Graph Algorithms - Lecture 4

October 25, 2024

Table of contents

A directed acyclic graph (dag) is a digraph without cycles.

A topological ordering of (the vertices) of the digraph $G = (V, E)$, with $|G| = n$, is an injective function $ord : V \to \{1, 2, \ldots, n\}$ ($ord[u] =$ the ordering number of the vertex u, $\forall u \in V$) such that

$$uv \in E \Rightarrow ord[u] < ord[v], \forall uv \in E.$$

Lemma 1

$G = (V, E)$ is a digraph without cycles if and only if it has a topological ordering.

Proof: "$\Leftarrow$" Let ord be a topological ordering of G. If $C = (u_1, u_1u_2, u_2, \ldots, u_k, u_ku_1, u_1)$ is a cycle in G, then, by the property of the function ord, we obtain the following contradiction

$$ord[u_1] < ord[u_2] < \ldots < ord[u_k] < ord[u_1].$$

"$\Rightarrow$" Let $G = (V, E)$ be a digraph of order n without cycles. We show by induction on n that G has a topological ordering. The induction step:

let $v_0 \in V$;
while $(d^-_G(v_0) \neq 0)$ do
  take $u \in V$ such that $uv_0 \in E$;
  $v_0 \leftarrow u$;
return v_0.

Obviously, since G has no cycles and V is finite, the algorithm terminates and the returned vertex v_0 has no incoming arc. The digraph $G - v_0$ has no cycles and by induction hypothesis has a topological ordering ord'. The topological ordering of G is

$$ord[v] = \begin{cases} 1, & \text{if } v = v_0 \\ ord'[v] + 1, & \text{if } v \in V \setminus \{v_0\}. \end{cases}$$

□

From the above proof we get the following algorithm for recognizing dags and constructing a topological ordering for the "yes" instances:
Input: $G = (\{1, \ldots, n\}, E)$ digraph with $|E| = m$.
Output: "yes" if G is a dag, and a topological ordering ord; "no" otherwise.

Shortest paths - Solving problem P2 for dags

construct the array $d_G^-[u], \forall u \in V$;
$count \leftarrow 0$; $S \leftarrow \{u \in V : d_G^-[u] = 0\}$; // S is a stack or a queue;
while $(S \neq \varnothing)$ do
  $v \leftarrow pop(S)$; $count++$; $ord[v] \leftarrow count$;
  for $(w \in A[v])$ do
    $d_G^-[w]--$;
    if $(d_G^-[w] = 0)$ then
      $push(S, w)$;
// time complexity $\mathcal{O}(n + m)$;
if $(count = n)$ then
  return "yes" ord;
return "no";

Shortest paths - Solving problem P2 for dags

P2 Given $G = (V, E)$ digraph; $a : E \to \mathbb{R}$; $s \in V$.
Find $P^*_{si} \in \mathcal{P}_{si}, \forall i \in V$, s. t. $a(P^*_{si}) = \min\{a(P_{si}) : P_{si} \in \mathcal{P}_{si}\}$

P2 with $G = (\{1, \ldots, n\}, E)$ dag, with $ord[i] = i$, $\forall i \in V$, and $s = 1$.
Property (I) holds and the system (B) can be solved by "substitution".

$u_1 \leftarrow 0$; $before[1] \leftarrow 0$;
for $(i = \overline{2, n})$ do
  $u_i \leftarrow \infty$; $before[i] \leftarrow 0$;
  for $(j = \overline{1, i-1})$ do
    if $(u_i > u_j + a_{ji})$ then
      $u_i \leftarrow u_j + a_{ji}$; $before[i] \leftarrow j$;
// The time complexity is $\mathcal{O}(n^2)$ or $\mathcal{O}(n + m)$ if we use adjacency lists instead of cost-adjacency matrix;

P2 Given $G = (V, E)$ digraph; $a : E \to \mathbb{R}$; $s \in V$.
Find $P^*_{si} \in \mathcal{P}_{si}, \forall i \in V$, s. t. $a(P^*_{si}) = \min\{a(P_{si}) : P_{si} \in \mathcal{P}_{si}\}$

P2 with $a(e) \geqslant 0, \forall e \in E$.
Property (I') holds and a solution of the system (B) can be obtained with the Dijkstra's algorithm, which maintains the following invariant: $S \subseteq V$ and

(D)
$$\begin{cases} \forall i \in S & u_i = \min\{a(P_{si}) : P_{si} \in \mathcal{P}_{si}\} \\ \forall i \in V \setminus S & u_i = \min\{a(P_{si}) : P_{si} \in \mathcal{P}_{si}, V(P_{si}) \setminus S = \{i\}\} \end{cases}$$

Initially $S = \{s\}$ and, in each step (of a sequence of $n-1$ steps), a new vertex is added to S, until $S = V$. Hence, by the invariant (D) from above, P2 is solved.

