

COMP251: Final Review

Jérôme Waldispühl

School of Computer Science

McGill University

Overview of the exam

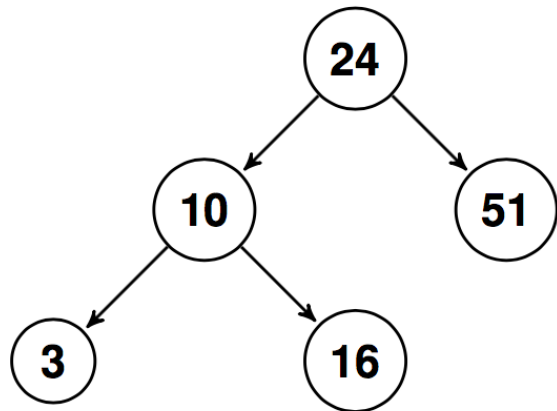
- 11 questions.
- 200 points + 30 bonus
- 20 point True or False (Warning: -1 penalty for wrong answers!)
- 20 point for *multiple* choices answers.
- 28 points for short answers (no justification)
- 97 points questions/applications
- 35 points + 30 bonus problems
- Unless specified, all answers must be justified.
- Partial answers will receive credits.
- The clarity and presentation of your answers is an integral part of the grading. **Be neat!**
- Books and electronic devices are not allowed.
- 2 crib sheets (4 pages).

Advices

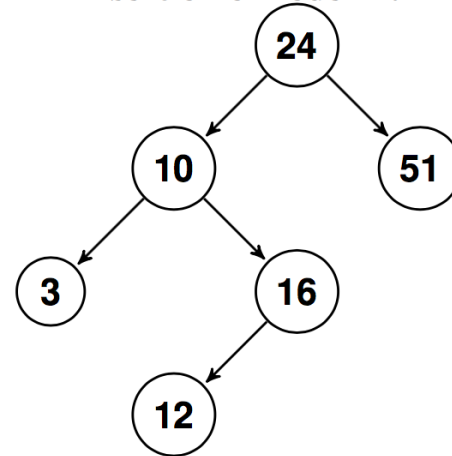
- When practicing, do not look at the solution immediately.
- Look at resources from other similar classes (e.g. Mike Langer COMP 251 web page, OCW at MIT, etc.)
- At the exam, do the questions you know the answer to. Do not spend time on the difficult ones. You will do them after.
- Sleep well!

MIDTERM SOLUTIONS

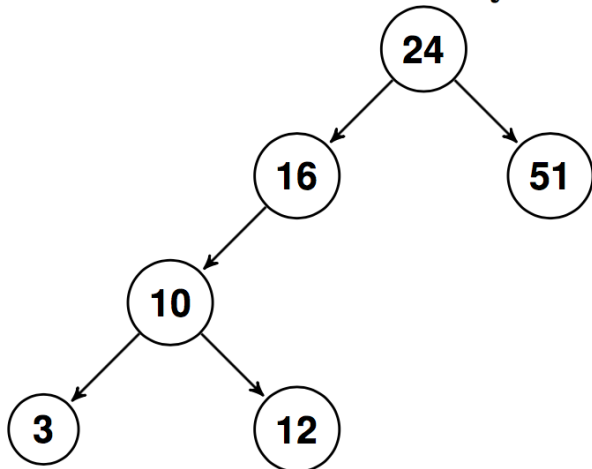
3. (10 points) Insert the key 12 into the AVL tree shown below. If necessary, balance the tree with rotations to restore the AVL properties of the new tree. **Show your work.** Show the tree before/after each operation and write down which operation you are doing (i.e. insertion, rotation left or right) with all its parameters.



