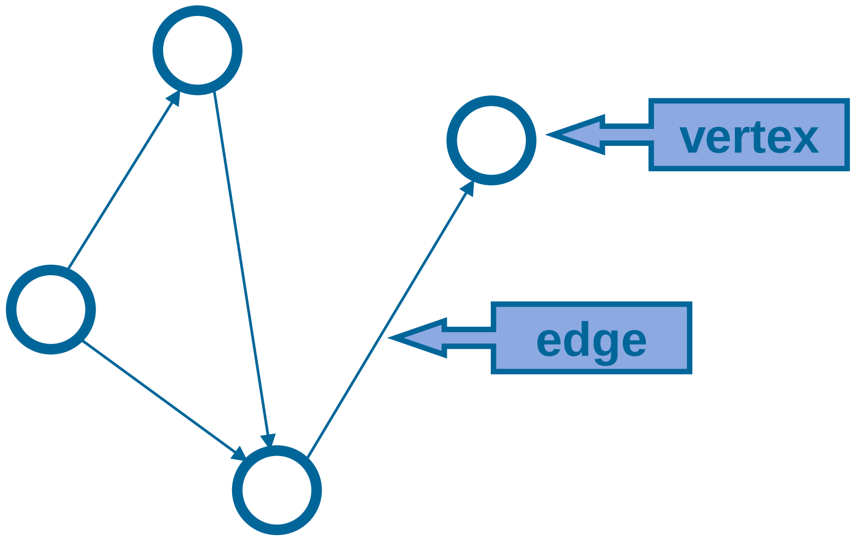
Lab 8
Graphs: DFS & BFS

What is a graph?

- A set of vertices and edges
  - Directed/Undirected
  - Weighted/Unweighted
  - Cyclic/Acyclic



Representation of Graphs

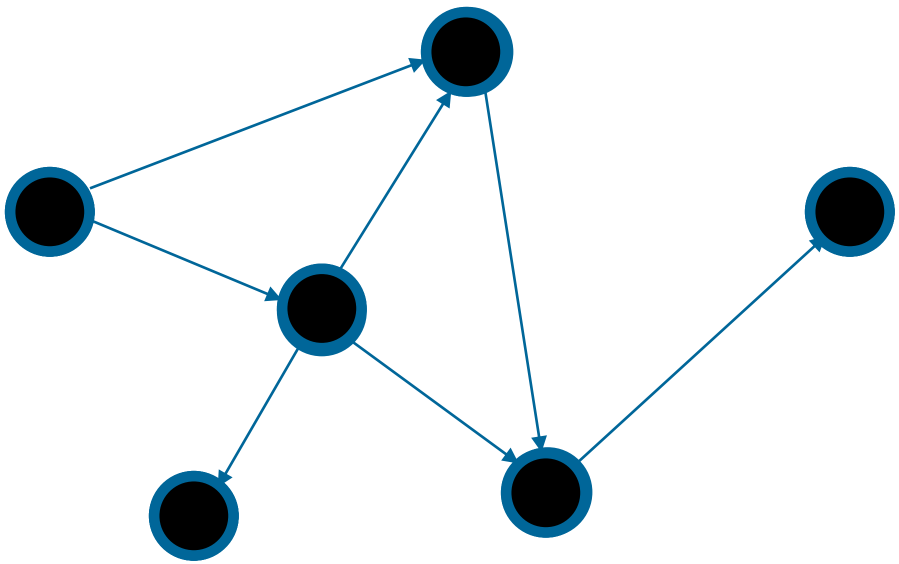
- Adjacency Matrix
  - A V x V array, with matrix[i][j] storing whether there is an edge between the i^{th} vertex and the j^{th} vertex
- Linked List of Neighbours
  - One linked list per vertex, each storing directly reachable vertices

Graph Searching

- Why do we do graph searching? What do we search for?
- What information can we find from graph searching?
- How do we search the graph? Do we need to visit all vertices? In what order?

