HS19 T1(b) 4pts

A connected graph is called a tree.

- what is a tree ?
- when is a graph connected ?

Tree [edit]

A *tree* is an undirected graph G that satisfies any of the following equivalent conditions:

- G is connected and acyclic (contains no cycles).
- G is acyclic, and a simple cycle is formed if any edge is added to G.
- G is connected, but would become disconnected if any single edge is removed from G.
- G is connected and the 3-vertex complete graph K_3 is not a minor of G.
- Any two vertices in G can be connected by a unique simple path.

If G has finitely many vertices, say n of them, then the above statements are also equivalent to any of the following conditions:

- G is connected and has $n - 1$ edges.
- G is connected, and every subgraph of G includes at least one vertex with zero or one incident edges. (That is, G is connected and 1-degenerate.)
- G has no simple cycles and has $n - 1$ edges.

$\Longrightarrow$ False counterexample

HS19 T1(b) 4pts

If a tree contains at least two vertices, then any longest path in this tree has leaves as endpoints.

- what is a tree?
- $G = (V, E)$, $|V| \geq 2$
- what is a path
- when is a path considered longest?
- what are leaves?

$\Longrightarrow$ Idea: If P wasn't longest, we could extend P. How long can we possibly extend P? Until we reach a leaf. ...

$\Longrightarrow$ True

HS19 T1(b) 4pts **Any directed acyclic graph has a vertex with in-degree 0.**

- what is a directed graph?
- what is an acyclic graph?
- what is in-degree?
- what do we know about DAG? } connections
- degree sum formula? ...

⟹ Idea: Since they say "Any" can I find a counter example?

⟹ Idea: DAG ⟹ Topological Ordering
⟹ sink, source
⟹ source has in-degree 0

⟹ True! Topological ordering ...

FS20 T1 (5) 6 pts

A tree with n vertices must have $n-1$ edges.

- what is a tree ?

Tree [edit]

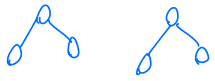
A *tree* is an undirected graph G that satisfies any of the following equivalent conditions:

- G is connected and acyclic (contains no cycles).
- G is acyclic, and a simple cycle is formed if any edge is added to G.
- G is connected, but would become disconnected if any single edge is removed from G.
- G is connected and the 3-vertex complete graph K_3 is not a minor of G.
- Any two vertices in G can be connected by a unique simple path.

If G has finitely many vertices, say n of them, then the above statements are also equivalent to any of the following conditions:

- G is connected and has $n-1$ edges.
- G is connected, and every subgraph of G includes at least one vertex with zero or one incident edges. (That is, G is connected and 1-degenerate.)
- G has no simple cycles and has $n-1$ edges.

Counter example ?



NO! Because the tree definition already implies connectedness !

$\Longrightarrow$ True

FS20 T1(b) 6pts An Eulerian walk visits every edge exactly once.

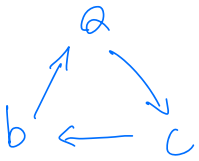
- what is a Eulerian walk?

=> True

FS20 T1 (5) 6 pts

Let $B = A^k$ be the adjacency matrix raised to the k-th power. If $B_{ij} > 0$ then there is a *path* of length k from i to j.

- what is an adjacency matrix?
- what is a path?



$$\begin{array}{c} \\ a \\ b \\ c \end{array}\begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{array} = A$$

$$\begin{array}{ccc|ccc} & & & 1 & 0 & 1 \\ & & & 1 & 1 & 0 \\ & & & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 & 1 & 2 \\ 1 & 1 & 0 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 & 2 & 1 \end{array} = A^2$$

Proof by induction.

Let A be the adjacency matrix of an undirected Graph G.

We prove that $A_{ij}^{(k)}$ is the number of walks of length k from vertex i to j.

IB: $k = 1$

$$A_{ij}^{(1)} = A_{ij} = \begin{cases} 1 & \text{if } \{i,j\} \in E \\ 0 & \text{else} \end{cases}$$

thus it's easy to see that A_{ij} is the number of walks of length 1 from i to j.