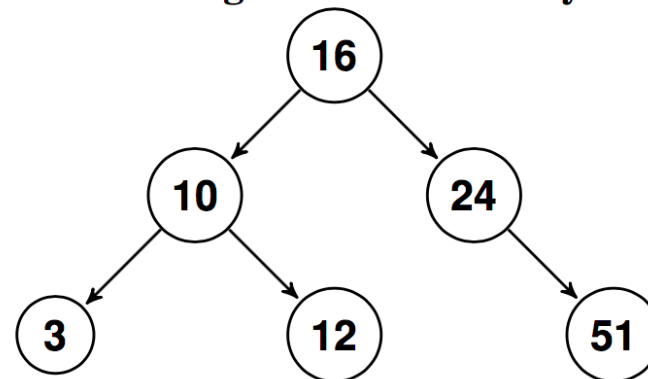
Insertion of node 12:



Rotation left at node with key 10:



Rotation right at node with key 24:



4. We consider an hash function $h(k) = k \bmod 7$. Draw the content of the table after the successive insertion of the keys: 20, 8, 7, 3, 27, 15, and 19.

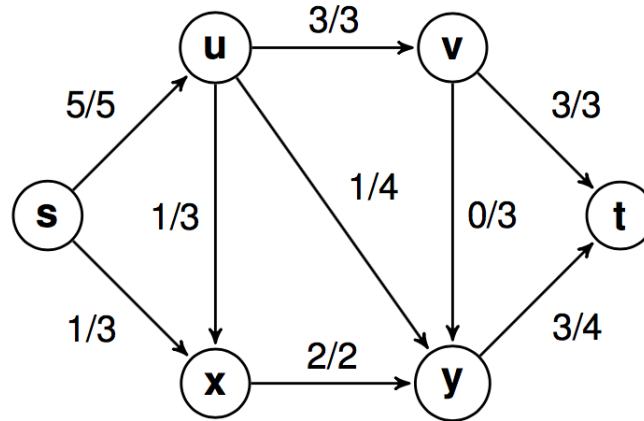
(a) (5 points) Draw the content of the hash table after insertion of the keys when we use direct addressing (resolve collision by chaining).

0	7
1	15,8
2	
3	3
4	
5	19
6	27,20

(b) (5 points) Draw the content of the hash table after insertion of the keys when we use open addressing and linear probing to resolve collisions.

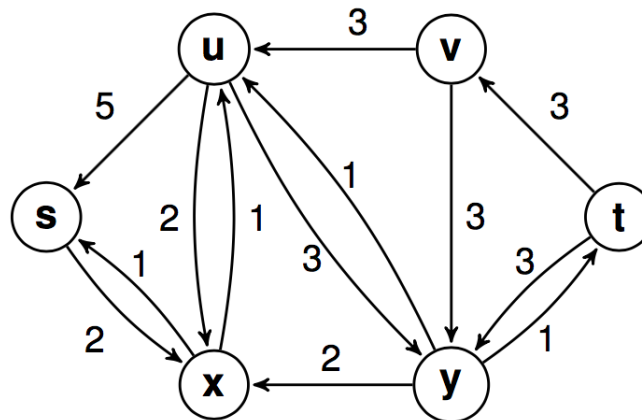
0	7
1	8
2	27
3	3
4	15
5	19
6	20

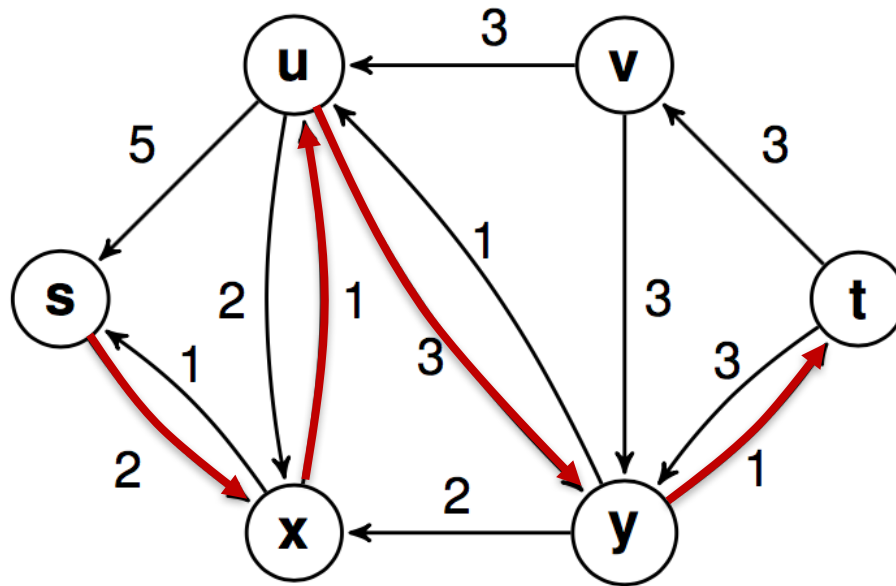
5. We consider the flow network G below. Each edge is annotated with its flow followed by the capacity of the edge.



