Technical University of Denmark

Written exam,

Course name: Algorithms and data structures II.

Course number: 02110.

Aids allowed: All aids. No open internet.

Exam duration:

Weighting: Question 1: 32% - Question 2: 15% - Question 3: 7% - Question 4: 20% - Question 5: 26%

The weighting is only an approximative weighting.

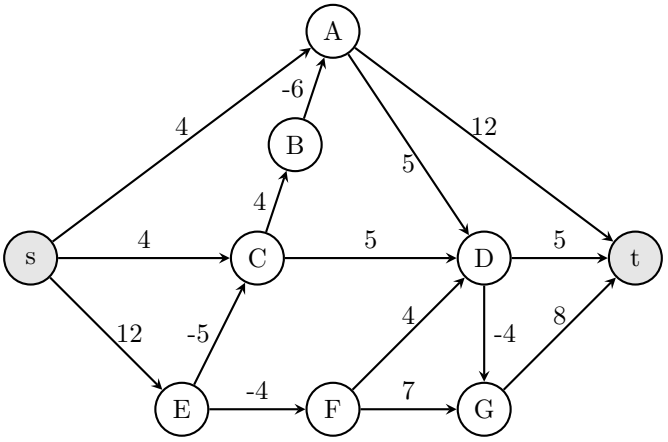
You can answer the exam in either Danish or English.

All questions should be answered by filling out the box below the question.

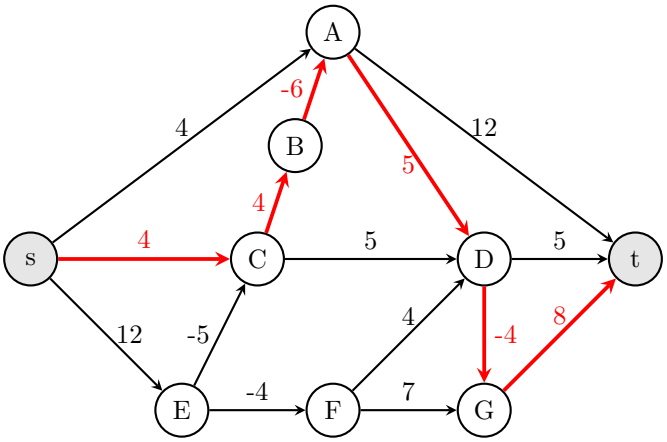
We recommend that you use a pdf-editor to do so. Please make sure that the resulting file is readable by standaed pdf-readers e.g. by printing it to a pdf-file and uploading this.

Question 1

Question 1.1 (8%) Find a shortest path from s to t in the graph below. Mark the edges on the shortest path and write the length of the path.



Solution:



Length of shortest path: 11

Question 1.2 (8%) Compute the Fenwick tree F for the array A below:

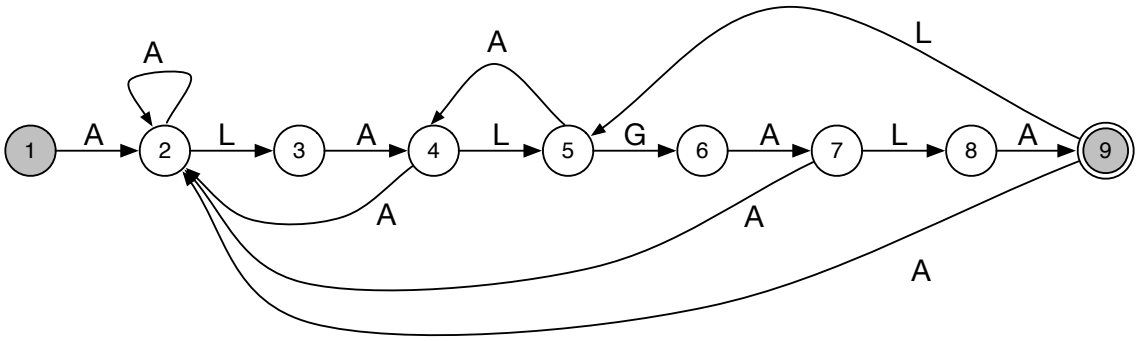
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$A[i]$	–	5	3	6	1	1	3	2	4	4	5	2	5	7	6	5	3

Solution:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$F(i)$	–	5	8	6	15	1	4	2	25	4	9	2	16	7	13	5	62

Question 1.3 (8%) Draw the string-matching finite automaton for the string `ALALGALA`: (You don't have to draw the strings going back to the start state.)

