CHECKLIST fiz ☐ imcompleto ☐ não entendi ☐

- ☐ 1. descrever uma classe para representar grafos
- ☐ 2. diferença do ex. 1 para digrafos
- ☐ 3. escrever função achaFonte
- ☐ 4. escrever função testaCaminho
- ☐ 5. escrever função Kcaminho
- ☐ 6. escrever função - movimentos de um cavalo
- ☐ 7. escrever função ancestralComum
- ☐ 8. escrever função, falar complexidade
- ☐ 9. percorrer digrafo em profundidade
- ☐ 10. justificar
- ☐ 11. algoritmo de Dijkstra
- ☐ 12. escrever função (pais)
- ☐ 13. escrever função (clique)
- ☐ 14. usar função do 13, falar complexidade

1. Descreva uma classe para representar `grafos`, implementando:

   - construtor: devolve um grafos com V vértices, sem arestas; V
   - inclui aresta: inclui uma aresta $u - v$ no grafo; V
   - remove aresta: remove a aresta $u - v$ (se existir) do grafo; V

```
class Grafo {
    private:
        int V;                  // número de vértices
        int E;                  // número de arestas
        vector <int> *adj;      // vetor de adjacência
    public:
        Grafo (int V) {
            this -> V = V;
            E = 0;
            adj = new vector <int> [V];
        }
        void add (int u, int v) {
            adj[u].push(v)
            adj[v].push(u)
        }
        void remove (int u, int v) {
            adj[u].pop(v);
            adj[v].pop(u);
        }
};
```

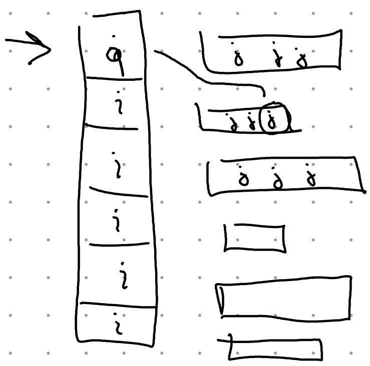
2. O que muda nas funções acima para digrafos?

Para adicionar uma aresta u-v em um digrafo usando a função add() apenas inserimos a aresta dirigida de u até v (adj[u].push(v)) ao invés de adicionar nas adjacências de u e v. Assim como na função remove, removemos apenas de adj[u], pois as arestas são dirigidas.

3. Escreva uma função **achaFonte** que devolve, se existir, um vértice fonte no grafo e -1, caso contrário.

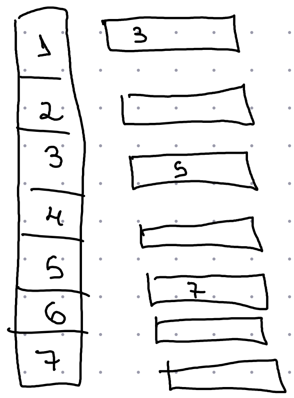
Uma fonte (= source) é um vértice que tem grau de entrada nulo.

```
int achaFonte() {
    for (int q = 0; q < V; q++) {
        bool fonte = false
        for (int i = 0; i < V && q != i; i++) {
            if (std::count(adj[i].begin(), adj[i].end(), q) {
                fonte = true;
            }
        }
        if (fonte == false) return q;
    }
    return -1;
}
```



4. Escreva uma função **testaCaminho** que recebe um vetor **seq** e verifica se seq é um caminho no grafo.

```
bool testaCaminho ( vector<int> seq ) {
    for(int i = 0; i < seq.size() -1 ; i++) {
        bool caminho = false;
        for (int j = 0 ; j < adj[seq[i]].size() ; j++) {
            if ( adj[seq[i]].at(j) == seq[i+1]) caminho = true;
        }
        if (caminho == false) return false;
    }
    return true;
```



5. Escreva uma função **kcaminho** que recebe dois vértices u e v e um inteiro k e verifica se existe no grafo um caminho de comprimento menor ou igual a k.

podemos buscar o menor caminho e comparar com K

```
bool Kcaminho (int u, int v, int K){
    int * distância = new int[V];
    for (
    return Rbfs(u, distancia) <= K;
```

6. Escreva uma função que recebe um inteiro n e constroi o grafo correspondente aos movimentos de um cavalo no tabuleiro de xadrez $n \times n$, ou seja, os vértices são as posIcões do tabuleiro e dois vértices são ligados por uma aresta se é possível que um cavalo vá de uma posição para a outra. Observe que este grafo é bipartido :)

   Faça um programa que determine, neste grafo, a distância do vértice correspondente ao canto superior esquerdo do tabuleiro a todas as outras posições do tabuleiro.

   Exemplo: Para o tabuleiro 4×4, as distâncias entre a posição do canto e as posições do tabuleiro são:

$$\begin{matrix} 0 & 3 & 2 & 5 \\ 3 & 4 & 1 & 2 \\ 2 & 1 & 4 & 3 \\ 5 & 2 & 3 & 2 \end{matrix}$$

7. Você pode representar uma genealogia por um grafo dirigido acíclico, em que os vértices representam as pessoas e existe um arco $u - v$ se u é pai ou mãe de v. Faça uma função `ancestralComum` que recebe três vértices u, v e w e devolve 1 se w é ancestral de u e v. Qual a complexidade da função?

```
bool   ancestralComum ( int u, int v , int w)
        bool * marked = new bool [V];
        dfsR (w , marked);
        return marked[u] && marked[v];
}
```

8. Usando a função do item anterior, faça uma função que recebe dois vértices u e v que desejam se casar, e devolve a lista dos ancestrais comuns deles. Qual a complexidade da função?

```
int * listaAncestral (int u, int v)
    vector <int> ancestrais;
    for (int w = 0 ; w < V ; w++) {
        if( w != u  && w != v ) {
            if( ancestralComum (u, v, w) {
                ancestrais.push_back (w);
            }
        }
    }
    new * lista = new int[ancestrais.size()];
    for (int i = 0, i < ancestrais.size() ; i++) {
        lista[i] = ancestrais[i] ;
    }
    return lista;
}
```

