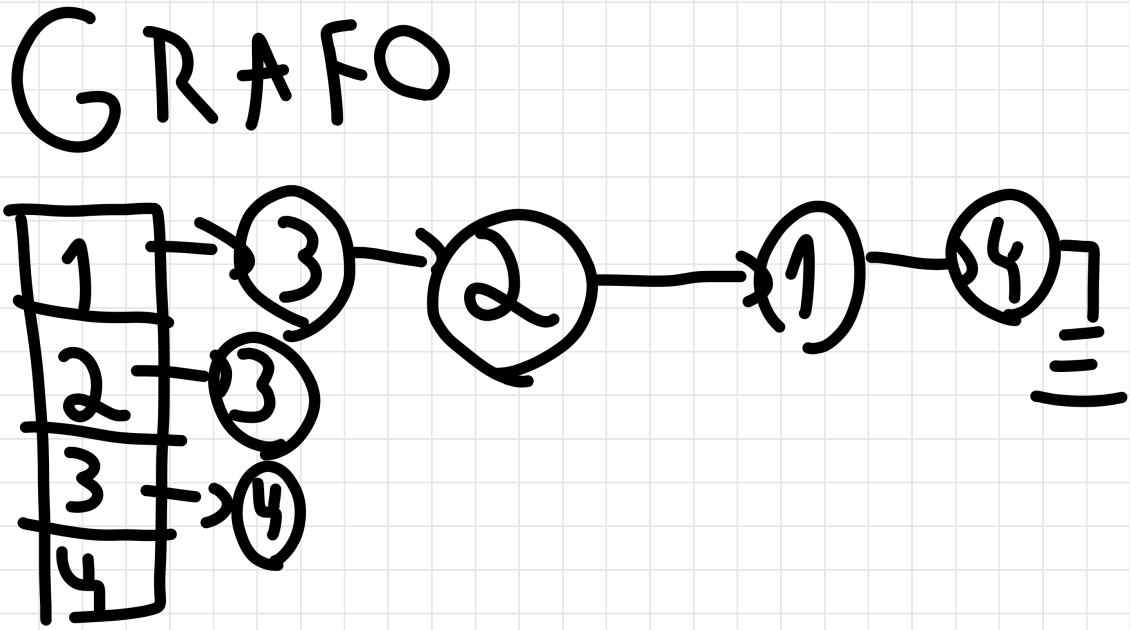
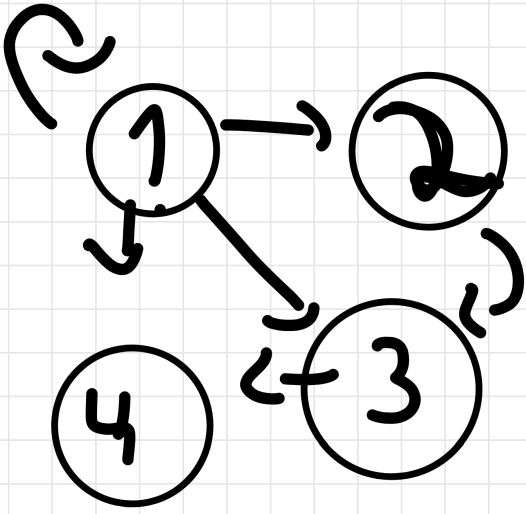
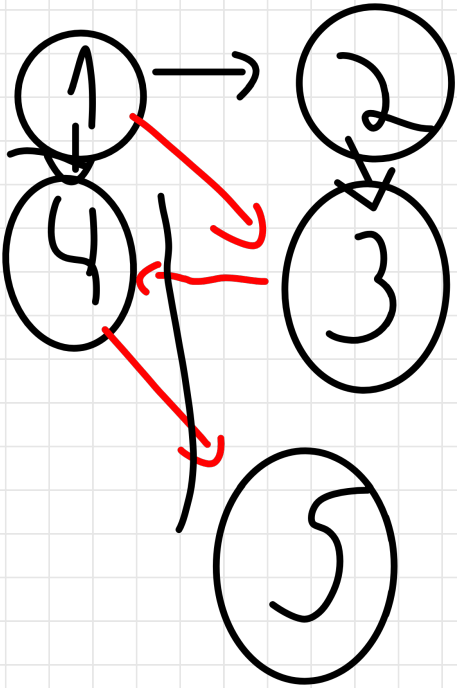