Solution:



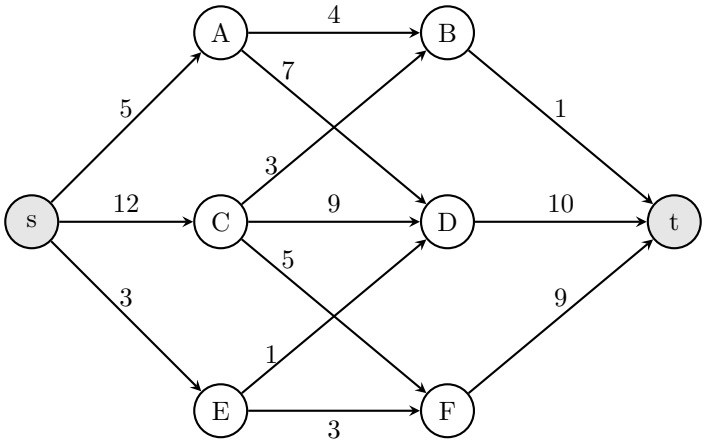
Question 1.4 (8%) Compute the prefix function as used in the Knuth-Morris-Pratt algorithm for the string `ALALGALA`:

Solution:

i	1	2	3	4	5	6	7	8
$P[i]$	`A`	`L`	`A`	`L`	`G`	`A`	`L`	`A`
$\pi[i]$	0	0	1	2	0	1	2	3

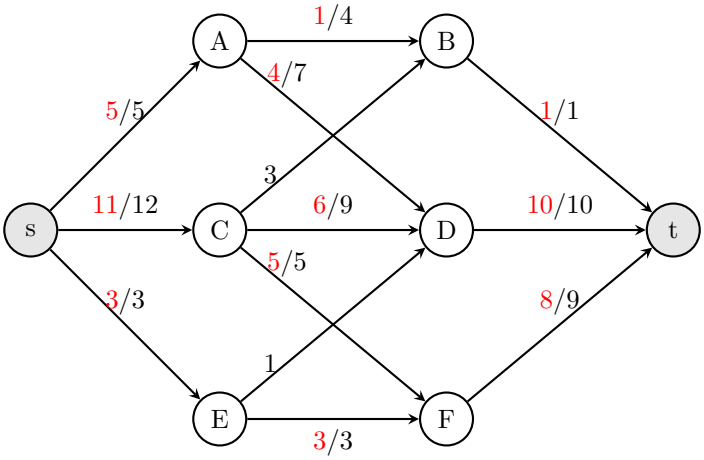
Question 2

Consider the network N below with capacities on the edges.



Question 2.1 (8%) Give a maximum flow from s to t in the network N (write the flow for each edge along the edges on the graph below), give the value of the maximum flow, and give a minimum $s-t$ cut (give the partition of the vertices).

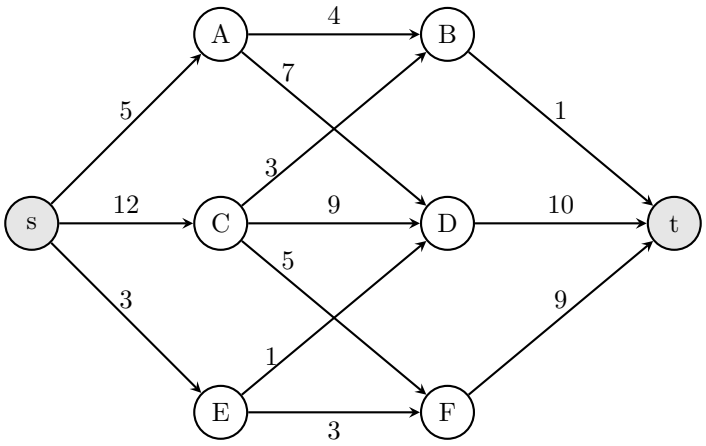
Solution:



value of flow: 19

minimum cut: $\{s, A, B, C, D\}$, $\{E, F, t\}$ or $\{s, A, B, C, D, E\}$, $\{F, t\}$

