Department of Informatics
Database Technology Group
University of Zurich
Prof. Dr. Anton Dignös

# Informatics II
# Exercise 12

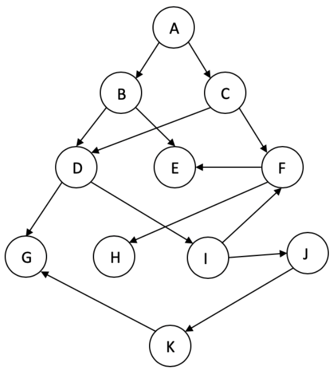
May 17, 2020

**Goals:**

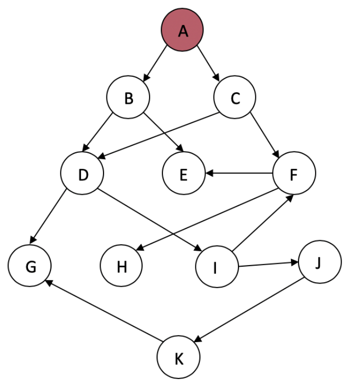
- Practice DFS and BFS
- Implementing a search algorithm in a 2-D array.
- Understand alternative graph representations and discuss them

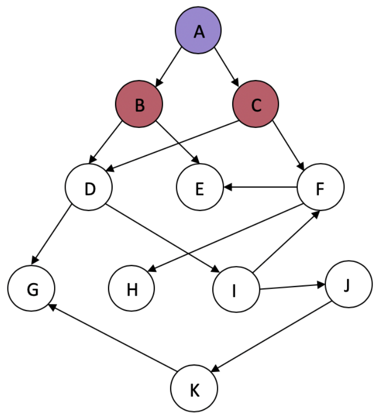
## Graphs(BFS, DFS)

**Task 1.** In the given graph below, each vertex has an unique label. For example, we use vertex A to denote the vertex with label "A". Write a breadth first search (BFS) starts at vertex A using a queue. In this task, during the BFS search, neighbors of a vertex are visited in the alphabetical order of their labels. For example, in the BFS that starts at vertex A, vertex B is visited before vertex C. The first two steps of the solution are shown below.

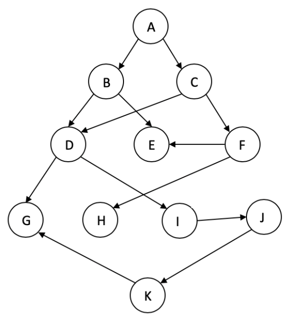



Solution: Vertices colored in blue are already visited by the BFS, and vertices colored in red are in queue and to be visitedby the BFS. Rememebr that the first-in-first-out principle in Queue.




**Task 2.** Given a graph below, each vertex has an unique label. Write the depth first search (DFS) starts at vertex A using a stack. In this task, during the DFS search, neighbors of a vertex are traversed in the alphabetical order of their labels. For example, in the DFS that starts from vertex A, vertex B is visited before vertex C. Note that the recursive solution of DFS is different from the one using a stack.




**Task 3.** This task is about implementing the depth-first search (DFS) on graphs. Graphs are represented by adjacency lists.

The following code snippet is everything you need for your graph:

```
struct node {
```

```c
  int vertex;
  struct node* next;
};

struct node* createNode(int v);

struct Graph {
  int numVertices;
  // We need int** to store a two dimensional array.
  // Similary, we need struct node** to store an array of Linked lists
  struct node** adjLists;
};


struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}


struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
  }
  return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}
```

Implement the function `void DFS(struct node** graph, int start)` that prints a DFS on the Graph from start vertex. Hint: You can find Advanced Data Structure implementations on previous exercise sheets.

**Task 4.** So far we have studied search in graphs. This task is about search in a 2D array (2D matrix). Imagine a 2D array, where each element shows a physical location in a maze. In this exercise, we use the following notation:

- **.** shows plain terrain
- # shows unpassable walls
- **S** and **E** shows start and end position respectively (also plain terrain)

The maze is a 7 by 7 matrix that is shown below. The possible movements for a position in the matrix are towards **west, south, east, north**

| | | | | | | |
|---|---|---|---|---|---|---|
| S | . | # | . | . | # | # |
| . | . | # | . | . | . | . |
| . | . | # | . | # | . | E |
| . | . | # | . | # | . | # |
| # | . | # | . | # | . | . |
| . | . | # | . | . | # | . |
| # | . | . | . | . | . | . |

1. What graph search algorithm should be used to find the minimum amount of steps required to travel from start to end?
2. Run this algorithm by hand(no need for the progress of the algorithm, just show the path and total number of steps)
3. Implement the algorithm that returns the shortest distance from the start position to the end position. You can use the following C codes.

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define GRIDSIZE 7
#define MAXQSIZE 49

struct Point{
    int x;
    int y;
};

struct queueNode
{
    struct Point pt; // The cordinates of a cell
    int dist; // cell's distance of from the source
};

bool isValid(int row, int col)
{
    // return true if row number and column number
    // is in range
```

```
    return (row ≥0) && (row < GRIDSIZE) &&
            (col ≥0) && (col < GRIDSIZE);
}
```

```
int sp_algo(int A[GRIDSIZE][GRIDSIZE], struct Point start, struct Point end)
```

```
int main() {
  int A[7][7] =
    {
        { '.', '.', '#', '.', '.', '#', '#'},
        { '.', '.', '#', '.', '.', '.', '.'},
        { '.', '.', '#', '.', '#', '.', '.'},
        { '.', '.', '#', '.', '#', '.', '#'},
        { '#', '.', '#', '.', '#', '.', '.'},
        { '.', '.', '#', '.', '.', '#', '.'},
        { '#', '.', '.', '.', '.', '.', '.'},
    };
  struct Point start = {0, 0};
  struct Point end = {2, 6};

  int spDist = sp_algo(A, start, end);
  printf("%d", spDist);
}
```

Hint: You can find Advanced Data Structure implementations in previous exercise sheets.

4. Is there a chance multiple shortest paths can be found? If so, show another path and explain what implementation detail determines which of the two is chosen?