- (a) (5 points) Determine its residual graph.

Solution:





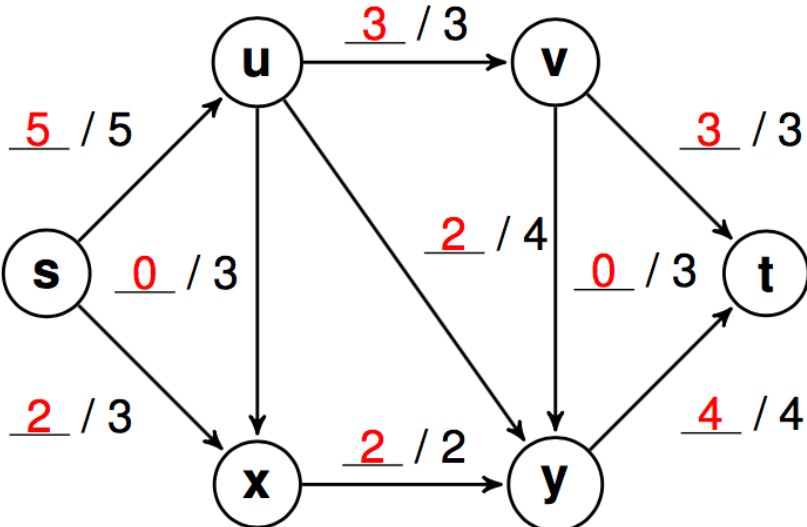
(b) (5 points) Find an augmenting path in the residual graph. Write its sequence of vertices below and indicate the bottleneck β (i.e. the maximum value of the flow that can be augmented on that path).

Solution:

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

$$\beta = 1$$

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



(d) (5 points) Can this resulting flow network be augmented? Justify your answer in one sentence.

Solution: No. Consider the cut $\{\{s, u, x, v, y\}, \{t\}\}$, its capacity is 7. After augmenting the flow, the total flow in the sink t is equal to 7. But the maximum flow is lower or equal to the capacity of the minimal cut. Thus, the flow is maximal.

Suppose that we are given a weighted, directed acyclic graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are strictly positives.

- (a) (4 points) We will show that that $d[v] = \delta(s, v)$ for every $v \in V$ when v is added to S . First, lets start to argue that this is the case at the beginning of the algorithm.

Solution: At the beginning $u = s$ and $d[s] = 0$, thus the proposition is satisfied.

- (b) (4 points) Then, we address the general case where $v \in V - \{s\}$. Let u be the vertex preceding v on the shortest path $s \rightsquigarrow v$. By the convergence property, we need to verify that $u \in S$ to guarantee that $d[v] = \delta(s, v)$ after relaxation of $u \rightarrow v$. Write the shortest path estimate $d[v]$ as a function of $d[u]$ and $w(u, v)$.

Solution: $d[v] = d[u] + w(u, v)$.

- (c) (4 points) Assume that $u \notin S$. What relationship does it imply between $d[u]$ and $d[v]$?

Solution: Then u and v are in the priority queue Q . Since v is extracted before u , we have $d[v] \leq d[u]$.

(d) (8 points) Show now that u must already be in S (i.e. show a contradiction with the hypothesis $u \notin S$).

Solution: If $u \neq s$, then $w(u, v) > 0$, which implies that $d[v] > d[u]$ because $d[v] = d[u] + w(u, v)$. Contradiction with the hypothesis $u \notin S$ that requires $d[v] \leq d[u]$. If $u = s$, then we have a contradiction because s is the first vertex to be added in S and this is already in S .

MASTER THEOREM

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:

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

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 $T(n) = \Theta(f(n))$.

$$\bullet T(n) = 3 \cdot T\left(\frac{n}{2}\right) + n^2 \quad \left\{ \begin{array}{l} \bullet n^2 = \Omega(n^{\log_2 3 + (2 - \log_2 3)}) \\ \bullet 3 \cdot \left(\frac{n}{2}\right)^2 \leq \frac{3}{4} \cdot n^2, n \geq 0 \end{array} \right. \quad (\text{case 3}) \Rightarrow \Theta(n^2)$$

$$\bullet T(n) = T\left(\frac{n}{2}\right) + 2^n \quad \left\{ \begin{array}{l} \bullet 2^n = \Omega(n^{\log_2 1 + 1}) \\ \bullet 2^{\frac{n}{2}} \leq \frac{1}{2} \cdot 2^n = 2^{n-1}, n \geq 2 \end{array} \right. \quad (\text{case 3}) \Rightarrow \Theta(2^n)$$

$$\bullet T(n) = \frac{1}{2} \cdot T\left(\frac{n}{2}\right) + \frac{1}{n} \quad \text{Theorem does not apply (a < 1)}$$

$$\bullet T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2 \quad n^2 = \Theta(n^{\log_2 4} \cdot (\log n)^0) \quad (\text{case 2}) \Rightarrow \Theta(n^2 \cdot \log n)$$

DYNAMIC PROGRAMMING

Knapsack problem

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ and has value $v_i > 0$.
- Knapsack has capacity of W .
- Goal: fill knapsack so as to maximize total value.

Ex. $\{1, 2, 5\}$ has value 35.

Ex. $\{3, 4\}$ has value 40.

Ex. $\{3, 5\}$ has value 46 (but exceeds weight limit).

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

knapsack instance
(weight limit $W = 11$)

Greedy by value. Repeatedly add item with maximum v_i .

Greedy by weight. Repeatedly add item with minimum w_i .

Greedy by ratio. Repeatedly add item with maximum ratio v_i / w_i .

Observation. None of greedy algorithms is optimal.


False start...

Def. $OPT(i)$ = max profit subset of items $1, \dots, i$.

Case 1. OPT does not select item i .

- OPT selects best of $\{1, 2, \dots, i-1\}$.

optimal substructure property
(proof via exchange argument)



Case 2. OPT selects item i .

- Selecting item i does not immediately imply that we will have to reject other items.
- Without knowing what other items were selected before i , we don't even know if we have enough room for i .

Conclusion. Need more subproblems!

New variable

Def. $OPT(i, w) = \max$ profit subset of items $1, \dots, i$ with **weight limit** w .

Case 1. OPT does not select item i .

- OPT selects best of $\{1, 2, \dots, i-1\}$ using weight limit w .

Case 2. OPT selects item i .

- New weight limit = $w - w_i$.
- OPT selects best of $\{1, 2, \dots, i-1\}$ using this new weight limit.

↙ optimal substructure property
↘ (proof via exchange argument)

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{OPT(i-1, w), v_i + OPT(i-1, w - w_i)\} & \text{otherwise} \end{cases}$$

Dynamic programming algorithm

KNAPSACK ($n, W, w_1, \dots, w_n, v_1, \dots, v_n$)

FOR $w = 0$ TO W

$M[0, w] \leftarrow 0$.

FOR $i = 1$ TO n

 FOR $w = 1$ TO W

 IF ($w_i > w$) $M[i, w] \leftarrow M[i-1, w]$.

 ELSE $M[i, w] \leftarrow \max \{ M[i-1, w], v_i + M[i-1, w - w_i] \}$.

RETURN $M[n, W]$.

Example

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$$OPT(i, w) = \begin{cases} 0 & \text{if } i=0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{OPT(i-1, w), v_i + OPT(i-1, w-w_i)\} & \text{otherwise} \end{cases}$$

		weight limit w											
		0	1	2	3	4	5	6	7	8	9	10	11
subset of items $1, \dots, i$	{ }	0	0	0	0	0	0	0	0	0	0	0	0
	{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
	{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
	{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
	{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
	{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	34	40

$OPT(i, w) = \text{max profit subset of items } 1, \dots, i \text{ with weight limit } w.$

Analysis

Theorem. There exists an algorithm to solve the knapsack problem with n items and maximum weight W in $\Theta(nW)$ time and $\Theta(nW)$ space.

Pf.

weights are integers
between 1 and W

- Takes $O(1)$ time per table entry.
- There are $\Theta(nW)$ table entries.
- After computing optimal values, can trace back to find solution:
take item i in $OPT(i, w)$ iff $M[i, w] < M[i-1, w]$. ■

Remarks.

- Not polynomial in input size! ← "pseudo-polynomial"
- Decision version of knapsack problem is NP-COMPLETE. [CHAPTER 8]
- There exists a poly-time algorithm that produces a feasible solution that has value within 1% of optimum. [SECTION 11.8]

AMORTIZED ANALYSIS

Dynamic tables

Scenario

- Have a table - maybe a hash table.
- Don't know in advance how many objects will be stored in it.
- When it fills, must reallocate with a larger size, copying all objects into the new, larger table.
- When it gets sufficiently small, *might* want to reallocate with a smaller size.

Goals

1. $O(1)$ amortized time per operation.
2. Unused space always \leq constant fraction of allocated space.

Load factor $\alpha = (\# \text{ items stored}) / (\text{allocated size})$

Never allow $\alpha > 1$; Keep $\alpha >$ a constant fraction \Rightarrow Goal 2.

Table expansion

Consider only insertion.

- When the table becomes full, double its size and reinsert all existing items.
- Guarantees that $\alpha \geq \frac{1}{2}$.
- Each time we insert an item into the table, it is an *elementary insertion*.

```
TABLE-INSERT(T, x)
```

```
  if size[T] = 0
```

```
    then allocate table[T] with 1 slot
```

```
      size[T]  $\leftarrow$  1
```

```
  if num[T] = size[T] then
```

```
    allocate new-table with  $2 \cdot \textit{size}[\textit{T}]$  slots
```

```
    insert all items in table[T] into new-table
```

```
    free table[T]
```

```
    table[T]  $\leftarrow$  new-table
```

```
    size[T]  $\leftarrow$   $2 \cdot \textit{size}[\textit{T}]$ 
```

```
  insert x into table[T]
```

```
  num[T]  $\leftarrow$  num[T] + 1
```

(Initially, $\textit{num}[\textit{T}] = \textit{size}[\textit{T}] = 0$)

Aggregate analysis

- Charge 1 per elementary insertion.
- Count only elementary insertions (other costs = constant).

c_i = actual cost of i^{th} operation

- If not full, $c_i = 1$.
- If full, have $i-1$ items in the table at the start of the i^{th} operation.
Have to copy all $i-1$ existing items, then insert i^{th} item $\Rightarrow c_i = i$.

n operations $\Rightarrow c_i = O(n) \Rightarrow O(n^2)$ time for n operations

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is power of } 2 \\ 1 & \text{Otherwise} \end{cases}$$

$$\text{Total cost} = \sum_{i=1}^n c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j = n + \frac{2^{\lfloor \log n \rfloor + 1} - 1}{2 - 1} < n + 2n = 3n$$

Amortized cost per operation = 3.

Accounting method

Charge \$3 per insertion of x .

- \$1 pays for x 's insertion.
- \$1 pays for x to be moved in the future.
- \$1 pays for some other item to be moved.

Suppose we've just expanded, $size=m$ before next expansion, $size=2m$ after next expansion.

- Assume that the expansion used up all the credit, so that there's no credit stored after the expansion.
- Will expand again after another m insertions.
- Each insertion will put \$1 on one of the m items that were in the table just after expansion and will put \$1 on the item inserted.
- Have \$ $2m$ of credit by next expansion, when there are $2m$ items to move. Just enough to pay for the expansion...

The End

Do not focus on grades, what you learned is more important.

Be curious. Get involved in projects. You now have the tools and knowledge to do interesting things.

Questions? `jeromew@cs.mcgill.ca`