Technische Universität München
Institut für Informatik
Theoretische Informatik

WS 2015/16
Worksheet

Fundamental Algorithms

Exercise 1 – Hypergraphs

A *hypergraph* extends the concept of a graph in the sense that edges are allowed to connect an arbitrary number of vertices (instead of exactly two). Hence, a *hypergraph* is defined as a tuple (V, H), where V is a set of vertices and H is a set of *hyperedges*, where $H \subset \mathcal{P}(H) \setminus \{\emptyset\}$, with $\mathcal{P}(H)$ the power set (i.e., the set of all possible subsets) of H.

Let's assume a hypergraph where V is a set of authors, and each hyperedge $h \in H$ contains all authors of a specific scientific article.

Exercise 1a)

Give a suitable definition of the concept of a *path* in a hyperedge.

Solution:

We may define that a path (of length n) exists between two vertices v and w in a hypergraph, if:

- We have a sequence of hyperedges $h_1, h_2, h_3, \ldots, h_n$
- and a sequence of vertices $v_0, v_1, v_2, \ldots, v_n$ (where $v = v_0$ and $w = v_n$)
- such that $v_0, v_1 \in h_1$ and $v_1, v_2 \in h_2$ and $\ldots v_{n-1}, v_n \in h_n$

Exercise 1b)

Given is the hypergraph $S = (V_S, H_S)$ of "all" scientific articles. The Erdös number $\mathrm{Er}(a)$ of an author $a \in V_S$ is defined as the length of the shortest path in S that connects the specific vertex $e \in V$ (e corresponds to the author Paul Erdös) to a. Write down an algorithm to determine $\mathrm{Er}(a)$.

Solution:

For a (non-directed, non-weighted) graph, breadth-first traversal will find the shortest path from a given root node to any reachable node in the graph. Hence, we adapt breadth-first search for hypergraphs. Instead of using an array Mark that just labels vertices as already visited, we use

an array Erd that contains the Erdös number of a already visited vertices (non-visited vertices will be initialized to contain -1 in the beginning).

```
BF_Erdos(e:HyperNode, x:HyperNode, n:Integer) : Integer {
  // Input: e is the vertex that corresponds to Erdos
  //        x is the vertex that corresponds to the author
  //        n is the number of nodes of the graphs
  // Output: Erdos number of x (or -1 if no path if found)

  // use an array Erd to mark visited nodes and to store their Erdos number
  Array Erd[1..n];
  for i from 1 to n do { Erd[i] = -1 };
  Erd[ e.key ] = 0;  // Erdos has Erdos number 0

  // initialize a queue 'active' for breadth-first traversal
  Queue active = {e}; // initial node is e = Erdos

  while active <> {} do
    // take first vertex from queue:
    v = remove(active);
    // if vertex is author x then return Erdos number
    if v = x then return Erd[v.key];
    // determine all authors with a joint publication
    coAuthors = {}; // start with empty set
    forall articleAuthors in v.hyperedges do
      coAuthors = coAuthors union articleAuthors
    end do;
    // expand graph search by all coauthors
    forall author in coAuthors do
      if Erd[author.key] < 0
      then // set Erdos number of author
           Erd[author.key] = Erd[v.key]+1;
           append(active, author);
      end if;
    end do;
  end while;
  // at this point, no path has been found -> no Erdos number:
  return -1;
}
```

In this algorithm, HyperNode needs to be a suitable data structure to represent a node of a hypergraph. Here, we assume that a HyperNode represent one vertex and stores the set of hyperedges that contain this vertex (or a reference to these hyperedges). Each hyperedge is a set of vertices.

Exercise 1c)

Try to formulate the problem of 1b) as a graph problem!

Solution:

There is a simple solution, once we realize that it is insignificant for the Erdös number which articles are actually responsible for the connection. Hence, we may define a graph V_S, E_S, where two authors $a, b \in V_S$ are connected by an edge, i.e. $(a, b) \in E_S$, iff there exists a hyperedge H_S with $a \in H_S$ and $b \in H_S$. Then we can use the regular breadth-first traversal on graphs.