Dijkstra's algorithm

$S \leftarrow \{s\}$; $before[s] \leftarrow 0$; $u_s \leftarrow 0$;
for $(i \in V \setminus \{s\})$ do
  $u_i \leftarrow a_{si}$; $before[i] \leftarrow s$; // (D) holds
while $(S \neq V)$ do
  find $j^* \in V \setminus S$ such that $u_{j^*} = \min\{u_j : j \in V \setminus S\}$;
  $S \leftarrow S \cup \{j^*\}$;
  for $(j \in V \setminus S)$ do
    if $(u_j > u_{j^*} + a_{j^*j})$ then
      $u_j \leftarrow u_{j^*} + a_{j^*j}$; $before[j] \leftarrow j^*$;

Remark

In the initialisation step we can replace $u_i \leftarrow a_{si}$ with $u_i \leftarrow \infty$.

Proof of correctness of Dijkstra's algorithm

Since (D) holds after the initialization step, we have to prove that if (D) holds before a current while iteration, then (D) holds before next iteration.

Let $S \subseteq V$ and $u_1, \ldots u_n$ satisfying (D) before the current iteration.

First we show that if j^* is such $u_{j^*} = \min\{u_j \ : \ j \in V \setminus S\}$, then

$$u_{j^*} = \min\{a(P_{sj^*}) \ : \ P_{sj^*} \in \mathcal{P}_{sj^*}\}.$$

Suppose that $\exists P^1_{sj^*} \in \mathcal{P}_{sj^*}$ such that $a(P^1_{sj^*}) < u_{j^*}$. Since S and u_i satisfy (D), we have

$$u_{j^*} = \min\{a(P_{sj^*}) \ : \ P_{sj^*} \in \mathcal{P}_{sj^*}, V(P_{sj^*}) \setminus S = \{j^*\}\}.$$

It folows that $V(P^1_{sj^*}) \setminus S \neq \{j^*\}$; let k be the first vertex on $P^1_{sj^*}$ (starting from s) such that $k \notin S$.

Proof of correctness of Dijkstra's algorithm (cont'd)

Then $a(P^1_{sj^*}) = a(P^1_{sk}) + a(P^1_{kj^*})$. By the choice of k, we have $V(P^1_{sk}) \setminus S = \{k\}$ and, since (D) is satisfied we have $a(P^1_{sk}) \geqslant u_k$. We obtain that $u_{j^*} > a(P^1_{sj^*}) \geqslant u_k + a(P^1_{kj^*}) \geqslant u_k$ (the costs are $\geqslant 0$, so $a(P^1_{kj}) \geqslant 0$. But this contradicts the choosing of j^*.

It follows that in the current iteration, after the assignment $S \leftarrow S \cup \{j^*\}$, the first part of (D) holds.

The for loop after this assignment is necessary in order that the second part of (D) to be fulfilled after the while iteration: $\forall j \in V \setminus (S \cup \{j^*\})$

$$\min\{a(P_{sj}) \ : \ P_{sj} \in \mathcal{P}_{sj}, V(P_{sj}) \setminus (S \cup \{j^*\}) = \{j\}\} =$$

$$\min\{\min\{a(P_{sj}) \ : \ P_{sj} \in \mathcal{P}_{sj}, V(P_{sj}) \setminus S = \{j\}\}(= u_j),$$

$$\min\{a(P_{sj}) \ : \ P_{sj} \in \mathcal{P}_{sj}, V(P_{sj}) \setminus S = \{j, j^*\}\}(= \alpha_j)\}.$$

Proof of correctness of Dijkstra's algorithm (cont'd)

The first argument is u_j (the value before the updating from the for loop), and let the second argument be α_j.

Let P^1_{sj} an sj-path such that $\alpha_j = a(P^1_{sj})$, $j^* \in V(P^1_{sj})$ and $V(P^1_{sj} \setminus (S \cup \{j^*\}) = \{j\}$; we have $\alpha_j = a(P^1_{sj^*}) + a(P^1_{j^*j})$. Since we have proved that $S \cup \{j^*\}$ satisfies the first part of (D), it follows that $a(P^1_{sj^*}) = u_{j^*}$ and hence $\alpha_j = u_{j^*} + a(P^1_{j^*j})$.

If $a(P^1_{j^*j}) \neq a_{j^*j}$, it follows that there exists an $i \in V(P^1_{j^*j}) \cap S$, $i \neq j^*$. Hence $u_j \leqslant a(P^*_{si}) + a(P^1_{ij}) = u_i + a(P^1_{ij}) \leqslant a(P^1_{si}) + a(P^1_{ij}) = a(P^1_{sj}) = \alpha_j$.

We have obtained that the only possibility to have $\alpha_j < u_j$ is when $a(P^1_{j^*j}) = a_{j^*j}$, in this case $\alpha_j = u_{j^*} + a_{j^*j} < u_j$, which is the test in the for loop of algorithm (u_j is the value before the updating from the for loop).

□

Time complexity of Dijkstra's algorithm

Since the for-loop from the while step can be replaced (equivalently) by

for $(j \in N_G^+(j^*))$ do
  if $(u_j > u_{j^*} + a_{j^*j})$ then
    $u_j \leftarrow u_{j^*} + a_{j^*j}$; $before[j] \leftarrow j^*$;

the overall time spent by the algorithm to update the values of u_j is

$$\mathcal{O}(\sum_{j^* \in V \setminus \{s\}} d_G^+(j^*)) = \mathcal{O}(m).$$

Hence the time complexity is dominated by the sequence of minimum u_{j^*} findings.

Time complexity of Dijkstra's algorithm

- If the selection of the minimum u_{j^*} is made by parsing $(u_j)_{j \in V \setminus S}$, then the algorithm runs in time $\mathcal{O}((n-1)+(n-2)+\cdots+1) = \mathcal{O}(n^2)$.
- If the values of u_j for $j \in V \setminus S$ are maintained in a priority queue (e.g. a heap) then the extraction of each minimum takes $\mathcal{O}(1)$ time, but the time needed to execute all u_j reductions is in the worst case $\mathcal{O}(m \log n)$ - there are $\mathcal{O}(m)$ possible reductions, each needing $\mathcal{O}(\log n)$ time for the heap maintaining (Johnson,1977).
- The best implementation is obtained by using a Fibonacci heap with time complexity $\mathcal{O}(m + n \log n)$ (Fredman & Tarjan, 1984).

Remarks 1

- In order to solve the problem P1 using Dijkstra's algorithm, we add a test to stop the execution when the target vertex t is introduced in S. The worst-case time complexity remains the same.
- A nice heuristics directing the search toward t is obtained with the help of a consistent estimator, that is a function $g : V \to \mathbb{R}_+$ satisfying the following two conditions:
  (i) $\forall i \in V,\ u_i + g(i) \leqslant \min\{a(P_{st}) : P_{st} \in \mathcal{P}_{st} \text{ and } i \in V(P_{st})\}$
  (ii) $\forall ij \in E, g(i) \leqslant a_{ij} + g(j)$.
- Clearly, $g(i) = 0$, $\forall i$ is a trivial consistent estimator.

Remarks 2

- If $V(G)$ is a set of points in an Euclidean space then, by taking $g(i) =$ the Euclidean distance from i to t, we obtain a consistent estimator in the hypothesis that conditions (ii) are fulfilled.
- If g is a consistent estimator, then the choice of j^* in the Dijkstra's algorithm is made by

$$u_{j^*} + g(j^*) = \min\{u_j + g(j) \ : \ j \in V \setminus S\}.$$

The correctness of the proof is similar to that given for $g(i) = 0$, $\forall i$.

- This algorithm is part of the A* family of algorithms.

Bellman-Ford-Moore algorithm

If there is $ij \in E$ such that $a_{ij} < 0$ then Dijkstra's algorithm can fail ("best first" strategy does not work). Assuming that

(I') $$a(C) \geqslant 0, \forall C \text{ cycle in } G,$$

we will solve the system

(B) $$\begin{cases} u_s &= 0 \\ u_i &= \min_{j \neq i} (u_j + a_{ji}), \forall i \neq s \end{cases}$$

by successive approximation.

Bellman-Ford-Moore algorithm

Let us define

$$\text{(BM)} \qquad u_i^k = \min\{a(P) \,:\, P \in \mathcal{P}_{si}, |E(P)| \leqslant k\}, \forall i \in V, k = \overline{1, n-1}$$

Since the length (number of arcs) of every path in G is at most $n-1$, it follows that if we construct

$$\begin{aligned} \mathrm{u}^1 &= (u_1^1, \ldots, u_n^1), \\ \mathrm{u}^2 &= (u_1^2, \ldots, u_n^2), \\ &\vdots \\ \mathrm{u}^{n-1} &= (u_1^{n-1}, \ldots, u_n^{n-1}), \end{aligned}$$

then u^{n-1} is a solution of the system (B). Since the values of u^1 are clear, if we give a rule of passing from u^k to u^{k+1} we obtain the following algorithm to solve (B):

Bellman-Ford-Moore algorithm

$u_s^1 \leftarrow 0$;
for $(i \in V \setminus \{s\})$ do
  $u_i^1 \leftarrow a_{si}$; //obviously (BM) holds.
for $(k = \overline{1, n-2})$ do
  for $(i = \overline{1, n})$ do
    $u_i^{k+1} \leftarrow \min\left(u_i^k, \min_{j \neq i}\left(u_j^k + a_{ji}\right)\right)$;

In order to prove the correctness of this algorithm, we will show that if u^k satisfies (BM) then u^{k+1} satisfies (BM), for $k = \overline{1, n-2}$.

Bellman-Ford-Moore algorithm

(B)
$$\begin{cases} u_s &= 0 \\ u_i &= \min_{j \neq i} (u_j + a_{ji}), \forall i \neq s \end{cases}$$

For $i \in V$, let us consider the following sets of paths:

$$A = \{P \ : \ P \in \mathcal{P}_{si}, \text{ length of } P \leqslant k+1\}$$
$$B = \{P \ : \ P \in \mathcal{P}_{si}, \text{ length of } P \leqslant k\}$$
$$C = \{P \ : \ P \in \mathcal{P}_{si}, \text{ length of } P = k+1\}$$

Then, $A = B \cup C$, and

$$\min\{a(P) \ : \ P \in A\} = \min(\min\{a(P) \ : \ P \in B\}, \min\{a(P) \ : \ P \in C\})$$

Since u_i^k satisfies (BM) we have

$$\min\{a(P) \ : \ P \in A\} = \min(u_i^k, \min\{a(P) \ : \ P \in C\})$$

Bellman-Ford-Moore algorithm

Let $\min\{a(P) : P \in C\} = a(P^0)$, for $P^0 \in C$. If j is the vertex before $i \in P^0$ (such a j must exist since P^0 has at least two arcs), then $a(P^0) = a(P^0_{sj}) + a_{ji} \geqslant u_j^k + a_{ji}$ (since P^0_{sj} has k arcs and u^k satisfies (BM). Hence

$$\min\{a(P) : P \in A\} = \min\left(u_i^k, \min_{j \neq i}(u_j^k + a_{ji})\right),$$

that is, the value assigned to u_i^{k+1} in the algorithm.