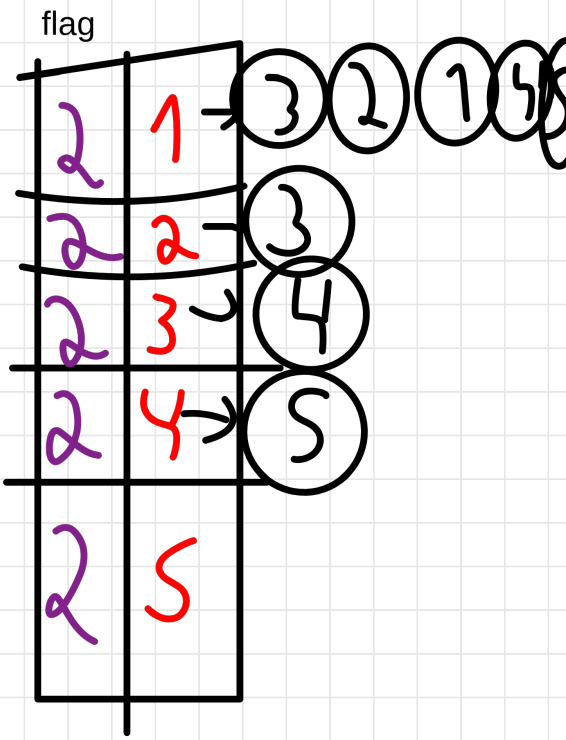
Vetor de Listas Ligadas



Busca em Grafo

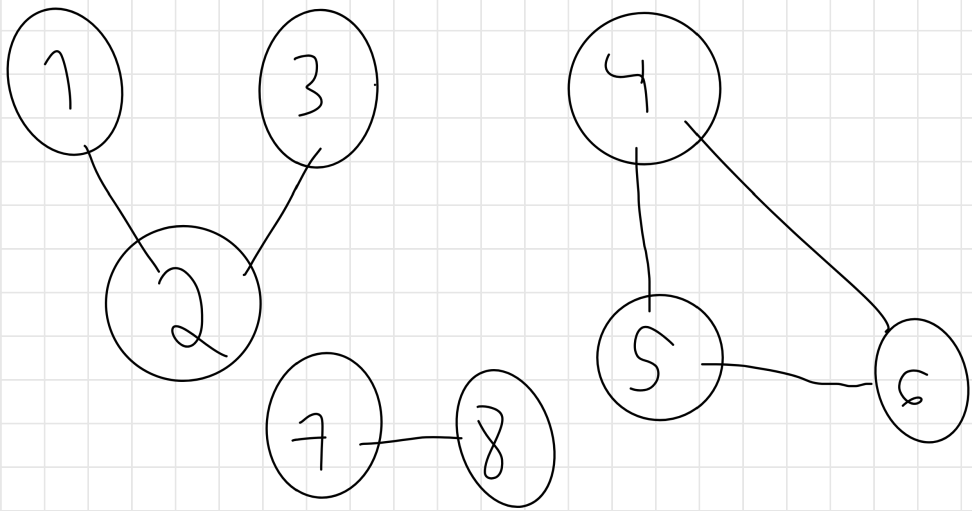
OBJ: a partir de um vertice inicial, percorre todos os vertices alcançaveis. Int flag é um auxiliar para controle da busca, para evitar repetição





Passos de resolver problemas com grafos:

1. Modelar o grafo:



2. É de busca?