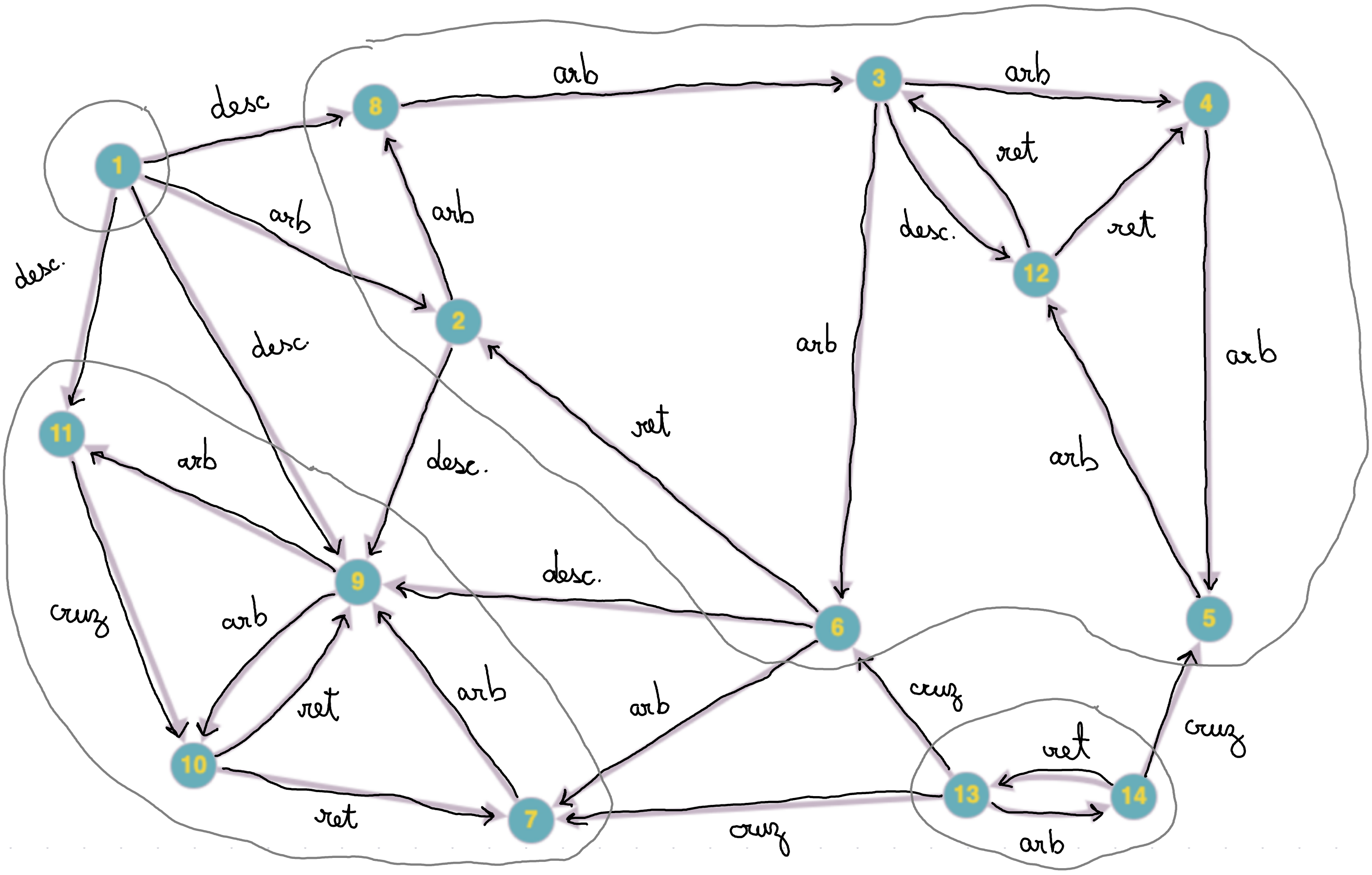
complexidade: $V(V+E)$

$V+E$ → dfsR

V → nº de vértices

9. Percorra o digrafo abaixo em profundidade (a partir do vértice de menor índice e supondo que as listas de adjacências estão ordenadas) fazendo o seguinte:

- anote o instante de tempo em que começa e termina o percurso dfs de cada vértice;
- identifique cada arco como arco da arborescência dfs, descendente, retorno, ou cruzado.
- identifique as componentes fortemente conexas do digrafo.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	→ vértice
pre	1	2	4	5	6	11	12	3	13	14	16	7	25	26	→ instante que chega

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	→ vértice
pos	24	23	21	10	9	20	19	22	18	15	17	8	28	27	→ instante que acaba

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	→
low	1	2	~~4~~	5	~~6~~	~~11~~	12	~~3~~	~~13~~	~~14~~	~~16~~	~~7~~	25	~~26~~	→
			2	4	4	2		2	12	12	12	4		25	

1 → 2 → 8 → 3 → 4 → 5 → 12

3 → 6 → 7 → 9 → 10

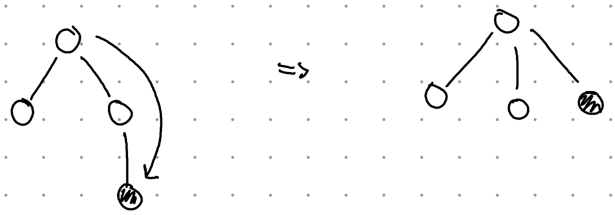
9 → 11

13 → 14

10. Considere agora o percurso bfs e a mesma classificação dos arcos. Podem existir arcos descendetes, de retorno e cruzados? Justifique.

dois tipos de cruzado e nenhum descendente

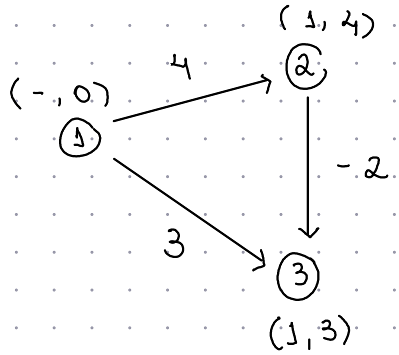
Arcos de retorno podem existir, cruzados também, no entanto, descendentes não, pois se há caminho de cima para esse vértice mais baixo, ele já teria sido visto antes.