Question 2.2 (7%) Use the *Scaling Max-Flow Algorithm* to compute a maximum flow in the network N (the network is shown again below). For each augmenting path write the nodes on the path and the value you augment the path with in the table below.



Solution:

Augmenting path	Value
$sCDt$	9
$sACDFt$	5
$sEFt$	3
$sCDt$	1
$sCBt$	1

Question 3

Consider the following recurrence: $T(n) = T(n/2) + 4T(n/4) + cn^2$. Solve the recurrence using either a recursion tree or the substitution method.

Solution:

Solution to recurrence: $O(n^2)$

Argumentation/proof:
First layer: cn^2
2nd layer: $c(n/2)^2 + 4c(n/4)^2 = cn^2/4 + cn^2\frac{4}{16} = \frac{1}{2}cn^2$
3rd layer: $c(n/4)^2 + 4c(n/8)^2 + 4(c(n/8)^2 + 4c(n/16)^2 = (\frac{1}{16} + \frac{4}{64} + \frac{4}{64} + \frac{4}{64})cn^2 = \frac{1}{4}cn^2$
Guess: $\frac{cn^2}{2^{k-1}}$.

Height of tree: $\lg n$.

Get:

$$\sum_{k=1}^{\lg n} \frac{cn^2}{2^{k-1}} = \sum_{i=0}^{\lg n-1} \frac{cn^2}{2^i} = cn^2 \sum_{i=0}^{\lg n-1} \frac{1}{2^i} < 2cn^2.$$

Substitution method: Guess $T(n) \leq 2cn^2$. Base case assume $n = 1$ then $T(1) = c$.

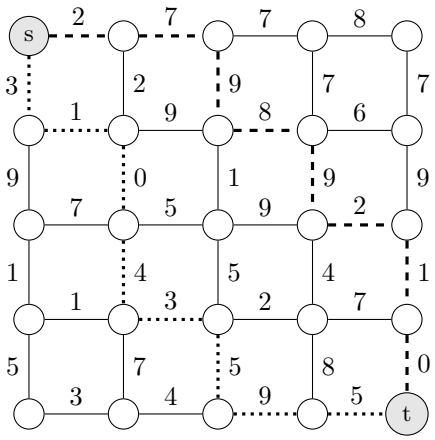
Base case: $T(1) = c \leq 2c1^2 = 2c$. Ok.
Induction:
$T(n) = T(n/2) + 4T(n/4) + cn^2$
$\leq 2c(n/2)^2 + 4 \cdot 2c(n/4)^2 + cn^2$
$= cn^2/2 + 4 \cdot 2cn^2/16 + cn^2$
$= cn^2/2 + cn^2/2 + cn^2$
$= 2cn^2$

Question 4

Your are helping the tourist guide company "Manhattan Tourists", that are arranging guided tours of the city. They want to find a walk between two points on the map that is both interesting and short. The map is a square grid graph. The square grid graph has n rows with n nodes in each row. Let node $v_{i,j}$ denote the jth node on row i. For $1 \leq i < n$ and for $1 \leq j \leq n$ node $v_{i,j}$ is connected to $v_{i+1,j}$. And for $1 \leq i \leq n$ and for $1 \leq j < n$ node $v_{i,j}$ is connected to $v_{i,j+1}$. The edges have non-negative edge weights that indicate how interesting that street is. See the graph below for an example of a 5×5 grid graph.