However, scientists who are interested in their Erdös number actually do want to know which articles define the connection, so we need to set up a different graph (V, E):

- the nodes are given as $V := V_S \cup H_S$, i.e., each author and each articles define a vertex;
- E contains an edge between an author $a \in V_S$ and an article $p \in H_S$, iff a was an author of p, i.e., $a \in p$.

Now, a shortest-path search in the resulting graph (using breadth-first search) will deliver the Erdös number and the connecting papers. Note that the constructed graph is an example for a bipartite graph (see Exercise 2).

Exercise 2 – Bipartite Graphs

The following exercise is based on the concept of so-called *bipartite* graphs:

> *A graph (V, E) is called* bipartite*, if there exist V_0 and V_1 with $V_0 \subset V$ and $V_1 = V \setminus V_0$, and for all $(v, w) \in E$ there is either $v \in V_0$ and $w \in V_1$, or $w \in V_0$ and $v \in V_1$.*

To put it simpler: for a bipartite graph, it is possible to attribute each node $v \in V$ with one of two "colors", say red and black, such that any edge $e \in E$ will connect a red and a black node (and no edge will connect edges of the same color).

Exercise 2a

Give a prove to the following claim:

> *If a graph (V, E) is bipartite, then it cannot contain an odd cycle (i.e., a cycle of odd length).*

Solution:

We proof this by contradiction:
Assume that a bipartite graph contains a cycle $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n = v_0)$, where n is odd. Assume that v_0 is coloured black, then v_1 has to be red (as it is connected to the black v_0), v_2 has to be black (as it is connected to the red v_1), etc. Thus all v_{2i} are black and all v_{2i+1} are red. However, if n is odd, i.e. $n = 2i + 1$ for some i, then v_n is red, and thus cannot be equal to v_0, which is black. This contradicts our initial assumption.

Exercise 2b

Try to find an algorithm that tests whether a given graph is bipartite.

Hint: you can build such an algorithm by extending one of the graph traversals we discussed in the lecture!

Solution:

We can change breadth-first traversal into a colouring algorithm. Assume that we have n nodes with distinct key values $\{1, \ldots, n\}$, then we can use an array M to store the colouring status: 0 means "uncoloured", 1 is for black, and 2 for red.

```
BFbipartite(x:Node) {
  bipartite = true;
  ! uses queue of "active" nodes
  Queue active = { x };
  Mark[x.key] = 1;
  while active <> {} do
    ! remove first node from queue
    V = remove(active);
    ! determine the opposite colour of V:
    if Mark[V.key]=1 then newcolour=2 else newcolour=1 end if;
    ! visit all nodes W connect to V by an edge (V,W):
    forall (V,W) in V.edges do
      ! assign a colour to W, if it is still uncoloured
      if Mark[W.key] = 0
      then
        Mark[W.key] = newcolour;
        ! check all nodes connected to W for a colour conflict:
        forall (W,Y) in V.edges do
          if Mark[y.key] = newcolour then bipartite = false;
        end do;
        ! for BFT: append W to queue of active nodes:
        append(active, W);
      end if;
    end do;
  end while;
  return bipartite;
}
```

Note: instead of the assignment bipartite = **false**, we could also immediately **return false**, and thus not necessarily traverse the entire graph (and thus be much more efficient). However, we implement a full traversal here, as we will discuss a quite similar algorithm in the next exercise.

Exercise 2c

Try to give a prove for the following claim (using the algorithm from Exercise 2):

> *If a graph (V, E) is not bipartite, then it will contain an odd cycle.*

Solution:

The algorithm of Exercise 3 contains a standard breadth-first search; we can simply change it by assigning a distance to the starting node s to each node, instead of marking it as black or red. We obtain the following algorithm:

```
BFdistance(x:Node) {
   ! array Mark contains value −1 in every element at start
   bipartite = true;
   ! uses queue of "active" nodes
   Queue active = { x };
   Mark[x.key] = 0;
   while active <> {} do
      ! remove first node from queue
      V = remove(active);
      ! determine the opposite colour of V:
      dist := Mark[V.key];
      ! visit all nodes W connect to V by an edge (V,W):
      forall (V,W) in V.edges do
         ! assign a distance to W, if it is still uncoloured
         if Mark[W.key] = −1
         then
            Mark[W.key] = dist+1;
            ! for BFT: append W to queue of active nodes:
            append(active, W);
         end if;
      end do;
   end while;
   return bipartite;
}
```

Note: as 0 is a valid distance (which is correct for the start node x), we have to use a different value (which is -1) to mark nodes that have not yet been visited.

Due to the identical breadth-first traversal structure of algorithms BFbipartite and BFdistance, it is obvious that nodes with an odd distance are marked red, while nodes with an even distance are marked black.

We now have two opposite situations:

1. There is no edge in the graph that connects two nodes with even distance (which would both be marked black by algorithm 2), and no edge that connects two nodes with odd distance (which would be red). In that case, we have a bipartite graph.
2. There is an edge in the graph that connects two nodes with odd or two nodes with even

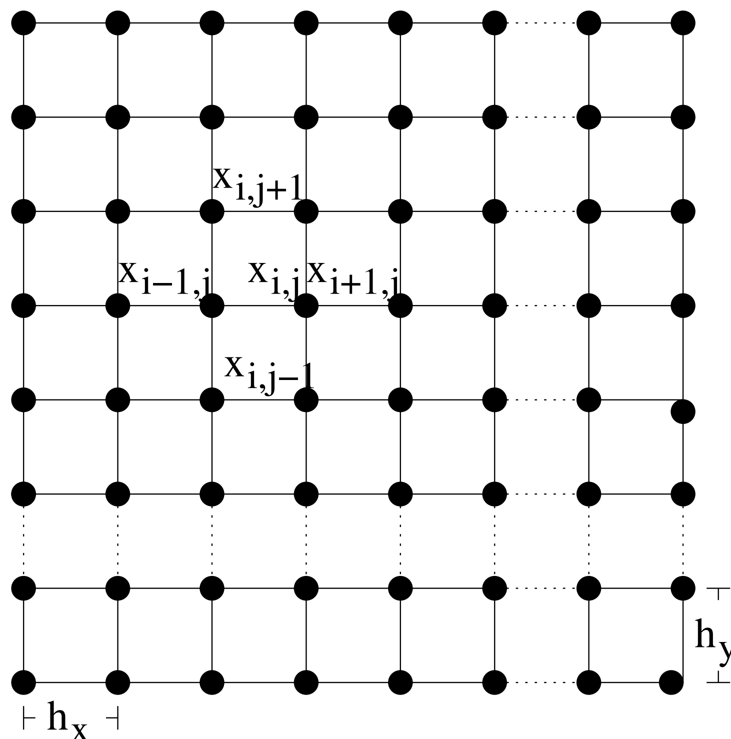
distances. As both nodes are connected to the starting node (via different paths) the edge between them generates a cycle. The length of this cycle is 1 plus either the sum of two odd numbers or the sum of two even numbers. In any case, the cycle length is an odd number.

Hence, a graph can either be bipartite or have an odd cycle.

Reference for Exercises 1a–1c:
Kleinberg, Tardos: *Algorithm Design*, Pearson Education, 2006.

Exercise 2d

Consider the following graph obtained from a Cartesian discretization mesh. Is this a bipartite graph?



Solution:

This algorithm is, of course, inspired by the so-called *red-black ordering* frequently used in numerical algorithms (such as the red-black Gauß-Seidel relaxation). If we colour the grid points in a red-black checkerboard fashion. then the four direct (left, right, top, down) neighbours of each black node will be red (and vice versa). Consider a system of equations where unknowns (the x_{ij} in the image) only depend on their direct neighbour. For relaxation methods, an important consequence is then that for an update of a "red" unknown x_{ij}, only the values of "black" unknowns (in addition to x_{ij} itself) are accessed. Hence, all red unknowns may be updated in parallel.