Algorithmic Discrete Mathematics 4. Exercise Sheet

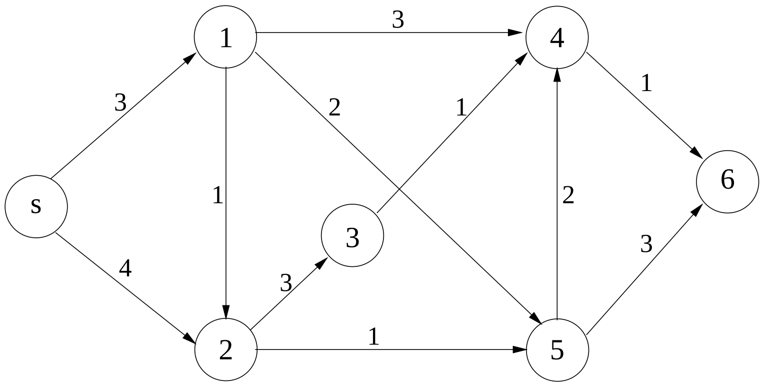
TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Mathematics
PD Dr. Ulf Lorenz
Dipl.-Math. David Meffert

SS 2012
30. and 31. May 2012
Version of May 23, 2012

Groupwork

Exercise G1 (Shortest paths I)



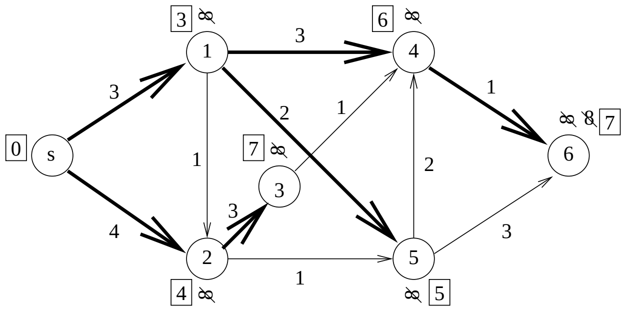
(a) Calculate the shortest path from s to all other vertices by using the Dijstra algorithm. Determine the shortest path tree.

(b) Is the shortest path tree unique?

(c) Now change the weight of the edge $(3,4)$ to -2. Show that the Dijkstra algorithm does not work in this case.

Solution:

(a) The solution is given by the followig picture. The numbers next to vertices are the distances to the starting vertex and crossing out a number means that there has been an update. The numbers in squares indicate the final distances (shortest distances). The shortest path tree is indicated by the thick lines. The order of visiting the vertices is $s, 1, 2, 5, 4, 6, 3$ or $1, 2, 5, 4, 3, 6$.



(b) No the shortest path tree is not unique. By substituting the edge $(s,3)$ with the edge $(1,2)$ one gets another possible shortest path tree.

(c) The Dijkstra algorithm will give the same solution as in part (a) although the path $(s,1,2,3,4)$ (dist 5) has a shorter distance than the path $(s,1,4)$ (dist 6). This is what happens:

After visiting the vertices $s, 1, 2$ the algorithm will look for the shortest path to a not yet visited vertex. This will be vertex 4 with distance 6. After visiting vertex 4 the algorithm will not make an update on vertex 4 because it was already visited and for this reason not find the shortest path (distance 5).

Exercise G2 (Shortest paths II)

Show that the problem finding a shortest odd cycle in a simple digraph $G = (V, E)$ with nonnegative weights c_e on the edges can be solved by using a shortest path algorithm.

Solution: We look at the graph $G = (V, E)$ with weights c_e for $e \in E$ and contruct another copy of its vertex set to get the graph $G' = (V', E')$. The set V' is definde by $V' := V \times \{0\} \cup V \times \{1\}$. We connect two vertices (v_1, σ_1) and (v_2, σ_2) by a directed edge iff $(\sigma_1 = 1 - \sigma_2)$ and $(v_1, v_2) \in E$. This means v_1 and v_2 are in different copies of V and they are adjacent in G. For the weights of these edges we choose the original weights of the corresponding edge in E.

The problem to find a shortest odd cycle in the orginal graph G can now be solved by looking at shortest paths from $(v, 0)$ to $(v, 1)$ in the graph G' for every vertex $v \in V$. All these paths in G' yield odd cycles in the original graph G by construction of G'. This holds because we have an odd number of changing between the two components $V \times \{0\}$ and $V \times \{1\}$. Hence by taking the shortest path we found we get the shortest odd cycle.

Exercise G3 (Crossing the river)

A man has to transport a wolf, a goat and a cabbage to the other side of a river. He has one boat to do this but it is so small that he can only take one of the three things with him each time. Is it possible to bring all three things to the other side of the river safely?

Notice that the wolf and the goat or goat and the cabbage must never be on the same side of the river without surveillance of the man. At least the wolf is no vegeterian and does not like to eat cabbage.

Solution: We construct a graph for this problem where every legal state is represented by a vertex in the graph and look for a shortest path in that graph.

So let $S = \{M, W, G, C\}$ where M, W, G, C represent the man, the wolf, the goat and the cabbage. A state for this problem is a pair (X, Y) where $\{X, Y\}$ is a partition of S. The elements in X are still on the starting side of the river and the elements in Y are already on the other side. A state is a legal state if $W, G \in X, Y \Rightarrow M \in X, Y$ and $G, C \in X, Y \Rightarrow M \in X, Y$. This represents that you may not leave alone the cabbage and the wolf or the goat and the cabbage without surveillance. Now construct a graph $G = (V, E)$ which has a vertex for every legal state of this problem. We have the directed edge $(v_1, v_2) \in E$ if we can get from the state $v_1 = (X_1, Y_1)$ to the state $v_2 = (X_2, Y_2)$ by just one boat trip across the river. All edges get the weight $c_e = 1$. Now the problem can be solved by calculating the shortest path from the state (vertex) $s = (S, \emptyset)$ to the state (vertex) $t = (\emptyset, S)$. We get 7 as the solution for the shortest path.

Homework

Exercise H10 (Algorithms) (10 points)

We are looking for an algorithm which gets an undirected, connected graph G given as an adjacency list and determines whether the graph is bipartite or not in runtime $O(|V| + |E|)$.

(a) Construct such an algorithm.

(b) Prove its correctness and analyse its runtime.

Solution:

(a) We modify the depth first search algorithm such that it assigns every vertex to one of the sets V_1 or V_2. The starting vertex will be assigned to the set V_1. One of the neighbours, which is not yet visited, will be assigned to the set V_2 and so on. If the algorithms gets to a neighbour of the actual vertex which has already been visited it checks whether the actual vertex and the neighbour are in the same set. If this is the case it returns false otherwise nothing happens. The algorithm keeps on doing this till all vertices are visited once.

Algorithm 1 DFS(G)

```
for each vertex $u \in V$ do do
  color[u] ← white
end for
G-bipartit ← true
for each vertex $u \in V$ do
  if color[u]= white then
    BIP-set[u] ← $V1$
    if DFS-Visit(u)=false then
      G-bipartit ← false
    end if
  end if
end for
return G-bipartit
```

Algorithm 2 DFS-Visit(u)

```
color[u] ← GRAY
for each vertex $v$ ∈Adj[u] do
  if color[v] = white then
    if BIP-set[u] = $V1$ then
      BIP-set[v]=$V2$
    else
      BIP-set[v]=$V1$
    end if
    DFS-Visit(v)
  else if BIP-set[u] = BIP-set[v] then
    return false
  end if
  return true
end for
```

(b) Now be prove the correctness of the algorithm and claim that G is bipartite iff the algorithm returns true.

first case: G is bipartite. We have to show that the algorithm returns true. The first vertex will be assigned to V_1. Every neighbour of that vertex will be assigned to V_2 (see line 3-6 in DFS-Visit). So two adjacent vertices will never be assigned to same set. In the end the algorithm will correctly return true, since G is bipartite and assigning all neighbours to other set as the vertex itself is necessary for getting a bipartition.

second case: G is not bipartite. We have to show that the algorithm does not return true. Since G is not bipartite we know that it contains a cycle C of odd length. Lets say C consists of $n+1$ vertices with n being even. The algorithm will assign n vertices correctly by always assigning the adjacent vertex in the cycle to the opposite sets. When it visits the $n+1$ vertex in the cycle it will realize that one adjacent vertex was already visited and was assigned to the same set as itself. Therefore it will return false as desired.

The runtime of the algorithm is the same as for the DFS algorithm because we only made modifications adding constant running time in each relevant part of the algorithm (for each part). So the runtime is $O(|V|+|E|)$.

Exercise H11 (Graphs) (10 points)

Given the graph G in Figure 1.

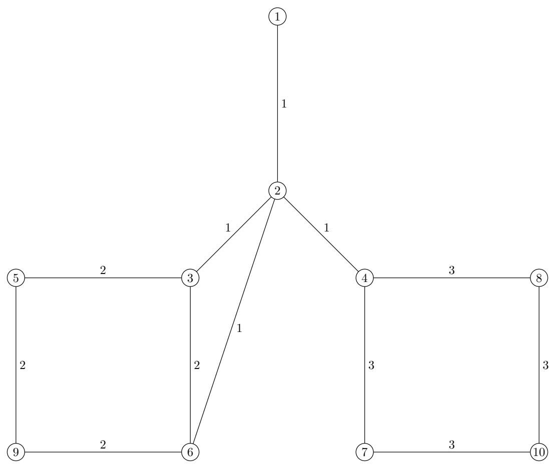


Figure 1: a graph

(a) Is the graph G eulerian? Justify your answer.

