

Algoritmo Dijkstra

Felipe Affonso
Feliphe Nogueira

Edsger Wybe Dijkstra

- * Matemático e Cientista da Computação
- * Em 1972 recebeu o Premio ACM Turing (atual Nobel)



Simplicidade é um pré requisito para a confiança.

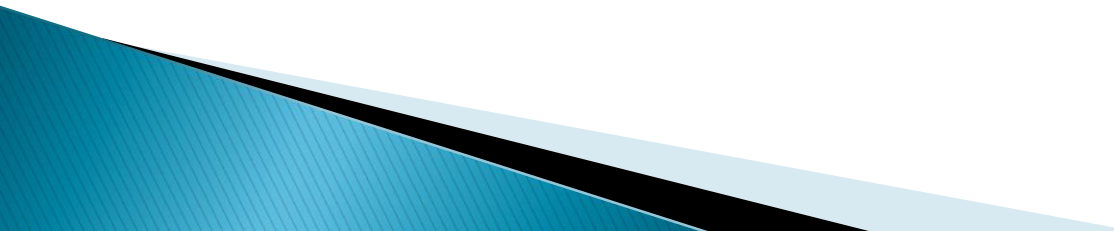
11 de maio 1930 / 2 de Agosto 2002

Edsger Wybe Dijkstra

Algoritmo de Dijkstra

- ▶ Cálculo de custo mínimo entre vértices de um grafo;
- ▶ Utilização de grafos orientados ou não;
- ▶ Aplicação em áreas de transportes e redes de computadores;

Informações essenciais na estrutura do algoritmo

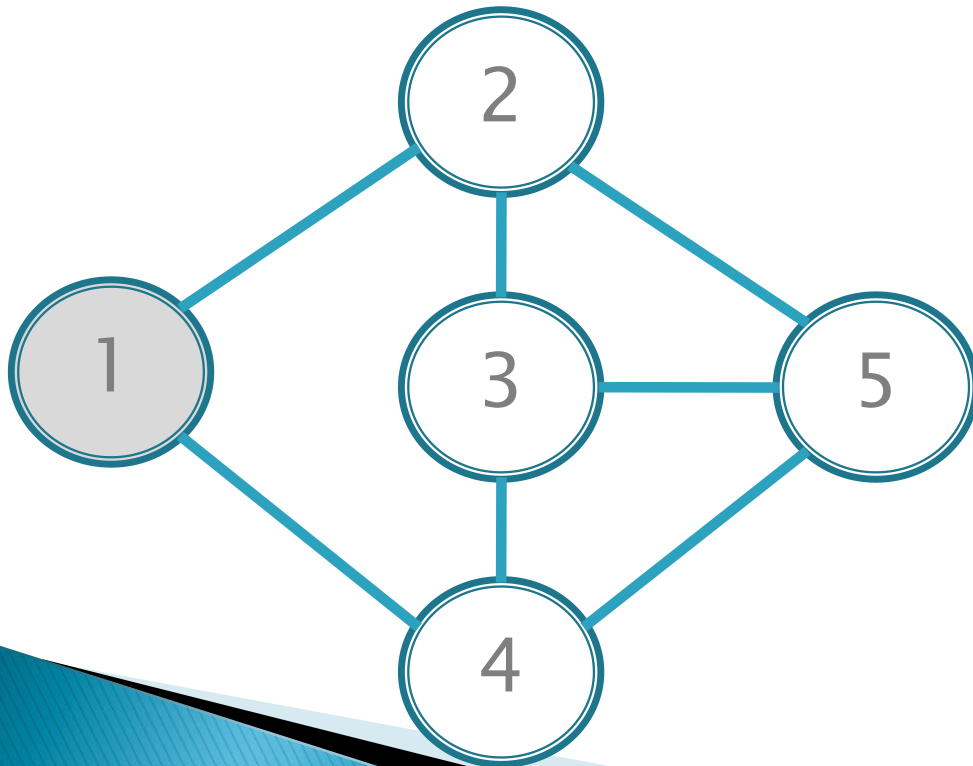
- ▶ Um flag determinando se o caminho mais curto do vértice fonte até um vértice qualquer é conhecido;
 - ▶ O comprimento do caminho mais curto do vértice fonte ao vértice destino. Ao ser iniciado o algoritmo, todos os comprimentos dos vértices do grafo possuem um valor desconhecido que é alterado durante a execução do algoritmo;
 - ▶ O antecessor do vértice destino no caminho mais curto do vértice fonte a seu destino, sendo que ao início o antecessor de um vértice destino não é conhecido.
- 

Primeiro teste:

Fazendo funcionar:

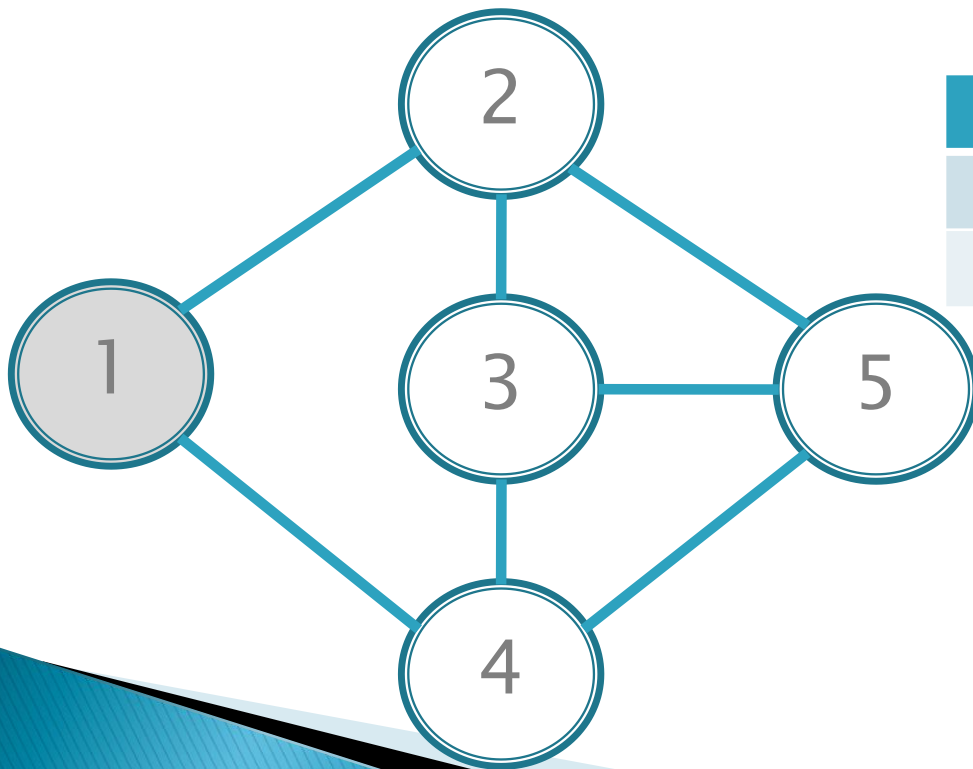
- ▶ Definir o vértice principal (Origem):