The time complexity is $\mathcal{O}(n^3)$ if the minimum in the second for loop needs $\mathcal{O}(n)$ time.

The shortest paths can be obtained as in Dijkstra's algorithm, if the array *before*[], trivially initialized, is appropriately updated when the minimum in the second for loop is found.

Remarks 3

Bellman-Ford-Moore algorithm

- We can add to the algorithm the following step:

  if ($\exists i \in V$ such that $u_i^{n-1} > \min_{j \neq i} (u_j^{n-1} + a_{ji})$) then
  return " a negative cost cycle exists";

  In this way we obtain an $\mathcal{O}(n^3)$ time test if the digraph G and the cost function a violates condition (I') (otherwise u_i^{n-1} cannot be decreased). The cycle of negative cost can be found using the vector $before[]$.
- If there is $k < n-1$ such that $\mathrm{u}^k = \mathrm{u}^{k+1}$, then the algorithm can be stopped. Based on this idea, it is possible to implement the algorithm in $\mathcal{O}(nm)$ time, by keeping the vertices i for which the value u_i is changed in a queue.

P3 Given $G = (V, E)$ a digraph and $a : E \to \mathbb{R}$ find
$P^*_{ij} \in \mathcal{P}_{ij}, \forall i, j \in V$, s. t. $a(P^*_{ij}) = \min\{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}\}$

- Let $u_{ij} = \min\{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}\}$. Hence, we have to find the matrix $U = (u_{ij})_{n\times n}$, when the cost-adjacency matrix A is given.
- Each shortest path can be obtained in $\mathcal{O}(n)$ time if during the construction of the matrix U we maintain the matrix $Before = (before_{ij})_{n\times n}$, in which $before_{ij}$ =the vertex before j on the shortest path from i to j in G.
- If the pair (G, a) satisfies condition (I'), we can solve P3 by calling Bellman-Ford-Moore algorithm for $s \in \{1, \ldots, n\}$, with overall time $\mathcal{O}(n^4)$. There are faster solutions that we'll discuss in the next slides.

Iterating Dijkstra's algorithm

If all costs are non-negative, we can solve P3 by applying Dijkstra's algorithm for $s \in \{1, \ldots, n\}$, with overall time $\mathcal{O}(n^3)$.

The iteration of Dijkstra's algorithm is also possible when we have negative costs, but condition (I') is fulfilled, after a nice pre-processing. This is the Johnson's algorithm.

Let $\alpha : V \to \mathbb{R}$ such that $\forall ij \in E$, $\alpha(i) + a_{ij} \geqslant \alpha(j)$.

Let $\overline{a} : E \to \mathbb{R}_+$ given by $\overline{a}_{ij} = a_{ij} + \alpha(i) - \alpha(j)$, $\forall ij \in E$.

We have $\overline{a}_{ij} \geqslant 0$ and it is not difficult to see that for any $P_{ij} \in \mathcal{P}_{ij}$,

$$(*) \qquad \overline{a}(P_{ij}) = a(P_{ij}) + [\alpha(i) - \alpha(j)].$$

Hence, we can iterate the Dijkstra's algorithm for finding the shortest paths with respect to costs $\overline{a}$.

Iterating Dijkstra's algorithm

Relation (*) shows that a path is of minimum cost with respect to cost $\overline{a}$ if and only if it is of minimum cost with respect to cost a (since $\overline{a}(P_{ij}) - a(P_{ij})$ is a constant which does not depend on P_{ij}). So, we have the following algorithm:

1: find α and build the matrix $\overline{A}$;
2: solve P3 for $\overline{A}$, returning $\overline{U}$ and $\overline{Before}$;
3: find U $(u_{ij} = \overline{u}_{ij} - \alpha(i) + \alpha(j))$;

The step 2 needs $\mathcal{O}(n^3)$ by iterating Dijkstra's algorithm. The step 1 can be also performed in $\mathcal{O}(n^3)$, by choosing a node $s \in V$ and solving P2 with the Bellman-Ford-Moore algorithm (which tests also if condition (I') is fulfilled).

Iterating Dijkstra's algorithm

Indeed, if $(u_i, i \in V)$ is a solution for P2, then $(u_i, i \in V)$ satisfies the system (B), so $u_j = \min_{i \neq j} u_i + a_{ij}$, that is, $\forall ij \in E$, we have $u_j \leqslant u_i + a_{ij}$. Hence, $a_{ij} + u_i - u_j \geqslant 0$, $\forall ij \in E$, which shows that we can take $\alpha(i) = u_i$, $\forall i \in V$ such that the condition (*) holds.

Floyd - Warshall algorithm

Let

$$u_{ij}^k = \min \{a(P_{ij}) \ : \ P_{ij} \in \mathcal{P}_{ij}, V(P_{ij}) \setminus \{i, j\} \subseteq \{1, 2, \ldots, k-1\}\}$$

$$\forall i, j \in V, k = \overline{1, n+1}.$$

Floyd - Warshall algorithm

Obviously, $u_{ij}^1 = a_{ij}$, $\forall i, j \in V$ (we suppose that $a_{ii} = 0$, $\forall i \in V$). Moreover,

$$u_{ij}^{k+1} = \min\{u_{ij}^k, u_{ik}^k + u_{kj}^k\}, \forall i, j \in V, k = \overline{1, n}.$$

This follows by induction on k. In the inductive step: a shortest path from i to j without internal vertices $\geqslant k+1$ either does not contain the vertex k and its cost is u_{ij}^k, or contains the vertex k and then its cost is $u_{ik}^k + u_{kj}^k$ (by the Bellman's optimality principle and induction hypothesis).

Obviously, if we get $u_{ii}^k < 0$, then there exists a negativ cost cycle C passing through the vertex i with $V(C) \setminus \{i\} \subseteq \{1, \ldots, k-1\}$.

Shortest paths - Solving problem P3


```
for (i = 1,n) do
    for (j = 1,n ) do
        before(i,j) ← i;
        if (i = j) then
            a_ii ← 0; before(i,i) ← 0;
for (k = 1,n ) do
    for (i = 1,n) do
        for (j = 1,n ) do
            if (a_ij > a_ik + a_kj) then
                a_ij ← a_ik + a_kj; before(i,j) ← before(k,j);
                if (i = j and a_ij < 0) then
                    return "negative cycle";
```

This method for solving P3 is known as Floyd - Warshall algorithm: Obviously, the time complexity of the algorithm is $\mathcal{O}(n^3)$.

Remark 1

If we know that the digraph has no negative cost cycle, then if the diagonal entries of A are initialized with ∞, the final value of each diagonal entry is the minimum cost of a cycle passing through the corresponding vertex.

(Fast) matrix-multiplication

Suppose that the condition (I') is fulfilled and in the cost-adjacency matrix the diagonal entries are 0. Let

$$u_{ij}^k = \min\{a(P_{ij}) \ : \ P_{ij} \in \mathcal{P}_{ij}, P_{ij} \text{ has at most } k \text{ arcs}\}$$

$$\forall i, j \in V, k = \overline{1, n-1}.$$

Let us denote by $U^k = (u_{ij}^k)_{1 \leqslant i,j \leqslant n}$ for $k \in \{0, 1, 2, \ldots, n-1\}$, where U^0 has all entries ∞, except the diagonal entries which are all 0. Then, iterating the Bellman-Ford-Moore algorithm can be described in a matrix form as:

(Fast) matrix-multiplication

for $(i, j \in V)$ do
  if $(i \neq j)$ then
    $u_{ij}^0 \leftarrow \infty;$
  else
    $u_{ij}^0 \leftarrow 0;$
for $(k = \overline{0, n-2})$ do
  for $(i, j \in V)$ do
    $u_{ij}^{k+1} = \min_{h \in V} (u_{ih}^k + a_{hj});$

(Fast) matrix-multiplication

Let us consider the following "matrix-multiplication":

$$\forall B, C \in \mathcal{M}_{n\times n}, B \otimes C = P = (p_{ij}), \text{ where } p_{ij} = \min_{k=\overline{1,n}} (b_{ik} + c_{kj}).$$

Note that the $\otimes$ operation is associative and that it is similar to the usual matrix-multiplication.

