|  |
| --- |
| INPT |
| Les journées Nationales des Jeunes Développeurs |
| Finale interne de INPT |
|  |
| **BOUZKRAOUI Hicham, ZARHRI Badr** |
| **17/03/2012** |

|  |
| --- |
| Ce concours servira pour sélectionner les équipes qui vont représenter INPT dans la finale nationale. |





## Problème A

aire.(c|cpp|java|in|out)

Problème :

Il s’agit de calculer l’aire d’un polygone convexe (s’il contient deux points A et B, alors il contient le segment [AB]).

Spécification de l’entrée :

Le fichier d’entrée contient plusieurs cas de teste, chaque cas représente un polygone.

Chaque cas est composé d’un entier **3≤n≤10** (le nombre de sommets du polygone), suivi de 2\*n entiers **-100≤xi,yi≤100** (les coordonnées de ses sommets x1, y1, x2, y2 … xn, yn). Les coordonnées sont présentées dans l’entrée selon le sens contraire des aiguilles d’une montre.

Spécification de la sortie :

Pour chaque cas, on affichera l’aire du polygone avec la précision de deux chiffres après la virgule.

|  |  |
| --- | --- |
| Exemple d’entrée : | Exemple de sortie : |
| 2  4  0 0 0 1 1 1 1 0  3  0 0 0 1 1 1 | 1.00  1.50 |

## Problème B

bidon.(c|cpp|java|in|out)

Problème :

bidon.exe est le fruit d’un projet de fin d’année. Ce programme permet d’accéder à des répertoires en se déplaçant à travers les raccourcis dans répertoire en cours. Il accepte comme paramètre de lancement le répertoire de départ.

Considérons l’exemple suivant : Si nous avons 3 répertoires nommés respectivement rep1, rep2, et rep3. Et si on crée un raccourci de rep2 dans rep1, et on lance le programme avec comme paramètre rep1, alors on peut se déplacer du répertoire rep1 vers rep2. De même si on ajoute un autre raccourci de rep1 dans rep2, alors on peut revenir dans rep1. Par contre rep3 reste injoignable, et il faudra arrêter le programme et le relancer avec comme paramètre rep3 pour pouvoir y accéder.

Avant de lancer notre programme, on relie les répertoires entre eux (créer un raccourci de chacun dans l’autre). L’annulation d’une liaison supprime les deux raccourcis créés.

Votre tâche est de trouver le nombre minimum de lancements de bidon.exe pour visiter **n** **(2≤n≤100000)** répertoires.

Spécification de l’entrée :

Le fichier d’entrée contient plusieurs cas de teste :

Chaque cas de teste commence par deux entiers : n (le nombre de répertoires qu’on désire visiter) et m **(1≤m≤100000)** (le nombre de liaisons effectués), les m lignes suivantes représentent les opérations de liaison effectués de la forme : « r1 r2 » **(1≤r1,r2≤n)**, où r1 er r2 sont les numéros des répertoires à relier. La ligne suivante contient un entier k **(1≤k≤m)**, qui est le nombre de liaisons annulées, suivi de k entiers représentant les indices des liaisons annulées, démarrant de 1.

Spécification de la sortie :

Pour chaque cas on affichera k entiers dans une ligne représentant le nombre de lancement de bidon.exe qu’il faut pour parcourir tous les répertoires. (le ième entier représente le nombre de lancements qu’il faut après la ième suppression d’une liaison).

|  |  |
| --- | --- |
| Exemple d’entrée : | Exemple de sortie : |
| 4 3  1 2  2 1  3 1  3  1 2 3 | 2 3 4 |

## Problème C

choix.(c|cpp|java|in|out)

Problème :

INPT a décidé de rendre hommage au scientifique qui a été les plus bénéfiques au monde.

Pour choisir cette personne, on a d’abord pensé à les classer par ordre d’importance de leurs inventions, mais on n’a pas pu le faire, car il n’y a pas de règle pour décider qu’une invention est plus importante qu’une autre. On a donc décidé de choisir le scientifique qui a vécu le plus longtemps.

Etant donné une liste de personnes, avec leurs dates de naissance, et dates de décès, vous devez afficher le nom de la personne qui a vécu la plus longue durée. S’il y en a plus qu’un, affichez le premier par ordre alphabétique (de ceux qui ont vécu cette durée).

Spécification de l’entrée :

Le fichier d’entrée contient plusieurs cas de teste :

Chaque cas commence par un entier **n≤100** (le nombre de savants), suivi de n lignes contenant chacune une chaine de caractères de longueur ≤ 20, qui est le nom du scientifique, suivie de deux dates, qui sont respectivement les dates de naissance et de décès.

Les dates sont données dans le format : **YYYY/MM/DD**, (exemple aujourd’hui : 2012/03/17).

Spécification de la sortie :

Pour chaque cas on affichera une chaine de caractères, qui est le premier nom par ordre alphabétique de ceux qui ont vécu la plus longue durée.

|  |  |
| --- | --- |
| Exemple d’entrée : | Exemple de sortie : |
| 10 10 | 120.71 |

## Problème D

ducobu.(c|cpp|java|in|out)

Problème :

Après ses notes catastrophiques en premier semestre, Ducobu a décidé de s’y mettre à fond et de travailler dur cette fois. Pour commencer, il a décidé de commencer par les mathématiques, et surtout par l’addition. Il a décidé de créer un programme qui ferait les devoirs d’addition comme s’ils étaient faits à la main ☺.

Votre tâche est d’écrire le même programme que Ducobu, et on va comparer votre fichier de sortie avec celui de Ducobu pour décider s’il est correct ou pas.

Etant donné deux entiers a et b, rédigez l’opération comme vous l’avez apprise au primaire.

Spécification de l’entrée :

Le fichier d’entrée contient plusieurs cas de teste.

Chaque cas est composé de deux entiers a et b (0≤a,b≤1020).

Spécification de la sortie :

Pour chaque cas, l’opération d’addition comme montré dans l’exemple.

La première ligne contient le nombre a justifié par des espaces à gauche si c’est nécessaire.

La deuxième contient le symbole ‘+’ et l’entier b. Le symbole + doit être à l’extrême gauche, et les chiffres du nombre b doivent se situer en dessous de ceux de a (les unaires en dessous des unaires, les dizaines en dessous des dizaines …).

La troisième ligne doit contenir des caractères ‘-‘, jusqu’au dernier chiffre à droite (unaires).

La 4ème ligne doit contenir le caractère ‘=’, et le nombre a+b. le symbole ‘=’ doit se situer dans la même colonne que le symbole ‘+’, et les chiffres du nombre a+b, dans les mêmes colonnes que ceux de a et de b. (les unaires en dessous des unaires, les dizaines en dessous des dizaines …).

|  |  |
| --- | --- |
| Exemple d’entrée : | Exemple de sortie : |
| 15 3  10 0 | 15  + 3  ---  =18  10  + 0  ---  =10 |

## Problème E

e.(c|cpp|java|in|out)

Problème :

Considérons les deux manières suivantes de représenter un Bitmap rectangulaire :

La première, notée ‘**B’,** consiste en un tableau à deux dimensions de 1 et de 0.

La deuxième, notée **‘D’**, est basée sur une technique de décomposition : On prend d’abord toute le bitmap, ensuite, si tous les bits sont égaux à 0, alors la sortie est 0, et si tous les bits sont égaux à 1, la sortie est 1. Si la sortie n’est ni 1 ni 0, alors on affiche D et on procède de la même manière pour les 4 quarts de la matrice initiale.

Rq : Si le nombre de lignes est impair, les quarts d’en haut contiennent 1 ligne de plus que ceux du bas. Si le nombre de colonnes est impair, alors les quarts de gauche contiennent une colonne de plus de ceux de droite. Les 4 quarts sont traités dans l’ordre suivant : Haut-Gauche, Haut-Droite, Bas-Gauche et Bas-Droite. Si la matrice est vide, alors on ne retourne rien.

Spécification de l’entrée :

Le fichier d’entrée consistera en une série de bitMaps. Chaque bitMaps commence avec une ligne donnant son format (B ou D) et ses dimensions n et m (**1≤n,m≤200**) séparés par des espaces.

Les lignes suivantes contiennent la séquence de « 1 , 0 et D » caractères qui représentent la carte de bits, sans espaces.

Chaque ligne (sauf la dernière, qui peut être plus courte) contiendra exactement 50 caractères. Le type « B » bitmap sera décrit de gauche à droite, de haut en bas. Le fichier se termine par une ligne constituée d'un seul #.

Spécification de la sortie :

Pour chaque Bitmap dans l’entrée, afficher sa représentation dans l’autre format, comme expliqué pour le fichier d’entrée :

(D->B, et B-> D).

|  |  |
| --- | --- |
| Exemple d’entrée : | Exemple de sortie : |
| B 3 4  001000011011  D 2 3  DD10111  # | D 3 4  D0D1001D101  B 2 3  101111 |

## Problème F

f.(c|cpp|java|in|out)

Problème :

Une manière très populaire de crypter mes messages, est de permuter l’ordre des caractères.

Par exemple, si on applique la permutation « 21 » à « abc » on aurait « ba?c ».

Votre tâche dans ce problème est, étant donné un « message\_claire », « message \_crypté1 » et « message \_crypté2 », vérifiez si « message \_crypté1 » a été généré depuis « message \_claire » par une permutation. Et si c’est le cas, décrypter « message \_crypté2 ».

Spécification de l’entrée :

Le fichier d’entrée contient plusieurs cas de teste, chaque cas est composé de 3 lignes contenant chacune un message.

La deuxième ligne est le cryptage de la première, donc elles sont de même longueur.

Le fichier d’entrée se termine par une ligne contenant un seul caractère ‘#’.

Spécification de la sortie :

Pour chaque cas, afficher le décryptage de la troisième ligne si la deuxième est une permutation de la première, et afficher la troisième ligne en claire dans le cas contraire.

S’il y a plusieurs permutations possibles, alors choisissez la plus courte.

Si la longueur de la troisième ligne n’est pas multiple de k (k= longueur de la permutation), alors ajouter des caractères ‘ ?’.

On vous garantit que le fichier d’entrée n’admet pas plus d’une permutation de longueur minimale.

|  |  |
| --- | --- |
| Exemple d’entrée : | Exemple de sortie : |
| Mary had a little lamb!!  aMyrh daa l tilt ealbm!!  hTsii s aetts  Foobar  blargg  No cycle  abc  bca  abcd  # | This is a test  No cycle  cab?d? |