IH: For some $k \in \mathbb{N}$ we have that $A_{ij}^{(k)}$ is the number of walks of length k from i to j

IS: $k \rightsquigarrow k+1$

Assume that $A_{ij}^{(k)}$ is the number of walks of length k from i to j (I.H.).

$A^{k+1} = A^k \cdot A^1$ and therefore we have

$$A^{(k+1)}_{ij} = \sum_{\ell=1}^{|V|} A^{(k)}_{i\ell} A^{(1)}_{\ell j}.$$

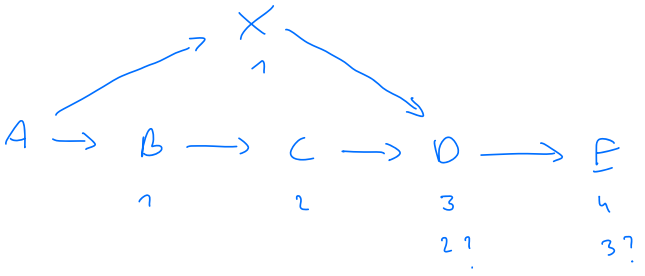
Now notice that a walk of length $k+1$ from i to j can be seen as a walk of length k from i to some $u \in V$ followed by a walk of length 1 from u to j, i.e. $W = \langle i, \dots, u, j \rangle$. Since the walk from i to u is of length k we know that $A^{(k)}_{iu}$ contains the number of walks of length k from i to u (I.H.). We can extend any walk from i to u by any walk from u to j thus the number of walks from i to j is the number of walks from i to u times the number of walks from u to j. We have: $A^{(k+1)}_{ij} = \sum_{u=1}^{|V|} A^{(k)}_{iu} A_{uj}$.

□

FS20 T1(b) 6pts

In an unweighted graph, both BFS and DFS can be used to determine shortest paths.

- unweighted graph?
- BFS / DFS?

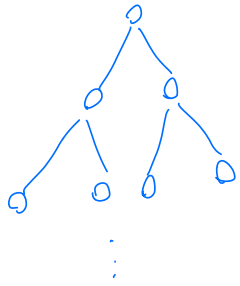


⟹ false

FS20 T1(b) 6pts

A binary tree of height h (the root has height 0) has at most 2^h leaves.

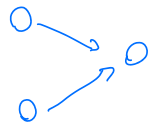
- what is a binary tree?
- what is the height of a binary tree?
- what is a leaf?



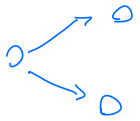
$\Rightarrow$ True

H S20 T1 (c) 5pts The topological ordering of a directed acyclic graph is unique.

- what is a topological ordering?
- when do you have a topo. ord.?



two sources?

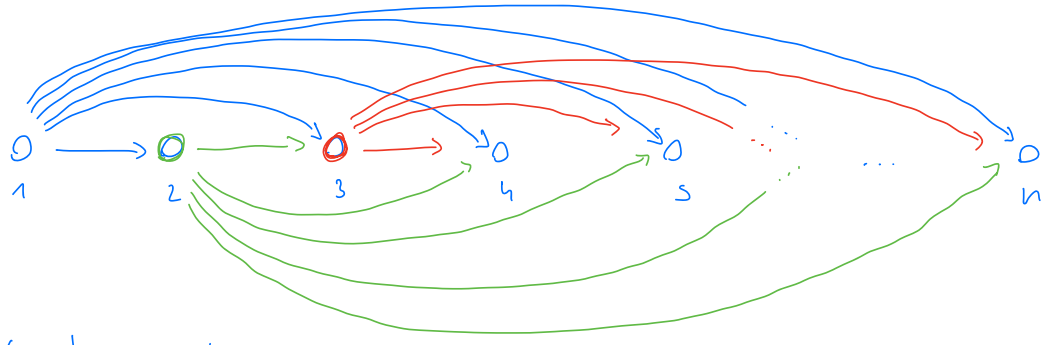


two sinks?

=> false

H S20 T1 (c) 5pts For all $n \in \mathbb{N}$, there exists a directed acyclic graph on n vertices with $\binom{n}{2}$ edges.

- what is the maximal number of edges in a graph?



first vertex $n-1$ edges

second vertex $n-2$ edges

$\vdots$

$(n-1)^{th}$ vertex 1 edge

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n!}{2!(n-2)!} = \binom{n}{2}$$

$\Longrightarrow$ True

H S20 T1 (c) 5pts

Let $v \in V$ be a vertex of an undirected graph $G = (V, E)$ with adjacency matrix A. It takes time $\Theta(1 + \deg(v))$ to compute $\deg(v)$ from A.

- adjacency matrix?
- degree? (undirected)

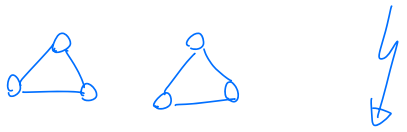
$\Rightarrow$ consider row v, what is the degree? number of 1's in the row. Have to go through entire row $\Theta(|V|)$.

$\Rightarrow$ False

~~HS20 T1(c) 5pts~~ If every vertex of an undirected graph G has even degree, then G has an Eulerian walk.

- Eulerian walk?

Too simple ?



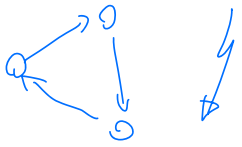
=> false

HS20 T1(c) 5pts

In order to run Dijkstra's algorithm on a directed graph G, you first need to have a topological ordering of G.

- Dijkstra's? (greedy, Priority Queue, ...)
- Topological ordering?

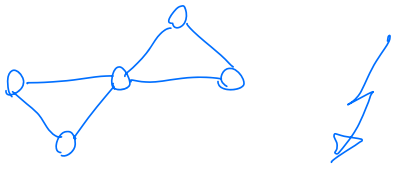
which graph has a topological ordering?



$\Longrightarrow$ false

~~H S21 T1 (c) 4pts~~ An undirected graph that contains a closed walk of even length always contains a cycle of even length as well.

- closed walk? length?
- cycle? length?



⟹ false

H521 T1(c) 4pts

For all $n \in \mathbb{N}$ and all $0 \leq m \leq \frac{n(n-1)}{2}$, there exists a directed acyclic graph on n vertices with m edges.

Same as:

H520 T1(c) 5pts

For all $n \in \mathbb{N}$, there exists a directed acyclic graph on n vertices with $\binom{n}{2}$ edges.

H S21 T1 (c) 4pts An undirected graph $G = (V, E)$ with $|E| = |V| - 1$ is always connected.

– $|E| = |V| - 1$? familiar?

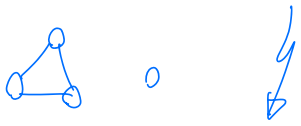
Tree [edit]

A *tree* is an undirected graph G that satisfies any of the following equivalent conditions:

- G is connected and acyclic (contains no cycles).
- G is acyclic, and a simple cycle is formed if any edge is added to G.
- G is connected, but would become disconnected if any single edge is removed from G.
- G is connected and the 3-vertex complete graph K_3 is not a minor of G.
- Any two vertices in G can be connected by a unique simple path.

If G has finitely many vertices, say n of them, then the above statements are also equivalent to any of the following conditions:

- G is connected and has $n - 1$ edges.
- G is connected, and every subgraph of G includes at least one vertex with zero or one incident edges. (That is, G is connected and 1-degenerate.)
- G has no simple cycles and has $n - 1$ edges.



$\Rightarrow$ false

FS22 T2 (c) 2pts . **justify your answers briefly.**

Let $G = (V, E)$ be a connected edge-weighted undirected graph where all the weights are nonnegative and distinct, and assume that $|V| \geq 2$. Let T be a minimum spanning tree of G.

Let $v \in V$ be some vertex and define T_v to be the tree of shortest paths that is obtained by applying Dijkstra's algorithm on G starting from the source v.

Is it possible that T and T_v do not have any edge in common? If the answer is yes, provide an example showing that it is possible. Otherwise, prove that it is impossible.

- MST is unique ?
- Dijkstra? Tree ?

Proof. (Sketch)

(1) Since non-neg. distinct weights ⇒ unique MST Kruskal takes the weights in ascending order ⇒ cheapest edge to v is picked to connect v to the MST. Same for Dijkstra ...

(2) Prim starting from v ⇒ takes cheapest edge to v. Same for Dijkstra ...

