DISCIPLINA: ESTRUTURA DE DADOS II

2018

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Implementação de Grafos

Prof. Luis Cuevas Rodríguez, PhD

EST
Escola Superior de Tecnologia da UEA

NUCOMP
NÚCLEO DE COMPUTAÇÃO

Grafos

- Representação de Grafos
- Criação de um grafo
- Incluir uma aresta no grafo
- Visualizar o grafo

Usando listas de adjacência

Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE
DO ESTADO DO
AMAZONAS

REPRESENTAÇÃO DE GRAFOS

Prof. Luis Cuevas Rodríguez, PhD

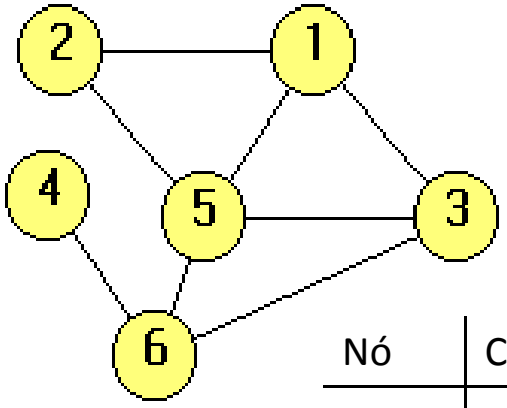
UEA
UNIVERSIDADE
DO ESTADO DO
AMAZONAS

Representação de Grafos

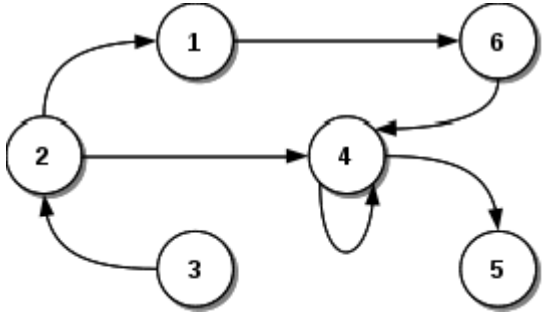
- Listas de adjacência: utiliza uma coleção de listas de adjacência.
  - Útil para representar *grafos esparsos*
    - $(|E| \ll \binom{V}{2})$
- Matrizes de adjacência:
  - Preferível quando o grafo é denso $(|E| \approx \binom{V}{2})$
  - Necessidade de conhecer com rapidez se uma aresta conecta dois vértices

Grafos

- Cada nó vai ter uma lista de nós aos quais ele esta conectado



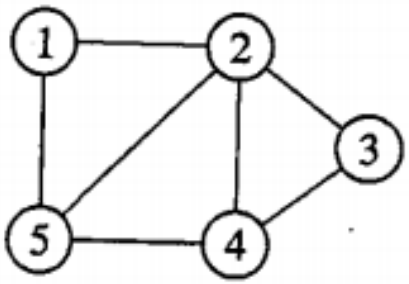
Nó	Conectado a
v1	v2, v5, v3
v2	v1, v5
v3	v1, v5, v6
v4	v6
v5	v1, v2, v3, v6
v6	v3, v4, v5

