

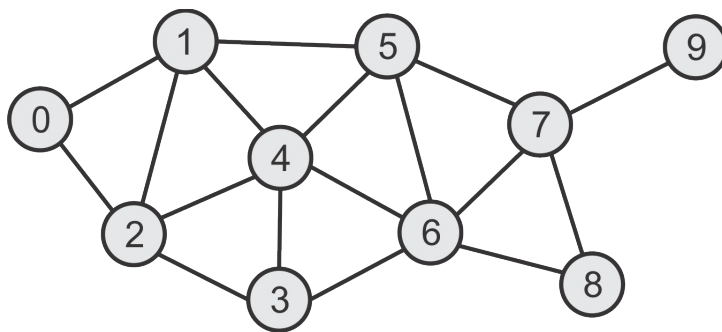


LABORATÓRIO DE PROGRAMAÇÃO AVANÇADA
QUARTO TRABALHO PRÁTICO
GRAFOS

OBJETIVO

Relembrar os conceitos básicos de estruturas de dados e ponteiros obtidos nas disciplinas de AED1 e AED2 através da implementação de um Grafo Não-Direcionado que represente uma rede sem fio. Neste grafo, os vértices terão as informações de um nó (dispositivo) da rede (`int id`; `double pos_x`, `double pos_y`) e, caso a distância entre dois nós seja menor que o raio de comunicação (i.e., os nós conseguem se comunicar), haverá uma aresta entre os dois vértices do grafo (i.e., eles serão vizinhos).

QUESTÃO ÚNICA



O Grafo acima representa uma rede sem fio formada por 10 nós com raio de comunicação de, por exemplo, 38m.

Grafos podem ser implementados (principalmente) de duas formas:

- (1) usando uma **matriz de adjacência** $N \times N$, onde N é a quantidade de vértices; ou
- (2) usando uma **lista de adjacência**, que é um vetor de tamanho N em que cada item possui uma lista de vizinhos.

Como no caso de uma rede sem fio, o grafo será relativamente **esparso** (o que significa que não haverá muitas arestas), a melhor representação é através da **lista de adjacência**. *Esta deverá ser a representação usada neste trabalho.*

Neste trabalho, você deverá implementar um **grafo** onde os **nós** serão os **dispositivos da rede sem fio**. Cada nó terá informações de `id` (`int`), `posição X` e `Y` (`double`) e a sua lista de vizinhos (lista encadeada de inteiros). Sua rede (o grafo) será um **vetor** destes nós.

As informações da rede (número de nós e raio de comunicação) e dos nós serão **todos passados por um arquivo** de entrada (`argv[1]`). A primeira linha deste arquivo terá o seguinte formato:

```
<número-de-nós>\t<raio-de-comunicação>\n
```

onde `<número-de-nós>` é a quantidade de nós na rede e `<raio-de-comunicação>` é o raio de comunicação entre os nós. Caso a distância física entre dois nós seja menor que este raio de comunicação, **eles podem se comunicar** e, portanto, existirá uma **aresta** entre eles.

As linhas seguintes do arquivo de entrada, terão o seguinte formato:

`<ID>\t<posição-X>\t<posição-Y>\n`

Onde `<ID>` é o id do dispositivo móvel (número da linha) e `<posição-X>` e `<posição-Y>` a posição física do nó. Para cada linha, adicione o nó atual no grafo. Depois de inserir todos os nós, execute uma função para **popular** (atualizar) a lista de vizinhos de todos os nós. Esta função, para cada nó (nó “i”), irá compará-lo com **todos os nós da rede** (nó “j”) e, caso a distância entre i e j seja menor que o raio de comunicação, adicione i à lista de vizinhos de j. Após ler todo o arquivo de entrada, para cada nó do grafo, liste os seus vizinhos como no formato especificado a seguir:

Exemplo de Entrada (arquivo_entrada.txt):

```
10 38.0
0 6.0 20.5
1 33.4 6.2
2 23.7 41.6
3 51.0 54.8
4 50.9 27.8
5 71.1 7.2
6 79.1 41.9
7 99.6 21.6
8 109.3 51.6
9 128.7 7.3
```

Saída Esperada (./grafo arquivo_entrada.txt):

```
NÓ 0: 2 1
NÓ 1: 5 4 2 0
NÓ 2: 4 3 1 0
NÓ 3: 6 4 2
NÓ 4: 6 5 3 2 1
NÓ 5: 7 6 4 1
NÓ 6: 8 7 5 4 3
NÓ 7: 9 8 6 5
NÓ 8: 7 6
NÓ 9: 7
```

DICAS

Sugestão de funções a serem implementadas:

```
bool lista_vizinhos_adicionar(int vizinho, lista_vizinhos_t **lista);
=> Adiciona um inteiro (id de um vizinho) ao início de uma lista encadeada
=> lista_vizinhos_t será uma lista de inteiros, portanto não há necessidade de criar um tipo “vizinho_t”.
```

```
void lista_vizinhos_imprimir(lista_vizinhos_t *lista);
=> Imprime os números (ids) da lista se vizinhos
```

```
grafo_t grafo_criar(int tam);
=> Aloca memória para um vetor de nós.
=> grafo_t pode ser definido como: typedef no_t* grafo_t;
```

```
void grafo_atualizar_vizinhos(int tam, double raio, grafo_t grafo)
=> Para cada nó (nó “i”) do vetor de nós do grafo, compará-lo com todos os outros nós da rede (nó “j”, sendo j != i) e, caso a distância entre i e j seja menor que o raio de comunicação, adicione i à lista de vizinhos de j.
```

=> Cálculo de distâncias Euclidianas:

```
sqrt(pow(grafo[i].x - grafo[j].x, 2) + pow(grafo[i].y - grafo[j].y, 2))
```

```
void grafo_imprimir(int tam, grafo_t grafo);  
=> Para cada posição do vetor, executa a função lista_vizinhos_imprimir.  
  
int main(int argc, char **argv);  
=> Abre o arquivo de entrada;  
=> Lê a primeira linha usando o fscanf com o formato "%d\t%lf\n" para ler a quantidade de nós e o  
raio de comunicação.  
=> Cria o grafo;  
=> Cria um contador i;  
=> Para cada linha do arquivo de entrada, lê os dados de id, x e y diretamente para o grafo[i] e  
faça com que grafo[i].lista_vizinhos = NULL; Incremente i.  
=> Atualiza os vizinhos;  
=> Imprime.
```

Este trabalho deve ser entregue no dia 23/05/2014. Entretanto, vou liberar a entrega até a data máxima de 26/05/2014 (segunda) até meia-noite.

Coloque no assunto: [LPAV-TP04].