FS22 T2 (c) 2pts . **justify your answers briefly.**

We define the (undirected) complete graph K_n on n vertices as $K_n = (\{v_1, \ldots, v_n\}, E_n)$, where $E_n = \{\{v_i, v_j\} \mid 1 \leq i < j \leq n\}$.

Do the following claims hold true? For each of them, either provide a counter-example or briefly justify why they hold.

i) For all n, K_n contains a Hamiltonian path.

- complete graph ? Hamiltonian path ?

$P = \langle v_1, v_2, \ldots, v_n \rangle$

ii) For all n, K_n contains a Eulerian circuit.

Let v from k_n, what is the degree of v ? $\deg(v) = n-1$.

For even n we have $n-1$ is odd ...

iii) For all n, one can always add a single vertex v_0, and at most n edges adjacent to v_0, to K_n so that the resulting graph contains a Eulerian path.

Let v from k_n, what is the degree of v ? $\deg(v) = n-1$. We add $+1$ to all $\deg(v)$ through v_0 ...

For odd n, we don't add v_0.

For even n, ... all degrees equal n ...

HS20 T2 (c) 4 pts

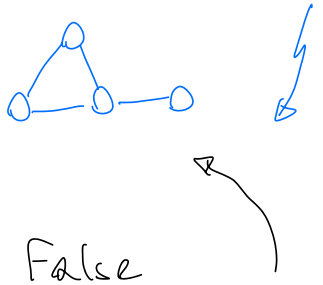
Definition 1 *A vertex v in a connected graph is called a* cut vertex *if the subgraph obtained by removing v (and all its incident edges) is disconnected.*

Definition 2 *An edge e in a connected graph is called a* cut edge *if the subgraph obtained by removing e (but keeping all the vertices) is disconnected.*

In the following, we always assume that the original graph is connected. Prove or find a counterexample to the following statements:

i) If a vertex v is part of a cycle, then it is not a cut vertex.

- cycle, cut vertex?
- seems correct?



=> False

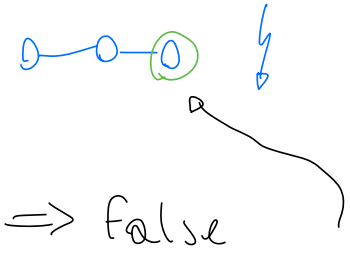
HS20 T2 (c) 4pts ii) If a vertex v is not a cut vertex, then v must be part of a cycle.

- comparing with i) this must be true now right? NO!

(i) v is in cycle $\Rightarrow$ v is no cut vertex

(ii) v is no cut vertex $\Rightarrow$ v is in cycle

(i) showed that v can be in a cycle and be a cut vertex at the same time.



not cut, but not cyle...

$\Rightarrow$ false

H S20 T2 (c) 4 pts

iii) If an edge e is part of a cycle (that is, e connects two consecutive vertices in a cycle), then it is not a cut edge.

- how to prove it?

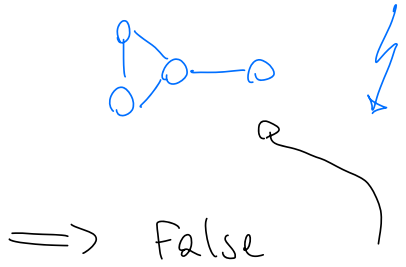
- if u, w connected in G, after deleting e, u, w still connected.

This statement is correct. Let G be a connected graph and let $e = \{v_1, v_2\}$ be an edge of G that is part of a cycle $v_1 \dots v_k$ for some $k \geq 3$. Let u and w be arbitrary vertices and consider any walk between u to w in G. Let's replace every appearance of v_1v_2 in this walk by the path $v_1v_kv_{k-1} \dots v_2$ and every appearance of v_2v_1 by $v_2v_3 \dots v_kv_1$. This yields a walk from u to w that does not use the edge e. Hence there exists a walk between any two vertices in the subgraph of G obtained by removing e, so e is not a cut edge.

HS20 T2 (c) 4 pts

iv) If v is a cut vertex and e is an edge incident to v, then e is a cut edge.

- relationship cut vertex and cut edge?



