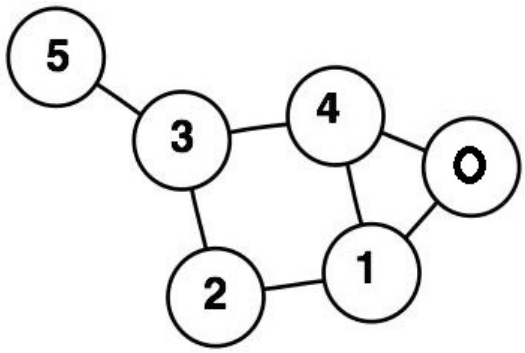
Introdução a Grafos.

Daiii galera, chegou uma hora muito esperada, vamos começar a falar sobre GRAFOS, como esse é um assunto extremamente vasto nesse post vamos começar devagar mostrando os conceitos básicos e algumas implementações pra não ficar só na ~~falaçada~~ teoria. Um breve conhecimento de C++ seria interessante para acompanhar as implementações ~~maaaaaaaaas se você não programa em C++ também vai intender com facilidade~~.

O que é são Grafos?

Um grafo é uma estrutura formada por vértices e arcos (~~bolinhas e linhas~~) esse é um exemplo de grafo:



Não se esqueça dele que usaremos nas implementações e exemplos.

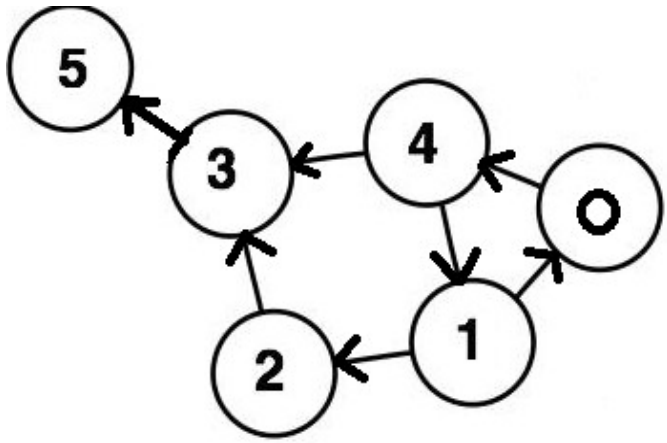
Nós colocamos nossas informações nas vértices, onde cada uma possui um número de identificação e os arcos são responsáveis pela ligação das vértices formando um conjunto. Com o conjunto formado podemos fazer várias coisas interessantes (~~ou não~~) por exemplo executar buscas, logo falaremos mais sobre elas.

Mais falaçada... Para implementar os grafos ainda precisamos intender mais alguns conceitos importantes.

- **Vértice vizinho**: Dizemos que dois vértices são vizinhos se existe um arco ligando eles (na imagem acima podemos ver que o vértice 0 é vizinho do 4 e do 1, o vértice 1 é vizinho do 0, 4 e 2 e assim segue.

- O **tamanho** de um grafo é a soma de suas vértices e arcos.

- **Dígrafos**, ou grafos dirigido ou ainda grafo orientado é um grafo onde seus arcos possuem um sentido, acrescentando o sentido no grafo do exemplo fica assim.



Isso significa que para percorrer o dígrafo temos que seguir um caminho limitado sempre seguindo a seta, ou seja, para chegar no vértice 3 partindo do 0 precisamos passar primeiro pelo 4 ou como caminho alternativo 0 – 4 – 1 – 2 – 3.

- **Grau de saída e entrada** representa o número de arcos entrando e saindo de um vértice, exemplo o grau de saída do vértice 4 é 2 e de entrada é 1.

Mas como eu programo isso?

Existem várias maneiras de representar grafos, as clássicas são a matriz de adjacência e lista de adjacência vamos falar sobre as duas e você escolhe qual usar tendo em vista que cada uma possui vantagens e desvantagens.

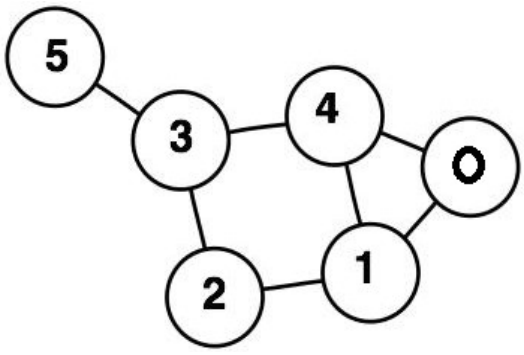
Matriz de adjacência

A matriz de adjacência é uma matriz booleana com colunas e linhas representando os vértices. Representamos como true (1) na matriz se dois vértices são vizinhos e false (0) se não são.

Ta confuso? Vamos montar a matriz do grafo anterior pra ver se melhora, lembrando que a primeira linha e a primeira coluna representam os vértices e o tamanho da matriz é igual ao número de vértices no grafo. Chamaremos a matriz de Adjacência de *matrizAdj* .

matrizAdj[5][5]

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



Como o vértice 0 é vizinho dos vértices 5 e 1 os índices da *matrizAdj[0][4]* e *matrizAdj[0][1]* são marcados como true.

Como o vértice 3 é vizinho dos vértices 2, 4, 5 os índices *matrizAdj[3][2], matrizAdj[3][4], matrizAdj[3][5]* são marcados true.

Essa lógica segue até marcarmos todos os índices.

Agora vamos ver isso em c++...

A implementação da Matriz de adjacência é bem simples. Primeiro definimos a matriz:

```
#include <iostream>

using namespace std;

int matrAdj[5][5] = {
        { 0, 1, 0, 0, 1, 0,},
        { 1, 0, 1, 0, 1, 0,},
        { 0, 1, 0, 1, 0, 0,},
        { 0, 0, 1, 0, 1, 1,},
        { 1, 1, 0, 1, 0, 0,},
        { 0, 0, 0, 1, 0, 0,}
};
```

Nossa matriz de ascendência está declarada, logo o grafo está construído, como esse exemplo é didático não estamos guardando informações nos nós exceto a identificação. Feito isso agora podemos implementar uma função interessante para verificar se dois vértices são ou não vizinhos.

A função *vizinho* vai receber o número de dois vértices e retorna verdadeiro se eles são realmente vizinhos ou retorna false se não (~~ela só verifica se o índice dos vértices na matriz é igual a 1 ou 0~~).

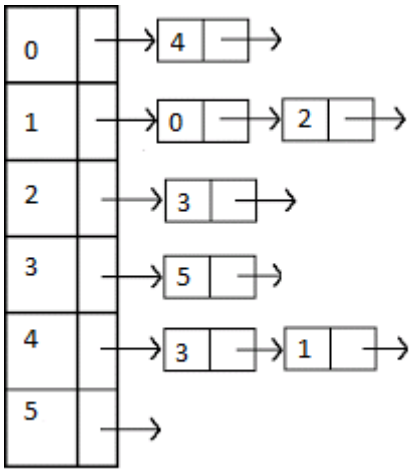
```
bool vizinho(int vertice1, int vertice2) {

        if (matrAdj[vertice1][vertice2]) {
                return true;
        return false;
        }
}
```

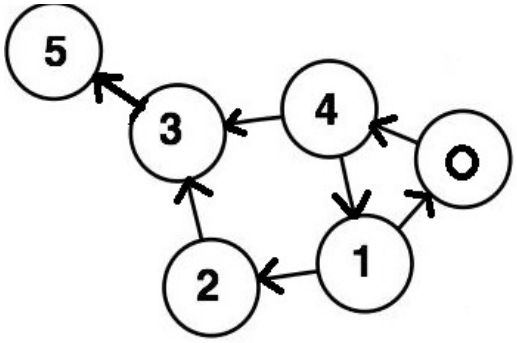
Lista de adjacência

Tá na hora de dar uma complicadinha vamos a lista de adjacência essa representação possuímos um array de listas encadeadas e cada vértice do grafo representa um índice do array, essa lista possui todos os vizinhos do vértice em questão.

Atenção que para esse exemplo vamos usar o dígrafo.

Como a melhor maneira de aprender é na prática, vamos montar a lista de adjacência do nosso dígrafo já conhecido. A primeira coluna representa o array de vértices e ao lado a lista encadeada com seus vizinhos **atenção que agora precisamos respeitar o sentido do dígrafo**.