1

para ser um problema de busca, eu tenho que estar preocupado se há conexões a partir de certos elementos

3. Se for de busca, escolher entre profundidade x largura:

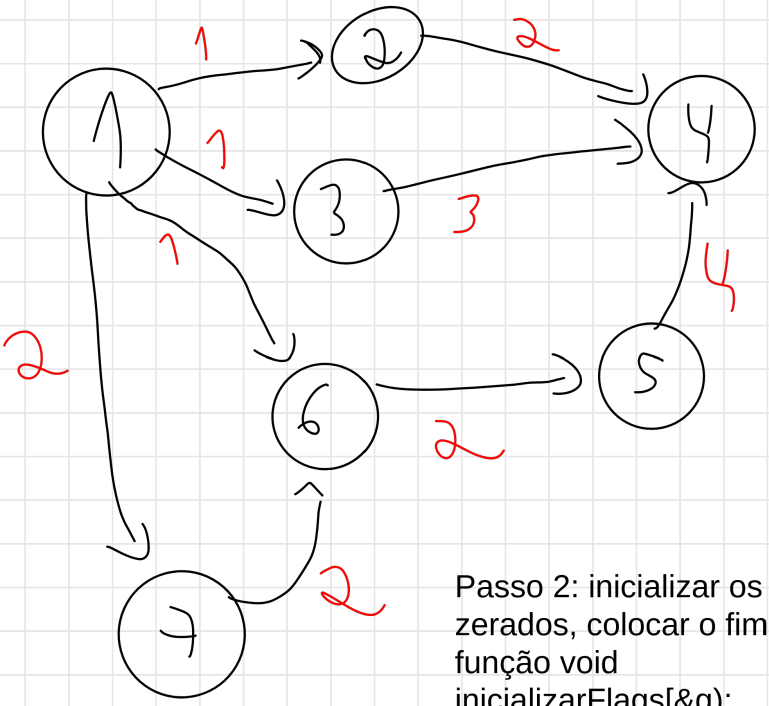
no caso de profundidade, oque eu faço com os outros desconexos? simples, inicie a busca pelo proximo NO que possui flag = 0

NO	flags
1	2
2	2
3	2
4	0
5	0
6	0
7	0
8	0

EX 2

```
ZeroFlags(Vertice* g){
        int i = 0;
        int j;
        for(j = 0; j <= V; j++){
                if(g[j].flag == 0){
                        i = j;
                        break;
        }
        }
        if(i == 0) return NULL;
        prof(g, i);
        printf("\n");
}
```

EX 3



Minha área:

Dado uma origem **i**, e um destino **j**, e cia áerea **C**, verifique se é possivel voar **i => j** usando apenas vôos de **C**

Passo 1: mude o struct de NO:

```
typedef struct s{
        struct s* prox;
        int adj;
        int cia;
}NO;
```

Passo 2: inicializar os flags para todos serem zerados, colocar o fim como falso, e fazer uma função void
inicializarFlags[&g);
bool fim = false;

```
void prof(Vertice* g, int i, int j, int c, bool* fim){
        g[i].flag = 1;
        if(*fim) return;
        NO* p = g[i].inicio;
        while(p){
                if(p->cia == c && g[p->adj].flag == 0){

                        prof(g, p->adj, j, c, fim);
                        if(*fim) return;
                }
                p = p->prox;
        }
        g[i].flag = 2;
                if(p->adj == j) *fim = true;

}
```

Mas as vezes, no caminho não queremos transitar por um lugar especifico, para evitar isso basta colocar um campo "percorrível" no struct vertice

EX 4: encontrar caminho i => j sem passar por zonas invalidas

Passo 1: mudar o Struct vertice

```
typedef struct Vertice{
        NO* inicio;
        int flag;
        bool valido;
};
```

Passo 2: re fazer a função prof sabendo que:
I) não há mais parametro c (pois não é limitado a apenas uma companhia)
II) testar se g[p->adj].valido é true (ou seja, só mais uma checagem)

Passo 3, função:

```
void prof(Vertice* g, int i, int j, bool* fim){
        g[i].flag = 1;
        if(*fim) return;
        NO* p = g[i].inicio;
        while(p){
                if(g[p->adj].valido == true && g[p->adj].flag == 0{
                        prof(g, p->prox, j, fim);
                        if(p->adj == j) *fim = true;
                                if(*fim) return;
                        prof(g, p->prox, j, fim);
                        if(*fim) return;
                }
                p = p->prox
        }
        if(p->adj == j) *fim = true;
}
```

Ex 5 : Quantas horas de Voo entre i e j?

PS: Não é o custo mínimo, é só o tempo de qualquer caminho que for encontrado

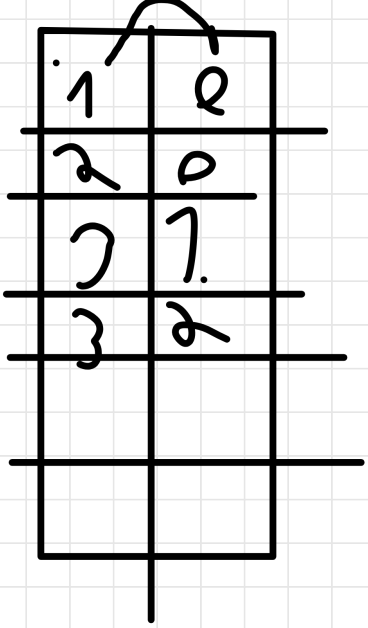
Passo 1: altere o struct do NO(pois a propriedade de custo é do vôo, e não do local) e coloque o campo duração no struct do Vertice (para guardar a informação de quanto tempo demora pra chegar do ponto de origem até aquele local)

```
typedef struct s {
    struct s* prox;
    int adj;
    int horas;
}
typedef struct Vertice{
    NO* inicio;
    int flag;
    int duracao
}
```

Passo 2: re-faça a função de busca por profundidade:

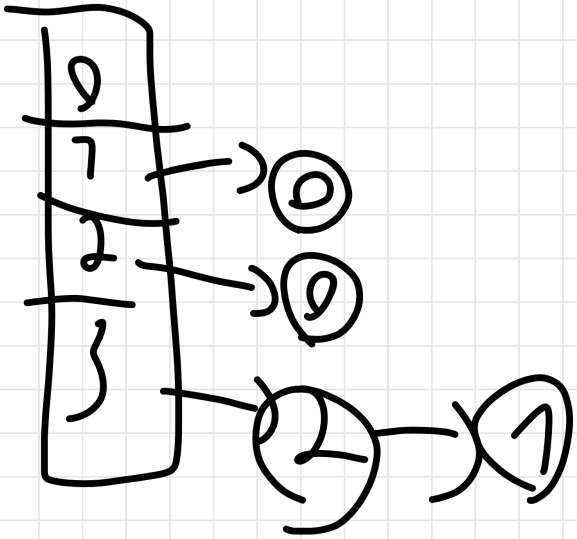
```
int x;
for(x = 0; x <= V; x++) g[x].duracao = 0;
inicializarFlags(&g);
bool fim = false;

void prof(Vertice* g, int i, int j, bool* fim){
    g[i].flag = 1;
    if(*fim) return;
    NO* p = g[i].inicio;
    while(p){
        if(g[p->adj].flag == 0){
            g[p->adj].duracao = g[i].duracao + p->horas;
                if(p->adj == j) *fim = true;
                if(*fim) return;
            prof(g, p->prox, j, fim);
            if(*fim) return;
        }
        p = p->prox
    }
}
```

