id;

            for(int j = 0; j < S; j++){
                string skill;
                int pts;
                cin >> skill >> pts;
                player[skill] = pts;
            }
            players.push_back(player);
        }

        vector<vector<int>> dp(N+1, vector<int>((1<<M), -1));

        dp[0][0] = 0;

        for(int i = 1; i < N+1; i++){
            for(int j = 0; j < (1<<M); j++){
                if(dp[i-1][j] != -1) dp[i][j] = dp[i-1][j];
                for(int k = 0; k < M; k++){
                    if((j&(1<<k)) && (dp[i-1][(j ^ (1<<k))] != -1)){
                        dp[i][j] = max(dp[i][j], dp[i-1][j ^ (1<<k)] + players[i-1][skills[k]]);
                    }
                }
            }
        }
        cout << dp[N][(1<<M)-1] << endl;
    }

        return 0;
```

```
}
```

Python

```python
#!/bin/env python3

from collections import defaultdict
import sys

def solve(N, S, M, required_skills_list, players):
    dp = [[-1 for _ in range(pow(2, M))] for __ in range(N+1)]
    dp[0][0] = 0

    for i in range(1, N+1):
        for j in range(pow(2, M)):
            if dp[i-1][j] != -1:
                dp[i][j] = dp[i-1][j]
            for k in range(M):
                if j & (1 << k) and dp[i-1][j ^ (1 << k)] != -1:
                    dp[i][j] = max(dp[i][j], dp[i-1][j ^ (1 << k)] +
                                   players[i][required_skills_list[k]])

    return dp[N][-1]


T = int(fin.readline().strip())

for _ in range(T):
    N, M, S = list(map(int, fin.readline().strip().split()))
    required_skills_list = list(map(str, fin.readline().strip().split()))
    players = {}
    for i in range(N):
        players[i+1] = defaultdict(int)
        inutile = fin.readline()
        for j in range(S):
            skill, level = fin.readline().strip().split()
            level = int(level)
            players[i+1][skill] = level

    print(solve(N, S, M, required_skills_list, players))
```