We can write $U^{k+1} = U^k \otimes A$ and, by induction, we obtain

$$U^1 = A, U^2 = A^{(2)}, \ldots, U^{n-1} = A^{(n-1)},$$

$$\text{where } A^{(k)} = A^{(k-1)} \otimes A \text{ and } A^{(1)} = A.$$

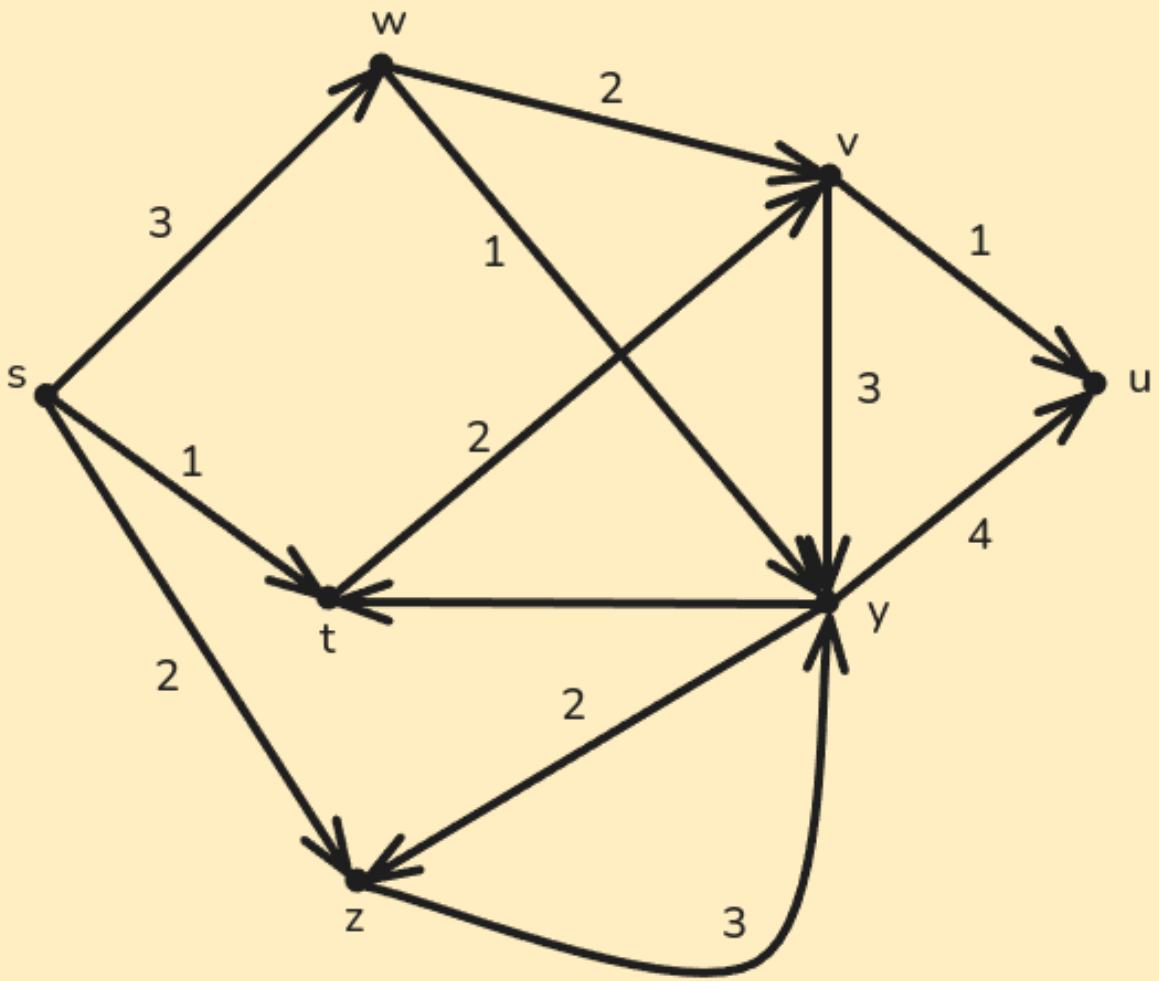
In the hypothesis (I') we have: $A^{(2^p)} = A^{(n-1)}$, $\forall p$ with $2^p \geqslant n-1$.

Hence, computing successively, $A, A^{(2)}, A^{(4)} = A^{(2)} \otimes A^{(2)}, \ldots$, we obtain an $\mathcal{O}(n^3 \log n)$ time algorithm for solving P3.

Remark 2

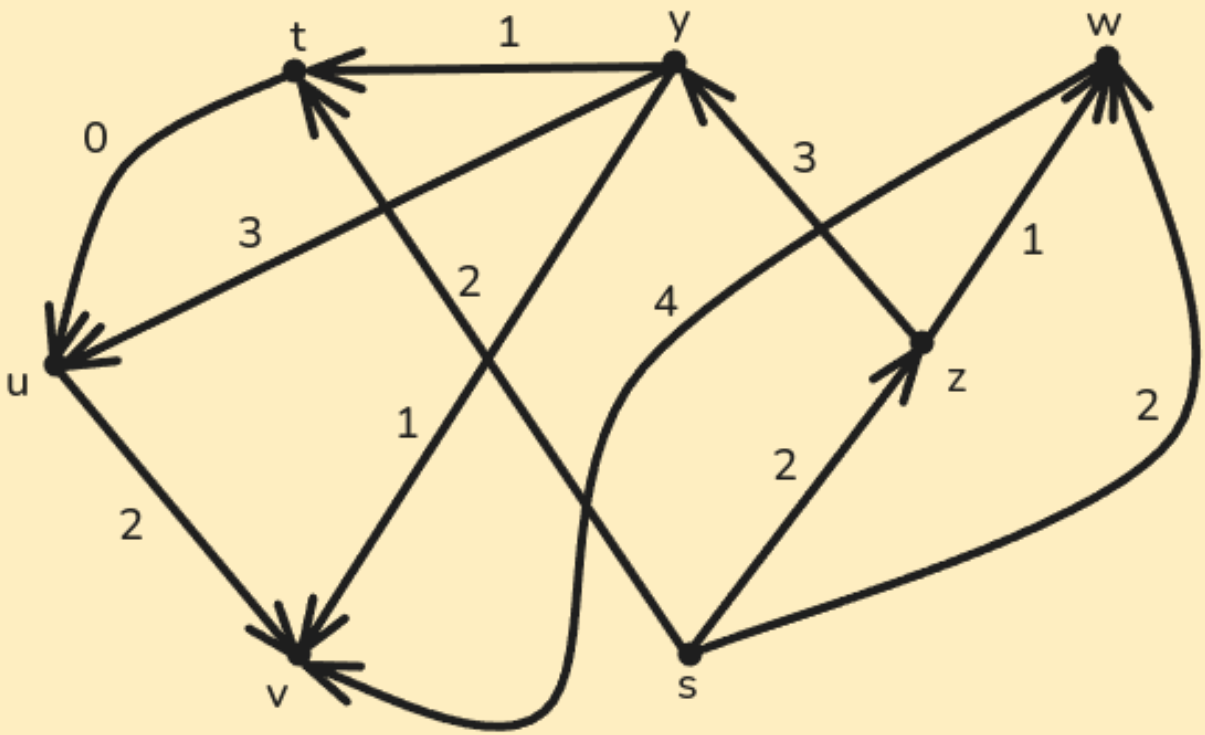
If $\otimes$ operation is implemented with faster algorithms, the above n^3 in the time complexity of the algorithm can be replaced by $n^{\log_2 7} = n^{2.81}$ (Strassen 1969), or by $n^{2.3728639}$ (Cooppersmith & Winograd 1987, Le Gall 2014).

Exercise 1. Run the Dijkstra's algorithm on the following digraph.



Exercises for the 5th seminar

Exercise 2. Find the minimum cost paths starting in s on the following acyclic digraph using the 'substitution' algorithm.



Exercise 3. A graph G is given functionally: for each vertex $v \in V(G)$ we obtain $N_G(v)$ at cost 1; alternatively, after a preprocessing of cost T ($T >> 1$), we obtain $N_G(v)$ and also $N_G(w)$, for all vertices $w \in V(G)$. In G, a path P is constructed by starting from an arbitrary vertex and choosing an unvisited neighbor from each current vertex until no choice is possible. After the path is completed we can compare its cost, $Online(P)$, with the best possible cost at which it can be obtained, $Offline(P)$. Devise a strategy of accessing the sets of neighbors such that

$$Online(P) \leqslant \left(2 - \frac{1}{T}\right) \cdot Offline(P).$$

Exercises for the 5th seminar

Exercise 4. Let D be a digraph and $a : E(D) \to \mathbb{R}_+$, $b : E(D) \to \mathbb{R}_+^*$. Devise an efficient algorithm for finding a cycle C^* of D, such that

$$\frac{a(C^*)}{b(C^*)} = \min\left\{\frac{a(C)}{b(C)} \ : \ C \text{ cycle in } D\right\}.$$

Exercise 5. In the problem of finding the shortest paths from a given vertex s to all vertices of a digraph $G = (V, E)$, we have a cost function $c : E \to \{0, 1, \ldots, C\}$ where $C \in \mathbb{N}$ does not depend on $n = |V|$ or $m = |E|$. How do you modify the Dijkstra's algorithm in order to reduce the time complexity to $\mathcal{O}(n + m)$?

Exercise 6. Let $D = (V, E)$ be a strongly connected digraph of order n and $a : E \to \mathbb{R}$ a cost function on its arcs. If X is a walk, a path or a cycle in D, then $a(X)$, the cost of X, is the sum of the cost on its arcs, $len(X)$, the length of X, is the number of its arcs, and $a_{avg}(X)$, the average cost of its arcs, is $a_{avg}(X) = \dfrac{a(X)}{len(X)}$. Let

$$a^*_{avg} = \min_{C \text{ cycle in } D} a_{avg}(C).$$

For a given vertex $s \in V$ and $k \in \mathbb{N}^*$, we denote by $A_k(v)$ the minimum cost of a sv-walk of length k in D (if any, otherwise $A_k(v) = \infty$).

Exercise 6. (cont'd)

a) If D doesn't contain negative cost cycles, but there exists a cycle C with $a(C) = 0$, prove that there exists a vertex $v \in V$ such that

$$A_n(v) = \min\{a(P) \ : \ P \text{ is a } sv\text{-path from } s \text{ to } v \text{ in } D\}.$$

b) Show that if $a^*_{avg} = 0$, then

$$(MMC) \quad a^*_{avg} = \min_{v \in V} \max_{0 \leqslant k \leqslant n-1} \frac{A_n(v) - A_k(v)}{n-k}.$$

c) If $a^*_{avg} \neq 0$, transform the function a such that the new function a' satisfy the hypothesis from b) and from (MMC) which holds a' follows that (MMC) holds for every function a.

Exercises for the 5th seminar

Exercise 7. In various applications for a given digraph $G = (V, E)$, $a : E \to \mathbb{R}_+$ we have to persistently answer to a question of the following type: What is the shortest path between s and t? $(s, t \in V,\ s \neq t)$. For a very large digraph, G, it is proposed the following bidirectional Dijkstra algorithm:

- build up the inverse of G, G', with a cost function $a' : E(G') \to \mathbb{R}_+$, given by $a'_{ij} = a_{ji}$, $\forall ij \in E(G')$;
- successively apply a step from Dijkstra's algorithm to G and a (starting from s) and to G' and a' (starting from t);
- when vertex u is introduced in S (the set of labeled nodes in Dijkstra's algorithm) by both instances of the algorithm we stop;
- return the path from s to u in G joined with the inverse of the path from t to u in G'.

Prove that this procedure it is not correct, by giving an example that is inconsistent with it.

Exercise 8. Give an example of a digraph with negative-cost arcs for which Dijkstra's algorithm fails.

Exercise 9. Let $G = (V, E)$ be a digraph, $a : E \to \mathbb{R}_+$ a cost function defined on its arcs, and $x_0 \in V$ from wich any other vertex in G is accessible. An SP-tree for the triple (G, a, x_0) is a directed rooted (in x_0) tree of G, $T = (V, E')$ such that the cost of the path from x_0 to u in T is the cost of the shortest path from x_0 to u in G, for every $u \in V$.

(a) Prove that an SP-tree always exists.

(b) Devise an algorithm that finds an SP-tree.

Exercises for the 5th seminar

Exercise 10. Let G be a connected graph. Prove that

a) any two maximum length paths of G have a non-empty intersection.

b) if G is a tree, then all maximum length paths in G have a non-empty intersection.

Exercise 11. Let $G = (V, E)$ be a connected graph.

a) Show that there exists a stable set, S, such that the spanning (bipartite) graph $H = (S, V \setminus S; E')$ is connected, where $E' = E \setminus \binom{V \setminus S}{2}$.

(b) Prove that $\alpha(G) \geqslant \dfrac{|G| - 1}{\Delta(G)}$.

Exercises for the 5th seminar

Exercise 12. Prove that any tree, T, has at least $\Delta(T)$ pendant nodes.

Exercise 13. Let G be a graph with $n \geqslant 2$ vertices and m edges.

a) Prove that G contains at least two vertices of the same degree.

b) Let $r(G)$ the maximum number of vertices having the same degree in G. If we denote by $d_{med} = 2m/n$, show that

$$r(G) \geqslant \left\lceil \frac{n}{2d_{med} - 2\delta(G) + 1} \right\rceil .$$

Exercise 14. Prove that a digraph has an unique topological ordering if and only if it has a Hamiltonian path and all other arcs are directed forward with respect to a traversal of this path.

Exercise 15. Let $G = (S, T; E)$ be a bipartite graph with the following properties:

- $|S| = n, |T| = m$ $(n, m \in \mathbb{N}^*)$;
- $\forall t \in T, |N_G(t)| > k > 0$ (for a given $k < n$);
- $\forall t_1, t_2 \in T$, if $t_1 \neq t_2$, then $|N_G(t_1) \cap N_G(t_2)| = k$;

Prove that $m \leqslant n$;

Exercise 16. Let $G = (V, E)$ be a digraph; we define a function $f_G : \mathcal{P}(V) \to \mathbb{N}$ by $f_G(\varnothing) = 0$ and $f_G(S) = |\{v \ : \ v \text{ is accesible from } S\}|$, for $\varnothing \neq S \subseteq V$.

(a) Prove that, for every digraph G, we have

$$f_G(S) + f_G(T) \geqslant f_G(S \cup T) + f_G(S \cap T), \forall\, S, T \subseteq V.$$

(b) Prove that (a) is equivalent with

$$f_G(X \cup \{v\}) - f_G(X) \geqslant f_G(Y \cup \{v\}) - f_G(Y), \forall X \subseteq Y \subseteq V, \forall v \in V \setminus Y$$

Exercises for the 5th seminar

Exercise 17. A student decided to run each day from his home to the university campus. The city map contains m two-ways street segments (a segment lies between two intersections) whose strictly positive lengths are known. He wants to go only uphill (hence slower in the first part of his running) before going only downhill (hence faster in the second part). (An uphill street segment is a segment ending in an intersection of strictly larger altitude than its starting intersection; a downhill street segment is a segment ending in an intersection of strictly lower altitude than its starting intersection.)

(a) Describe an efficient algorithm ($\mathcal{O}(n+m)$ time complexity, where n is the number of intersections) for finding the minimum cost path meeting the student requirements (if such a path exists).

Exercise 17. (cont'd)

(b) After a few days of running he finds that the structure of the path makes it too exhausting and decide to accept routes that contains level street segments, hence, in the first part of the path he wants to go uphill or level, while in the second part he will accept downhill or level segments.
Give an efficient algorithm to solve this new problem; what is the time complexity of such an algorithm?

Exercise 18. Let $G = (V, E)$ be a digraph and let $\mathcal{V}$ be the family strongly connected components of G. Define the digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where for every $H', H'' \in \mathcal{V}$, $H'H'' \in \mathcal{E}$ if there exists an arc $v'v'' \in E$ such that $v' \in V(H')$ and $v'' \in V(H'')$. An ordering $v_1, v_2, \ldots, v_n$ of the vertices of G is called a *SP-topological ordering* if for every arc $v_j v_i \in E$, with $1 \leqslant i < j \leqslant n$, the vertices $v_i, v_{i+1}, \ldots, v_j$ belong to the same strongly connected component of G.

Exercise 18. (cont'd)

(a) Prove that $\mathcal{G}$ is an acyclic digraph.

(b) Devise an $\mathcal{O}(|V|+|E|)$ time complexity algorithm for finding a SP-topological ordering of the vertices of G.

(c) Prove that for any path D in G there exists a SP-topological ordering of the vertices of G $v_1, v_2, \ldots, v_n$ such that for every two vertices $v_i, v_j \in V(D)$, if v_i occurs before v_j on D, then $i < j$.