Prova-04

Prof. Msc. Elias Batista Ferreira
Prof. Dr. Gustavo Teodoro Laureano
Profa. Dra. Luciana Berretta
Prof. Dr. Thierson Rosa Couto

Sumário

1 Rede de Contatos no Twitter (+++)

 (+++)

Um determinado pesquisador precisa representar em uma tabela informações sobre relacionamentos entre um conjunto $\mathscr{U} = \{u_1, u_2, \ldots u_n\}$ de usuários do Twitter. Para isso, ele quer guardar para cada par $(u_1, u_2), u_1, u_2 \in \mathscr{U}$ as seguintes informações, quando existirem:

- O número de *likes* que u_1 fez em *tweets* escritos por u_2.
- O número de *retweets* que u_1 fez em fez em *tweets* escritos por u_2.
- O número de *menções* que u_1 fez em *tweets* escritos por u_2.

Par isso, o pesquisador quer criar uma matriz quadrada $n \times n$ que permita armazenar para cada par u_i, u_j, $1 \leq i, j \leq n$ as informações acima caso elas existam. Ele pensou em criar uma matriz de *structs* onde cada *structs* têm três campos correspondentes aos totais para cada um dos números de interações descritos acima. O problema dessa representação é que o pesquisador sabe que para um grande número de pares de usuários não existe nenhum tipo de interação entre eles, ou seja, os totais para os três tipos de interação seriam iguais a zero. Além disso, há vários casos em que u_1 interage com u_2, mas u_2 não interagem com u_1. Armazenar todas as $n \times n$ *structs* na tabela é, portanto, um desperdício de memória e n pode ser muito grande impossibilitando o programa de funcionar. O pesquisador quer que você faça um programa em que a tabela $n \times n$ seja seja uma tabela de ponteiro para as *structs* que possuem os três campos comentados anteriormente e que aloque as structs sob demanda, somente quando forem necessárias. O programa deve ler os dados dos relacionamentos entre usuários e imprimir para cada usuário o total de de *likes*, o total de de *retweets* e o total de *menções* que ele fez a *tweets* de outros usuários.

Entrada

A primeira linha da entrada é constituída por um único inteiro positivo $N(N \leq 1000)$, o qual corresponde à dimensão da matriz de relacionamentos. A segunda linha contém outro número inteiro M, $1 \leq M \leq N^2$ que corresponde ao número de pares não nulos da matriz. Em seguida há M linhas, cada uma com o seguinte formato: u_1 u_2 *num_likes num_retweets num_menções*, onde os campos estão separados entre si por um espaço. O primeiro campo u_1 corresponde ao usuário que fez as interações. O segundo campo (u_2) corresponde ao usuário que recebe as interações. Os demais três campos correspondem a, respectivamente, o número de likes, o número de retweets e o número de menções que u_1 fez em tweets de u_2.

Saída

O programa deve imprimir uma linha para cada usuário contendo as seguintes informações: Usuario u_i - num. likes feitos: x, num. retweets feitos: y e num. mencoes feitos: z. Os valores de x, y e z correspondem ao totais de likes, retweets e menções, respectivamente, feitos pelo usuário u_i. Os usuários devem ser impressos na ordem crescente do seu número (ordem das linhas da tabela).

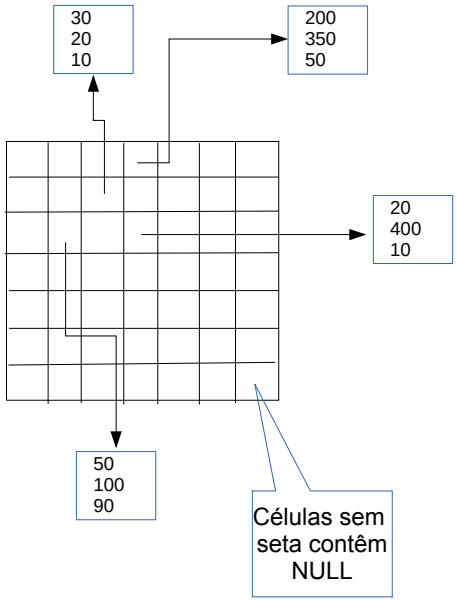
Observações

1. Os compiladores da linguagem C não inicializam os campos de uma tabela. Você deve garantir que antes da leitura todos as células da tabela tenham NULL como valor de ponteiro.
2. Ao final do programa, todo espaço alocado dinamicamente deve ser liberado
3. Uso da função malloc() : x = (Tipo_do_x*) malloc(Num_bytes_tipo_x * quantidade_elem). A função malloc() retorna NULL caso não consiga alocar epaço. Seu programa deve verificar se foi possível alocar espaço e terminar o programa em caso contrário.
4. Deve ser feito #include<stdlib.h> para usar a função malloc().