//Escolhemos o 1;



Fazendo funcionar:

- ▶ Atribuir a cada nó, um peso provisório. Igualamos a zero o nosso vértice inicial, e infinito para todos os outros vértices.

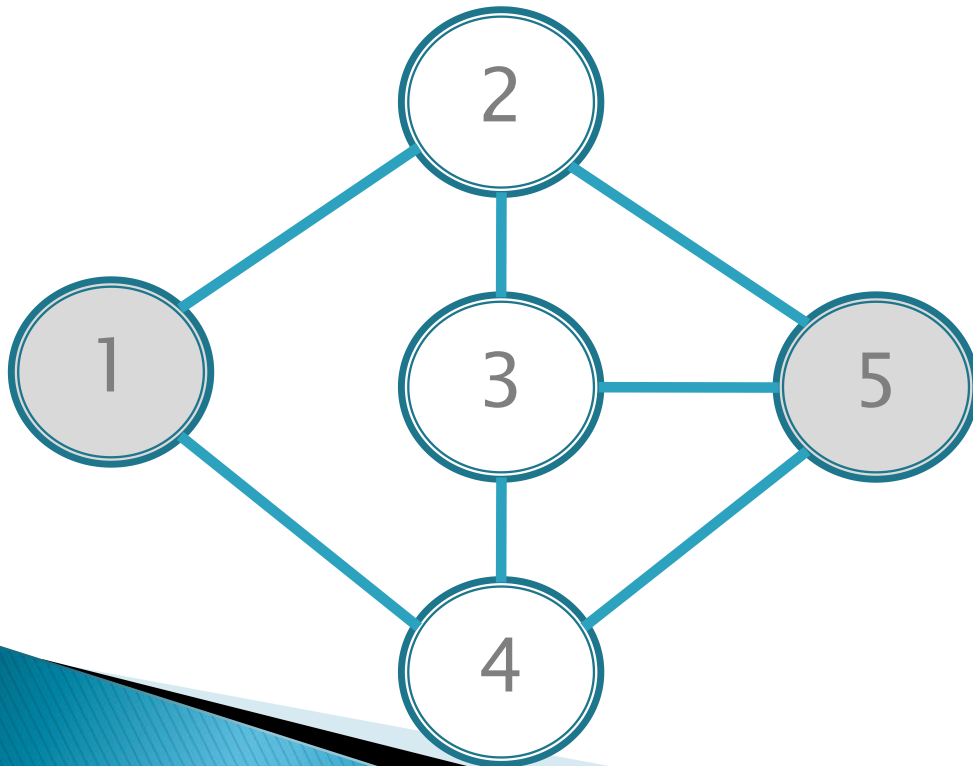


Custo				
1	2	3	4	5
0	∞	∞	∞	∞

Fazendo funcionar:

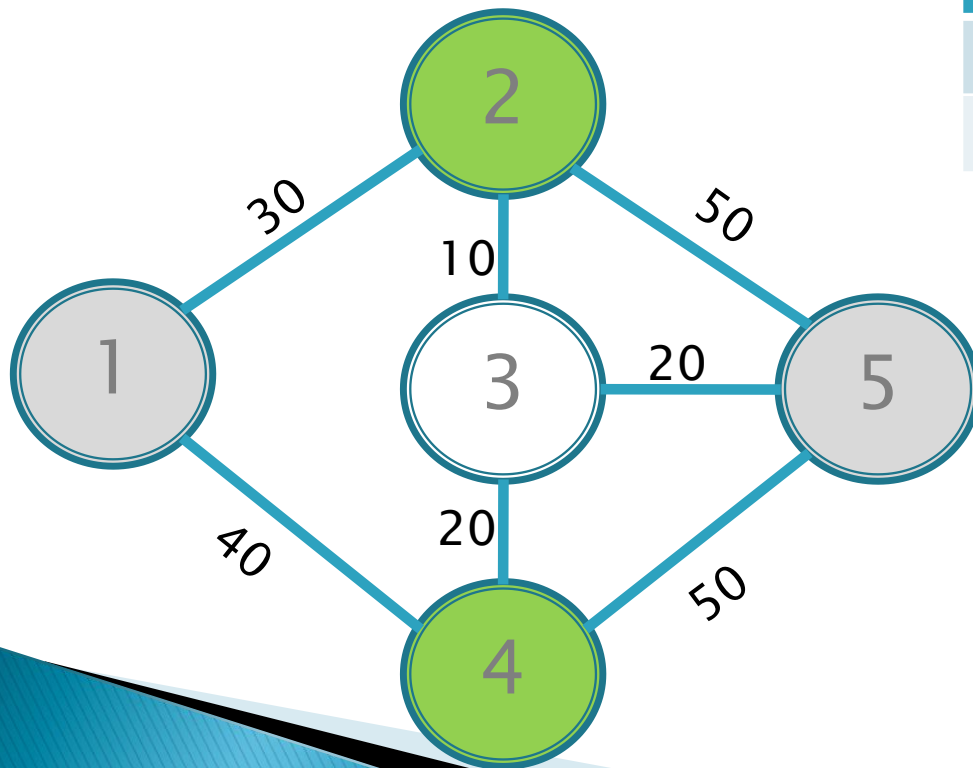
- ▶ Definir o vértice final (Destino):

- ▶ //Escolhemos o 5;



Fazendo funcionar:

- ▶ Encontramos os vértices adjacentes à origem e adicionamos seu peso no vetor:

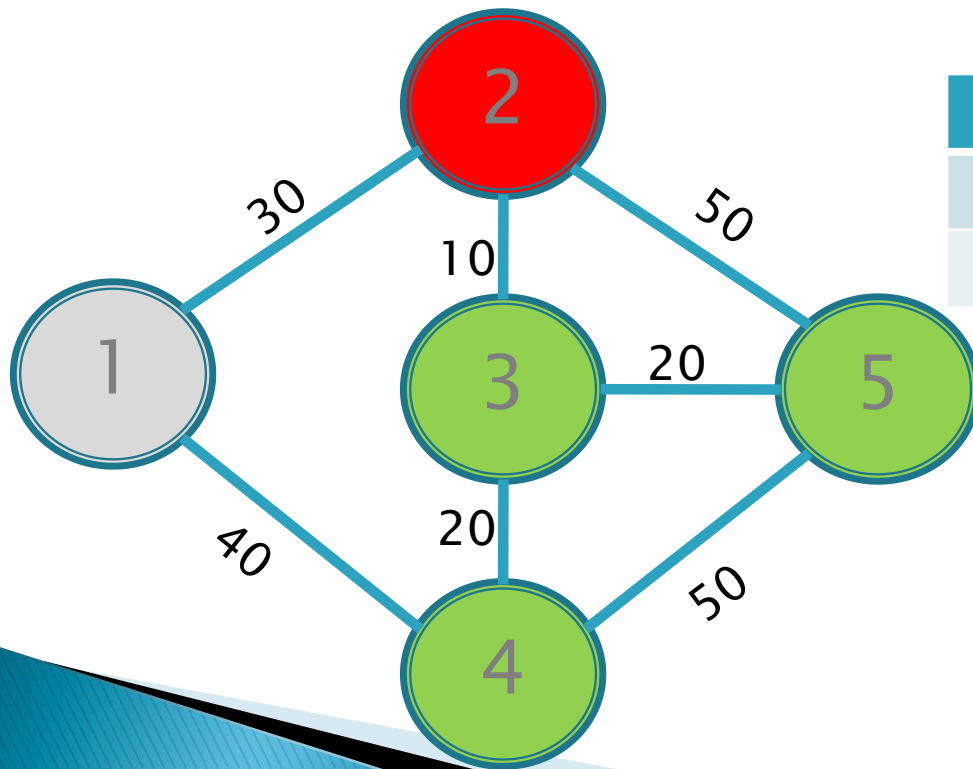


Custo				
1	2	3	4	5
0	30	∞	40	∞

Legenda	
	Não visitado
	Origem/Destino
	Vert. Adjacentes
	Vert. Percorrido
	Fim do caminho

Fazendo funcionar:

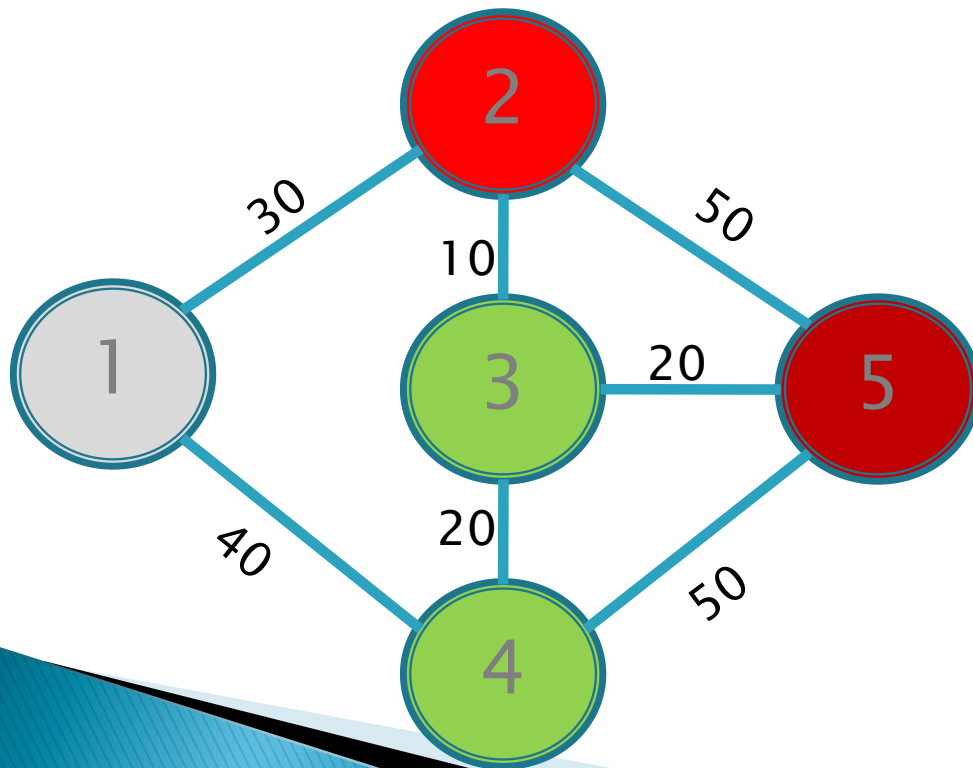
- ▶ Visitamos o vértice com menor custo:



Custo				
1	2	3	4	5
0	30	∞	40	∞

Fazendo funcionar:

- ▶ Verificamos se conseguimos chegar no destino e qual o custo total.

