Graph Algorithms

Benjamin Schmid

Swiss Olympiad in Informatics

November 6, 2016

Table of Contents

Traverse a graph

- Visit all nodes in a graph
- Check whether all nodes are connected
- Find connected components
- ...

Traverse a graph

- Visit all nodes in a graph
- Check whether all nodes are connected
- Find connected components
- ...

Two main algorithms

DFS and BFS

Shortest path

- Find shortest path between two nodes
- Find shortest path from one node to all other nodes
- Many real life applications
- And many applications in our tasks ;)

Shortest path

- Find shortest path between two nodes
- Find shortest path from one node to all other nodes
- Many real life applications
- And many applications in our tasks ;)

One important algorithm

Dijkstra's shortest path algorithm

DFS

- Start at arbitrary node
- Walk as far as possible
- Mark all discovered nodes
- If we can't find new undiscovered nodes, turn around
- Like searching the exit in a labyrinth

Visulization

Visualization

DFS

Time Complexity

$\mathcal{O}(V + E)$

Implementation

- `vector<vector<int>> graph` stores graph as adjacent list
- `vector<int> visited` to mark discovered nodes
- recursive function `void dfs(int v)` performs DFS starting from vertex `v`

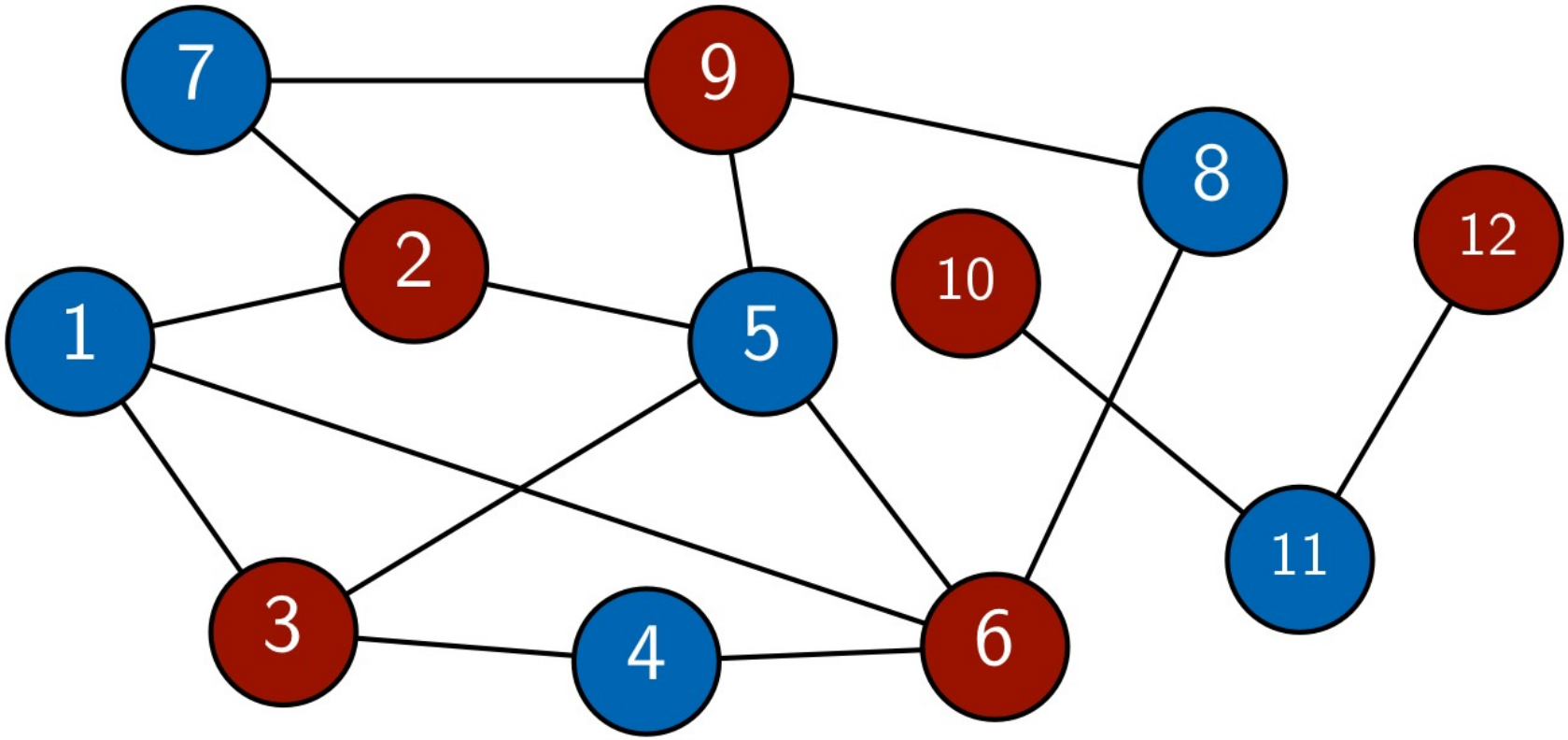
Implementation

```
vector<vector<int>> graph;
vector<int> visited;
void dfs(int v) {
  // We are at node v and explore neighbors
  for(int e = 0; e < graph[v].size(); e++) {
    int w = graph[v][e];
    // Check whether neighbor w
    // is not yet visited
    if(!visited[w]) {
      visited[w] = true;
      dfs(w); // recursive call
    }
  }
}
```

Sample Problem

- Given an undirected graph
- Two colors available
- No two neighbors may have the same color
- Is it possible to give each node a color?

Sample Problem



BFS

- Similar to DFS
- Instead of visiting a node the moment we discover it, we add it to a queue
- Choose start node
- Visit all neighbors of start node
- Visit all neighbors of neighbors of start node
- ...
- In step i we visit all not yet discovered nodes that we discovered in step i-1

BFS

- Allows determining shortest path
- If a node is discovered in step i,
  the shortest path has a length of i
- If there would have been a shorter path,
  we would have discovered the node already in a previous step

Visulization

Visualization

BFS

Time Complexity

$\mathcal{O}(V + E)$

Implementation

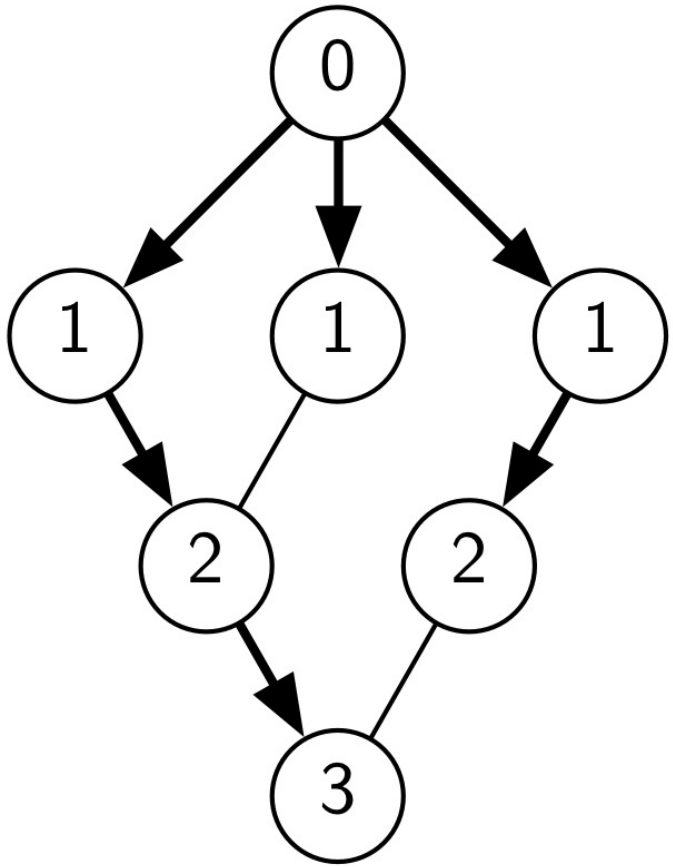
- `vector<vector<int>> graph` stores graph as adjacent list
- `vector<int> visited` to mark discovered nodes
- `queue<int> Q` to store the next nodes

Implementation

```
void bfs(int start) {
  queue<int> Q;
  Q.insert(start);
  while(!Q.empty()) {
    int v = Q.front(); Q.pop();
    for(int e = 0; e < graph[v].size(); e++) {
      int w = graph[v][e];
      if(!visited[w]) {
        visited[w] = true;
        Q.insert(w); // replaces recursive call
      }
    }
  }
}
```

Sample Problem

- Given a connected, undirected graph
- Find length of shortest path to each node from a given starting point



Dijkstra's shortest path algorithm

- BFS can only calculate the shortest path if all edges have the same length (so called unweighted graph)
- Dijkstra's Algorithm can be used on a weighted graph
- Calculates all shortest paths from a given start point
- Often only path from A to B are of interest

Ants

- Imagine an ant colony
- All ants walk at the same speed
- Thousands of ants start at the anthill (start node)
- Each time there are multiple paths the ants split
- If a path was already taken by an other group it is ignored
- If an ant reaches an undiscovered node it writes the passed time next to the node

Visualization

Visualization

Ants to Code

How do we efficiently simulate the ants?

- Only arrivals of ants at a node are of interest
- Once an ant starts walking we know when it will arrive

Algorithm

1. Place ant at start
2. Take ant that will arrive next
3. If it is the first ant at this node note the found shortest path, add an ant for each outgoing path and calculate when it will arrive. Add the ant at the corresponding arrival time to the list
4. Go to 2 as long as there are still ants walking

Visulization

Visualization

Dijkstra's shortest path algorithm

Time Complexity (easy to implement)

$\mathcal{O}(E \log V)$

Time Complexity (using fancy data structures)

$\mathcal{O}(V \log V + E)$

Implementation

```
int dijkstra(int start, int destination){
  vector<int> mindist(graph.size(), INT_MAX);
  vector<bool> visited(graph.size(), false);
  priority_queue<pair<int, int>> pq;
  pq.push({0, start});
  mindist[start] = 0;
  while(!pq.empty()){
    int active = pq.top().second;
    pq.pop();
    if(visited[active]){
      continue;
    }
    if(active == destination){
      return mindist[destination];
    }
    visited[active] = true;
    for(auto edge : graph[active]){
      if(mindist[active] + edge.second < mindist[edge.first]){
        mindist[edge.first] = mindist[active] + edge.second;
        pq.push({-mindist[edge.first], edge.first});
      }
    }
  }
  return INT_MAX;
}
```

"Beautiful" Implementation by Timon

```
#include <queue>
using namespace std;

#define OG(T) bool operator>(const T& o)const
struct E{ int t,w; OG(E){ return w>o.w; } };
using VE=vector<E>;
struct V{ VE in; bool v; };
using VV=vector<V>;
using HE=priority_queue<E,VE,greater<E>>;

int dijkstra(VV& g,int s,int t){
  HE q; q.push({s,0});
  while(!q.empty()){
    E c=q.top(); q.pop();
    if(g[c.t].v) continue;
    g[c.t].v=1;
    if(c.t==t) return c.w;
    for(auto&e:g[c.t].in)
      q.push({e.t,c.w+e.w});
  }
  return INT_MAX;
}
```

Implementation using Set by Michal Forišek

```
struct edge { int to, length; };

int dijkstra(const vector< vector<edge> > &graph, int source, int target) {
  vector<int> min_distance( graph.size(), INT_MAX );
  min_distance[ source ] = 0;
  set< pair<int,int> > active_vertices;
  active_vertices.insert( {0,source} );

  while (!active_vertices.empty()) {
    int where = active_vertices.begin()->second;
    if (where == target) return min_distance[where];
    active_vertices.erase( active_vertices.begin() );
    for (auto ed : graph[where])
      if (min_distance[ed.to] > min_distance[where] + ed.length) {
        active_vertices.erase( { min_distance[ed.to], ed.to } );
        min_distance[ed.to] = min_distance[where] + ed.length;
        active_vertices.insert( { min_distance[ed.to], ed.to } );
      }
  }
  return INT_MAX;
}
```

Tasks

Training tasks in the grader:

- components (DFS / BFS)
- shortestpath (BFS)
- mole (Dijkstra)