5. A função exit(1) encerra o programa.
6. A função free(x) libera a área de memória cujo endereço está em x.
7. dada uma matriz mat, para acessar o campo num_likes da struct cujo endereço está em mat[i][j]: (*mat[i][j]).num_likes. Exemplo scanf("%d", &((*mat[i][j]).num_likes)).

Exemplo

Entrada:
7
4
0 3 200 350 50
1 2 30 20 10
2 1 50 100 90
2 3 20 400 10



Saída:
Usuario 0 - num. likes: 200, num retweets: 350 e num. mencoes: 50
Usuario 1 - num. likes: 30, num retweets: 20 e num. mencoes: 10
Usuario 2 - num. likes: 70, num retweets: 500 e num. mencoes: 100

2 Polinômios (++)

 (++)

Faça um programa que implemente a leitura e a soma, subtração e multiplicação de uma sequência de polinômios de qualquer ordem. Neste exercício você deverá usar a estrutura Poly, disponível no código abaixo, para armazenar um polinômio. Nessa estrutura, o atributo `ordem` representa a maior ordem do polinônio e o vetor `coef` representa os coeficientes do polinômio. Os coeficientes são armazenados de modo que sua potência é o seu índice correspondente. Por exemplo, a representação do polinônio $2x^3 - 1x^2 + 1$ é: `ordem=3` e `coef={1,0,-1,2}`.

```
typedef struct {
   int ordem;      // Ordem do polinomio
   double * coef; // Coeficientes. Cada indice representa a potência do coeficiente.
} Poly;
```

Neste exercício, a impressão de um polinômio segue o seguinte padrão: $s_c c_p \hat{} p_c$, onde s_c é o sinal do coeficiente, c_p é o coeficiente da potência p e p_c é a potência do coeficiente c. Desse modo, o polinômio dado como exemplo no parágrafo anterior seria impresso como: `+2.0x^3-1.0x^2+0.0x^1+1.0x^0`. Note que deve ser usada somente uma casa decimal.

Você deverá implementar as funções faltantes no código abaixo.

```
typedef struct {
   int ordem;      // Ordem do polinomio
   double * coef; // Vetor de coeficientes
} Poly;

/**
* Funcao que cria um polinomio com alocacao dinamica
* @param Poly* Ponteiro para o novo polinomio
*/
Poly * poly_new(int ord);

/**
* Funcao que imprime um polinomio na tela
* @param p Ponteiro para o polinomio
*/
void poly_print(Poly * p);
/**
* Funcao que libera a memoria alocada a um polinomio
* @param p Ponteiro para o polinomio
*/
void poly_free(Poly * p);

/**
* Cria o polinomio resultante da soma
* @param A Ponteiro para o primeiro polinomio
* @param B Ponteiro para o segundo polinomio
* @return Poly* <- A + B
*/
Poly * poly_sum( Poly * A, Poly * B );

/**
* Cria o polinomio resultante da subtracao
* @param A Ponteiro para o primeiro polinomio
* @param B Ponteiro para o segundo polinomio
* @return Poly* <- A - B
*/
Poly * poly_sub( Poly * A, Poly * B );

/**
```

```
40 * Cria o polinomio resultante da multiplicacao
41 * @param A Ponteiro para o primeiro polinomio
42 * @param B Ponteiro para o segundo polinomio
43 * @return Poly* <- A * B
44 */
45 Poly * poly_mult( Poly * A, Poly * B );
46
47 int main() {
48    Poly **P;    // Vetor de polinomios
49    int n;    // Quantidade de casos
50    // Demais declaracoes
51    // ...
52
53    scanf("%d", &n);  // Definicao da quantidade de polinomios
54
55    // Controle o laco de repeticao
56    // Execute n repeticoes
57
58    // Demais instrucoes
59
60    return 0;
61 }
```

Entrada

Seu programa deve ler um inteiro correspondente à quantidade de polinômios a serem lidos. Em seguida, para cada polinômio da sequência, deverá ler a ordem seguido dos ses coeficientes.

Saída

O programa deve apresantar, para cada par de polinômios os três polinômios resultantes da soma, subtração e multiplicação. Os pares são formados sempre pelos polinômios de índice i e $i+1$, ou seja, o primeiro forma par com o segundo, o segundo com o terceiro e assim por diante.

Exemplo