Algorithms and Data Structures

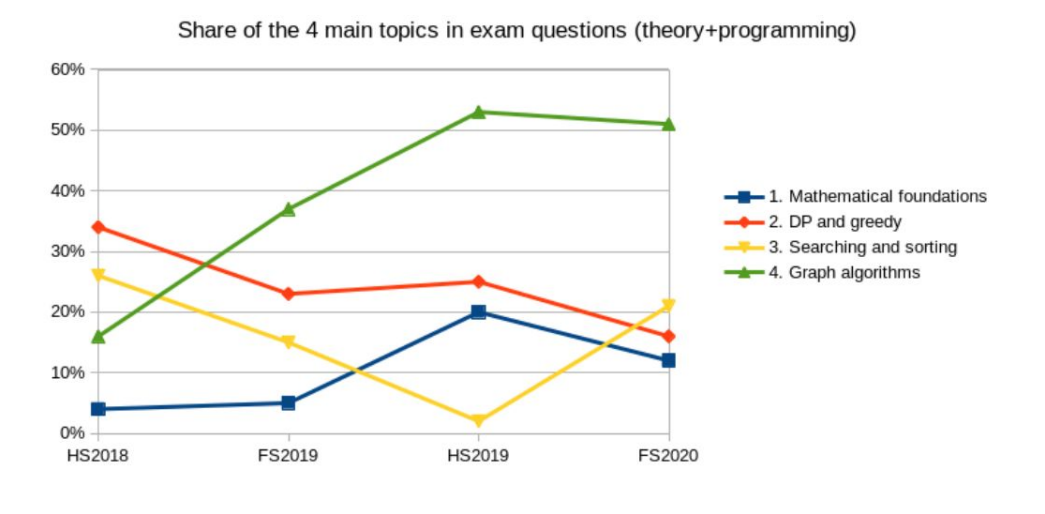
Exercise Session 10

https://n.ethz.ch/~ahmala/and

EXAM

Theory Exam: 8:30-10:30 am in 22nd January

Programming Exam: 9:30-12:30 am in 29th January.



https://cdn.vis.ethz.ch/luk/pvw_algorithmenunddatenstrukturen.pdf#page=5

Recap

Topological Sorting = Topological Ordering

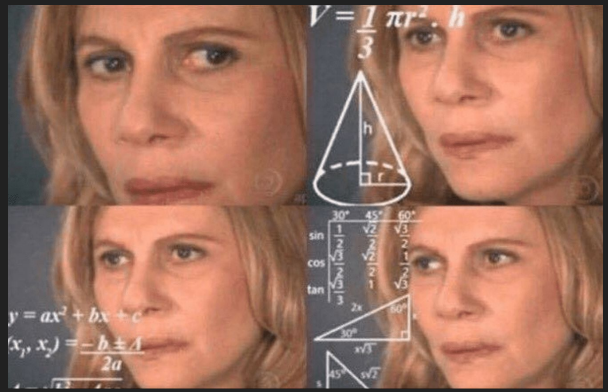
Directed edge = Arc

(u,v) : directed edge from u to v

{u,v} : undirected edge between u and v

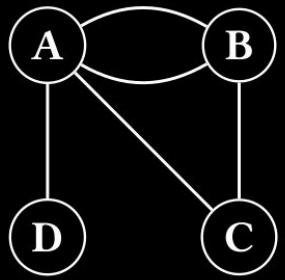
Confusion

German Kreis = English cycle; German Zyklus = English circuit.



Exercise 10.1 *Eulerian tours in multigraphs* **(1 point)**.

A *multigraph* $G = (V, E)$ is a graph which is permitted to have multiple copies of the same edge. That is, the edges E form a *multiset* (a set in which elements are allowed to occur multiple times). For example, the multigraph with $V = \{1, 2, 3, 4\}$ and $E = \{\{A, B\}, \{A, B\}, \{A, D\}, \{B, C\}, \{A, C\}\}$ is depicted below. To avoid confusion, the term *simple graph* is sometimes used to indicate that duplicate edges are not allowed.