Depth-First Search (DFS)

- Strategy: Go as far as you can (if you have not been there), otherwise, go back and try another way



Implementation

```
DFS (vertex u) {
   mark u as visited
   for each vertex v directly reachable from u
         if v is unvisited
                  DFS (v)
}
```

- Initially all vertices are marked as *unvisited*

Application: Topological Sort

- Topological order:
  A numbering of the vertices of a directed acyclic graph such that every edge from a vertex numbered i to a vertex numbered j satisfies $i<j$
- Topological Sort:
  Finding the topological order of a directed acyclic graph

Example: Teacher's Problem

- Emily wants to distribute candies to *N* students one by one, with a rule that if student A is teased by B, A can receive candy before B.
- Given lists of students teased by each students, find a possible sequence to give the candies

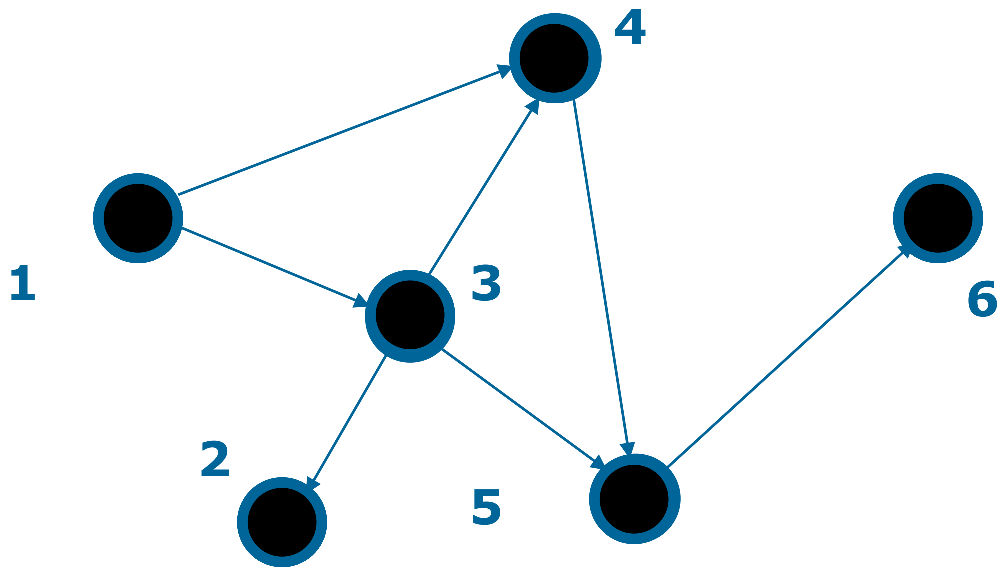
Breadth-First Search (BFS)

- Instead of going as far as possible, BFS tries to search all paths.
- BFS makes use of a queue to store visited (but not dead) vertices, expanding the path from the earliest visited vertices.

Simulation of BFS

- Queue:

1	4	3	5	2	6



Implementation

```
while queue Q not empty
  dequeue the first vertex u from Q
  for each vertex v directly reachable from u
        if v is unvisited
              enqueue v to Q
              mark v as visited
```

- Initially all vertices except the start vertex are marked as *unvisited* and the queue contains the start vertex only

Implementation with the adjacency matrix

```
#include <stdio.h>
#include "queue.h"
// http://www.cplusplus.com/reference/queue/queue/

template<typename TnodeInfo, typename TedgeInfo>
class Graph {
  public:
    int N;
    char **A;
    TnodeInfo *nodeInfo;
    TedgeInfo **edgeInfo;

    Graph(int numNodes) {
      int i, j;

      N = numNodes;

      // allocate the adjacency matrix
      A = new char*[N];
      for (i = 0; i < N; i++)
        A[i] = new char[N];
      for (i = 0; i < N; i++)
        for (j = 0; j < N; j++) A[i][j] = 0;

      // allocate the array with node information
      nodeInfo = new TnodeInfo[N];

      // allocate the matrix of edge information
      edgeInfo = new TedgeInfo*[N];
      for (i = 0; i < N; i++)
        edgeInfo[i] = new TedgeInfo[N];
    }

    void setNodeInfo(int i, TnodeInfo info) {
      nodeInfo[i] = info;
    }

    TnodeInfo getNodeInfo(int i) {
      return nodeInfo[i];
    }

    void addEdge(int i, int j) {
      A[i][j] = A[j][i] = 1;
    }
```

```
    void removeEdge(int i, int j) {
        A[i][j] = A[j][i] = 0; }

    void setEdgeInfo(int i, int j, TedgeInfo info) {
        edgeInfo[i][j] = edgeInfo[j][i] = info; }

    TedgeInfo getEdgeInfo(int i, int j) {
        return edgeInfo[i][j]; }

    ~Graph() {
        int i;
        for (i = 0; i < N; i++) {
            delete A[i];
            delete edgeInfo[i];
        }
        delete A;
        delete edgeInfo;
        delete nodeInfo;
    }
};
```

```
Graph<int, int> g(10);
char* visited;

void dfs(int x) {
    int y;
    printf("%d\n", x);
    visited[x] = 1;

    for (y = 0; y < g.N; y++)
        if (g.A[x][y] && !visited[y])
            dfs(y);
}
```

```
void bfs(int S) {
   std::queue<int> Q;
   int x, y;

   Q.push(S);
   visited[S] = 1;

   while (!Q.empty()) {
      x = Q.front();
         Q.pop();
      printf("%d\n", x);
      for (y = 0; y < g.N; y++)
         if (g.A[x][y] && !visited[y]) {
            visited[y] = 1;
           Q.push(y);
         }
   }
}
```

```
int main() {
   int i;
   g.addEdge(9, 4);   g.addEdge(9, 6);
   g.addEdge(4, 6);   g.addEdge(6, 2);
   g.addEdge(4, 2);   g.addEdge(4, 1);
   g.addEdge(4, 5);   g.addEdge(1, 5);
   g.addEdge(8, 2);   g.addEdge(8, 7);
   g.addEdge(8, 0);   g.addEdge(8, 3);
   g.addEdge(0, 3);   g.addEdge(7, 3);

   visited = new char[g.N];
   for (i = 0; i < g.N; i++)
      visited[i] = 0;
   printf("DFS:\n");
   dfs(2);

   for (i = 0; i < g.N; i++)
      visited[i] = 0;
  printf("BFS:\n");
   bfs(6);
  return 0;
}
```

Application: Shortest Path

- If all edges have the same cost, we find the minimum distance between two nodes A and B by performing a BFS from node A and stop when node B was found.

Example: The travelling salesman problem is the problem of finding the shortest path that goes through every vertex exactly once, and returns to the start

There is more…

- Other Graph Searching Algorithms:
  - Bidirectional search (BDS)
  - Iterative deepening search (IDS)

Graph Modeling

- Conversion of a problem into a graph problem
- Essential in solving most graph problems

Basics of graph modeling

- Identify the vertices and the edges
- Identify the objective of the problem
- State the objective in graph terms
- Implementation:
  - construct the graph from the input instance
  - run the suitable graph algorithms on the graph
  - convert the output to the required format

Well-known Applications

- Social networks
- The salesman problem
- The timetable problem
- ……

Application

- Let's consider un undirected graph, representing a social network. Given an user, display all his friends (or information about them) having the degree <=N (N is given). A is friend with B if there is an edge between A and B; we say that the degree of friendship is 1. Friends of friends have the degree of friendship 2.

Homework

- Check if a graph is bipartite and if so, display the components of those two sets A and B. The graph will be displayed through list of neighbors.
- Check your code for the following graphs:
  - G1=({ 1,2,3,4,5,6,7,8,9},{ 12, 13, 45, 56, 75, 24, 58, 79, 43, 89})
  - G2=({ 1,2,3,4,5,6,7,8,9},{ 12, 13, 45, 56, 75, 24, 58, 79, 43, 89,47})

Bipartite graph

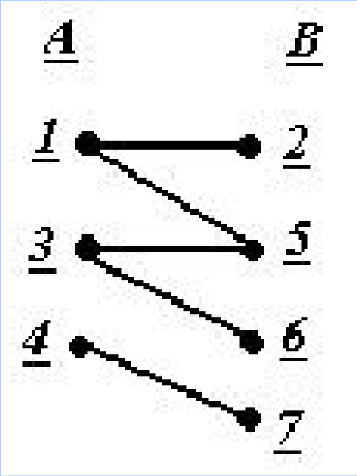
$G=(X,U)$

$X=\{1,2,3,4,5,6,7\}$

$U=\{[1,2];[1,5];[3,5];[3,6];[4,7]\}$

$A=\{1,3,4\}$

$B=\{2,5,6,7\}$



Tips

- In the mathematical field of graph theory, a bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets and such that every edge connects a vertex in to one in ; that is, and are each independent sets. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles.(Wikipedia)

- Use BFS: http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm