Exercise sheet 9.

Advanced Algorithms

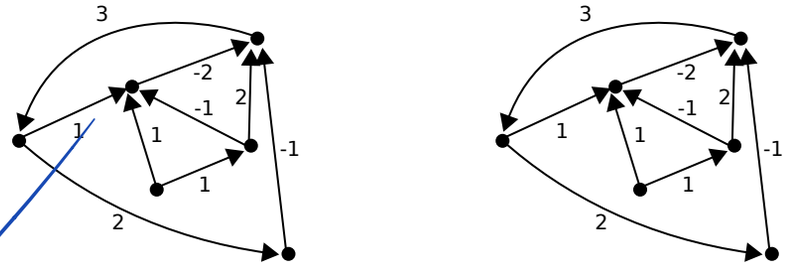
WiSe 2020/21

Instructor: László Kozma

Due 18:00, January 24th, 2021

Exercise 1 All-pairs shortest paths *11 pts*

(a) (3 pts) Consider the edge-weighted directed graph shown on the figure (in two copies, to help with sketching). Find a valid vertex potential function, and use it to reweight the graph, as in Johnson's algorithm discussed in the lecture.



(b) (4 pts) The Floyd-Warshall algorithm for a graph with vertex set $\{1, \ldots, n\}$ and edge-weights $w(i, j)$, where a missing edge has weight ∞, can be described as follows:

```
d(:,:) = w(:,:)
for k=1..n:
  for i=1..n:
     for j=1..n:
        d(i,j) = min{d(i,j), d(i,k)+d(k,j)}
```

Suppose that you forgot the correct nesting of the loops, and instead of the order k, i, j, you write them in the order (i) i, k, j, or (ii) i, j, k, leaving the last line unchanged.

Are any of these variants still correct? If yes, argue why, if not, give a small counterexample.

(c) (4 pts) Describe the modification necessary to the Floyd-Warshall algorithm, that allows the efficient construction of the actual shortest paths. ($O(n^2)$ extra storage should be sufficient.)

Exercise 2 Approximate distances *7 pts*

Consider the following, "simplified" version of the *APD* algorithm discussed in the lecture. (The original APD algorithm computes, given the adjacency matrix A of an unweighted, undirected, connected graph, the pairwise distance matrix D, in time $O(M(n)\log n)$, where $M(n)$ is the time necessary to multiply two $n \times n$ matrices.)

$AAPD(A)$:
if $A_{i,j} = 1$ for all $i \neq j$ **then return** A
$Z = A^2 + A$
let B be an $n \times n$ matrix, where

$$B_{i,j} = \begin{cases} 1 \text{ if } Z_{i,j} \geq 1, & \text{and } i \neq j \\ 0 \text{ otherwise} \end{cases}$$

let $T = AAPD(B)$
return $n \times n$ matrix D, where:

$$D_{i,j} = \begin{cases} 1 \text{ if } A_{i,j} = 1 \\ 2 \cdot T_{i,j} \text{ otherwise.} \end{cases}$$

(a) (4 pts) The distances returned by this simplified algorithm are clearly not always correct. Give a small example that makes this evident, and sketch the execution by hand.

(b) (3 pts) Show that the distances returned by the algorithm (denoted D) are never smaller than the true distances (denoted Δ), meaning $\Delta_{i,j} \leq D_{i,j}$ for all i, j.

(c) (bonus 3 pts) Show that the distances returned by the algorithm are never more than twice the true distances: $D_{i,j} \leq 2\Delta_{i,j}$ for all i, j.

Exercise 3 Set disjointness *4 pts*

This problem is not directly related to graphs, its purpose is to illustrate the usefulness of matrix multiplication in solving problems.

Suppose we are given n subsets of the set $\{1, 2, \ldots, n\}$, denoted $A_1, \ldots, A_n$. Describe an algorithm that runs in time $o(n^3)$ and finds a pair i, j of indices $1 \leq i, j \leq n$ such that A_i and A_j are disjoint (or reports that there are no such pairs).

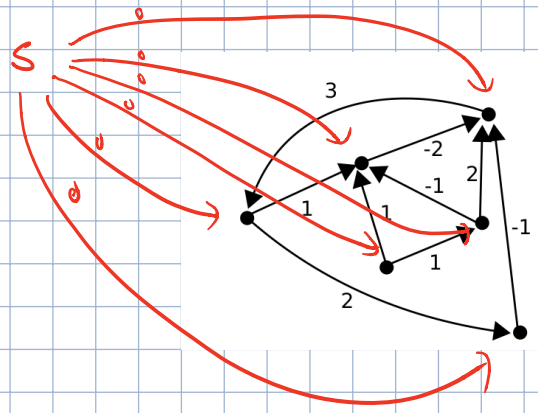
Hint: Matrix multiplication.

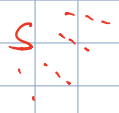
Total: 22 points. Have fun with the solutions!

From: Yumeng Li and Thore Brehmer

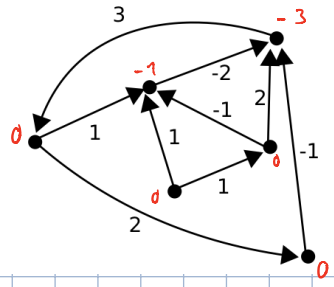
Exercise 1 All-pairs shortest paths *11 pts*

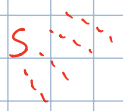
(a) (3 pts) Consider the edge-weighted directed graph shown on the figure (in two copies, to help with sketching). Find a valid vertex potential function, and use it to reweight the graph, as in Johnson's algorithm discussed in the lecture.



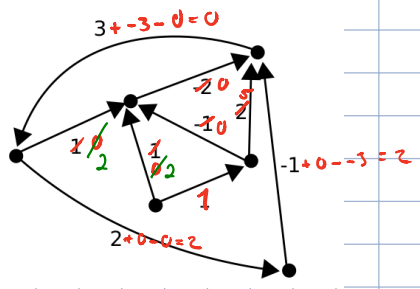


d(s, ·)





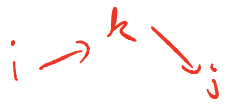
new-edge-weights:



$$d(u,v) = d'(u,v) + p(u) - p(v)$$

3

(b) (4 pts) The Floyd-Warshall algorithm for a graph with vertex set $\{1, \ldots, n\}$ and edge-weights $w(i, j)$, where a missing edge has weight ∞, can be described as follows:



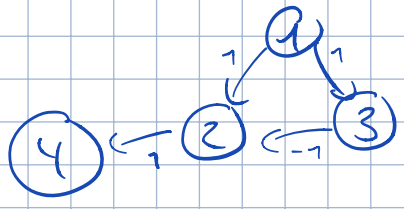

```
d(:,:) = w(:,:)
for k=1..n:
  for i=1..n:
     for j=1..n:
        d(i,j) = min{d(i,j), d(i,k)+d(k,j)}
```

Suppose that you forgot the correct nesting of the loops, and instead of the order k, i, j, you write them in the order (i) i, k, j, or (ii) i, j, k, leaving the last line unchanged.

Are any of these variants still correct? If yes, argue why, if not, give a small counterexample.

(i) i, k, j Not working

For i = 1:



init =

i\j	1	2	3	4
1	0	1	1	∞
2	∞	0	∞	1
3	∞	-1	0	∞
4	∞	∞	∞	0

k=1:

d(1,1) = 0
d(1,2) = 1
d(1,3) = 1
d(1,4) = ∞

k=2:

d(1,1) = 0
d(1,2) = 1
d(1,3) = 1
d(1,4) = 2

k=3:

d(1,1) = 0
d(1,2) = 0
d(1,3) = 1
d(1,4) = 2

k=4:

d(1,1) = 0
d(1,2) = 0
d(1,3) = 1
d(1,4) = 2

i=1:

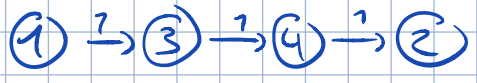
k\j	1	2	3	4
1	0	1	1	∞
2	0	1	1	2
3	0	0	1	2
4	0	0	1	2

d(1,4) = 2 but should be 1 ↯

distance from i=1 cannot be changed anymore, as the for loop is now on the second (i=2) iteration

⇒ (i) i, k, j will not work

(ii) i, j, k Not working e.g.



init =

i \	1	2	3	4
1	0	∞	1	∞
2	∞	0	∞	∞
3	∞	∞	0	1
4	∞	1	∞	0

$$d(1,2) = \min_{k=0}^{4} \{ d(1,2),\ d(1,k) + d(k,2) \}$$

$$= \min \{ d(1,2),\ d(1,1) + d(1,2),\ d(1,2) + d(2,2),\ d(1,3) + d(3,2),\ d(1,4) + d(4,2) \}$$

$$= \min \{ \infty,\ 0 + \infty,\ \infty + 0,\ 1 + \infty,\ \infty + 1 \} = \infty$$

$\Rightarrow$ $d(1,2) = \infty$ even tho it should be 3

distance from i=1 cannot be changed anymore, as the for loop is now on the second (i=2) iteration

$\Rightarrow$ (i) i, k, j will not work 4

(c) (4 pts) Describe the modification necessary to the Floyd-Warshall algorithm, that allows the efficient construction of the actual shortest paths. ($O(n^2)$ extra storage should be sufficient.)

```
d(:,:) = w(:,:)
for k=1..n:
  for i=1..n:
     for j=1..n:
        d(i,j) = min{d(i,j), d(i,k)+d(k,j)}
```

old = d(i,j)

```
# if a new min distant was found the path needs to be updated
  if old ≠ d(i,j):
      path_matrix [i][j] = path_matrix [i][k]
```

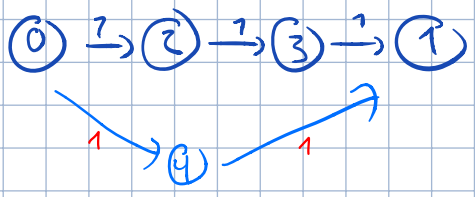
path_matrix is initialized similar to the matrix for the Floyd-Warshall algorithm. But every entry ≠ ∞ is = j

- Modification should work as it only updates the path_matrix when a new shorter path has been found.
- Also the path always updates to path_matrix [i][k] ⇔ as the shortest path from i j has k as middle edge

to print the path (s, t):

```
path =[s]
while s ≠ t:
    s = path_matrix [s][t]
    path.append(s)
```

e.g.



init_A =

	0	1	2	3	4
0	0	∞	1	∞	1
1	∞	0	∞	∞	∞
2	∞	∞	0	1	∞
3	∞	1	∞	0	∞
4	∞	1	∞	∞	0

⇒ D =

	0	1	2	3	4
0	0	2	1	2	1
1	∞	0	∞	∞	∞
2	∞	2	0	1	∞
3	∞	1	∞	0	∞
4	∞	1	∞	∞	0

init path_matrix =

	0	1	2	3	4
0	0	∞	~~0~~ 2	∞	~~0~~ 4
1	∞	1	∞	∞	∞
2	∞	∞	2	~~2~~ 3	∞
3	∞	~~3~~ 1	∞	3	∞
4	∞	~~4~~ 1	∞	∞	4

⇒ path_matrix =

	0	1	2	3	4
0	0	4	2	2	4
1	∞	1	∞	∞	∞
2	∞	3	2	3	∞
3	∞	1	∞	3	∞
4	∞	1	∞	∞	4

path(0, 1) ?

[0] + path(path_matrix[0][1], 1)

= [0] + [4] + path(path_matrix[4][1], 1)

= [0] + [4] + [1] s==t (1==1)

= [0, 4, 1]

4

3. for each set A_i, we define a n-dim row vector V_i.
where $V_{ij} = \begin{cases} 1 & \text{if } j \in A_i \\ 0 & \text{otherwise} \end{cases}$

then $A_i \cap A_j = \phi$ iff $V_i V_j^T = 0$, i.e the inner product of V_i & V_j is 0. We can build a $n \times n$ matrix V,

$V = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \end{bmatrix}$ then we compute $W = VV^T$,

if $W_{ij} = 0$, then A_i & A_j are disjoint.

The time complexity is the same as the matrix multiplication VV^T, which is $O(n^3)$. 4

2. (a)

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

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

$= \begin{pmatrix} 1&0&1&0 \\ 0&2&0&1 \\ 1&0&2&0 \\ 0&1&0&1 \end{pmatrix} + \begin{pmatrix} 0&1&0&0 \\ 1&0&1&0 \\ 0&1&0&1 \\ 0&0&1&0 \end{pmatrix} = \begin{pmatrix} 1&1&1&0 \\ 1&2&1&1 \\ 1&1&2&1 \\ 0&1&1&1 \end{pmatrix}$

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

T = AAPD(B):

A = B,

$Z' = A \hat{+} A = \begin{pmatrix} 1&1&1&0 \\ 1&1&0&1 \\ 1&1&1&1 \\ 0&1&1&1 \end{pmatrix} \begin{pmatrix} 1&1&1&0 \\ 1&1&0&1 \\ 1&1&1&1 \\ 0&1&1&1 \end{pmatrix} + \begin{pmatrix} 1&1&1&0 \\ 1&1&0&1 \\ 1&1&1&1 \\ 0&1&1&1 \end{pmatrix}$

$= \begin{pmatrix} 3&3&3&2 \\ 3&4&4&3 \\ 3&4&4&3 \\ 2&3&3&3 \end{pmatrix} + \begin{pmatrix} 1&1&1&0 \\ 1&1&1&1 \\ 1&1&1&1 \\ 0&1&1&1 \end{pmatrix} = \begin{pmatrix} 4&4&4&2 \\ 4&5&5&4 \\ 4&5&5&4 \\ 2&4&4&4 \end{pmatrix}$

$B' = \begin{pmatrix} 0&1&1&1 \\ 1&0&1&1 \\ 1&1&0&1 \\ 1&1&1&0 \end{pmatrix}$

T' = AAPD(B'):

T' = B' since $B'[i,j] = 1 \ \forall \ i,j$.

$D' = \begin{pmatrix} 1&1&1&2 \\ 1&1&1&1 \\ 1&1&1&1 \\ 2&1&1&1 \end{pmatrix} = \begin{cases} 1 & \text{if } B[i,j] = 1 \\ 2 \cdot B'[i,j] & \text{otherwise.} \end{cases}$

T = D'

$D = \begin{pmatrix} 2&1&2&4 \\ 1&2&1&2 \\ 2&1&2&1 \\ 4&2&1&2 \end{pmatrix}$

So $d[1,4] = D[0,3] = 4$, however, the correct distance between vertex 1 & 4 is 3. 4

(b). In the APD algorithm, the update of D is

APD:

$D_{ij} = \begin{cases} 2T_{ij} & \text{if } D_{ij} \text{ even} \\ 2T_{ij}-1 & \text{if } D_{ij} \text{ odd} \end{cases}$

However, in AAPD algorithm, the update of D is

AAPD

$D_{ij} = \begin{cases} 1 & \text{if } A_{ij} = 1 \\ 2T_{ij} & \text{otherwise.} \end{cases}$

if $A_{ij} = 1$, both algorithms get $D_{ij} = 1$,

if $A_{ij} \neq 1$, then by above formulas, the D_{ij} in AAPD is never smaller than D_{ij} in APD, therefore the true distance $\Delta_{ij} \leq D_{ij}$. correct intuition, but would need to be done more formally :)