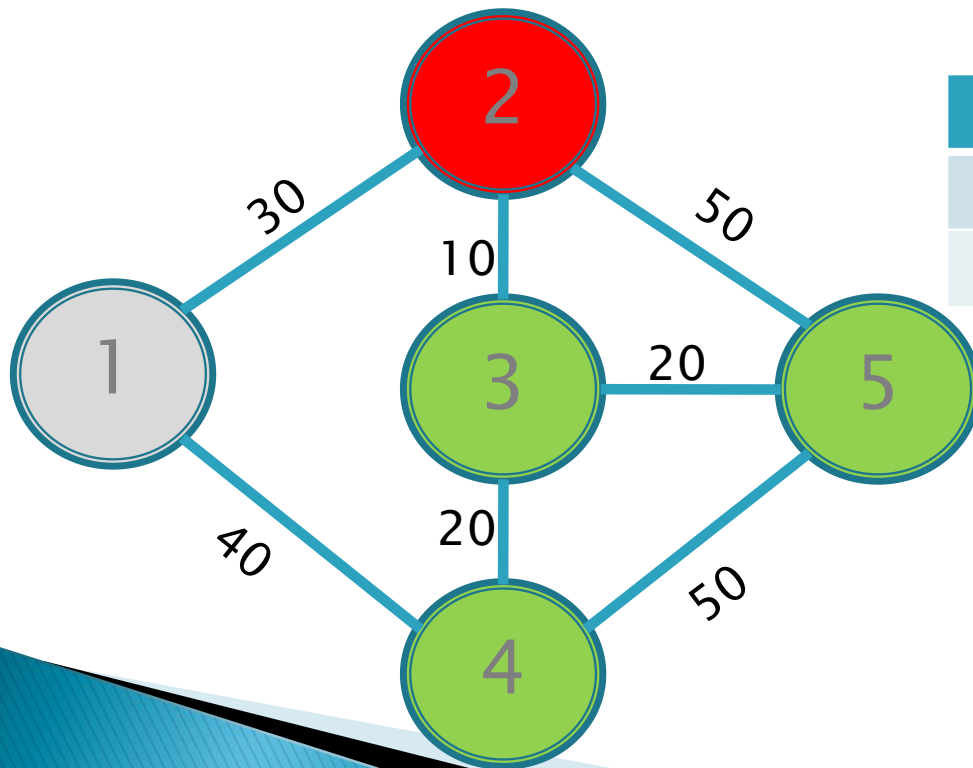


Custo 1-2-5
80

Segundo teste:

Fazendo funcionar:

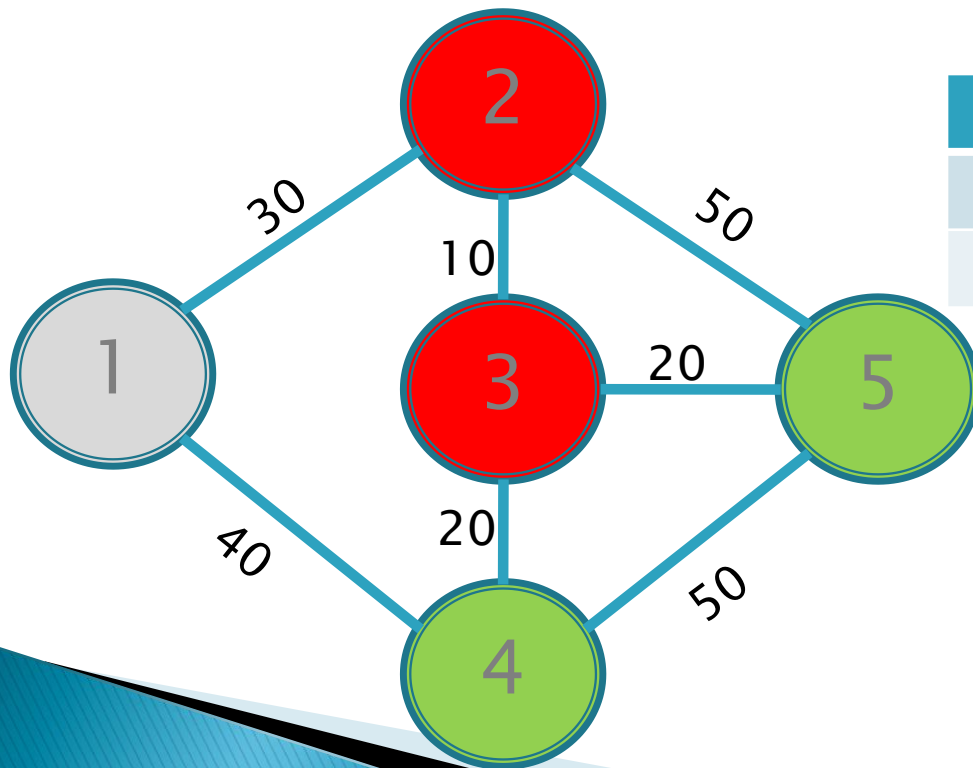
- ▶ Verifica se existem vértices adjacentes não visitados:



Custo				
1	2	3	4	5
0	30	∞	40	∞

Fazendo funcionar:

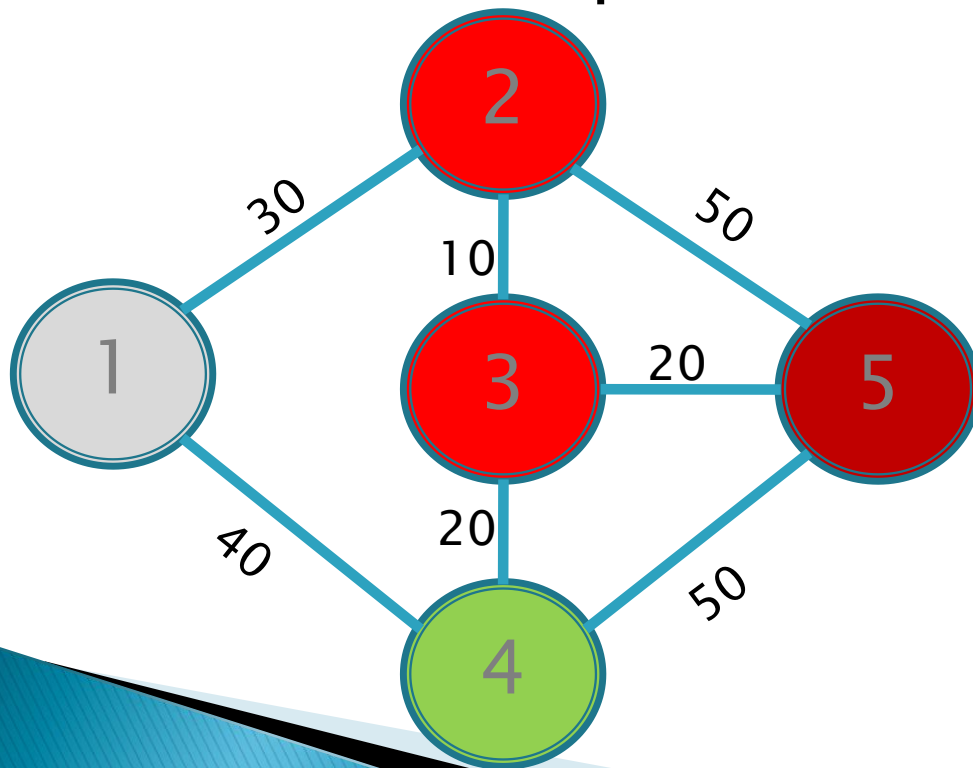
- ▶ Escolhemos o vértice não visitado.
- ▶ Adicionamos o seu custo



Custo				
1	2	3	4	5
0	30	40	40	∞

Fazendo funcionar:

- ▶ Encontramos o nosso destino.
- ▶ Calculamos o custo.
- ▶ Verificamos qual o melhor caminho.



Custo 1-2-5

80

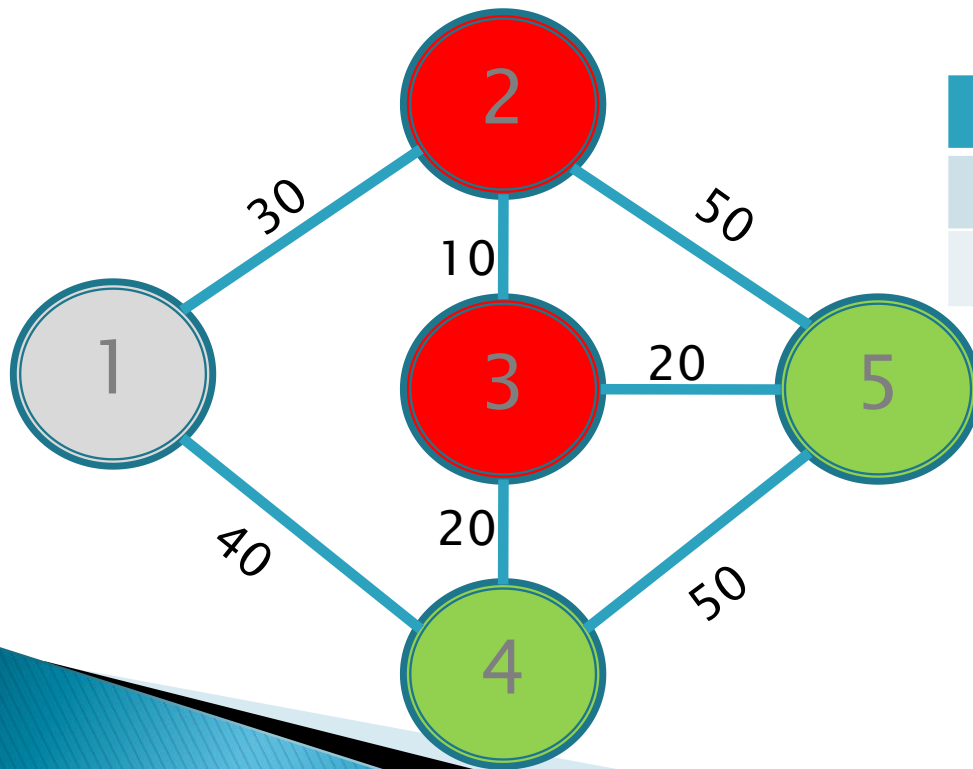
Custo 1-2-3-5

60

Terceiro Teste:

Fazendo funcionar:

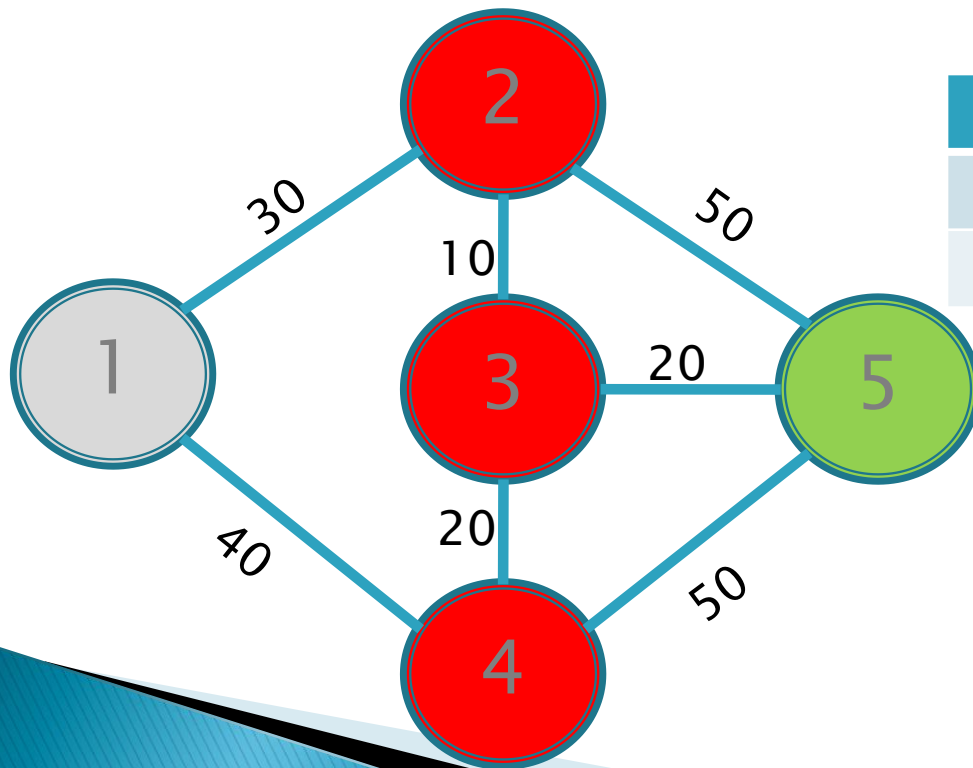
- ▶ Verificamos se ainda existem vértices adjacentes não visitados.



Custo				
1	2	3	4	5
0	30	40	40	∞

Fazendo funcionar:

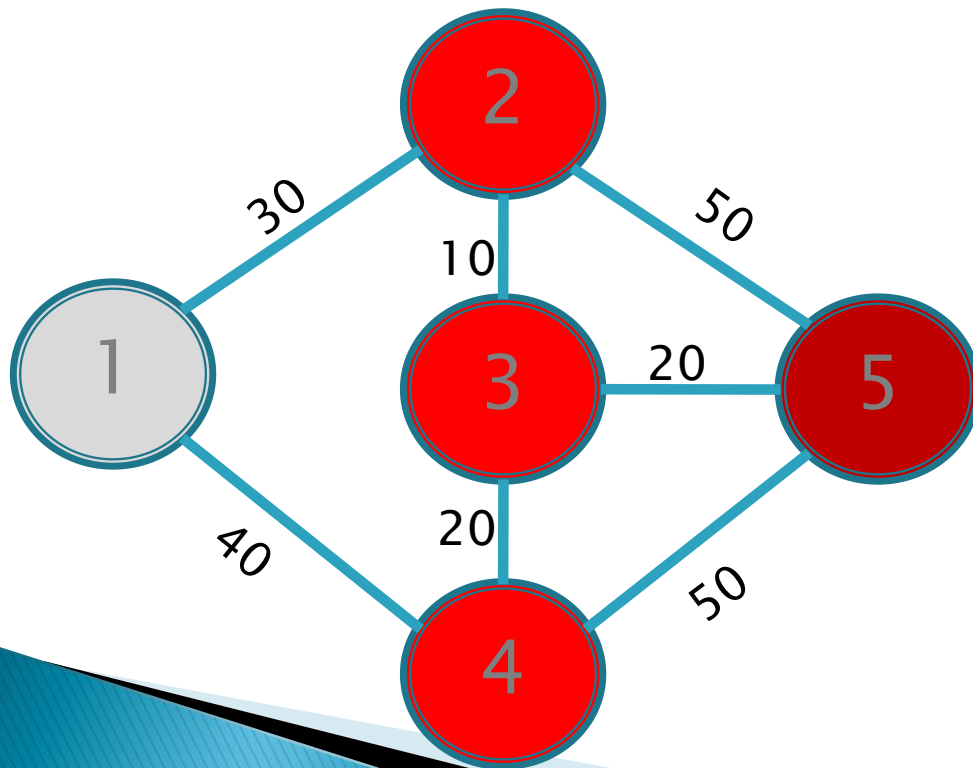
- ▶ Escolhemos esse vértice que ainda não foi visitado.



Custo				
1	2	3	4	5
0	30	40	40	∞

Fazendo funcionar:

- ▶ Calculamos o custo total.

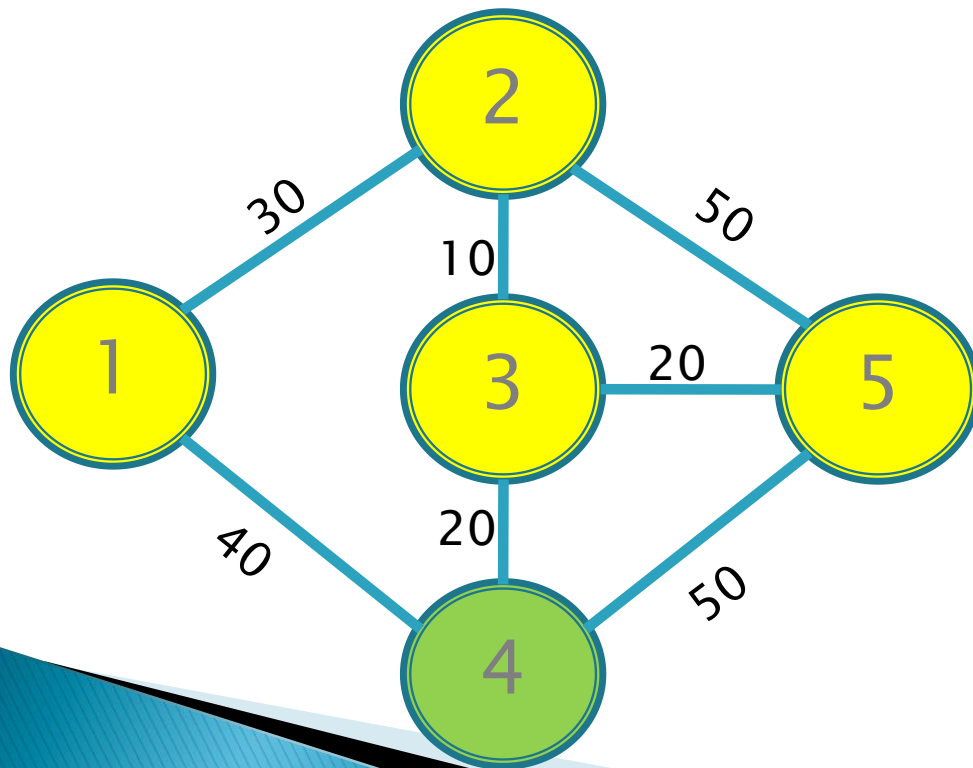


Custo 1-4-5

90

Os três casos dos exemplos:

- ▶ O melhor caminho foi escolhido: 1-2-3-5



Custo 1-2-5

80

Custo 1-2-3-5

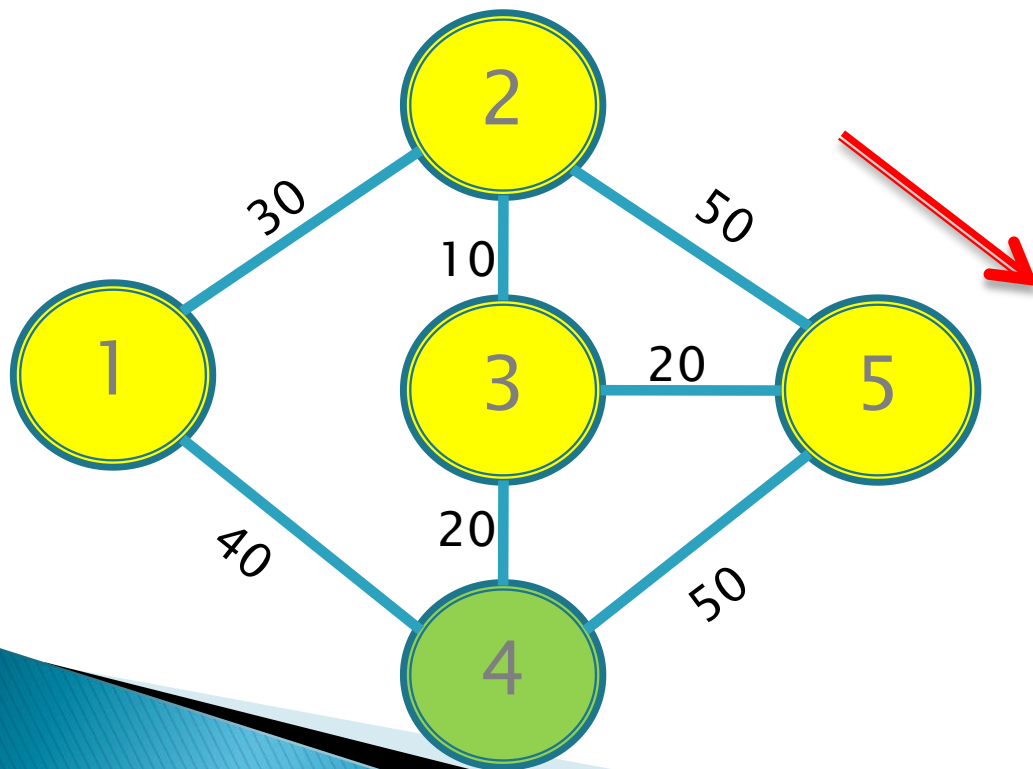
60

Custo 1-4-5

90

Os três casos dos exemplos:

- ▶ O melhor caminho foi escolhido: 1-2-3-5



Custo 1-2-5

80

Custo 1-2-3-5

60

Custo 1-4-5

90

Código:

Utilizaremos a biblioteca stdio.h para as funções de scanf e printf

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Código:

Delimitamos o nosso infinito como
999

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++){
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
            }
        }
    }
}
```

Chamada da função dijskra.
Passamos como parâmetro a
quantidade de vertices, o nosso
vertice origem, a matriz de custo
e o vizinhos a esse vertice

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Declaramos as variáveis utilizadas nessa função.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

U: Gravar o indice de quando encontramos o menor caminho. É sempre reutilizada.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Count: para definir quando o termino da busca pelo menor caminho. Ela será utilizada no nosso While.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

i: contador para o for.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Flag: vetor que indica se um determinado vértice já foi percorrido ou não.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Min: uma distância inicial minima.
Funciona como se fosse o nosso infinito.
Todas as distâncias deverão ser menores
que essa variável.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

J: Pode ser utilizada caso seja necessário imprimir a matriz de custo. Nessa apresentação essa variável não foi utilizada.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

For que percorrer todos os vertices e os marca como não visitados.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Preenche o vetor de vizinhos adjacentes adicionando o valor da matriz de custos na linha especificada como nosso vertice de origem.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++)
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Após essas rotinas, o nosso contador passa a valer dois.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

While que irá percorrer todos os vértices buscando o menor caminho. Enquanto o contador for menor ou igual ao numero de vértices ele irá buscar o menor caminho.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

No começo da nossa repetição nós colocamos a variável min como 99, para que todas as distâncias verificadas abaixo sejam menores que ela.

Note que o nosso infinito vale 999, e o min 99.
999 indica que os vértices não são adjacentes.
Menores que 99 indica que existe uma distância entre estes vértices.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++)
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Percorre todos os vertices.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;
    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }
    count=2;
    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }
        }
        flag[u]=1;
        count++;
        for(i=1;i<=vertices;i++){
            if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Verifica se os vértices são adjacentes e se esse vértice ainda não foi visitado.
Flag[i] -> verifica se é verdadeiro (1)
!flag[i] -> verifica a negação da variável (0)

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++){
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
            }
        }
    }
}
```

Se este vértice passar pelas condições acima, a distância mínima será a distância até esse vértice.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++){
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
            }
        }
    }
}
```

A variável U receberá esse vértice.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++)
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

O Vértice escolhido anteriormente como menor é marcado como visitado.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){
        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++){
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
            }
        }
    }
}
```

Aumentamos o valor do nosso contador.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){

        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++)
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Percorremos todos os vértices novamente...