⟹ false

Relation to vertex connectivity [edit]

An undirected connected graph v no bridge edges

Bridges are closely related to the concept of articulation vertices, vertices that belong to every path between some pair of other vertices. The two endpoints of a bridge are articulation vertices unless they have a degree of 1, although it may also be possible for a non-bridge edge to have two articulation vertices as endpoints. Analogously to bridgeless graphs being 2-edge-connected, graphs without articulation vertices are 2-vertex-connected.

AlgoWahr ...

HS21 T2 (c) 4 pts

Describe an algorithm which, given an unweighted directed graph $G = (V, E)$ and a vertex $v \in V$, finds a shortest cycle containing v. If there is no such cycle, the algorithm should report that v is not a vertex of any cycle. Faster algorithms are worth more points. To get full points, aim for $O(|V| + |E|)$ runtime.

- shortest cycle ? shortest path ?
- path vs. cycle ?
- runtime hints at DFS / BFS ?

$C = \langle \underbrace{v_1, v_2, \dots, v_n}_{\text{path ?}}, v_1 \rangle$

$\langle v_1, v_2, \dots, v_n \rangle$ + " " $\langle v_1 \rangle$

shortest path from 1 to n + 1 ?

We start a breadth-first search (BFS) from v:

- If we do not encounter v for a second time, then v is not a vertex of any cycle.
- If we do encounter v for a second time, let u be the vertex from which we reached v, and let P be the shortest path from v to u which was found by the BFS. Adding the edge (u, v) to the path P will yield a shortest cycle containing v.

HS21 T2 (c) 4 pts

A *tournament* is a directed graph $G = (V, E)$ such that:

- G has no self loops, i.e., $(v, v) \notin E$, for all $v \in V$. (Note that the graphs that we usually consider have no self loops.)
- For every two distinct vertices $u, v \in V$, either $(u, v) \in E$ or $(v, u) \in E$ but not both.

Let G be a directed acyclic graph that is also a tournament. Show that G has a unique topological sorting.

- tournament ?
- DAG ?

Since DAG $\Rightarrow$ topo. sort.

but unique ?

$S = (v_1, v_2, \dots, v_n)$ (topo. sort.)

v_i all distinct (by def. of)

tournament: v_i, v_{i+1} have edge.

edge must be (v_i, v_{i+1}) not (v_{i+1}, v_i) !

(because topo. sort.)

$v_1 \to v_2 \to v_3 \to \dots \to v_n$ path ?

Let n be the number of vertices of G. Since G is a directed acyclic graph (DAG), it has at least one topological sorting $(v_1, \dots, v_n)$. On the other hand, since G is a tournament, for every $1 \leq i < n$, there is an edge between v_i and v_{i+1} in G. Furthermore, since v_i appears before v_{i+1} in the topological sorting $(v_1, \dots, v_n)$ of G, the edge between v_i and v_{i+1} must be (v_i, v_{i+1}) and cannot be (v_{i+1}, v_i). Therefore, $(v_1, \dots, v_n)$ is a path in G, hence v_i must appear before v_{i+1} in any topological sorting of G. We conclude that $(v_1, \dots, v_n)$ is the unique topological sorting of G.

H520 T3 9 pts

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$ summing to $A := \sum_{i=1}^{n} a_i$, which is a multiple of 3. You want to determine whether it is possible to partition $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that the corresponding elements of the array yield the same sum, i.e.

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{A}{3}.$$

Note that I, J, K form a partition of $\{1, \ldots, n\}$ if and only if $I \cap J = I \cap K = J \cap K = \emptyset$ and $I \cup J \cup K = \{1, \ldots, n\}$.

For example, the answer for the input $[2, 4, 8, 1, 4, 5, 3]$ is *yes*, because there is the partition $\{3, 4\}$, $\{2, 6\}$, $\{1, 5, 7\}$ (corresponding to the subarrays $[8, 1]$, $[4, 5]$, $[2, 4, 3]$, which are all summing to 9). On the other hand, the answer for the input $[3, 2, 5, 2]$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a partition exists. Your algorithm should have an $\mathcal{O}(nA^2)$ runtime to get full points. Address the following aspects in your solution:

- seems similar to subset sum
- subset sum is in $O(nB)$

In subset sum we take or not take elements.
Since I, J, K are disjoint we have to be careful not to take a single element more than once.
Since I, J, K are disjoint and $A = \sum_{i=1}^{n} a_i$ we have that if

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \frac{1}{3} A \quad \text{then} \quad [n] \setminus (I \cup J) =: K$$

$$\sum_{k \in K} a_k = \frac{1}{3}$$

Idea: show I, J exist such that

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \frac{1}{3} A .$$

Idea: we consider each element one by one as in subset sum and see what sums we can construct with them.

Idea: two times subset sum, but only give each element once.

$DP[m, B, C]$ is 1 if we can construct B and C using the first m elements (only once).

$$DP[m+1, B, C] = \max\{DP[m, B, C], DP[m, B - a_{m+1}, C], DP[m, B, C - a_{m+1}]\} .$$

Size of the DP table / Number of entries: $(n+1) \times (A+1) \times (A+1)$

Meaning of a table entry:

$$DP[m, B, C] = \begin{cases} 1 & \text{if there are two disjoint sets } I, J \subseteq \{1, \ldots, m\} \\ & \text{such that } \sum_{i \in I} a_i = B \text{ and } \sum_{j \in J} a_j = C, \\ 0 & \text{otherwise.} \end{cases}$$

Computation of an entry (initialization and recursion):

$$DP[0, B, C] = \begin{cases} 1 & \text{if } B = C = 0, \\ 0 & \text{otherwise.} \end{cases}$$

The other entries are then computed as

$$DP[m+1, B, C] = \max\{DP[m, B, C], DP[m, B - a_{m+1}, C], DP[m, B, C - a_{m+1}]\}.$$

In this formula we assume that if $a_{m+1} > B$, then $DP[m, B - a_{m+1}, C] = 0$, and if $a_{m+1} > C$, then $DP[m, B, C - a_{m+1}] = 0$.

Order of computation:

increasing order of m. Order of B and C doesn't matter.

Extracting the result:

YES if $DP[n, A/3, A/3] = 1$

NO else

Running time:

$(n+1)(A+1)^2$ in $O(1)$ $\Rightarrow$ $O(nA^2)$

HS21 T4 13pts

It is 2050 and the city of Zurich has finally built some proper cycling lanes to travel around the city. It turns out that one of the cycling lanes starts exactly at the road crossing where you are living, and of course there are also cycling lanes at the ETH road crossing. Since you are commuting to ETH by e-bike every day and you care about the environment, you would like to know which way from your place to ETH requires the least power from your e-bike. Your commuting way should only go through cycling lanes, and not use any other streets.

There are several cycling lanes around the city and you can switch between them at crossings. For each pair of crossings that are connected directly by a cycling lane, you know how much battery is needed to travel from the first crossing to the second one. Note that your e-bike battery gets charged when you go downhill, so there are some crossing pairs for which you gain some battery by going from the first one to the second one. However, due to the laws of physics, it is impossible that you leave a crossing, bike around cycling lanes and come back to the same crossing with more battery than you started with. Biking from a crossing C to another crossing C' does not necessarily require the same amount of power than biking from C' to C (for example, think of a steep slope from C to C').

i) Model the problem as a graph problem. Describe the set of vertices, the set of edges and the weights in words. What is the corresponding graph problem ?

- shortest path
- negative weights
- no negative cycle

Solution: The network of cycling lanes defines a directed graph. The vertices V are the crossings, and for each pair of crossings $u, v \in V$ the two directed edges (u, v) and (v, u) are present if u and v are connected directly by a cycling lane. The weight $w((u, v))$ of an edge (u, v) is the amount of power needed to go from u to v using this direct connection. If you gain battery by going from u to v then $w((u, v))$ will be negative. Note that the assumption about the laws of physics guarantees that the graph does not contain a negative cycle.

The graph problem corresponding to finding the most energy-economic way from your place (denoted by vertex s) to ETH (denoted by vertex t) is the computation of a shortest path from s to t.

ii) Which algorithm from the lecture can you use to solve the graph problem ? Justify why you can use this algorithm, and state its running time in terms of $|V|$ and $|E|$ in Θ-notation.