(b) Does the graph G contain a Hamiltonian cycle? This a cycle that contains each vertex exactly once. Justify your answer.

(c) Determine the adjacency matrix and adjacency list of the graph G.

Solution:

(a) The graph is not eulerian since it contains vertices not having an even degree.

(b) The graph contains no Hamiltonian cycle since the vertex number 2 has to be visisited at least two times to visit the vertex 1.

(c) The adjacency matrix looks as follows:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

For the adjacency list we have the following information in the given order.

- Number of vertices
- Number of edges
- For each vertex the degree and the incident edges

So the list looks as follows:

- 10
- 12
  - 1;2
  - 4;1,3,4,6
  - 3;2,5,6
  - 3;2,7,8
  - 2;3,9
  - 3;2,3,9
  - 2;4,10
  - 2;4,10

- 2;5,6
- 2;7,8

Exercise H12 (Shortest paths) (10 points)

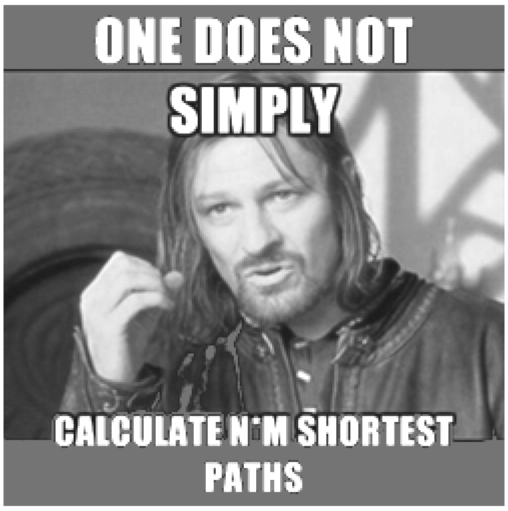
(a) Let $G = (V, E)$ be a directed graph.

i. Assume G has no negative cycles and $(s = i_0, i_1, \ldots, i_k = t)$ is a shortest simple path from s to t. Then every subpath from s to i_j with $j \in \{0, \ldots, k\}$ is a shortest simple path from s to i_j.

ii. Show that this statement is wrong for general graphs which may contain negative cycles.

(b) Given a directed graph $G = (V, E)$ we want to calculate the shortest path from the start to the destination. The problem is that start consists of n vertices and the destination consists of m vertices. So we are looking for the shortest path possibly starting at any s_i with $i \in \{1, \ldots, n\}$ and possibly going to any destination t_j with $j \in \{1, \ldots, m\}$. How can this be done efficiently? Justify your answer.

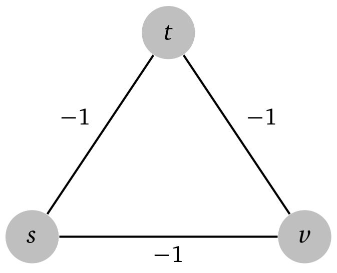
Hint:



Solution:

(a) i. Assume there is a shorter simple path P' from s to i_j. By connecting the paths P' and $(i_j, i_{j+1}, \ldots, t)$ we get a path from s to t which is shorter than the original shortest simple path from s to t. This path does not need to be simple. By eliminating all cycles in this path, which makes it ever shorter because we assumed no negative cycles, we get a shorter simple path from s to t than the original one. This is a contradiction

ii. Here is an example how the statement is not true for negative cycles. The shortest simple path from s to t is (s, v, t) with distance -2. Taking the subpath to (s, v) with distance -1 we find a shorter simple path from s to v namely (s, t, v) with distance -2.



(b) We modify the given graph $G = (V, E)$ to a graph $G' = (V', E')$. The vertex set is definded by $V' := V \cup \{s, t\}$ and the edge set is defined by $E' := E \cup \{(s, s_i) : 1 \leq i \leq n\} \cup \{(t_j, t) : 1 \leq j \leq m\}$. All the weights for the additional edges are set to zero. Notice that any weight would do the job. It is just important that they all get the same weight. Now we determine the shortest path from s to t in the graph G' which we will be $(s, s_i, \ldots, t_j, t)$ for some $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. The subpath $(s_i, \ldots, t_j)$ is the desired shortest path we wanted to find. If there would be a shorter path than this one we would have found it by using the shortest path algorithm on the graph G', since the edges (s, s_i) and (t_j, t) will not add any weight to the path (or add the same amount for every path from s to t if we have chosen nonzero weights).