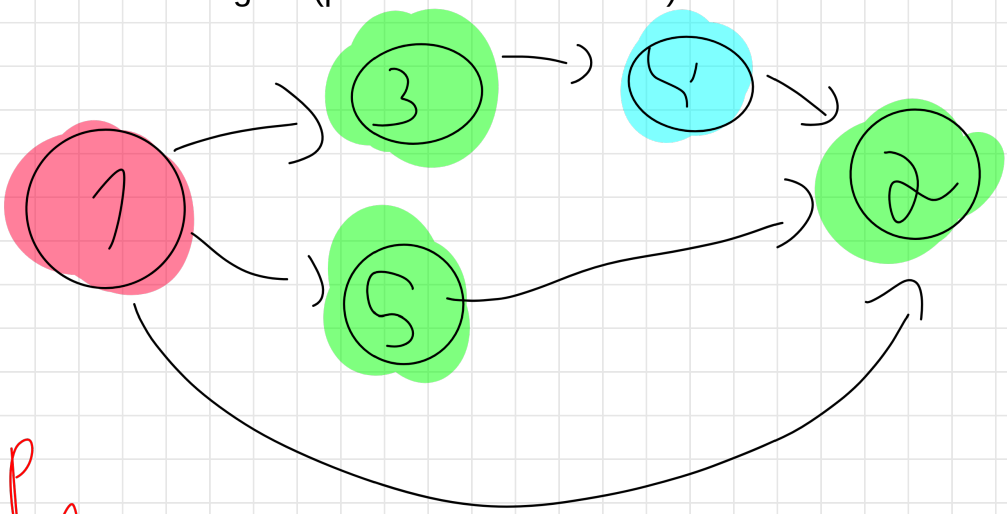


Q 4

Q 123



Busca em Largura (p/ caminho mais curto)



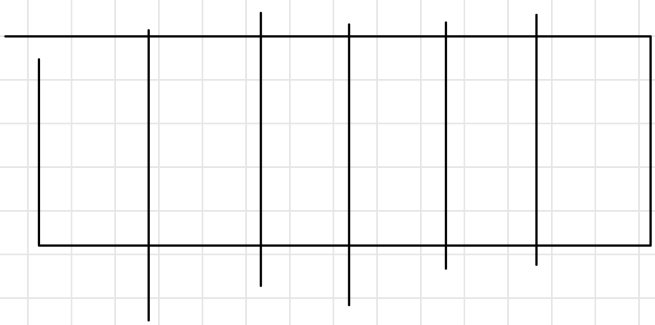
P1

Busca em nivel

P2

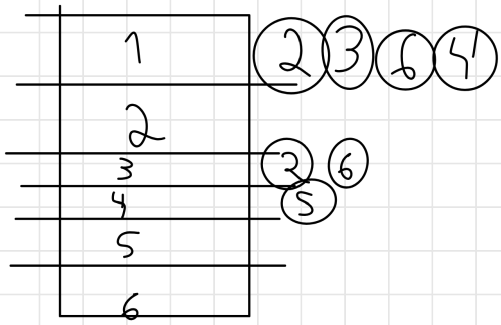
P3

Fila f



Passos:
1. Entrar em i e marcar flag = 1
2. enquanto houver fila
3. i = sair;
4. marcar flag como 2
5. enfileira o entrar na fila todos os adjascentes não descobertos
6.repete

```
void largura(Vertice* g, int i){    zerarFlags(g);
Fila* f;
inicializarFila(&f);
EntrarFila(&f, i);
g[i].flag = 1; //discovered
while(f){
        i = SairFila(&f);
        g[i].flag = 2;
        NO* p = g[i].inicio;
        while(p){
                if(g[p->adj].flag == 0){
g[p->adj].flag = 1;
                        EntrarFila(&f, p->adj);
                }
                p = p->prox;
        }
}
}
```

