Tirgul 7

- Review of graphs
- Graph algorithms:
  - BFS

  (next tirgul)
  - DFS
  - Properties of DFS
  - Topological sort

Graph – a definition:

- A directed graph, ***G***, is a couple (***V,E***) such that ***V*** is a finite set and ***E*** is a subset of $V \times V$. The set ***V*** is denoted as the vertex set of ***G*** and the set E is denoted as the edge set of ***G***. Note that a directed graph may contain self loops (an edge from a vertex to itself).
- In an undirected graph, the edges in ***E*** are not ordered, in the sense of that an edge is a set {***u***,***v***} instead of an ordered couple (***u***,***v***).

Some important graph definitions:

- **Sub-graph**: Let G(V,E) be a graph. We say that G'(E',V') is a *sub-graph* of G if $V' \subseteq V$ and $E' \subseteq E \cap V' \times V'$
- **Path**: Let u,v be vertices in the graph. A *path* of length k between u and v is a sequence of vertices, $v_0,\ldots,v_k$, such that $v_0=v$, $v_k=u$, and for each $i \in \{0..k-1\}$, $(v_i, v_{i+1}) \in E$. We say that v_i is the *predecessor* v_{i+1} on the path
- If there is a path from v to u we say that v is an *ancestor* of u and u is a *descendant* of v.
- **Cycle**: In a directed graph, a *cycle* is a path $v_0,..,v_k$, such that $v_0=v_k$. If the vertices $v_1,\ldots,v_k$ are also pair wise disjoint, the cycle is called *simple*.
- In an undirected graph, a (simple) *cycle* is a path $v_0,\ldots,v_k$ such that $v_0=v_k$, $k \geq 3$ and $v_1,\ldots,v_k$ are pair wise disjoint.

more important definitions…

- **Connected graph**: An undirected graph G is said to be *connected* if for each two vertices u,v in the graph, there is a path between u and v.
- **Strongly Connected graph**: A directed graph G is said to be *strongly connected* if for each two vertices u,v in the graph, there is a path between u and v.
- **Tree**: A tree is an undirected, connected, a-cyclic graph.
- **Rooted Tree**: A directed graph G is called a *rooted tree* if there exists $s \in V$ s.t. for each $v \in V$, there is exactly one path between s and v.
- **Forest**: A *forest* (*rooted forest*) is a set of disjoint trees (rooted trees).

Graph representations: adjacency lists

- One natural way to represent graphs is to use adjacency lists.
- For each vertex ***v*** there is a linked list of his neighbors.
- This representation is good for sparse graphs, since we use only $|V|$ lists and in a sparse graph, each list is short (overall representation size is $V+E$).

Graph representations: adjacency matrix

- Another way to represent a graph in the computer is to use an adjacency matrix. This is a matrix of size $|V|\times|V|$, we will denote it by T. The vertices are enumerated, $v_1,\ldots,v_{|V|}$. Now, $T_{i,j}=1 \Leftrightarrow$ there is an edge between the vertices v_i and $v_j \Leftrightarrow (v_i,v_j)\in E$.
- If the graph is undirected: $T_{i,j}=1 \Leftrightarrow T_{j,i}=1$

**** what is the meaning of T^2, T^3, etc. ???***

Review of graphs

- Graphs are a very useful tool in Computer Science. Many problems can be reduced to problems on graphs, and there exists many efficient algorithms that solves graph problems.
- Today we will examine a few of these algorithms.
- We will focus on the shortest path problem (unweighted graphs) which is a basic routine in many graph related algorithm. We can define:
  - Shortest path between ***s*** and ***t***.
  - Single source shortest path (shortest path between ***s*** and {***V***}).
  - All pairs shortest path.

Breadth First Search (BFS)

- The ***Breadth First Search*** (*BFS*) is one of the simplest and most useful graph algorithms.
- The algorithm systematically explores the edges of ***G*** to find all vertices that are reachable from ***s*** and computes distances to those vertices.
- It also produces a "breadth first tree", with s being the root.
- It is called breadth first search since it expands the frontier between visited and non visited vertices uniformly across the breadth of the frontier.

Breadth First Search (cont.)

- To keep track of progress, *BFS* colors each vertex according to their status.
- Vertices are initialized in white and are later colored as they are discovered and being processed.
- It also produces a "breadth first tree", with s being the root.
- If $(u,v) \in E$ and u is black then v is non white.
- Gray vertices represent the frontier between discovered and undiscovered vertices.

Breadth First Search (cont.)

- The BFS algorithm constructs a *BFS* tree, initially containing only the root ***s*** (the source vertex).
- While scanning the neighbors of an already discovered vertex ***u***, whenever a white vertex ***v*** is discovered it is added to the tree along with the edge (***u***,***v***).
- ***u*** is the parent of ***v*** in the *BFS* tree.
- If ***u*** is on the pass in the tree from ***s*** to ***v*** then ***u*** is ancestor of ***v*** and ***v*** is a descendant of ***u***.
- The algorithm uses a queue (FIFO) to manage the set of gray vertices.

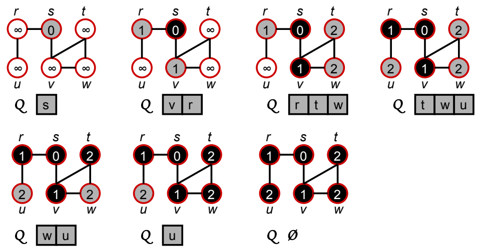
BFS – pseudo code

```
BFS(G,s)
     //initializing.
     for each vertex u∈V[G]\{s} {
          color[u] = white;
          dist[u] = ∞;
          parent[u] = NULL;
     }
     color[s] = GRAY;
     dist[s] = 0;
     parent[s] = NULL;
     Q <- {s};
```

BFS – pseudo code (cont.)

```
     ...
     while (not Q.isEmpty()) {
       u <- Q.head();
       foreach v ∈ u.neighbors() {
          if color[v] ≠ WHITE {
            color[v] = GRAY;
            dist[v] = dist[u]+1;
            parent[v] = u;
            Q.enqueue(v);
          }
       Q.dequeue();
       color[u] = BLACK;
     }
```

BFS, an example:



BFS, properties:

- What can we say about time complexity?
- Why does it works? (intuition):
  - We can think as if we have a set of nodes ***S*** and for all the nodes in ***S***, the distance is correct (***S*** begins with just ***s***).
  - At step *t*, ***S*** contains the *t* closest nodes to ***s***.
  - At each step, the algorithm adds to S the next closest node to s by finding the closest node to s in S that has neighbors out of S and adding these neighbors to S (greedy algorithm).
  - The proof of correctness uses the fact that we have already discovered closer nodes and assigned them the correct distance when we discover a new node that is a neighbor of one of them.

BFS, proof of correctness:

- Claim 1: Let ***G***=(***V***,***E***) be a graph and let s∈***V*** be an arbitrary vertex. Then for any edge (***u***,***v***) ∈***E*** : $\delta(s,v) \le \delta(s,u)+1$
- Proof 1: If ***u*** is reachable from ***s***, so is ***v***, otherwise $\delta(s,u)=\infty$
- Claim 2: Let ***G***=(***V***,***E***) be a graph, and suppose we run *BFS* on ***G*** from ***s***. Upon termination, $\forall v \in V, \text{dist}[v] \ge \delta(s,v)$
- Proof 2: The proof is by induction on the number of times a vertex is placed in ***Q***. The claim holds after placing ***s*** in ***Q*** (basis). For the induction step, let's look at a white vertex v discovered during the search from u. By the hypothesis $\text{dist}[u] \ge \delta(s,u)$. From claim 1 and the algorithm we get: $\text{dist}[v] = \text{dist}[u]+1 \ge \delta(s,u)+1 \ge \delta(s,v)$

BFS, proof of correctness *(cont.)*:

- Claim 3: Suppose that during the execution of *BFS* on graph ***G***, the queue ***Q*** contains the nodes <v_1, ..., v_r>. Then: $\text{dist}[v_r] \le \text{dist}[v_1]+1$ and $\text{dist}[v_i] \le \text{dist}[v_{i+1}]\ \forall i \in \{1,...,r-1\}$
- Proof 3: The proof is by induction on the number of queue operations. The basis holds (only ***s*** is in the queue). When dequeuing a vertex, $\text{dist}[v_r] \le \text{dist}[v_1]+1 \le \text{dist}[v_2]+1$ and the claim holds. When enqueuing a node ***w***, we have the node ***u*** at the head of the queue => $\text{dist}[v_{r+1}] = \text{dist}[w] = \text{dist}[u]+1 = \text{dist}[v_1]+1$ and we also have:

$$\text{dist}[v_r] \le \text{dist}[v_1]+1 = \text{dist}[u]+1 = \text{dist}[w] = \text{dist}[v_{r+1}]$$

BFS, proof of correctness *(cont.)*:

- Claim 4: Let ***G***=(***V***,***E***) be a graph and we run *BFS* from ***s***∈***V*** on ***G***. Then the *BFS* discovers every vertex ***v***∈***V*** that is reachable from s, and upon termination, $\forall v \in V, \text{dist}[v] = \delta(s,v)$
- Proof 4: If ***v*** is unreachable, we have $\text{dist}[v] \ge \delta(s,v) = \infty$, but since ***v*** hasn't been discovered since it has been initialized, we get: $\infty = \text{dist}[v] \ge \delta(s,v) = \infty \Rightarrow \text{dist}[v] = \delta(s,v)$

  For vertices that are reachable from s, we define $V_k = \{v \in V : \delta(s,v) = k\}$
  For each v ∈V_k we show by induction that during the execution of the *BFS*, there is at most one point at which:
  - ***v*** is grayed.
  - dist[***v***] is set to *k*.
  - if ***v***≠***s*** then parent[***v***] is set to ***u*** for some ***u*** ∈V_{k-1}.
  - ***v*** is inserted into the queue ***Q***.

BFS, proof of correctness *(cont.)*:

- Proof 4 (cont.): For *k*=0, the inductive hypothesis holds (basis). For the inductive step, we first note that *Q* is never empty during the algorithm execution and that once a vertex ***v*** is entered *Q*, dist[***v***] and parent[***v***] never changes. Let us consider an arbitrary vertex ***v*** ∈V_k (k > 1). From claim 3 (monotonicity),claim 2 (dist[***v***] ≥*k*) and the inductive hypothesis we get that v must be discovered after all vertices in V_{k-1} *are enqueued (if discovered at all).* Since $\delta(s,v) = k$, there is a path of length *k* from ***s*** to ***v*** => There is a vertex ***u*** ∈V_{k-1} such that (***u***,***v***) ∈***E***. Let ***u*** be the first such vertex grayed. ***u*** will appear as the head of *Q*, at that time, its neighbors will be scanned and ***v*** will be discovered => d[***v***] = d[***u***]+1 = *k* and parent[***v***] = ***u***.