Universidade do Estado de Santa Catarina (UDESC)
Curso: Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Estruturas de Dados (EDA0001)
Prof. Rui J. Tramontin Jr.

Trabalho 3: Conjuntos Disjuntos

Em ciência da computação, uma estrutura de dados ***união-busca***[1], também chamada de estrutura de dados *disjoint-set*, é uma estrutura de dados que mantém o controle de um conjunto de elementos particionados em ***subconjuntos disjuntos*** (ou seja, que não possuem valores em comum).

Há duas operações importantes nesse tipo de estrutura. A primeira consiste em *encontrar* em que subconjunto um dado elemento pertence. A outra é a *união* de dois subconjuntos. O foco deste trabalho é a implementação de um TAD (Tipo Abstrato de Dados) que implemente tal estrutura e suas operações, bem como sua aplicação na área de grafos, conforme detalhado mais adiante.

Uma estrutura ***união-busca*** pode ser implementada como uma ***multilista***, ou seja, uma lista de listas. Cada elemento da lista principal aponta para o descritor de outra lista, que neste caso é um dos conjuntos. Por fim, cada elemento das listas secundárias representam os membros propriamente ditos. Considere neste trabalho que os membros dos conjuntos são valores inteiros.

Cada conjunto é identificado por um *representante*, que é um de seus membros. Em algumas aplicações, o critério para a determinação do representante não é relevante. Em geral, pode-se utilizar o primeiro termo (que foi utilizado na criação do conjunto), ou mesmo o menor membro do conjunto.

Implementação do TAD

Sendo ***u*** a estrutura *união-busca* e ***x*** um membro do conjunto (portanto, um valor inteiro), o TAD deve implementar as três funções básicas:

1) **Cria_Conjunto (*u* , *x*)** : cria um novo conjunto cujo único membro (e, portanto, seu representante) é *x*. Já que os conjuntos são disjuntos, *x* não pode pertencer a nenhum outro conjunto da estrutura. A dica aqui é usar a função de *busca* para garantir tal condição.
2) **União (*u* , *x* , *y*)** : faz a união dos conjuntos S_x e S_y, cujos representantes são, respectivamente, x e y. A união gera um novo conjunto e destrói os conjuntos originais. O representante desse novo conjunto pode ser qualquer membro de S_x ou S_y. Já que a implementação solicitada utiliza listas, a operação pode ser facilmente implementada através de um simples ajuste de ponteiros fazendo uma *concatenação* entre S_x e S_y (e posterior destruição de S_y).

[1] Tradução do termo em Inglês *union-find data structure*. Tal estrutura é também conhecida em Inglês como *merge-find set* ou *disjoint-set data structure*.

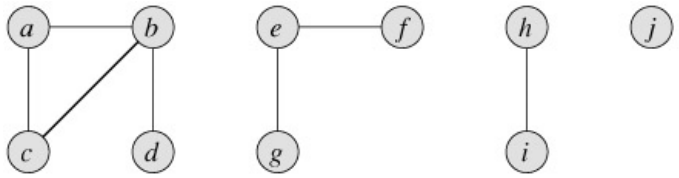
3) **Busca_Conjunto (*u* , *x*)** : retorna o índice do conjunto contendo x. Como a implementção é baseada em listas, saber o índice do conjunto na *multilista* torna simples o seu acesso, quando necessário.

Além dessas, outras operações gerais para TADs devem ser implementadas: *inicialização da estrutura, mostrar um dado conjunto (dado seu índice), mostrar todos os conjuntos, destruição de um conjunto (dado seu índice)* e *desalocação de toda a estrutura*.

Implementação da Aplicação

Um tipo de problema que pode ser resolvido através de conjuntos disjuntos é a **determinação dos componentes conexos de um grafo**. Em termos simples, um *componente conexo* é o conjunto de vértices (nós) conectados entre si.

O exemplo[2] da figura a seguir mostra quatro componentes conexos (a), e a sequência de operações de união que permitiram a identificação dos componentes conexos (b). Repare na última linha que os quatro componentes são definidos por conjuntos disjuntos.



(a)

Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

(b)

A aplicação deve ter como entrada um grafo (usando uma matriz de adjacência) e utilizar o TAD ***União-Busca*** para determinar os componentes conexos do grafo. O algoritmo é bem simples, conforme pode ser visto no pseudocódigo a seguir:

[2] Este exemplo, bem como a explicação para o enunciado do trabalho, foram retirados do livro *Introduction to Algorithms* (Thomas H. Cormen *et al.*), que pode ser encontrado na biblioteca.

```
 1. ENTRADA DE DADOS:
     - Matriz de adjacência (sua dimensão e seu conteúdo);
 2. INICIALIZA ESTRUTURA U;      // Estrutura União-busca.
 3. PARA CADA VÉRTICE X DO GRAFO FAÇA
 4.    CRIA_CONJUNTO( U, X );    // Cria conjunto contendo X.
 5. FIMPARA
 6. PARA CADA VÉRTICE X DO GRAFO FAÇA
 7.    PARA CADA VÉRTICE Y ADJACENTE A X FAÇA
 8.       SE X e Y ESTIVEREM EM CONJUNTOS DIFERENTES ENTÃO
 9.          UNIÃO( U, X , Y ); // União entre X e Y.
10.       FIMSE
11.    FIMPARA
12. FIMPARA
13. MOSTRA_CONJUNTOS( U );      // Mostra todos os conjuntos na tela.
```

Exemplo

Exemplo de entrada	Exemplo de saída
10 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	1 2 3 4 5 6 7 8 9 10

Critérios de avaliação

- Execução correta e alinhamento com o que foi solicitado neste enunciado;
- Uso apropriado das funções dos *tipos abstratos de dados*. Respeite o encapsulamento!

Informações importantes

- **Equipe:** 1 ou 2 alunos.
- **Entrega no Moodle**.