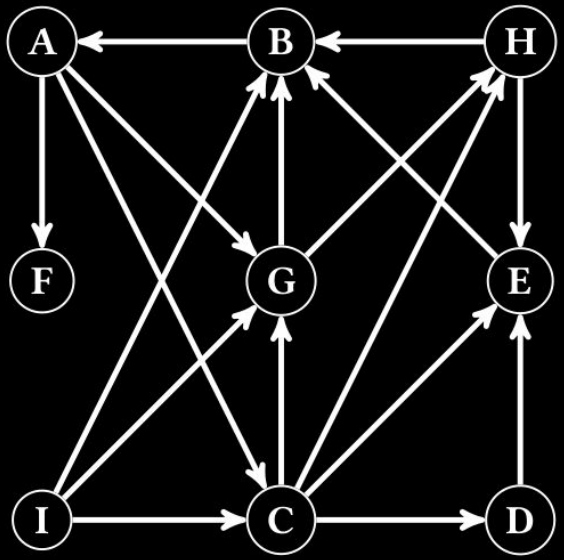


(a) An Eulerian tour in a multigraph is a tour which visits every edge exactly once. If multiple copies of an edge exist, the tour should visit each of them exactly once. Given a multigraph $G = (V, E)$, describe an algorithm which constructs a *simple* graph $G' = (V', E')$ such that G has a Eulerian tour if and only if G' has a Eulerian tour. The new graph should satisfy $|V'| \leq |V| + |E|$, and $|E'| \leq 2 \cdot |E|$. The runtime of your algorithm should be at most $O(n + m)$. You are provided with the number of vertices n and an adjacency list of G (if there are multiple edges between $v, w \in V$, then w appears that many times in the list of neighbours of v).

(b)* Let $G = (V, E)$ be a *simple* graph, and let $f : E \to \mathbb{N} \cup \{0\}$ be a function. A Eulerian f-tour of G is a tour which visits each edge $e \in E$ exactly $f(e)$ times. Describe an algorithm which constructs a simple graph $G' = (V', E')$ such that G has a Eulerian f-tour if and only if G' has a Eulerian tour. The new graph should satisfy $|V'| \leq |V| + \sum_{e \in E} f(e)$, and $|E'| \leq 2 \sum_{e \in E} f(e)$. The runtime of your algorithm should be at most $O(n + m + \sum_{e \in E} f(e))$.

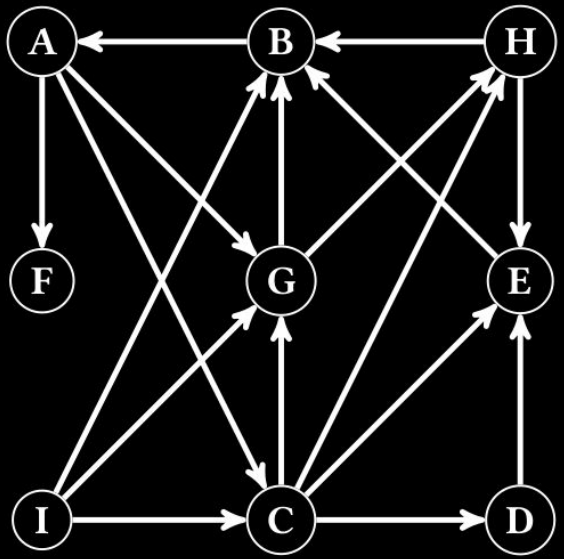
Exercise 10.2 *Depth-first search* **(1 point)**.