Solution: Since the graph contains edges with negative weights, but no negative cycle, we can use the Bellman-Ford algorithm to compute the shortest path from s to t, which has a running time of $\Theta(|V| \cdot |E|)$.

b) A friend of yours now tells you that no matter which pair of crossings you are considering, the most energy-economic way (i.e., the one that requires the least battery) from the first one to the second one always goes through at most k other crossings (for some natural number $k \leq \sqrt{|V|}$). How can you modify the previous algorithm to get a lower runtime? *In order to achieve full points, your algorithm should run in time* $O(k \cdot |E|)$.

i) Describe your algorithm (using text or pseudocode). A high-level description is enough.

any pair (v, u)

(at most k other crossings)

length of path ? $\Longrightarrow$ $k+1$

Solution: In this case, we know that any shortest path in the graph consists of at most $k+1$ edges. The algorithm therefore consists of running the "bound improvement" update of the Bellman-Ford algorithm $k+1$ times (instead of the usual $|V|-1$ times), with source vertex s. This is captured by the following pseudocode, where the distances from s to other vertices are saved in the array d:

Algorithm 3 `TruncatedBellmanFord`(V, E, w, s, k)

```
d[s] ← 0
for v ∈ V \ {s} do
    d[v] ← ∞
for ℓ = 1, ..., k + 1 do
    for (u, v) ∈ E do
        if d[v] > d[u] + w((u, v)) then
            d[v] ← d[u] + w((u, v))
return d[t]
```

ii) Prove the correctness of your algorithm and show that it runs in time $O(k \cdot |E|)$.

Solution: The correctness of the algorithm basically follows from the correctness of the Bellman-Ford algorithm and the guarantee that all shortest paths have at most $k+1$ edges. Indeed for any $\ell \in \mathbb{N}$, we know that after ℓ "bound improvement" updates of the Bellman-Ford algorithm, the values stored in the array $(d[v])_{v \in V}$ contain the length of the shortest paths from s to v among all paths of at most ℓ edges. Therefore, after $k+1$ updates, these will be the length of the shortest paths from s to all other vertices in V.

Since one "bound improvement" update runs in time $O(|E|)$, and we perform it $k+1$ times, the total running time is indeed $O(k \cdot |E|)$.

c) On the weekend you would like to explore the city with your e-bike. You start from the road crossing where you are living with your battery charged with an amount b of power, and you want to know what are the crossings that you can reach. In other words, you are interested in finding all places that can be visited without needing to charge your battery on the way. Describe an algorithm that gives you the set of such reachable places. *In order to achieve full points, your algorithm should run in time* $O(|V| \cdot |E|)$.

Note that this part builds on part (a), not on part (b). In other words, here we do not assume that the most energy-economic way from any crossing to any other crossing always goes through at most k other crossings.

i) Describe your algorithm (using text or pseudocode). A high-level description is enough.

Idea: we modeled the distances to correspond to the amount of battery is needed for traveling. Somehow we have to limit this distance ($\leq b$).

Idea: edge relaxation ($d[v] > d[u] + w((u,v))$) here add condition that battery $\leq b$?

Algorithm 4 `BatteryBellmanFord`(V, E, w, s, b)

$d[s] \leftarrow 0$
for $v \in V \setminus \{s\}$ **do**
  $d[v] \leftarrow \infty$
for $\ell = 1, \ldots, |V|$ **do**
  for $(u, v) \in E$ **do**
    if $d[v] > d[u] + w((u, v))$ and $d[u] \leq b$ **then**
      $d[v] \leftarrow d[u] + w((u, v))$
return $\{v \in V : d[v] \leq b\}$

ii) Prove the correctness of your algorithm and show that it runs in time $O(|V| \cdot |E|)$.

Solution: Adding the condition $d[u] \leq b$ in the update rule means that now, after the ℓ-th iteration of the for-loop, $d[v]$ contains the length of the shortest path from s to v among paths on at most ℓ edges such that any subpath from s to an intermediate node $v' \neq v$ has length at most b. Therefore after $|V|$ iterations, $d[v]$ contains the length of the shortest path from s to v such that you never run out of battery on the way. Clearly, the vertices we can reach are then those $v \in V$ with $d[v] \leq b$.

Again, each "bound improvement" update runs in time $O(|E|)$, so the total running time is $O(|V| \cdot |E|)$.