They want to find a short interesting walk from the upper left corner ($s = v_{1,1}$) to the lower right corner ($t = v_{n,n}$). More precisely, they want to find a path with the possible smallest number of edges, and among all paths with this number of edges they want the path with the maximum weight (the weight of a path is the sum of weights on the path).

All shortest paths have $2n - 2$ edges and go from s to t by walking either down or right in each step. In the example below two possible shortest paths (of length 8) are indicated. The dashed path has weigth 38 and the dotted path has weight 30.



Let $W[i, j]$ be the maximal weight you can get when walking from s to $v_{i,j}$ walking either down or right in each step. Let $D[i, j]$ be the weight of the edge going down from $v_{i,j}$ and let $R[i, j]$ be the weight of the edge going right from $v_{i,j}$.

Question 4.1 (5%) Fill out the table for the example above:

$W_{i,j}$	1	2	3	4	5
1	0	2	9	16	24
2	3	4	18	26	32
3	12	19	24	35	41
4	13	23	29	39	46
5	18	30	34	47	52

Question 4.2 (5%) Which of the following recurrences correctly computes $W[i,j]$:

A
$$W[i,j] = \begin{cases} 0 & \text{if } i=1 \text{ and } j=1 \\ W[i-1,1] + D[i-1,1] & \text{if } i>1 \text{ and } j=1 \\ W[1,j-1] + R[1,j-1] & \text{if } i=1 \text{ and } j>1 \\ W[i-1,j-1] + \max\{D[i-1,j-1] + R[i,j-1], R[i-1,j-1] + D[i-1,j]\} & \text{otherwise} \end{cases}$$

B
$$W[i,j] = \begin{cases} 0 & \text{if } i=1 \text{ and } j=1 \\ W[i-1,1] + D[i-1,1] & \text{if } i>1 \text{ and } j=1 \\ W[1,j-1] + R[1,j-1] & \text{if } i=1 \text{ and } j>1 \\ \max\{W[i,j-1] + D[i,j-1], W[i-1,j] + R[i-1,j]\} & \text{otherwise} \end{cases}$$

C
$$W[i,j] = \begin{cases} 0 & \text{if } i=1 \text{ and } j=1 \\ W[i-1,1] + D[i-1,1] & \text{if } i>1 \text{ and } j=1 \\ W[1,j-1] + R[1,j-1] & \text{if } i=1 \text{ and } j>1 \\ \max\{W[i-1,j] + D[i-1,j], W[i,j-1] + R[i,j-1]\} & \text{otherwise} \end{cases}$$

Question 4.3 (10%) Write pseudocode for an algorithm *based on dynamic programming and the recurrence from Question 4.2* that computes the maximum weight a shortest path can have in the grid graph.

Analyze the space usage and running time of your algorithm in terms of n.

Solution:

```
COMPUTE-MAXWEIGHT(R[1...n],D[1...n])
   W := n × n table
   W[1,1] = 0
   for i=2 to n do
      W[i, 1] := W[i-1,1] + D[i-1,1]
   for j=2 to n do
      W[1, j] := W[1,j-1] + R[1,j-1]
   for i = 2 to n do
      for j = 2 to n do
         if W[i-1,j]+D[i-1,j] > W[i,j-1]+R[i,j-1] then
            W[i,j] := W[i-1,j]+D[i-1,j]
         else
            W[i,j] := W[i,j-1]+R[i,j-1]
         end if
      end for
   end for
   return W[n,n]
```

Solution 4.3 continued:
Since we store the results to the subproblems in a table W of size $n \times n$ we use $\Theta(n^2)$ space. There are two nested for loops. All other operations can be done in constant time, and thus the total time of the algorithm is $\Theta(n^2)$.

Question 5

The tourist guide company "Manhattan Tourists" are arranging several tours of the city at the same time. As in Question 4 the map is a square grid graph, but now some of the streets are blocked, so they can't walk on these. They want to arrange several tours from s to t that do not overlap.