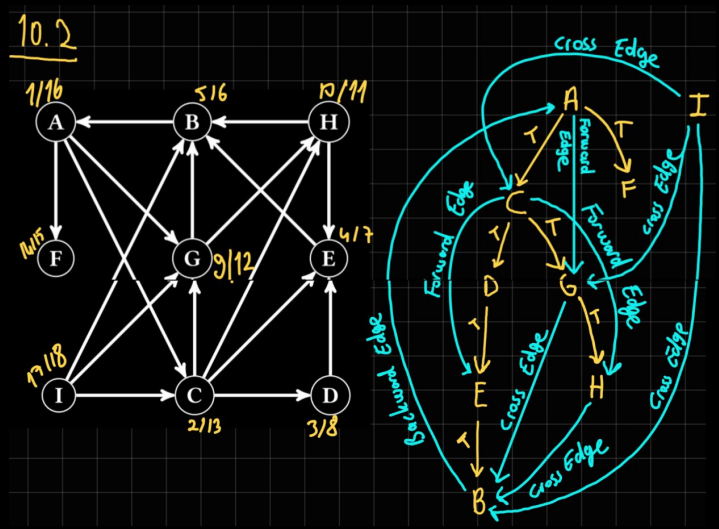
Execute a depth-first search (*Tiefensuche*) on the following graph. Use the algorithm presented in the lecture. Always do the calls to the function "visit" in alphabetical order, i.e. start the depth-first search from A and once "visit(A)" is finished, process the next unmarked vertex in alphabetical order. When processing the neighbors of a vertex, also process them in alphabetical order.



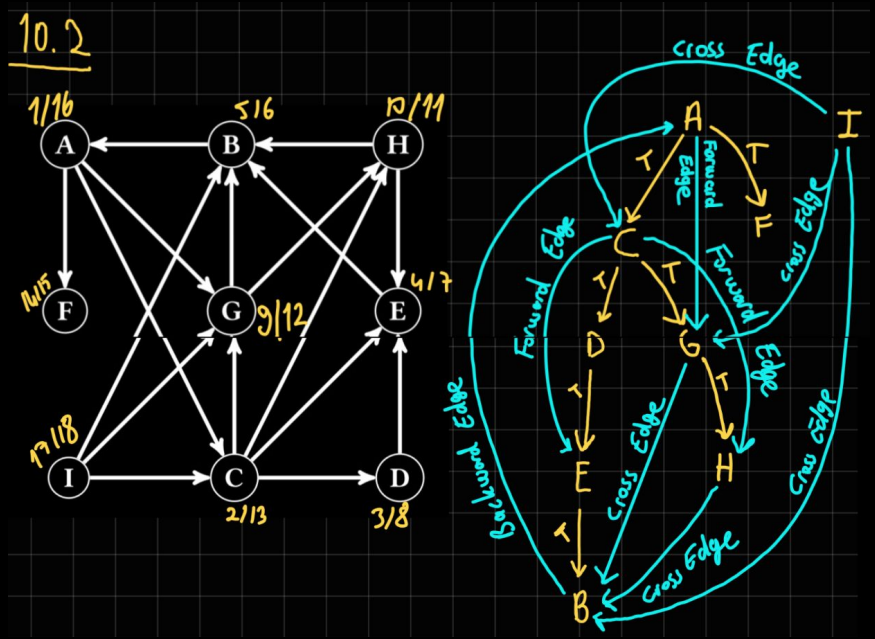
Exercise 10.2 *Depth-first search* **(1 point).**

Execute a depth-first search (*Tiefensuche*) on the following graph. Use the algorithm presented in the lecture. Always do the calls to the function “visit” in alphabetical order, i.e. start the depth-first search from A and once “visit(A)” is finished, process the next unmarked vertex in alphabetical order. When processing the neighbors of a vertex, also process them in alphabetical order.

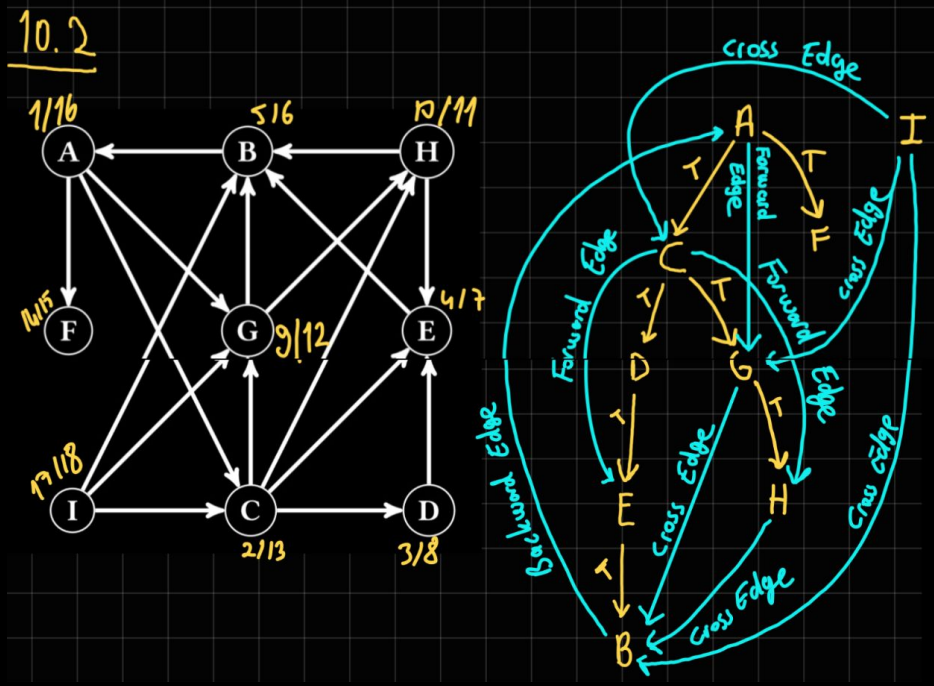




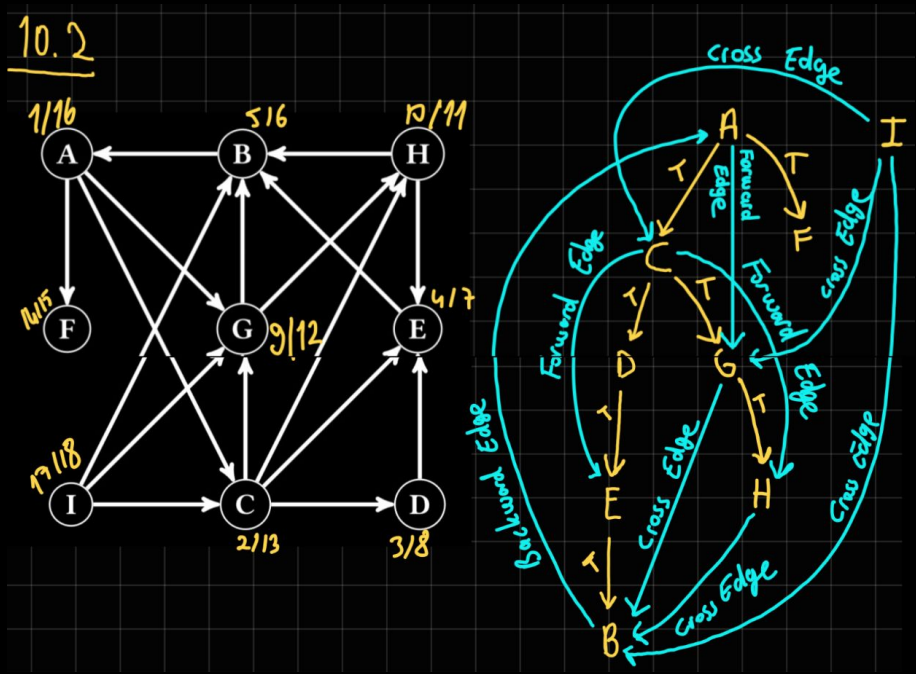
Does it have topological ordering?