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){

        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++)
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Verificamos se a distancia ate esse vértice com o menor caminho + o custo para chegar nesse vértice é menor do que a distância ate os vizinhos adjacentes e se ele não foi visitado.

Note que essa verificação é feita com todos os vértices utilizando a variável U como referência.

Código:

```
#include "stdio.h"
#define infinito 999

void dij(int vertices,int origem,int custo[10][10],int vizinhos[]){
    int u,count,i,flag[10],min,j;

    for(i=1;i<=vertices;i++){
        flag[i]=0;
        vizinhos[i]=custo[origem][i];
    }

    count=2;

    while(count<=vertices){

        min=99;
        for(i=1;i<=vertices;i++){
            if(vizinhos[i]<min && !flag[i]){
                min=vizinhos[i];
                u=i;
            }

            flag[u]=1;
            count++;

            for(i=1;i<=vertices;i++)
                if((vizinhos[u]+custo[u][i]<vizinhos[i]) && !flag[i])
                    vizinhos[i]=vizinhos[u]+custo[u][i];
        }
    }
}
```

Se o vértice i for aprovado nas condições acima, o nosso vetor de vizinhos irá receber a distância até o vertice U + custo ate o vértice U . Esse valor é incrementado até o final do for.

Dessa forma conseguimos obter a menor distancia partindo de um ponto até todos os outros vértices.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Declaração das variáveis que usaremos no nosso código

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Leitura da quantidade de vértices do nosso grafo.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
        printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
}
```

For para ler o o peso das arestas.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Se for a diagonal principal não é necessário ler o peso da aresta.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Leitura das arestas.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Se o custo for zero, não existe ligação entre as arestas, dessa forma o custo adicionado será infinito (999)

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Se for a matriz principal, o custo também será infinito.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Leitura do vértice de origem para calcularmos a distancia até os demais vértices.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Leitura do vértice de destino.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Chamamos a função passando q
a quantidade de vértices, nosso
vértice de origem, nossa matriz de
custos e a matriz de distancias.

Continuação do Código:

```
int main(){
    int vertices,origem,i,j,custo[10][10],dist[10],destino;

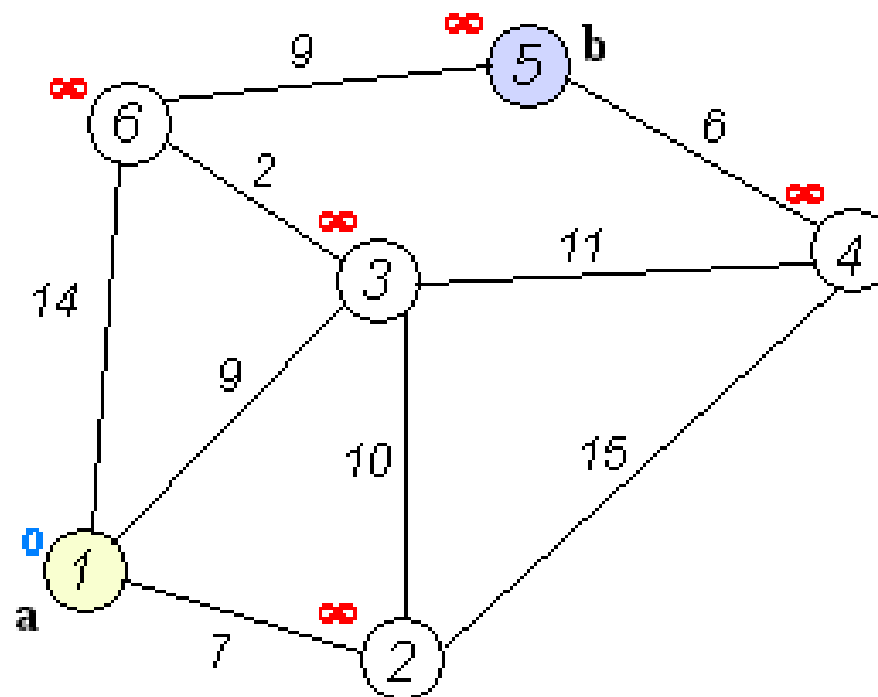
    printf("\n Digite o numero de vertices:");
    scanf("%d",&vertices);
    printf("\n Digite o custo da matriz:\n");
    for(i=1;i<=vertices;i++){
        for(j=1;j<=vertices;j++){
            if(i!=j){
                printf("Vertice %d a %d: ",i,j);
                scanf("%d",&custo[i][j]);
                if(custo[i][j]==0)
                    custo[i][j]=infinito;
            } else
                custo[i][j] = infinito;
        }
    }

    printf("\n Digite a origem:");
    scanf("%d",&origem);
    printf("\n Digite o destino:");
    scanf("%d",&destino);

    dij(vertices,origem,custo,dist);
    printf("\n Menor caminho:\n");
    printf("%d->%d,cost=%d\n",origem,destino,dist[destino]);
}
```

Exibimos o vertice de origem, o vértice de destino e o vetor de distancias na posição do vertice de destino.

Aplicando Dijkstra em um Grafo



Desvantagens

- ▶ Não funciona para grafos com ciclos com pesos negativos;
- ▶ Complexidade Total: $O((n+m)\log n)$.

Conclusão

- ▶ O algoritmo de Dijkstra busca o menor caminho partindo de um determinado vértice até os outros vértices do grafo.