11. Mostre que o algoritmo de Dijkstra pode dar resultados incorretos na presença de arcos de custo negativo.

os pesos das arestas devem ser não-negativos, pois o algoritmo pressupõe que uma vez que o vértice é fechado, não há um caminho menor para chegar até ele, assim, se há uma aresta com peso negativo na frente que faça o caminho de um certo vértice anterior ser menor, esse caminho não será atualizado, pois o vértice já estará fechado.

exemplo:



procuramos todos os vértices adjacentes de 1 e relaxamos, vamos ter:

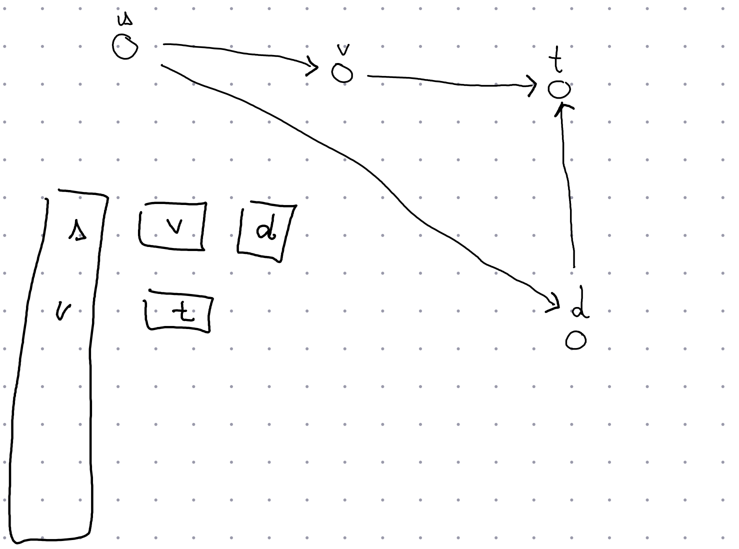
- 2 com precedente 1 e caminho de peso 4
- 3 com precedente 1 e caminho de peso 5

fechamos 1 e procuramos o próximo vértice aberto com menor caminho, que é o 3, fechamos o 3 e como não há nenhum adjacente, procuramos o próximo vértice aberto com menor caminho, que é o 2, fechamos o 2 e relaxamos seus adjacentes abertos, como 3 já foi fechado, seu caminho, que é 3, não será atualizado para 2, sendo assim, o valor do menor caminho até 3 está incorreto.

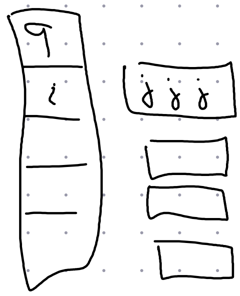
12. Considere um país situado em um arquipélago em que o presidente mandou construir diversas pontes de mão única ligando as diferentes ilhas do arquipélago. O ministro da infraestrutura está, agora, preocupado com a segurança do país frente a desastres naturais. Faça uma função que, dadas duas ilhas s e t, encontre, se existir, uma ponte no grafo, cuja remoção aumenta a distância entre s e t. Qual a complexidade da sua função?

```
pair <int, int>  aumentaDistancia (int s, int t) {
    int distanciaOriginal = dijkstra (s,t);
    for (int i = 0; i < adj.size(); i++) {
        for (int j = 0; j < adj[i].size(); j++) {
            if (adj[i].at(j) != t) remove (i, adj[i].at(j));
            int novaDistancia = dijkstra (s,t)
            if (novaDistancia > distanciaOriginal) return i, j;
        }
    }
    return -1, -1;
}
```

complexidade: $O(\underbrace{V^2}_{\text{2 for}}(V+E)\underbrace{\log V}_{\text{DijKstra}})$



13. Chamamos de *clique* de um grafo, um conjunto S de vértices em que todos são vizinhos de todos, ou seja, para todo par $u, v \in S$, $u-v$ é aresta do grafo. Faça uma função que receba um inteiro k e devolva, se existir, uma clique com k vértices de um grafo. Qual a complexidade de sua função?




```
bool clique (int K) {
    int contador = 0;
    for (int q = 0; q < V; q++) {
        for (int i = 0; i < V; i++) {
            if (q != i) {
                for (int j = 0; adj[i].size(); j++) {
```

14. Considere agora um grafo formado a partir de uma rede social, em que dois vértices estão ligados se as pessoas correspondentes são "amigas" nesta rede social. Use a função acima para encontrar o tamanho da maior clique do grafo. Qual a complexidade de seu algoritmo?

```
int maiorClique ( ) {
    for (int i = V; i >= 0 ; i++ ) {
        if(clique(i)) return i;
    }
    return 1;
}
```