Grafos – parte 1

SCC 218 – Alg. Avançados e Aplicações

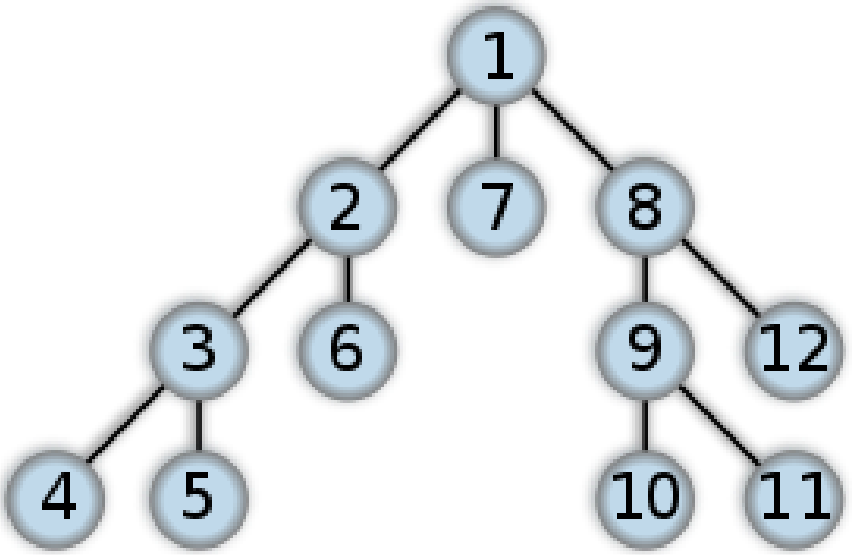
João Batista

Grafos

- Muitos problemas reais estão relacionados a grafos
- Alguns tópicos são vistos no curso de graduação, outros nem tanto.
  - Graph traversal (busca em grafo)
  - Minimum spanning tree (árvore geradora mínima)
  - Shortest path (caminho mínimo)
  - Network flow (fluxo em rede)
- Vamos assumir que você já conhece
  - Matriz de adjacência, lista de adjacência, lista de arestas, etc.

Busca em profundidade

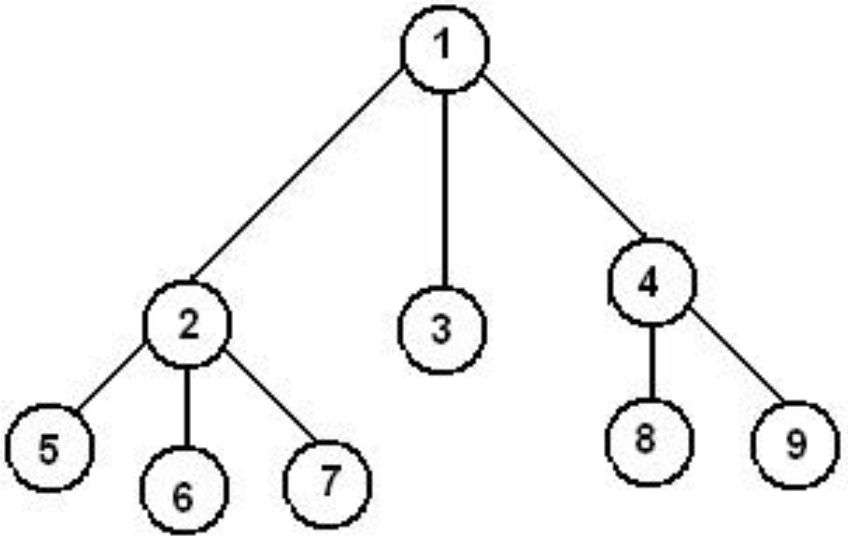
- Depth First Search (DFS)



- Para a árvore acima:
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Busca em profundidade

- Depth First Search (DFS)



- Para a árvore acima:
  - 1, 2, 5, 6, 7, 3, 4, 8, 9

```cpp
#define UNVISITED 0
#define VISITED 1

using namespace std;


// estruturas uteis para representacao de grafos...
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;



// vetor de vertices
vi vertices;

// a lista de adjacencia e um vetor de vetor de pares.
// cada par contem o vertice adjacente e o peso da aresta
// assumimos que o peco aqui eh inteiro...
vector<vii> AdjList(100);
```

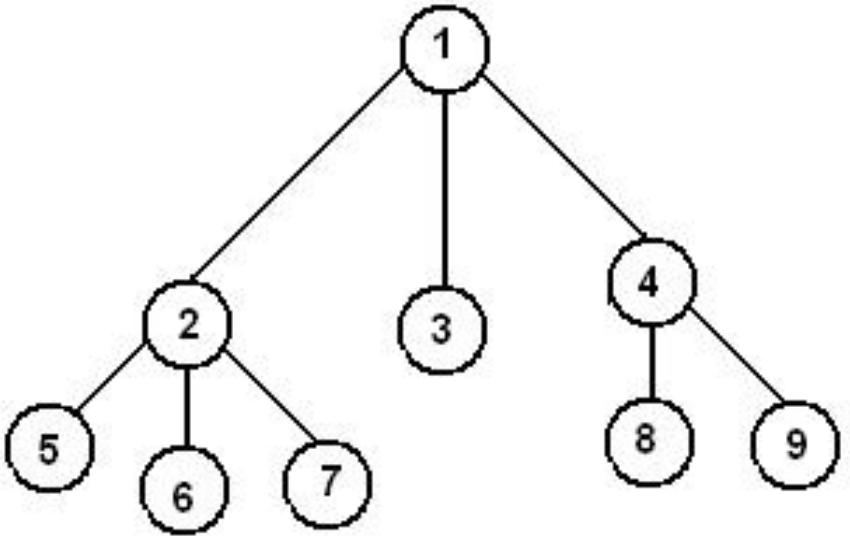
```cpp
// busca em profundidade a partir de um vertice u
void dfs(int u){
    vertices[u] = VISITED;
    printf("%d\n", u+1);

    for(int j=0; j<AdjList[u].size(); j++){
        ii v = AdjList[u][j];  // v  eh um vertice do tipo (vizinho de u, peso)
        if (vertices[v.first] == UNVISITED)
            dfs(v.first);
    }
}
```

Lendo o grafo...

```
int main (){
    int u,v;      // edge pair...
    int n;       // # of vertices...

    scanf("%d\n", &n);
    printf("%d\n", n);
    for (int i=0; i<n; i++){
        vertices.push_back(UNVISITED);
        //AdjList.pushback(0);
        printf("%d ", vertices[i]);
    }
    printf("\n");

    while (scanf("%d %d", &u, &v) != EOF) {
        AdjList[u-1].push_back(make_pair(v-1,0));
        AdjList[v-1].push_back(make_pair(u-1,0));

    }
    dfs(0);

    return 0;
}
```

•Busca em largura

- Breadth First Search (BFS)



- Para esta arvore
  - 1, 2,3,4,5,6,7,8,9

```cpp
// busca em profundidade a partir de um vertice u
void bfs(int u){
    queue<int> q;
    q.push(u);

    while (!q.empty()){
        //  retira da fila e marca como visitado..
        int k = q.front(); q.pop();
        vertices[k] = VISITED;

        printf("%d\n", k+1);

        // para todo adjacente, nao visitado poe na fila...
        for(int j=0; j<AdjList[k].size(); j++){
            ii v = AdjList[k][j];  // v  eh um vertice do tipo (vizinho de u, peso)
            if (vertices[v.first] == UNVISITED)
            q.push(v.first);
        }
    }

}
```

Encontrando componentes Conectados

- Observe o código de bfs e dfs. Se o grafo não for conectado o que acontece??
- Dê uma solução para achar o nro de componentes conectados em um grafo usando estes códigos!

```
int numCC = 0;
for (int i=0; i<n; i++){ // para todos os vertices..
    if (vertices[i] == UNVISITED) {
        printf("CC %d:", ++numCC);
        dfs(i);
        printf("\n");
    }
}
```

Basta chamar DFS (ou BFS) para todos os vértices do grafo!!!! Assim teremos o # de componentes conectados do grafo !

Ordenação Topológica

O grafo abaixo tem diversas ordenações topológicas possiveis:

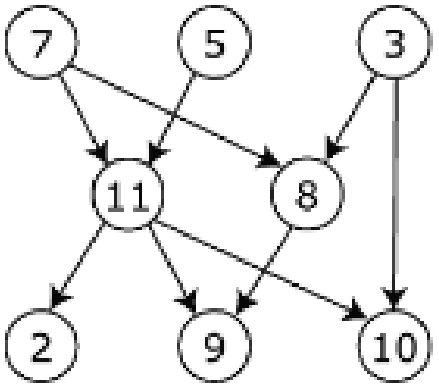


Figura por Derrick Coetzee

- 7, 5, 3, 11, 8, 2, 9, 10 (visual esquerda-para-direita, de-cima-para-baixo)
- 3, 5, 7, 8, 11, 2, 9, 10 (vértice de menor número disponível primeiro)
- 3, 7, 8, 5, 11, 10, 2, 9
- 5, 7, 3, 8, 11, 10, 9, 2 (menor número de arestas primeiro)
- 7, 5, 11, 3, 10, 8, 9, 2 (vértice de maior número disponível primeiro)
- 7, 5, 11, 2, 3, 8, 9, 10

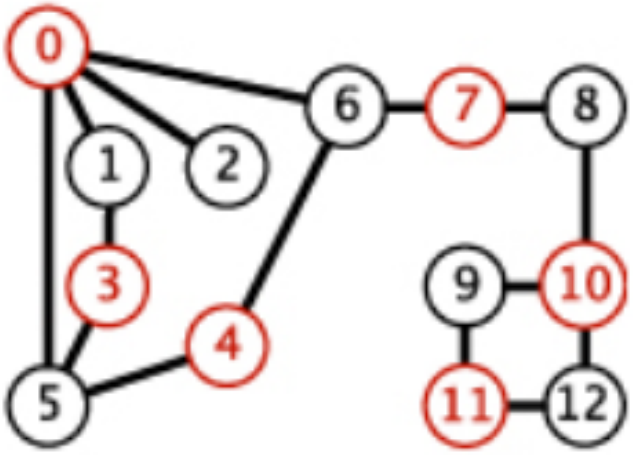
Implementação Ord.Top.

- Uma pequena modificação do DFS
  - Criar um vetor que armazena os vertices na seguinte ordem: das folhas para a raiz (usando DFS)
  - Imprimir o vetor de tras para frente

```cpp
// vetor de vertices
vi vertices;

vi ts; // armazena os vertices visitados em ordem inversa...

// a lista de adjacencia e um vetor de vetor de pares.
// cada par contem o vertice adjacente e o peso da aresta
// assumimos que o peco aqui eh inteiro...
vector<vii> AdjList(100);


// busca em profundidade a partir de um vertice u
void dfs2(int u){
    vertices[u] = VISITED;

    for(int j=0; j<AdjList[u].size(); j++){
        ii v = AdjList[u][j];  // v  eh um vertice do tipo (vizinho de u, peso)
        if (vertices[v.first] == UNVISITED)
            dfs2(v.first);
    }
    ts.push_back(u);
}
```

```
for (int i=0; i<n; i++){ // para todos os vertices..
    if (vertices[i] == UNVISITED)
        dfs2(i);
}

for (int i=ts.size()-1; i>=0; i--) // para todos os vertices..
    printf(" %d",ts[i]);
printf("\n");
```

Grafo Bipartido (bipartite)



[Copiado de 'Algorithms', 4th.ed., de Sedgewick e Wayne]

Grafo bipartido

- Várias aplicações, mas veremos como simplesmente verificar se o grafo é bipartido ou não.
- Podemos usar BFS ou DFS, embora o primeiro pareça ser mais natural.
  - Colorir vertice fonte v com 0 e os da segunda camada (vizinhos de v) com 1; e assim sucessivamente alternando cores 0 e 1.
- Se houver violação (aresta(u,v) onde u e v tem as mesmas cores, então o grafo não é bipartido..

```
bool bipartite(int u){
    queue<int> q;
    q.push(u);
    vi color(NumVert, SEMCOR);
    color[u] = 0;  // comeca com cor 0
    bool isBipartite = true;

    while (!q.empty()){
        //  retira da fila e marca como visitado..
        int k = q.front(); q.pop();
        printf("%d\n", k);

        // para todo adjacente, nao visitado poe na fila...
        for(int j=0; j<AdjList[k].size(); j++){
            ii v = AdjList[k][j];  // v  eh um vertice do tipo (vizinho de u, peso)
            if (color[v.first] == SEMCOR){
                color[v.first] = 1 - color[k];
                q.push(v.first);
            } else if (color[v.first] == color[k]){
                return (false);
            }
        }
    }
    return (true);

}
```