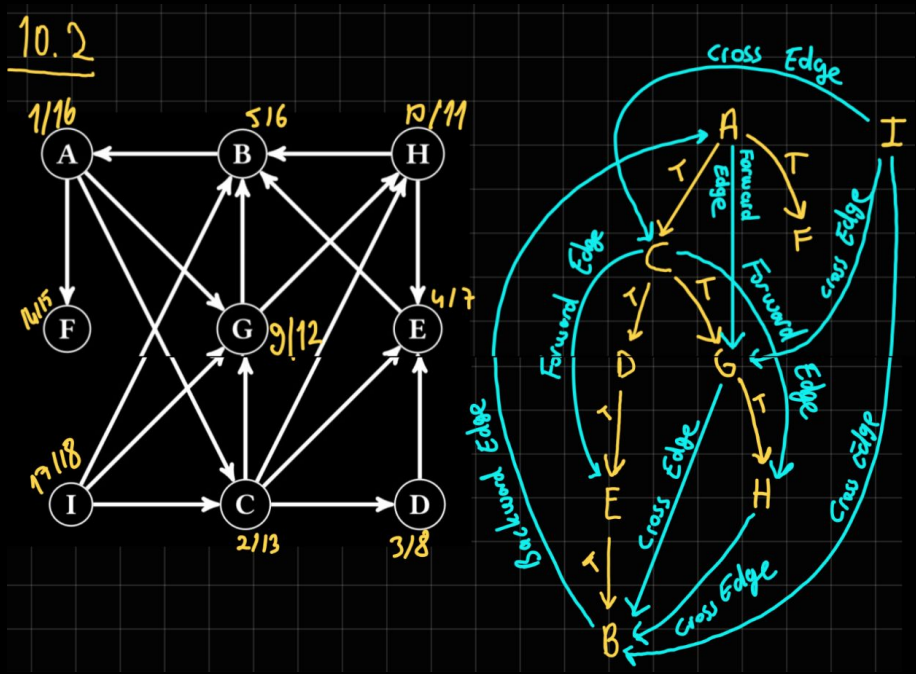
Does it have a topological ordering if we remove the edge from B to A and add an edge from F to I?



If you sort the vertices by pre-number, does this give a topological sorting?



If you sort the vertices by pre-number, does this give a topological sorting?



Cross Edges are a problem. For example I to B.

Exercise 10.3 *Driving on highways.*

In order to encourage the use of train for long-distance traveling, the Swiss government has decided to make all the m highways between the n major cities of Switzerland one-way only. In other words, for any two of these major cities C_1 and C_2, if there is a highway connecting them it is either from C_1 to C_2 or from C_2 to C_1, but not both. The government claims that it is however still possible to drive from any major city to any other major city using highways only, despite these one-way restrictions.

(a) Model the problem as a graph problem. Describe the set of vertices V and the set of edges E in words. Reformulate the problem description as a graph problem on the resulting graph.

Exercise 10.3 *Driving on highways.*

In order to encourage the use of train for long-distance traveling, the Swiss government has decided to make all the m highways between the n major cities of Switzerland one-way only. In other words, for any two of these major cities C_1 and C_2, if there is a highway connecting them it is either from C_1 to C_2 or from C_2 to C_1, but not both. The government claims that it is however still possible to drive from any major city to any other major city using highways only, despite these one-way restrictions.

(a) Model the problem as a graph problem. Describe the set of vertices V and the set of edges E in words. Reformulate the problem description as a graph problem on the resulting graph.

Solution:

V is the set of major cities in Switzerland (which is of size $|V| = n$), and there is a directed edge from $u \in V$ to $v \in V$ if and only if there is a highway going from city u to city v. The corresponding graph problem is to determine whether for any two vertices $u, v \in V$, there is a (directed) path from u to v in $G = (V, E)$.

(b) Describe an algorithm that verifies the correctness of the claim in time $O(n+m)$. Argue why your algorithm is correct and why it satisfies the runtime bound.

Hint: *You can make use of an algorithm from the lecture. However, you might need to modify the graph described in part (a) and run the algorithm on some modified graph.*

Exercise 10.4 *Strongly connected components* **(1 point)**.

Let $G = (V, E)$ be a directed graph with n vertices and m edges. Recall from Exercise 9.5 that two distinct vertices $v, w \in V$ are *strongly connected* if there exist both a directed path from v to w, and from w to v.