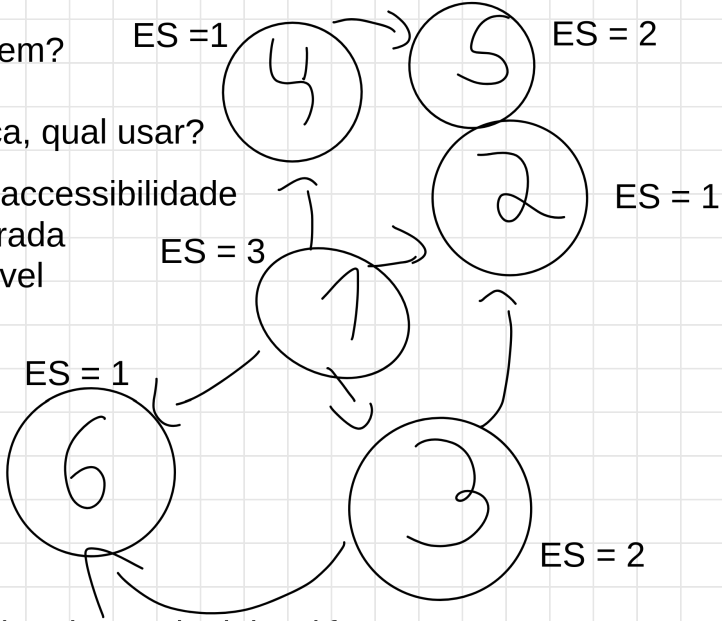


Ordem de conclusao (flag == 2)
Profundidade: 2, 6, 3, 5, 4, 1
Largura: 1, 2, 3, 6, 4, 5

Ordem de descoberta:
Profundidade: 1, 2, 3, 6, 4, 5
Largura: 1, 2, 3, 6, 4, 5

(I) modelagem?
(II) busca?
(III) se busca, qual usar?

ES 1 = Sem accessibilidade
ES 2 : Moderada
ES 3: Acessivel



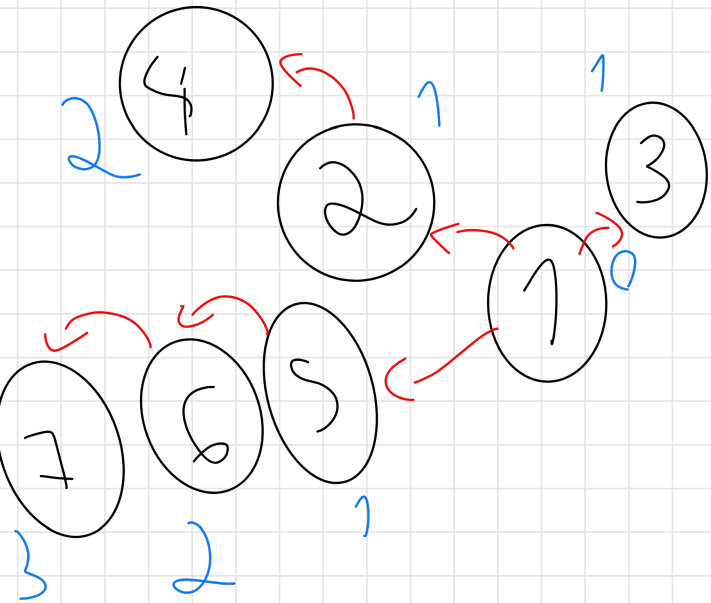

```
int largura(Vertice* g, int i, int x){
zerarFlags(g);
Fila* f;
inicializarFila(&f);
EntrarFila(&f, i);
g[i].flag = 1; //discovered
while(f){
        i = SairFila(&f);

        g[i].flag = 2;
        if(g[i].es == x){
                while(f) SairFila(&f);
                return i;
        }
        NO* p = g[i].inicio;
        while(p){
                if(g[p->adj].flag == 0){
g[p->adj].flag = 1;
                        EntrarFila(&f, p->adj);
                }
                p = p->prox;
        }
}
}
```

Encontrar o primeiro elemento com o ES solicitado

Busca em Largura final

(i) modular (tipo no vertice)
(ii) busca? sim
(iii) lagura x prof?



Problema, achar todos os vertices que estao a uma distancia x do vertice 1

nivel | flag | tipo

	1	0	1
	0	0	2
	0	1	3
	0	1	4
	0	0	5
	0	0	6
	0	0	7