Comp 251: Mid-term examination #3

Instructor: Jérôme Waldispühl

November 24th, 2020.

- Answer the questions on Crowdmark.
- You have 3 hours to complete this exam and 30 extra minutes to account for any technical issue. You must initiate your submission within 3 hours from the start. If you meet any difficulty while uploading your solution 15 minutes before the end of the submission timeframe (3h + 30min), email us at `cs251@cs.mcgill.ca` with you solution. We will NOT proceed to any manual upload or address any request beyond this time-point.
- You can update your submission on Crowdmark during the timeframe of the exam.
- The clarity and presentation of your answers is part of the grading. Answers poorly presented may not be graded. This includes the clarity of the writing or the quality of the image you uploaded. It is your responsibility to ensure that the image has the good resolution and contrast.
- Keep the size of any file you upload on Crowdmark as small as possible to avoid technical issues.
- Unless specified, all answers must be explained.
- Partial answers will receive credits.
- The conciseness of your answer is part of the grading. An answer that is unnecessarily long or poorly structured will be penalized.
- This is an open book examination.
- It is strictly forbidden to use any external help, including online tutoring systems, or to provide aid to someone else. You are not allowed to communicate to anyone during the exam.
- It is strictly forbidden to share or disseminate this exam or any information related to this exam.
- This exam contains a mandatory academic integrity statement that you should agree with and sign. *We will not grade the exam otherwise*.
- This exam contains 11 pages.

Question:	1	2	3	4	5	6	Total
Points:	8	16	20	24	12	20	100
Score:							

Statement of Academic Integrity

In submitting this exam, I confirm that my conduct during this exam adheres to the Code of conduct and academic integrity (`https://www.mcgill.ca/students/srr/academicrights`). I confirm that I did NOT act in such a way that would constitute cheating, misrepresentation, or unfairness, including but not limited to, using unauthorized aids and assistance, personating another person, and committing plagiarism. I will not share or disseminate this exam on any platform or through personal communication.

Write your name and date to sign this statement. *We will not grade your exam if you do not agree with and sign this statement.*

Short answers

1. True or False? Circle your answers. No justification. **Wrong answers will receive a penalty of -1**.

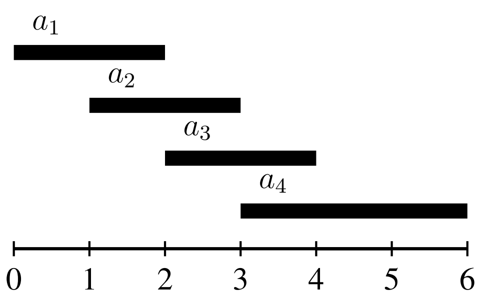
   (a) (2 points) When a graph G has no negative weight cycles, the Bellman-Ford algorithm and Dijkstra's algorithm always produce the same output (i.e. same shortest path estimates $d[v]$ and predecessors $\pi[v]$ for all vertices v of G).
   A. True **B. False**

   (b) (2 points) The total amortized cost of a sequence of n operations (i.e. the sum over all operations, of the amortized cost per operation) gives a lower bound on the actual cost of the sequence of operations.
   A. True **B. False**

   (c) (2 points) All dynamic programming algorithms satisfy an optimal substructure property.
   A. True B. False

   (d) (2 points) We apply the dynamic programming algorithm we have seen in class to solve the weighted activity scheduling problem. An instance of this problem is show below (Figure A). The weight of an activity a_i is noted V_i and is equal to the length (or duration) of the activity. The predecessor of an activity a_i is noted $pred(a_i)$. We filled the dynamic programming table M below (Figure B). Has this dynamic programming table (Figure B) been correctly filled?

   A. True B. False



(A) Instance of the weighted activity scheduling problem.

activity (a_i)	a_1	a_2	a_3	a_4
pred	-	-	a_1	a_2
$M[a_i]$	2	2	4	5
$V_i + M[pred(a_i)]$	2	2	4	5
$M[a_{i-1}]$	0	2	2	4

(B) Dynamic programming table M

Master theorem

2. Recall the master theorem.

Theorem 1 (Master theorem) *Let $a \geq 1$ and $b \geq 1$ be two constants, and $f(n)$ a function. $\forall n \in \mathbb{N}^+$ we define $T(n)$ as:*

$$T(n) = aT(\frac{n}{b}) + f(n), \text{where } \tfrac{n}{b} \text{ is interpreted as } \lfloor \tfrac{n}{b} \rfloor \text{ or } \lceil \tfrac{n}{b} \rceil.$$

Then, we can find asymptotical bounds for $T(n)$ such that:

1. *If $f(n) = O(n^{\log_b a - \epsilon})$ with $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.*
2. *If $f(n) = \Theta(n^{\log_b a} \cdot \log^p n)$, then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$.*
3. *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and $a \cdot f\left(\frac{n}{b}\right) \leq cf(n), \forall n > n_0$ with $c < 1$ and $n_0 > 0$. Then $T(n) = \Theta(f(n))$.*

When possible, apply the master theorem to find the asymptotic behaviour of $T(n)$ below. Indicate which case has been applied (no justification needed), or alternatively explain why you cannot apply it.

Grading scheme: Correct Θ notation: 2 pt. Correct case: 2pt.

(a) (4 points) $T(n) = \sqrt{2} \cdot T(\frac{n}{2}) + \log n$

Solution: $\Theta(\sqrt{n})$ (Case 1)

(b) (4 points) $T(n) = 4 \cdot T(\frac{n}{2}) + n^2$

Solution: $\Theta(n^2 \log n)$ (Case 2)

(c) (4 points) $T(n) = 6 \cdot T(\frac{n}{3}) + n^2 \cdot \log(n)$

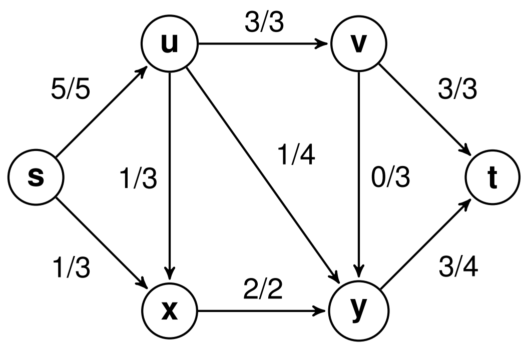
Solution: $\Theta(n^2 \log n)$ (Case 3)

(d) (4 points) $T(n) = T(\frac{n}{2}) + n \cdot (2 - \cos n)$

Solution: Does not apply. Regularity condition of case 3 is violated.

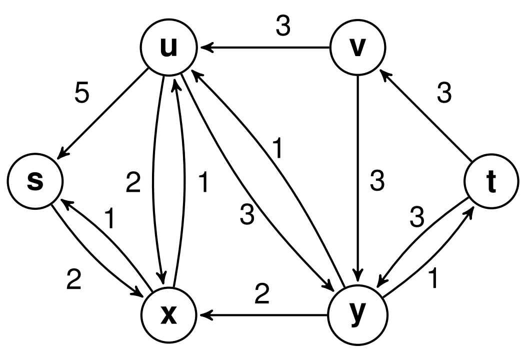
Flow networks

3. We consider the flow network G below. Each edge is annotated with its flow followed by the capacity of the edge.



(a) (5 points) Compute the residual graph.

Solution:



Grading scheme:
All/Most forward edges: 2/1 pt.
All/Most backward edges: 2/1 pt.
Same vertices: 1 pt.

(b) (5 points) Find an augmenting path in the residual graph. Write its sequence of vertices below and indicate the bottleneck β (i.e. the maximum value of the flow that can be augmented on that path).

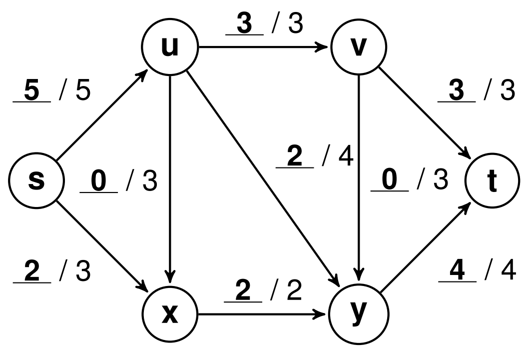
Solution:

$$\langle s, x, u, y, t \rangle$$

$$\beta = 1$$

Grading scheme: Good path: 3pt. Bottleneck: 2 pt.

(c) (5 points) Add the flow of the augmenting path to G, and show the values of the flow .



Grading scheme: -1 pt per mistake

(d) (5 points) The flow is now maximal. Compute the minimum cut and give its capacity.

Solution: The minimal cut is $\{\{s, x\}, \{u, v, y, t\}\}$ and its capacity is 7.

Grading scheme: Good cut: 3pt. Capacity: 2 pt.

Stable matching

4. Recall the Gale-Shapley algorithm for solving the stable matching problem. Let $G = (V, E)$ be a bipartite graph whose vertices are divided between two disjoint sets A and B. Each vertex $\alpha \in A$ has a full list of preferences of vertices $\beta \in B$ and, each vertex $\beta \in B$ has a full list of preferences of vertices $\alpha \in A$. We call $S \subset E$ a *perfect* matching if $\forall\ \alpha \in A,\ \exists!\ \beta \in B$ such that $(\alpha, \beta) \in S$, and $\forall\ \beta \in B,\ \exists!\ \alpha \in A$ such that $(\alpha, \beta) \in S$. We say a matching S is *stable* if $\nexists (\alpha, \beta)\ and\ (\alpha', \beta') \in S$ such that α and β' would prefer to be matched together rather than with their current assignment in S.

Algorithm 1 Gale-Shapley

```
S ← ∅
while ∃ α ∈ A not yet matched do
  β ← first β ∈ B to which α has not yet proposed.
  if β is not matched then
    S ← S ∪ (α, β)
  else if β prefers α to its current match α′ then
    S ← S \ (α′, β) ∪ (α, β)
  end if
end while
return S
```

We say that $\alpha \in A$ is a *valid* match of $\beta \in B$ if it exists a stable matching S in which they are matched. We showed in class that the Gale-Shapley algorithm returns a stable matching that is optimal (or A-optimal) from the point-of-view of A (i.e. all $\alpha \in A$ receive their best valid assignment).

Here, we want to show that the matching computed with this algorithm produces the *worst* valid assignment for all the $\beta \in B$. You will demonstrate this claim with a contradiction. Let $\beta \in B$ be the first element of B that is not receiving its worst valid assignment in a matching S^* computed by the Gale-Shapley algorithm. We call α the partner of β in S^* (i.e. $(\alpha, \beta) \in S^*$).

Since α is *not* the worst valid assignment for β, it exists another stable matching S in which β is matched with its worst valid assignment, say α'.

Note: Unnecessary long answers may not be graded. Respect the limit of words. At the discretion of the grader.

(a) (6 points) Justify why α must have a different partner β' than β in S. In other words, show that $(\alpha, \beta') \in S$ and $(\alpha', \beta) \in S$ with $\alpha \neq \alpha'$ and $\beta \neq \beta'$(max 50 words).

Solution: In S, β is paired with α'. Since S is a perfect matching, β cannot be paired with another vertex of A. But for the same reason, α has a partner in S, which must be different than β.

Grading scheme: Statement S is perfect matching: 1 pt; Statement about the partner of α: 2 pt; Statement about the partner of β: 2 pt; Conciseness: 1 pt.

(b) (5 points) Argue that β prefers α to α' (max 50 words).

Solution: By construction. α' is the worst valid assignment, thus β prefers α.

Grading scheme: Statement about worst valid assignment: 2 pt; Statement about the preference of β: 2 pt; Conciseness: 1 pt.

(c) (5 points) Show that α prefers β to β' (max 50 words).

Solution: By A-optimality, β is the best valid partner for α in S^*.

Grading scheme: Statement about A-optimality: 2 pt; Statement about the preference of α: 2 pt; Conciseness: 1 pt.

(d) (8 points) Conclude (max 50 words).

Solution: (α, β) is an unstable pair in S. Thus, S cannot be a stable matching.

Grading scheme: Statement about unstable pair: 3 pt; Contradiction: 3 pt; Conciseness: 2 pt.

Amortized analysis

5. Consider the implementation of a queue with two stacks S_1 and S_2. The method *enqueue* pushes a new element in S_1. The method *dequeue* pops an element from S_2 if the latter is not empty. Otherwise (if S_2 is empty), it pops all elements from S_1, pushes them into S_2, and then returns the last element inserted in S_2. We want to determine the amortized complexity the operations *enqueue* and *dequeue*.

   (a) (4 points) Let n be the number of elements in the queue. What is the individual worst-case running time complexity of the functions *enqueue* and *dequeue*? *No justification needed.*

   Solution: enqueue: $O(1)$, dequeue: $O(n)$.

   Grading scheme: Enqueue: 2 pt; Dequeue: 2 pt.

   (b) (8 points) Using the *accounting method*, calculate the amortized cost per operation of any sequence of m *enqueue* and *dequeue* operations. In particular, demonstrate that the credit can never be negative.
   Note: Make a concise answer (4-6 sentences). Unnecessary long answers may not be graded.

   Solution:

   Allocate $2 to each element stored in the queue structure. This will cover the cost of popping it and pushing it from stack1 to stack2 if that ever needs to be done and ensure that the credit will never be negative. There is an additional cost of $1 for the initial push onto S_1, for a cost of $3. The dequeue operations cost $1 to pop from S_2. The maximum amortized cost of a sequence of m operations is thus $3m$, which implies an amortized cost per operation of $O(1)$.

   Grading scheme:
   Cost of Enqueue: 2 pt; Cost of Dequeue: 2 pt.
   Credit never goes negative: 3 pt; Conclusion: 1 pt.

Divide-and-Conquer

6. Consider 2D arrays of integers b such that each row and column is sorted by increasing order. We show below an example of such array with 4 rows and 4 columns.

2	14	25	30
3	15	28	30
7	15	32	43
20	28	36	58

The objective of this problem is to design an *efficient* algorithm to find an element with value v in such array. *Here, we assume that the array b has the same number n of rows and columns. Moreover, this number n of rows and columns is a power of 2 (i.e. $n = 2^k$ with $n \geq 0$).*

(a) (5 points) Let $x = b(\frac{n}{2}, \frac{n}{2})$ (i.e. the value stored in b at index $(\frac{n}{2}, \frac{n}{2})$). We want to show that if $v > x$ we can safely eliminate a region (to be determined by yourself) of the array b from the search. Indicate which region of the array (i.e. which indices) remains to be explored.
Note: Briefly justify your answer in (2-3 sentences). You can also illustrate your answer with a drawing of the array if it helps.

Solution:
We can eliminate the sub-array $b[1, \cdots, \frac{n}{2}; 1, \cdots, \frac{n}{2}]$ from the search.
x is the largest integer of this sub-array. Thus, if $v > x$ then $v > b(i,j)\ \forall\ i \leq \frac{n}{2}$ and $\forall\ j \leq \frac{n}{2}$. We still need to explore the $b(i,j)$ such that $i > \frac{n}{2}$ or $j > \frac{n}{2}$.
Grading scheme: Isolate the bad array: 2 pt; Statement about x: 2 pt; Resursive search: 1 pt.

(b) (5 points) Let $y = b(\frac{n}{2}+1, \frac{n}{2}+1)$. We want to show that if $v < y$ we can safely eliminate a region (to be determined) of the array b from the search. Indicate which region of the array (i.e. which indices) remains to be explored.
Note: Briefly justify your answer (2-3 sentences).

Solution: This is the symmetric case. We can eliminate the sub-array $b[\frac{n}{2}+1, \cdots, n; \frac{n}{2}+1, \cdots, n]$ from the search.
y is the smallest integer of this sub-array. Thus, if $v < y$ then $v < b(i,j)\ \forall\ i \geq \frac{n}{2}+1$ and $\forall\ j \geq \frac{n}{2}+1$. We still need to explore the $b(i,j)$ such that $i < \frac{n}{2}+1$ or $j < \frac{n}{2}+1$.
Grading scheme: Isolate the bad array: 2 pt; Statement about x: 2 pt; Resursive search: 1 pt.

In the following, we denote a sub-matrix $b[i,i'][j,j']$ of b including rows i to i' and columns j to j' as follow. We also note the coordinates of x and y as (x_1, x_2) and (y_1, y_2).

$$b[i,i'][j,j'] = \begin{bmatrix} b_{i,j} & \dots & b_{i,x_2} & b_{i,y_2} & \dots & b_{i,j'} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{x_1,j} & \dots & b_{x_1,x_2} & b_{x_1,y_2} & \dots & b_{x_1,j'} \\ b_{y_1,j} & \dots & b_{y_1,x_2} & b_{y_1,y_2} & \dots & b_{y_1,j'} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{i',j} & \dots & b_{i',x_2} & b_{i',y_2} & \dots & b_{i',j'} \end{bmatrix}$$

(c) (10 points) Deduce from your previous observations a *divide-and-conquer* method to search a value v in a sorted 2D array of integers. Using the notations above, write a pseudo-code describing your algorithm. Your algorithm must use as an input the array b, the value v to be searched for, and the indices (i, i') and (j, j') of the region to explore. Here, (i, i') are the indices of the first and last rows and (j, j') the indices of the first and last columns to explore. It will return the index of the row and column if the value is found, and the pair $(-1, -1)$ otherwise.

Solution:

Algorithm 2 2Dsearch(v, b, i, i', j, j')

$x_1 \leftarrow \frac{i'-i+1}{2}$
$x_2 \leftarrow \frac{j'-j+1}{2}$
$y_1 \leftarrow x_1 + 1$
$y_2 \leftarrow x_2 + 1$
$x \leftarrow b(x_1, x_2)$
$y \leftarrow b(y_1, y_2)$
if $v = x$ **then**
  return (x_1, x_2)
else if $v > x$ **then**
  2Dsearch(v, b, i, x_1, y_2, j')
  2Dsearch(v, b, y_1, i', j, x_2)
  2Dsearch(v, b, y_1, i', y_2, j')
end if
if $v = y$ **then**
  return (y_1, y_2)
else if $v < y$ **then**
  2Dsearch(v, b, i, x_1, y_2, j')
  2Dsearch(v, b, y_1, i', j, x_2)
  2Dsearch(v, b, i, x_1, j, x_2)
end if
return $(-1, -1)$

Initialization: 2 pt; Recursive cases: 4 pt; Return statements: 2 pt; Clarity/conciseness: 2 pt.