The vertices of G can be partitioned into disjoint subsets $V_1, V_2, \ldots, V_k \subseteq V$ with $V = V_1 \cup V_2 \cup \ldots \cup V_k$, such that any two distinct vertices $v, w \in V$ are strongly connected if and only if they are in the same subset V_ℓ, for some $1 \leq \ell \leq k$. The subsets V_ℓ are called the *strongly connected components* of G.

As in Exercise 9.5, you are provided with the number of vertices n, and the adjacency list Adj of G.

(a) Describe an algorithm that outputs the strongly connected components of G in time $O(n \cdot (n + m))$.

Hint: *Apply the algorithm of Exercise 9.5 several times. After each application, remove a vertex from* G.

(b)* Let $L = [v_1, v_2, \ldots, v_n]$ be a list containing the vertices of G in the *reversed* post-order of a DFS. Show that L has the following property:

'For any distinct $v, w \in V$, if there is a directed path from v to w, then

(1) v and w are strongly connected; and/or

(2) there exists a $u \in V$ which is in the same strongly connected component as v, and which appears before w in L.'

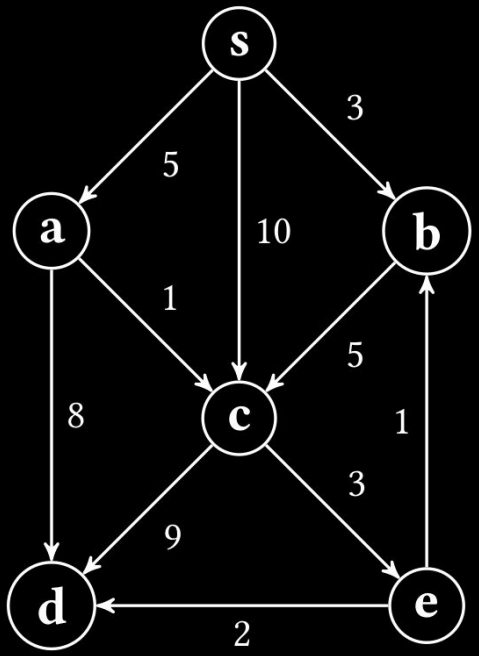
Remark. You are allowed to use this part in the rest of the exercise, even if you do not solve it.

(c) Let $\overleftarrow{G} = (V, \overleftarrow{E})$ be the directed graph obtained by inverting all edges in G. Let v_1 be the first element of L. Let $W \subseteq V$ be the set of vertices w for which there is a directed path from v_1 to w in $\overleftarrow{G}$. Show that W is a strongly connected component of G.

(d) Describe an algorithm that outputs all strongly connected components of G. The runtime of your algorithm should be at most $O(n + m)$. Prove that your algorithm is correct, and achieves the desired runtime.

Hint: *Use DFS on the inverted graph* $\overleftarrow{G}$*. Make* visit*-calls based on the list* L*.*

Apply Dijkstra!



Algorithm 1

Input: a weighted graph, represented via $c(\cdot, \cdot)$. Specifically, for two vertices u, v the value $c(u, v)$ represents the cost of an edge from u to v (or ∞ if no such edge exists).

function DIJKSTRA(G, s)

$d[s] \leftarrow 0$ ▷ upper bounds on distances from s

$d[v] \leftarrow \infty$ for all $v \neq s$

$S \leftarrow \emptyset$ ▷ set of vertices with known distances

while $S \neq V$ **do**

choose $v^* \in V \setminus S$ with minimum upper bound $d[v^*]$

add v^* to S

update upper bounds for all $v \in V \setminus S$:

$d[v] \leftarrow \min_{\text{predecessor } u \in S \text{ of } v} d[u] + c(u, v)$

(if v has no predecessors in S, this minimum is ∞)

Dijkstra with Fibonacci-Heaps: $O(|E| + |V| \log |V|)$
Dijkstra with Binary-Heaps: $O((|E| + |V|) \log |V|)$

Dijkstra only works only with non-negative weights.

You are given a maze that is described by a $n \times n$ grid of blocked and unblocked cells (see Figure 1). There is one start cell marked with 'S' and one target cell marked with 'T'. Starting from the start cell your algorithm may traverse the maze by moving from unblocked fields to adjacent unblocked fields. The goal of this exercise is to devise an algorithm that given a maze returns the best solution (traversal from 'S' to 'T') of the maze. The best solution is the one that requires the least moves between adjacent fields.

Hint: *You may assume that there always exists at least one unblocked path from 'S' to 'T' in a maze.*

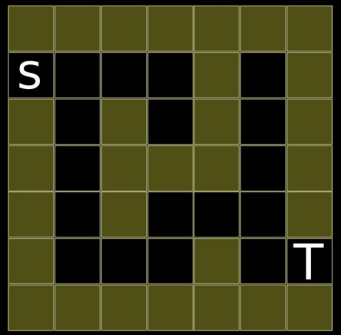


Figure 1: An example of 7×7 maze in which purple fields are blocked, white fields can be traversed (are unblocked). The start field is marked with 'S' and the target field with a 'T'.

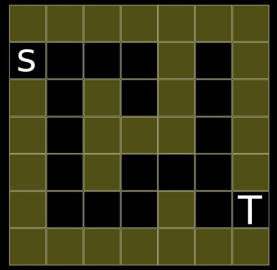


Figure 1: An example of 7×7 maze in which purple fields are blocked, white fields can be traversed (are unblocked). The start field is marked with 'S' and the target field with a 'T'.

a) Model the problem as a graph problem. Describe the set of vertices V and the set of edges E in words. Reformulate the problem description as a graph problem on the resulting graph.

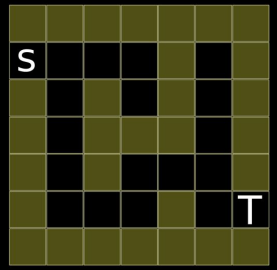


Figure 1: An example of 7×7 maze in which purple fields are blocked, white fields can be traversed (are unblocked). The start field is marked with 'S' and the target field with a 'T'.

a) Model the problem as a graph problem. Describe the set of vertices V and the set of edges E in words. Reformulate the problem description as a graph problem on the resulting graph.

Solution: V is the set of unblocked fields, and there is an edge between v_i and v_j if and only if v_i and v_j are adjacent unblocked fields. The corresponding graph problem is to find a shortest path between vertices 'S' and 'T' in $G = (V, E)$.

b) Choose a data structure to represent your maze-graphs and use an algorithm discussed in the lecture to solve the problem.

b) Choose a data structure to represent your maze-graphs and use an algorithm discussed in the lecture to solve the problem.

Solution: The data structure is adjacency list, the algorithm is BFS starting from 'S'. Once we know all the distances from 'S', we append vertices to a sequence starting from 'T' using the following rule: if the last appended vertex is v, we append some neighbour u of v such that $d_G(\text{'S'}, v) = d_G(\text{'S'}, u) + 1$. We stop after appending 'S'. Then we return a reverse sequence.

c) Determine the running time and memory requirements of your algorithm in terms of n in Θ notation.

c) Determine the running time and memory requirements of your algorithm in terms of n in Θ notation.

Solution: Adjacency list requires $\Theta(|V| + |E|)$ memory, where V is a number of vertices and $|E|$ is a number of edges in the graph. BFS requires $\Theta(|V| + |E|)$ time and appending procedure also requires $\Theta(|V| + |E|)$ time, so the total running time is $\Theta(|V| + |E|)$. Since each vertex has degree at most 4, $|E| = O(|V|)$, so the running time and memory are $\Theta(|V|)$ which is $\Theta(n^2)$ in the worst case.

Peer Grading

Exercise 10.4

You will find the file to peergrade in your polybox folder

While emailing your peer grading to me please include the group you corrected their work in cc.

https://docs.google.com/spreadsheets/d/1owPsJsd9THBWInwFcVjKCc0f_r6n4pGwwDKMDdwaCjM/edit?usp=sharing