Question 5.1 (10%)

The company wants to arrange two different tours from s to t that are disjoint in the sense that they don't share any streets. In this question you don't care whether the tours are interesting (the weight on the path does not matter). The tours should still only walk down and right in each step and you cannot walk on the blocked streets. Give an efficient algorithm that determines wether it is possible to make two such tours. Analyze the asymptotic running time of your algorithm in terms of n. Remember to argue that your algorithm is correct.

Solution:
Construct flow network. Source is s, sink is t. Orient all edges to the right and down. Edges that corresponds to blocked streets are deleted. Set all capacitites to 1 and find a maximum flow. If the maximum flow is at least 2 there are two edge disjoint paths from s to t not using blocked roads. and we return yes. Otherwise we return no.s
Running time: We have n^2 nodes and $O(n^2)$ edges. The flow network can be constructed in linear time in the size of the graph by running through the input graph and constructing the adjacency list on the way by inserting each non-blocked edge as an arc with capacity 1. The maximum flow is at most two as there are at most two edges out of s. Thus using Ford-Fulkerson we get $O(2n^2) = O(n^2)$.
Correctness We will argue that finding two edge disjoint s-t paths in the directed network is equivalent to solving the problem. That is, that there are two edge disjoint s-t paths in the flow network if and only if there are two street disjoint tours in the original problem.

- $\Rightarrow$: The two edge disjoint paths only use edges corresponding to non-blocked roads. Due to the direction on the edges the paths always move down or right. Thus each of the paths corresponds to a valid tour. Since they don't share any edges the two corresponding tours do not share any streets.
- $\Leftarrow$: All streets corresponding to the two tours are directed edges in the network as all non-blocked streets corresponds to an edge. The tours always move down or right thus following the directions on the edges in the network. Therefore the two tours corresponds to two paths in the network. Since the tours are street disjoint the paths are edge disjoint.

We know from the book (7.43) that there are 2 edge disjoint paths in a directed graph G if and only if the value of the maximum flow in G is at least 2. This concludes the argument that the algorithm is correct.

Remark: Just running BFS twice does not work. Consider the graph:

$$\begin{array}{ccccc} x & \to & x & & x \\ \downarrow & & \downarrow & & \\ x & \to & x & \to & x \\ \downarrow & & & & \downarrow \\ x & \to & x & \to & x \end{array}$$

Solution 5.1 continued:

Question 5.2 (8%)

It turned out that the tours were not different enough. The company now wants to arrange two different tours from s to t that are disjoint in the sense that the two tours only meet in s and t. In this question you don't care wether the tours are interesting (the weight on the path does not matter). The tours should still only walk down and right in each step and you cannot walk on the blocked streets. Give an algorithm that determines wether it is possible to make two such tours. Analyze the asymptotic running time of your algorithm in terms of n. Remember to argue that your algorithm is correct.

Solution: Same network as before, but add vertex capacities: Replace each node v (except s and t) with two nodes v_i and v_o. Add an edge from v_i to v_o with capacity 1. Edges going into v now goes into v_i and edges out of v goes out of v_o.
As before find max-flow....
Running time as before as number of nodes is $O(n^2)$ and number of edges is $O(n^2)$ and the max flow is still at most 2.
Correctness: ...

Solution 5.2 continued:

Question 5.3 (8%)

The company would like their tours to be not only disjoint but also interesting. So they want to arrange two *street disjoint* tours where the least interesting street on the tours is as interesting as possible. That is, find the maximum value w such that there are two street disjoint tours that both uses only streets of weight w or higher. The tours should still only walk down and right in each step and you cannot walk on the blocked streets. The weights on the edges are integers between 1 and W. Give an efficient algorithm that solves the problem. Analyze the asymptotic running time of your algorithm in terms of n and W. Remember to argue that your algorithm is correct.

Solution:
Do a binary search for w using algorithm from 5.1 as a subroutine. In a round only add edges to the network that has value at least w'. If two tours are found increase w' we know that $w \geq w'$ so we can increase w'. If $w < w'$ we decrease w'.
Running time is $O(n^2 \log W)$.
Correctness:....

Solution 5.3 continued: