Comp 251: Final examination

Instructor: Jérôme Waldispühl

April 21th, 2017

True or False

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

   (a) (2 points) Using the array representation seen in class, the following array `[20 15 18 7 9 5 12 3 6 2]` forms a max-heap.
   A. True B. False

   (b) (2 points) The height of binary search tree with n nodes is $\Omega(\log n)$.
   A. True B. False

   (c) (2 points) Given a weighted graph $G = (V, E, w)$ and a set $S \subseteq V$, let (u, v) be an edge such that (u, v) is the minimum weight edge between any vertex of S and any vertex of $V - S$. In this case, the minimum spanning tree of G must include the edge (u, v) (you can assume that the weights of all edges are distinct).
   A. True B. False

   (d) (2 points) Let T be a minimum spanning tree of a graph G. Then, for any pair of vertices s and t, the shortest path from s to t in G is the path from s to t in T.
   A. True **B. False**

   (e) (2 points) When a graph has no negative weight cycles, the Bellman-Ford algorithm and Dijkstra's algorithm always produce the same output.
   A. True **B. False**

   (f) (2 points) Given a weighted directed graph $G = (V, E, w)$ and a shortest path p from s to t, if we doubled the weight of every edge to produce $G' = (V, E, w')$, then p is also a shortest path in G'.
   A. True B. False

   (g) (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**

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

   (i) (2 points) Suppose that a hash table with collisions resolved by chaining contains n items and has a load factor of $\alpha = 1/\log n$. Assuming simple uniform hashing, the expected time to search for an item in the table is $\mathcal{O}(1/\log n)$.
   A. True **B. False**

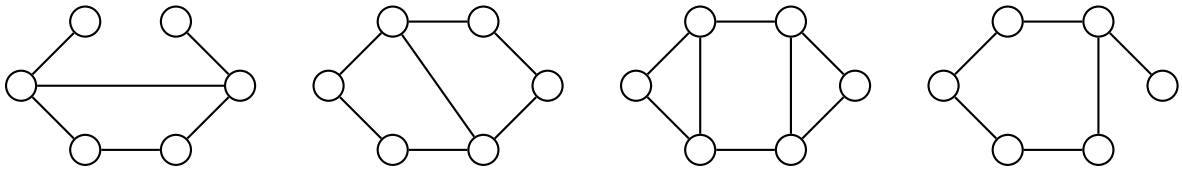
   (j) (2 points) An adversary can provide randomized quicksort with an input array of length n that forces the algorithm to run in $\Omega(n \log n)$ time on that input.
   A. True B. False

Multiple choice questions

2. Multiple choice questions (Circle your answers. No justification needed. You may circle zero, one or multiple answers.)

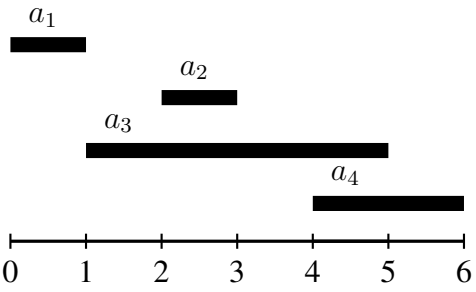
   (a) (4 points) We perform a rotation on a binary search tree. Which assertions are true?
   A. The resulting tree is a binary search tree.
   B. The height of the tree is preserved.
   C. The resulting tree is balanced.
   D. The running time of a rotation is $\mathcal{O}(1)$.

   (b) (4 points) Which of the following graphs are bipartite?



(A) G_1 * (B) G_2 * (C) G_3 (D) G_4

   (c) (4 points) We are applying the dynamic programming algorithm we have seen in class to solve the weighted activity scheduling problem. An instance of this problem is show below. 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 started to fill the dynamic programming table M for the first, second and third activity (i.e. column a_1, a_2, and a_3 in the table below).



(A) Instance of the weighted activity scheduling problem.

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

(B) Dynamic programming table M

Which values should appear in the column of the last activity (i.e. a_4)?

A. $\begin{bmatrix}4\\5\\4\end{bmatrix}$ B. $\begin{bmatrix}4\\4\\2\end{bmatrix}$ C. $\begin{bmatrix}5\\5\\4\end{bmatrix}$ **D.** $\begin{bmatrix}5\\4\\5\end{bmatrix}$

(d) (4 points) We use a tree representation to model disjoint sets. We implement unions as *union-by-size* with path compression. What is the worst case running time for the operations *find* and *union*?
A. $\mathcal{O}(1)$ for both.
B. $\mathcal{O}(\log n)$ **for both.**
C. $\mathcal{O}(n)$ for *find*, and $\mathcal{O}(1)$ for *union*.
D. $\mathcal{O}(1)$ for *find*, and $\mathcal{O}(n)$ for *union*.

(e) (4 points) What is the ratio of random boolean variable assignments that satisfies a clause of 3-SAT (i.e. the clause is True)?
A. $1/2$ B. $2/3$ **C.** $7/8$

Short answers

3. (4 points) Let S be a set of n integers. One can create a data structure for S so that determining whether an integer x belongs to S can be performed in O(1) time in the worst case. How?

> **Solution:** Use a hash table (Note: Solution such as a simple table can be accepted albeit not optimal)

4. For each algorithm listed below,

   - give a recurrence that describes its running time complexity, and
   - give its running time complexity using Θ**-notation**.

   No justification needed.

   (a) (8 points) Binary search (i.e. The classic dichotomic search algorithm for finding a value in a sorted array)

   > **Solution:** $T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$

   (b) (8 points) Merge sort

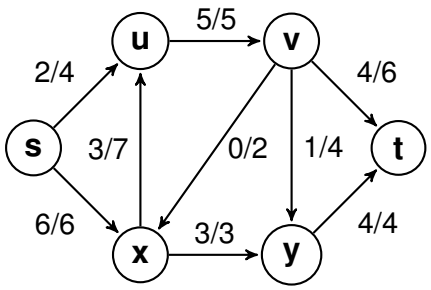
   > **Solution:** $T(n) = 2 \cdot T(n/2) + \Theta(n) = \Theta(n \cdot \log n)$

   (c) (8 points) Karatsuba

   > **Solution:** $T(n) = 3 \cdot T(n/2) + \Theta(n) = \Theta(n^{\log 3})$

Flow network

5. Let G be a flow network shown in the figure below. The source is the vertex s and the sink the vertex t. The notation a/b describes a units of flow in an edge of capacity b. Each edge is annotated with the value of the flow traversing it and its capacity.

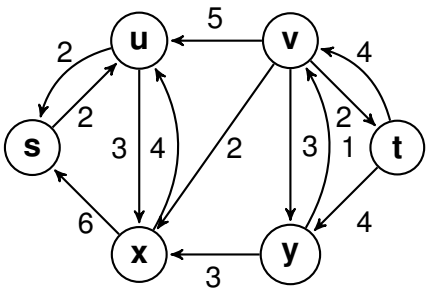


(a) (5 points) Is the flow maximal? Justify your answer with a constructive argument.

Solution: Yes. Consider the capacity of the cut {s,u,x} and {v, t,y} that is 5. The flow is already equal to 5. The flow in a graph cannot exceed the capacity of any cut, thus it is maximal.

(b) (5 points) Draw the residual graph G_f of G.

Solution:



(c) (5 points) Use the residual graph G_f to find a minimal cut in G. Give the minimal cut (i.e. the partition of vertices) and explain how you calculate it.

Solution: {s,u, x} and {v, t, y} is a minimal cut. We run DFS (or BFS) to determine all vertices reachable from the source s. The latter determine the cut.

Master method

6. We remind you the master method.

Theorem 1 (Master method) *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:*

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

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 $a \cdot f\big(\frac{n}{b}\big) \leq cf(n), \forall n > n_0$ with $c < 1$ and $n_0 > 0$. Then $T(n) = \Theta(f(n))$.*

When possible, apply the master method to find the asymptotic behaviour of $T(n)$ below. Indicate which case has been applied, or alternatively why you cannot use the method.

(a) (8 points) $T(n) = 4 \cdot T(\frac{n}{2}) + \log n$

Solution: $\Theta(n^2)$ (Case 1)

(b) (8 points) $T(n) = 2^n \cdot T(\frac{n}{2}) + n^n$

Solution: Does not apply (a is not constant)

(c) (8 points) $T(n) = 7 \cdot T(\frac{n}{3}) + n^2$

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

(d) (8 points) $T(n) = 3 \cdot T(\frac{n}{3}) + \frac{n}{2}$

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

Stable matching

7. (10 points) We have a group of 3 applicants (i.e. candidates) and 3 companies with preferences. Apply the Gale-Shapley algorithm to find a stable matching. Show your work.

Candidates	Preferences			Companies	Preferences		
	1^{st}	2^{nd}	3^{rd}		1^{st}	2^{nd}	3^{rd}
Xavier	Baidu	Alphabet	Campbell	Alphabet	Xavier	Zhang	Yulia
Yulia	Baidu	Campbell	Alphabet	Baidu	Yulia	Xavier	Zhang
Zhang	Alphabet	Campbell	Baidu	Campbell	Zhang	Yulia	Xavier

Solution: Xavier $\rightarrow$ Alphabet
Yulia $\rightarrow$ Baidu
Zhang $\rightarrow$ Campbell

Amortized analysis

8. We consider the implementation of a queue with two stacks S_1 and S_2. At any time, at least one of the two stacks is empty. The method *enqueue* pushes a new element in the non-empty stack (say S_1). While the method *dequeue* pops all elements from the non-empty stack (Say S_1) and pushes them into the empty one (S_2), and then returns the last element inserted in S_2. We want to determine the amortized complexity of the operations *enqueue* and *dequeue*.

   (a) (5 points) Let n be the number of elements in the queue. What is the worst running time complexity of *enqueue* and *dequeue* (No justification needed).

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

   (b) (15 points) Calculate the amortized cost per operation of a sequence of m operations *enqueue* and *dequeue* (You can choose to use the aggregate or accounting method).

   Solution:
   Aggregate method: As each element is processed, it is clearly pushed at most twice and popped at most twice. However, if an element is enqueued and never dequeued then it is pushed at most twice and popped at most once. Thus the cost of each enqueue is 3 and of each dequeue is 1.

   Accounting method: 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. 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.

Minimum Spanning Tree

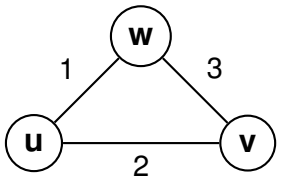
9. We give pseudocode for different algorithms to compute minimum spanning trees. Each one takes a graph as input and returns a set of edges T. For each algorithm, you must either prove that T is a minimum spanning tree or show a counter-example.

   (a) (10 points)

Algorithm 1 MAYBE-MST-B(G,w)

```
T ← ∅
for all edge e, taken in arbitrary order do
  if T ∪ {e} has no cycle then
    T ← T ∪ {e}
  end if
end for
return T
```

Solution: It will not give a minimum spanning tree, that can be proven by a counter-example.



The MST would have edges (w;u) and (v;w) with weight 3.
MAYBE-MST-B could add edges (u;v) and (v;w) to T, then try to add (w;u) which forms a cycle, then return T (weight 5), since the algorithm takes edges in arbitrary order.

   (b) (10 points)

Algorithm 2 MAYBE-MST-A(G,w)

```
Sort the edges into non-increasing (decreasing) order of edge weights w
T ← E
for all edge e, taken in non-increasing order by weight do
  if T − {e} is a connected graph then
    T ← T − {e}
  end if
end for
return T
```

Solution: Observe that MAYBE-MST-A removes edges in non-increasing order as long as the graph remains connected. Then, the resulting T is a minimum spanning tree. The correctness of the algorithm can be shown as follows:
let S be a MST of G. When an edge e is about to be removed, either $e \in S$ or $e \notin S$. If $e \notin S$, then we can just remove it.
If $e \in S$, removing e will disconnect S into two trees (Note: this does not disconnect the graph). It is clear that no other edge can connect these trees with smaller weight than e, because by assumption S is a MST, and if a larger edge existed that connected the trees the algorithm would have removed it before removing e. Since the graph is still connected, this means a path exists. So, there must be another edge with equal weight that has not been discovered yet, which means we can remove e.

Longest common substring

10. We define the *longest common substring* problem as: Given two strings, α of length m and β of length n, find the longest string which is a substring of both α and β (Note: a substring is a sequence of consecutive letters). For instance, 011 is the longest common substring of $\alpha = 00011$ and $\beta = 0111$. This common substring can be visualized with a simple alignment:

$$\begin{array}{ll} \alpha = & \texttt{00011-} \\ \beta = & \texttt{--0111} \end{array}$$

As illustrated above, we want to find an alignment with the longest series of consecutive matches.

(a) We aim to design a dynamic programming algorithm to solve the longest common substring problem. We propose below a recurrence to solve this problem. Let α and β be the input substrings of length m and n. First, we define a recursive equation to calculate the longest common *suffix* for all pairs of *prefixes* of the strings. Let LCS be the array such that $LCS(i, j)$ stores the length of longest common suffix of $\alpha[1, i] = \alpha_1 \cdots \alpha_i$ (i.e. prefix of α) and $\beta[1, j] = \beta_1 \cdots \beta_j$ (i.e. prefix of β). Our recurrence is:

$$LCS(i, j) = \begin{cases} LCS(i-1, j-1) + 1 & \text{if } \alpha[i] = \beta[j] \\ 0 & \text{Otherwise} \end{cases}$$

i. (5 points) What is the base case? (i.e. how do you initialize the array LCS?)

> **Solution:** LCS[i,j] = 0 if $i = 0$ or $j = 0$

ii. (5 points) Assume that the array LCS is now fully calculated. Where will you potentially find (i.e. in which cell(s)) the value of the length of the longest substring value in this table? (Note: This will also be the starting point of the backtracking procedure used to retrieve the solution)

> **Solution:** It can be anywhere in the dynamic array.

(b) (5 points) What is the optimal substructure of the longest common substring problem? You do not need to prove your answer.

> **Solution:** It is found in the recursive equation given above. Any prefix ending at index $(i-k, j-k)$ ($k \leq LCS(i, j)$) of the longest common substring ending at index (i, j) is also a longest common substring ending at index $(i-k, j-k)$.
> More generally, you could also say that the longest common suffix of a prefix of your prefix (I know it starts to be convoluted...) should be itself optimal.

(c) (10 points) Based on the recurrence described above, write the pseudo-code of a dynamic programming algorithm that fills/calculates the array LCS, and returns the length of the longest common substring (Note: We do not ask you to write the backtracking procedure).

Solution:

```
best=0;
for (i=1; i<=m; i++)
  LCS[i,0] = 0
for (j=1; j<=n; i++)
  LCS[0,i] = 0
for (i=1; i<=m; i++)
  for (j=1; j<=n; j++)
    if a[i] == b[j]
      LCS[i,j] = LCS[i-1,j-1] + 1;
      if LCS[i,j] > best:
       best = LCS[i,j]
    else
      LCS[i,j] = 0
return best
```

(d) (5 points) Execute your algorithm with $\alpha = 1100$ and $\beta = 101$. Show the complete dynamic programming table filled by your algorithm. Then, show the cell that contains the value of the longest common substring, as well as the path in the dynamic table corresponding to the longest common substring.

Solution:

	-	1	1	0	0
-	0	0	0	0	0
1	0	1	1	0	0
0	0	0	0	**2**	1
1	0	1	1	0	0

(e) (5 points) What is the running time complexity (worst and best case) of this algorithm? No justification needed.

Solution: $\Theta(m \cdot n)$

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	20	20	4	24	15	32	10	20	20	35	200
Score:											