Problem Solving (C43)

هذا البرنامج التدريبي مُصاغ بعناية لتمكين المتدربين من تطوير قدراتهم الفكرية على غرار المبرمجين المحترفين، والتعاون بكفاءة ضمن فريق محترف في شركة "جمال تك" أو أي مؤسسة متعددة الجنسيات أخرى. نظرًا لأهمية اللغة الإنجليزية في بيئة العمل العالمية، يتم تقديم المحتوى التدريبي بالإنجليزية. لا يشترط إتقان اللغة بشكل كامل، لكن من الضروري امتلاك القدرة الكافية لفهم المتطلبات وتنفيذها بشكل فعّال. يُمكن للمتدربين استخدام مترجم جوجل أو الاستعانة بـ "شات جي بي تي" للتغلب على أية عقبات لغوية، المهم هو الفهم الدقيق للمطلوب وتحقيقه بنجاح.

لتعظيم الاستفادة من التدريب، يُنصح بمحاولة حل التمارين بشكل مستقل لمدة ساعة واحدة على الأقل قبل الرجوع إلى الحل المرفق في نهاية الملف.

قد يتضمن الحل كودًا برمجيًا غير مفسر بعد، والغرض من ذلك هو تشجيعك على محاولة فهم الأكواد البرمجية الجديدة التي لم تتعرض لها من قبل. هذه المهارة ضرورية في سوق العمل، حيث تتطور لغات البرمجة باستمرار ويظهر كل يوم لغات جديدة. ستواجه دائمًا أكوادًا لم تدرسها من قبل، ومن المهم أن تكون قادرًا على فهمها بنفسك دون الحاجة إلى دراسة مسبقة. يمكنك الاستعانة بمحرك البحث جوجل، أو استخدام ChatGPT، أو حتى اللجوء لأصدقائك للمساعدة. الهدف الأساسي هو أن تصل إلى فهم معنى كل كود بأي طريقة ممكنة لتتمكن من إيجاد موقعك في سوق العمل.

إن وجود كود برمجي غير مفسر يشكل تحديًا يتوجب عليك إيجاد حل له. هذا النوع من التدريبات يعد جزءًا أساسيًا من تدريبات 'Problem Solving'، التي تهدف إلى تمكينك من أداء عملك بفاعلية بغض النظر عن التحديات والعقبات. هذه القدرة على حل المشكلات هي ما يتمتع به العاملون في 'جمال تك'، ومن الضروري أن تطور في نفسك هذه المهارة لتصبح عضوًا فعالًا في فريق عمل 'جمال تك'.

Gammal Tech's Innovative Scheduler

Background

Gammal Tech, a pioneer in software development, is renowned for its cutting-edge office facilities and a work culture that promotes innovation. Their latest venture is to tackle a critical industry issue: optimizing the scheduling of software deployment across various server clusters.

Problem Statement

Gammal Tech's servers are divided into multiple clusters, each capable of handling a specific number of software deployments per day. Due to recent expansions, the company needs an efficient way to schedule these deployments, ensuring maximum utilization of each cluster while avoiding overloading any single one.

Your Task

You are tasked with designing a system that allocates deployment tasks to server clusters. Each cluster has a limit to the number of deployments it can handle per day.

Your program should efficiently allocate these deployments across all clusters, maximizing the total deployments while ensuring no cluster is over-utilized.

Input Format

- The first line contains two integers `N` and `M` – the number of clusters and the number of deployment requests, respectively.
- The second line contains `N` integers, each representing the maximum deployments a cluster can handle.
- The third line contains `M` integers, each representing a deployment task needing allocation.

Output Format

Print the maximum number of deployments that can be scheduled without overloading any cluster. If it is not possible to schedule all deployments, print `-1`.

Constraints

- 1 ≤ `N`, `M` ≤ 1000
- 1 ≤ Deployments per cluster ≤ 10000
- 1 ≤ Deployment task ≤ 10000

Sample Input:

```
3 5
4 2 3
2 2 1 3 2
```

Sample Output:

```
5
```

Explanation

All deployment tasks can be scheduled across the clusters without exceeding their limits.

لتحقيق أقصى فائدة من التدريب، يُوصى ببذل محاولة مستقلة لحل التمارين لمدة لا تقل عن ساعة واحدة. تجنب الاطلاع على بقية الملف حتى تكمل عملية التفكير في الحل. بعد ذلك، جرب حلك بنفسك على المدخلات الموضحة. إذا واجهت مدخلات لم تتوقعها، فهذا يعد فرصة لتطوير مهارة جديدة ضمن مسيرتك التعليمية. المهندس المحترف يجب أن يضمن أن برنامجه يعمل مع جميع أنواع المدخلات، وهذه مهارة يتم تطويرها عبر التجربة والخطأ. لذا، من الضروري ألا تطلع على المدخلات المتوقعة قبل أن تجرب الحل بنفسك. هذه هي الطريقة الأمثل لتنمية هذه المهارة.

بعد اختبار المدخلات المقترحة، إذا كانت النتائج تختلف عما هو مدون في الملف، فيُنصح بمحاولة حل التمرين مرة أخرى لمدة ساعة على الأقل قبل الرجوع إلى الحل الموجود في نهاية الملف.

Test Case 1: Minimum Input Values

Input

```
1 1
1
1
```

Expected Output

```
1
```

Explanation

There is only one cluster and one deployment, which can be scheduled successfully.

Test Case 2: Clusters with Varied Capacities

Input

```
4 3
5 3 4 2
2 3 4
```

Expected Output

```
3
```

Explanation

- The first task (2 deployments) can be allocated to the second cluster (capacity 3).
- The second task (3 deployments) can be allocated to the first cluster (capacity 5).
- The third task (4 deployments) can be allocated to the third cluster (capacity 4).

Each cluster is utilized without exceeding its capacity, and all tasks are scheduled successfully.

Test Case 3: Insufficient Cluster Capacity

Input

```
3 4
2 2 2
3 3 3 3
```

Expected Output

```
-1
```

Explanation

None of the clusters can handle a deployment of size 3, so the tasks cannot be scheduled.

Test Case 4: Single Large Deployment

Input

```
5 1
1 1 1 1 10
9
```

Expected Output

```
1
```

Explanation

Only the last cluster can handle the large deployment task.

Test Case 5: More Tasks Than Clusters

Input

```
2 5
3 4
1 2 1 2 1
```

Expected Output

```
5
```

Explanation

All tasks can be distributed across the two clusters without exceeding their capacity.

لتحقيق أقصى استفادة من التدريب، من المستحسن أن تخصص وقتًا إضافيًا - لا يقل عن ساعة - لمحاولة حل التمرين مرة أخرى بمفردك قبل الرجوع إلى الحل المقترح. هذه العملية المتكررة من التجربة والخطأ تعتبر استراتيجية فعالة في تعزيز مهاراتك البرمجية وتعميق فهمك للمفاهيم. تذكر أن التحدي والمثابرة هما المفتاحان للتطور في مجال البرمجة.

C Programming Solution:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int capacity;
    int used;
} Cluster;

int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

int main() {
    int N, M, i, j, totalDeployments = 0;
    scanf("%d %d", &N, &M);
    Cluster clusters[N];
    int tasks[M];

    for (i = 0; i < N; i++) {
        scanf("%d", &clusters[i].capacity);
        clusters[i].used = 0;
    }

    for (i = 0; i < M; i++) {
        scanf("%d", &tasks[i]);
    }

    qsort(tasks, M, sizeof(int), compare);

    for (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            if (clusters[j].used + tasks[i] <= clusters[j].capacity) {
                clusters[j].used += tasks[i];
                totalDeployments++;
                break;
            }
        }
        if (j == N) {
            printf("-1\n");
            return 0;
        }
    }

    printf("%d\n", totalDeployments);
    return 0;
}
```

Notes

- The code uses a `typedef` structure `Cluster` to manage each cluster's capacity and current usage.
- Tasks are sorted to try fitting smaller deployments first, ensuring efficient utilization of cluster capacities.
- The program iterates through each task and tries to fit it in a cluster without exceeding its capacity. If a task cannot be allocated, it prints `-1`.