Implementação da lista de adjacência

Para facilitar essa implementação e não fugir do foco usaremos algumas bibliotecas nativas do c++ então começamos nosso código assim:

```
#include <iostream>

#include <list> //biblioteca para criação de listas encadeadas
#include <algorithm> //Vamos usar a função find da biblioteca Algorithm

using namespace std;
```

Vale salientar que temos um post sobre listas encadeadas e sua implementação nesse projeto seria um bom treino.

Agora com as bibliotecas adicionadas temos que definir nossa classe Grafo.

```
class Grafo {

        int V;
        list<int> *adj; //ponteiro o para o array de listas
public:
        Grafo(int V); //construtor
        void adicionarArco(int vertice1, int vertice2);

        int grauDeSaida(int v);

        bool vizinho(int vertice1, int vertice2);
};
```

A variável V corresponde ao tamanho do grafo, também criamos um ponteiro para o array de listas.

Em public declaramos o construtor com o que recebe o tamanho do grafo, a função adicionarArco recebe dois vértices e adiciona o vertice 2 na lista de adjacencia do vertice 1, o

método grauDeSaida recebe um vértice e retorna seu grau de saída e a função vizinho recebe dois vértices e retorna True se forem vizinhos ou False se não. Esses são métodos básicos e bastante úteis.

O próximo passo é definir nosso construtor.

```
Grafo::Grafo(int V) {

	this->V = V; //atribui o numero de vértices
	adj = new list<int>[V]; //Alocamos espaço e criamos a lista de adjacência
}
```

Agora a função responsável por criar os arcos e adicionar os vértices, ela recebe 2 vértices e adiciona o vértice 2 a lista de vértices adjacentes do vertice1. O método push_back é responsável por adicionar o vértice no final da lista.

```
void Grafo::adicionarArco(int vertice1, int vertice2) {

	//adiciona vertice2 a lista de vertices adjacentes do vertice1
	adj[vertice1].push_back(vertice2);
}
```

Obtendo o grau de saída, para isso utilizamos o método size para nos mostrar o tamanho da lista encadeada (o tamanho dela corresponde ao grau de saída).

```
int Grafo::grauDeSaida(int v) { //arestas que sai do vertice

	return adj[v].size();
}
```

Finalmente a função vizinho, para isso usamos o método *find* da biblioteca *Algorithm* que vai buscar na lista de adjacência do vértice1 o vértice2, se o vértice for encontrado eles são vizinhos logo retornamos true, senão retornamos false.

```
bool Grafo::vizinho(int vertice1, int vertice2) {

	if (find(adj[vertice1].begin(), adj[vertice1].end(), vertice2) != adj[vertice1].end())
		return true;
	return false;
}
```

Agora só falta construirmos nossa *main* e nela usamos os métodos anteriores para criar o grafo e fazer algumas consultas.

```
int main() {


	//criando um grafo de 6 vértices
	Grafo grafo(6);

	//Inserindo....
	grafo.adicionarArco(0, 4);
	grafo.adicionarArco(1, 0);
	grafo.adicionarArco(1, 2);
	grafo.adicionarArco(2, 3);
	grafo.adicionarArco(3, 5);
	grafo.adicionarArco(4, 3);
	grafo.adicionarArco(4, 1);

	//Grau de saida do vertice 3 (por exemplo)
	cout << "Grau de saida do vertice 3: " << grafo.grauDeSaida(3) << endl;

	//Verificando vizinhos 0 e 2 por exemplo
	if (grafo.vizinho(0, 2))
		cout << "2 eh vizinho de 0" << endl;
	else
		cout << "2 nem eh vizinho de 0\n" << endl;


   return 0;
}
```

Conclusão

Então pessoal vou deixar disponível os códigos fontes tentei abstrair o máximo então espero que não tenha ficado muito complexo, ~~nem muito superficial~~. Muito obrigado e bons estudos qualquer dúvida basta entrar em contato.

Para saber mais sobre C++ e sua Standart Library
http://www.cplusplus.com/reference/

E para um conteúdo mais completo e formal sobre grafos

https://www.ime.usp.br/~pf/algoritmos_para_grafos/index.html#contents

Logo começaremos os algoritmos de busca :D