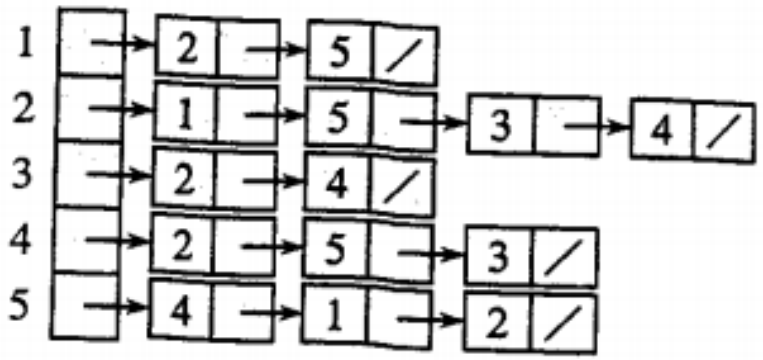


Nó	Conectado a
v1	V6
v2	v1, v4
v3	v2
v4	v4, v5,
v5	
v6	v4

Grafos – Lista de Adjacência

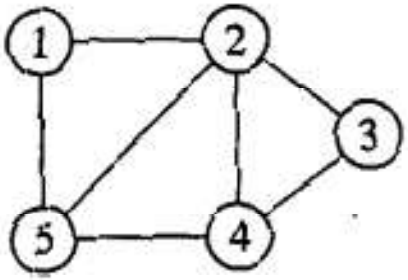
- Um arranjo de |v| listas, uma para cada vértice.
- A lista de adjacência de u∈V contem ponteiros para todos os vértices v tais que existe uma aresta (u,v).
- Vértices ordenados de forma arbitraria.

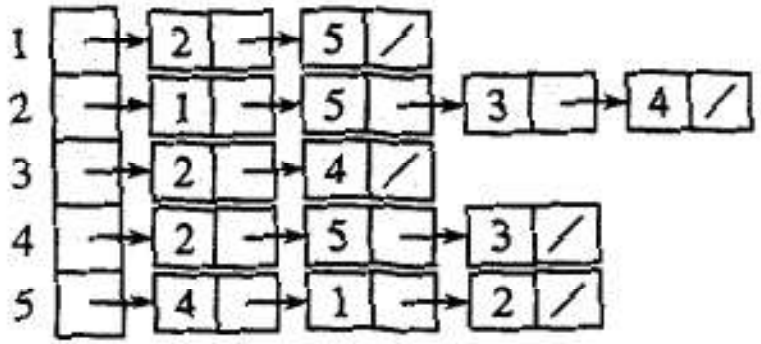


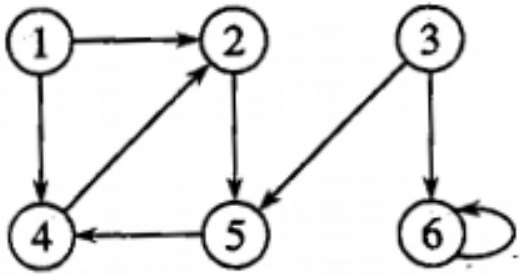


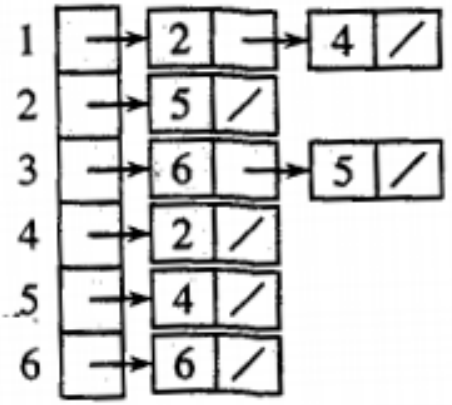
UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Grafos – Lista de Adjacência







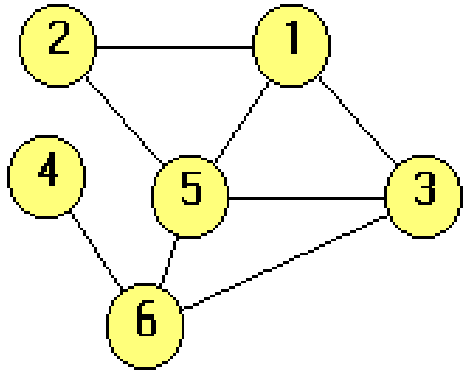


Prof. Luis Cuevas Rodríguez, PhD

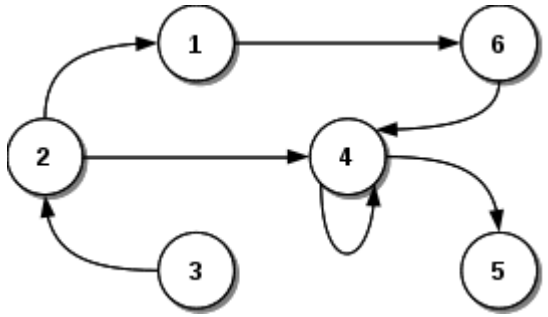
UEA
UNIVERSIDADE
DO ESTADO DO
AMAZONAS

Grafos – Lista de Adjacência

v1 ➔	v2, v5, v3
V2 ➔	v1, v5
V3 ➔	v1, v5, v6
v4 ➔	v6
v5 ➔	v1, v2, v3, v6
V6 ➔	v3, v4, v5



v1 ➔	V6
V2 ➔	v1, v4
V3 ➔	v2
v4 ➔	v4, v5,
v5 ➔	
V6 ➔	v4

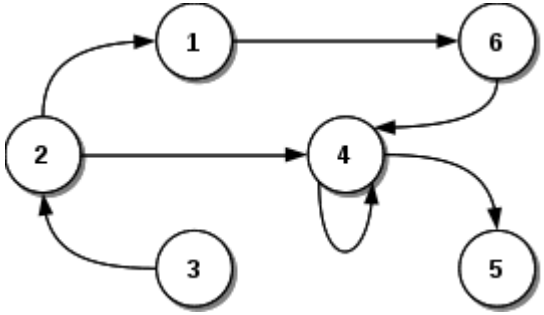


Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE
DO ESTADO DO
AMAZONAS

Grafos – Lista de Adjacência

- Arranjo de *n* **listas ligadas**, uma para cada vértice do grafo. (tem que ter o número de nó)
- Um **lista ligada**, os elementos ou dados de cada nó são listas ligadas como os vértices que tem aresta com esse vértice.



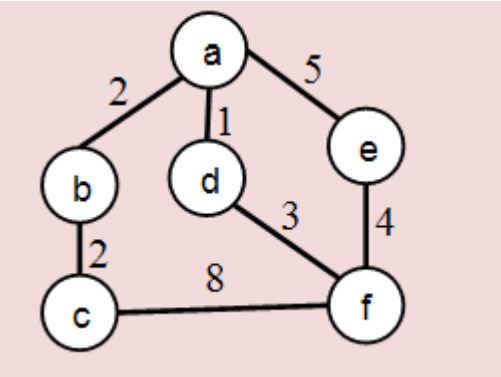
v1 ➔	V6
V2 ➔	v1, v4
V3 ➔	v2
v4 ➔	v4, v5,
v5 ➔	
V6 ➔	v4

Grafos – Lista de Adjacência

- G é um grafo orientado: a soma dos comprimentos de todas as listas de adjacência é |E|
- G é um grafo não orientado: a soma dos comprimentos de todas asa listas de adjacência é 2|E|

Grafos – Lista de Adjacência

- Grafo ponderado: a listas de adjacência se podem adaptar.



va ➔	vb,2 ➔ vd,1 ➔ ve,5
vb ➔	va,2 ➔ vc,2
vc ➔	vb,2 ➔ vf,8
vd ➔	va,1 ➔ vf,3
ve ➔	va,5 ➔ vf,4
vf ➔	ve,4 ➔ vd,3 ➔ vc,8

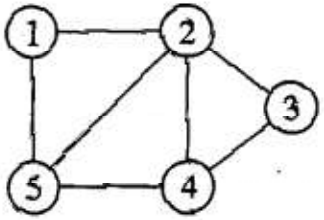
Grafos – Matriz de Adjacência

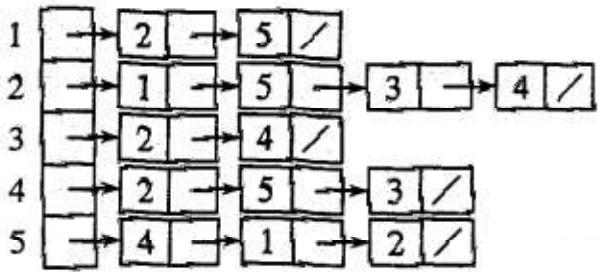
- Uma matriz *n x n (|V|x|V|)*, onde n é o numero de vértices.
- A[i,j] ➔ representa se houver uma aresta desde o vértice *i* até o vértice *j*

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E, \\ 0 & \text{em caso contrário} \end{cases}$$

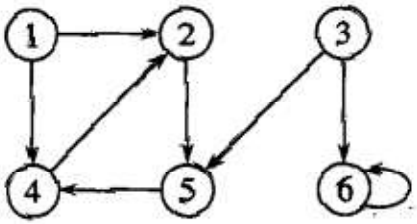
  - Se o grafo é ponderado o valor de a[i,j] é o peso
  - Grafo não ponderado o valor de a[i,j] é 0 ou 1

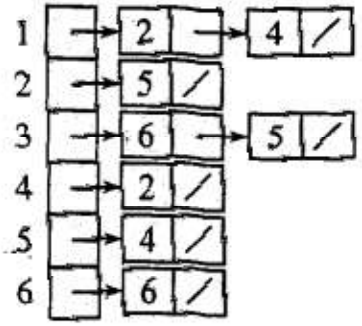
Grafos - Representação





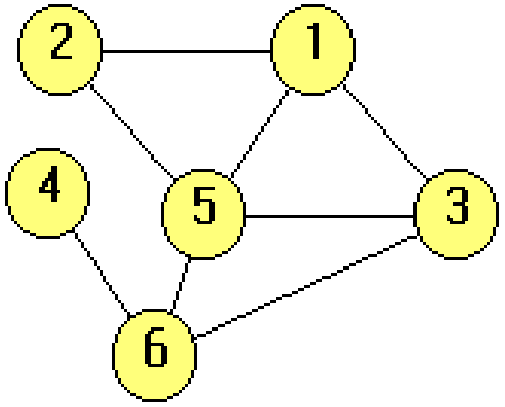
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



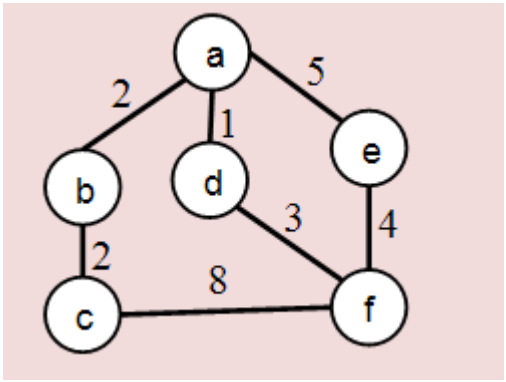


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Grafos – Matriz de Adjacência



0	1	1	0	1	0
1	0	0	0	1	0
1	0	0	0	1	1
0	0	0	0	0	1
1	1	1	0	0	1
0	0	1	1	1	0



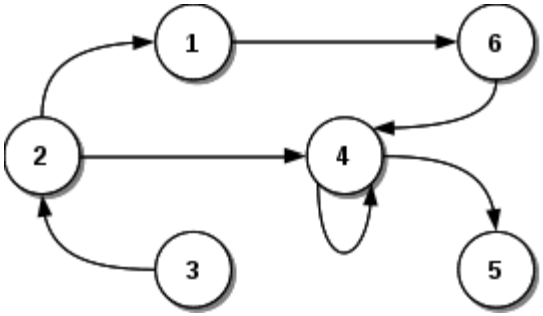
	a	**b**	**c**	**d**	**e**	**f**
a	0	2	0	1	5	0
b	2	0	2	0	0	0
c	0	2	0	0	0	8
d	1	0	0	0	0	3
e	5	0	0	0	0	4
f	0	0	8	3	4	0

Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Grafos – Matriz de Adjacência

- Exercícios: Mostrar as duas representações.



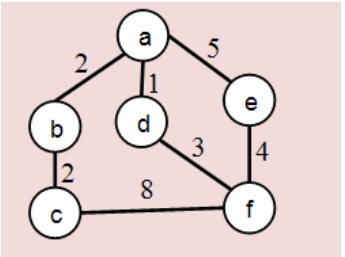
Grafos

- Grafos densos: muitas arestas em relação ao numero de vértices.

va ➔	vb,2 ➔ vd,1 ➔ ve,5
vb ➔	va,2 ➔ vc,2
vc ➔	vb,2 ➔ vf,8
vd ➔	va,1 ➔ vf,3
ve ➔	va,5 ➔ vf,4
vf ➔	ve,4 ➔ vd,3 ➔ vc,8

Listas enormes, mais memória

	a	b	c	d	e	f
a	0	2	0	1	5	0
b	2	0	2	0	0	0
c	0	2	0	0	0	8
d	1	0	0	0	0	3
e	5	0	0	0	0	4
f	0	0	8	3	4	0



UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

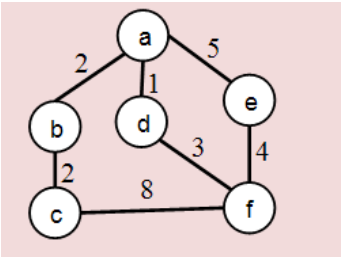
Grafos

- Grafos esparso: poucas arestas em relação ao numero de vértices.

va ➔	vb,2 ➔ vd,1 ➔ ve,5
vb ➔	va,2 ➔ vc,2
vc ➔	vb,2 ➔ vf,8
vd ➔	va,1 ➔ vf,3
ve ➔	va,5 ➔ vf,4
vf ➔	ve,4 ➔ vd,3 ➔ vc,8

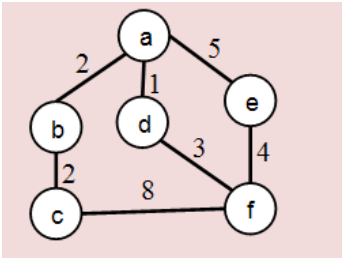
menos memória

	a	b	c	d	e	f
a	0	2	0	1	5	0
b	2	0	2	0	0	0
c	0	2	0	0	0	8
d	1	0	0	0	0	3
e	5	0	0	0	0	4
f	0	0	8	3	4	0



UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Grafos



Em dependência da operações.

- Busca ➔ mais rápida como lista de adjacência
- Testar existência de uma aresta ➔ mais rápida em matrizes.
- Encontrar predecessor de um nó➔ mais rápido em matrizes

va ➔	vb,2 ➔ vd,1 ➔ ve,5
vb ➔	va,2 ➔ vc,2
vc ➔	vb,2 ➔ vf,8
vd ➔	va,1 ➔ vf,3
ve ➔	va,5 ➔ vf,4
vf ➔	ve,4 ➔ vd,3 ➔ vc,8

	a	**b**	**c**	**d**	**e**	**f**
a	0	2	0	1	5	0
b	2	0	2	0	0	0
c	0	2	0	0	0	8
d	1	0	0	0	0	3
e	5	0	0	0	0	4
f	0	0	8	3	4	0

Grafos

Matrizes de adjacência	Listas de adjacência
• Grafos densos. • Operações como teste de aresta, identificação de predecessores.	• Grafos esparsos. • Operações que tenham como base um caminho de um vértice a outro (ex. busca)

IMPLEMENTAÇÃO LISTAS DE ADJACÊNCIA

Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE
DO ESTADO DO
AMAZONAS

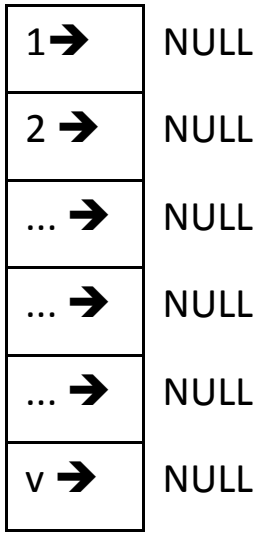
Grafo

```
typedef struct adjacencia {
int vertice;
int peso;
struct adjacencia *prox;
}ADJACENCIA;

typedef struct vertice {
ADJACENCIA *cad;
}VERTICE;

typedef struct grafo{
int vertices;
int arestas;
VERTICE *vert;
}GRAFO;
```

Criação de um grafo

- Número fixo de vértices

```
GRAFO* criaGrafo(int v){
GRAFO* g = new (GRAFO);
g->vertices=v;
g->arestas=0;
g->vert = new VERTICE[v];
for (int i=0; i<v; i++)
    g->vert[i].cad = NULL;
return g;
}
```




```
int main()
{
    GRAFO* gr = criaGrafo(5);
```

Incluir uma aresta no grafo

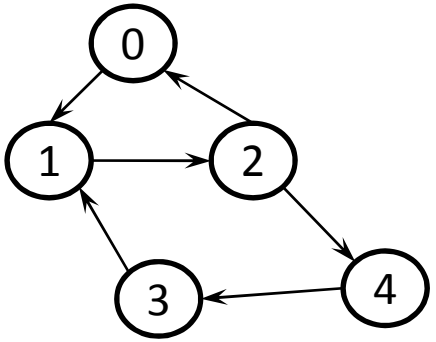
- Criação de adjacência

```
ADJACENCIA* criaAdj(int v, int peso){
ADJACENCIA* temp = new(ADJACENCIA);
temp->vertice=v;
temp->peso = peso;
temp->prox=NULL;
return (temp);
}
```

Incluir uma aresta no grafo

- Criação da aresta

```
bool criaAresta(GRAFO* gr, int vi, int vf, TIPOPESO p){
 if (!gr) return (false);
 if ((vf<0)|| (vf >= gr->vertices))
     return (false);
 if ((vi<0)|| (vi>=gr->vertices))
     return (false);
 ADJACENCIA* novo = criaAdj(vf,p);
 novo->prox=gr->vert[vi].cad;
 gr->vert[vi].cad = novo;
 gr->arestas++;
 return (true);
}
```



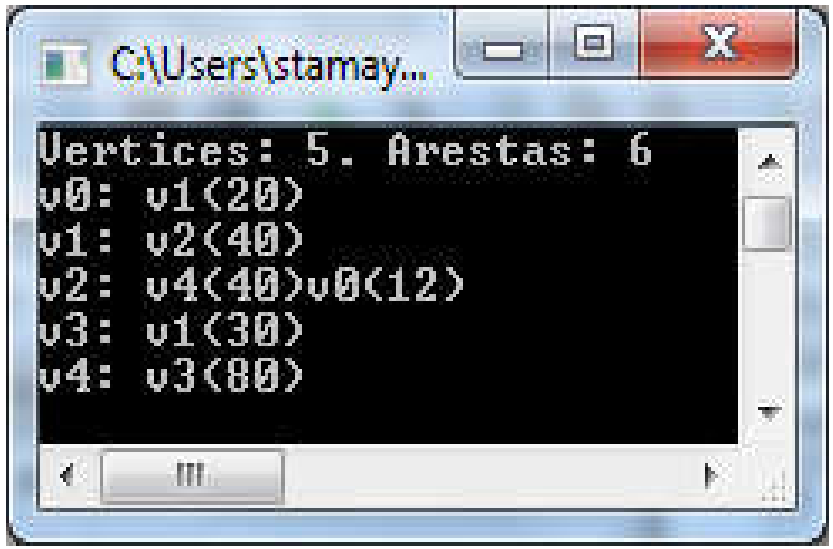

```
int main()
{
     GRAFO* gr = criaGrafo(5);
     criaAresta(gr,0,1,20);
     criaAresta(gr,1,2,40);
     criaAresta(gr,2,0,12);
     criaAresta(gr,2,4,40);
     criaAresta(gr,3,1,30);
     criaAresta(gr,4,3,80);
```

Visualizar o grafo

```
void imprime(GRAFO* gr){
 cout << "Vertices: " << gr->vertices <<". Arestas: "<< gr->arestas << endl;
for (int i=0; i < gr->vertices; i++){
     cout << "v" << i <<": ";
     ADJACENCIA* ad = gr->vert[i].cad;
     while(ad){
          cout << "v" << ad->vertice << "(" << ad->peso << ")";
          ad = ad->prox;
     }
     cout << endl;
}
 }
```

Grafo exemplo

```
int main()
{
    GRAFO* gr = criaGrafo(5);
    criaAresta(gr,0,1,20);
    criaAresta(gr,1,2,40);
    criaAresta(gr,2,0,12);
    criaAresta(gr,2,4,40);
    criaAresta(gr,3,1,30);
    criaAresta(gr,4,3,80);
    imprime(gr);
    return 0;
}
```



Exercício

- Escrever o código mostrado na aula e testar.
- Representar o problema do tabuleiro de xadrez e os cavalos
- Fazer uma implementação que procure o menor caminho para ir de uma casa a outra do tabuleiro de xadrez usando os movimentos do cavalo