CSCI 136:
Data Structures
and
Advanced Programming

Lecture 29

Hash tables, part 1

Instructor: Dan Barowy

Williams

Announcements

Two-week lab.

PRE-LAB: choose your own partner.

No design doc PRE-LAB.

May 8 lab meeting is optional.

Outline

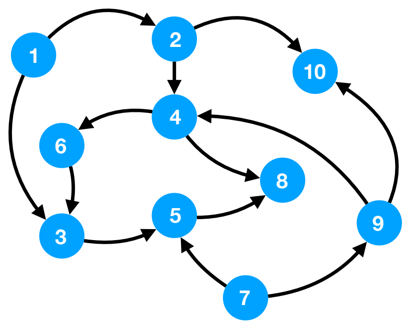
Topological ordering

Hash tables

DAGs / Topological ordering

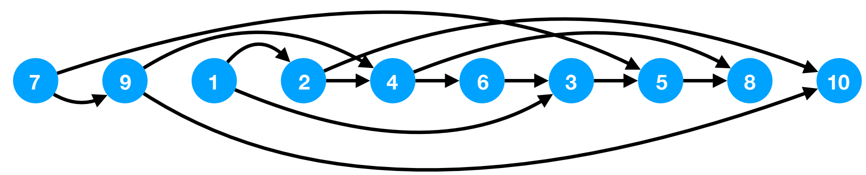
Topological ordering

A **topological ordering** of a **directed acyclic graph** is a **linear ordering of its vertices** such that for every directed edge **u,v** from vertex **u** to vertex **v**, **u** comes before **v** in the ordering.



Topological ordering

A **topological ordering** of a **directed acyclic graph** is a **linear ordering of its vertices** such that for every directed edge **u,v** from vertex **u** to vertex **v**, **u** comes before **v** in the ordering.



Good question

What makes a topological ordering "topological"?

Fun fact: graph theory used to be considered a branch of the field of topology in mathematics. Topology is the study of spaces under continuous deformations. Graphs can be thought of as "spaces" since many of their properties are invariant under continuous deformation.

Note that a topological sort produces an order with no regard to the values stored in a graph. Instead, the order is purely the result of the connectedness of the graph. The connectedness of a graph does not change if you stretch or twist it.

Topological ordering

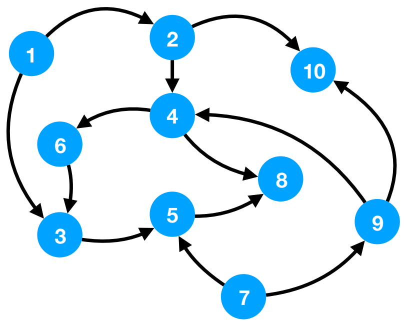
E.g., **how** does a factory decide what **parts of a car** to **assemble first**?

Produce a **topological ordering** of the vertices in the assembly dependence graph.

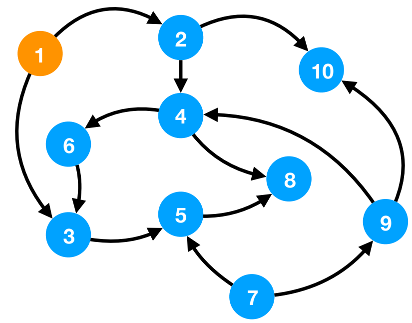
<u>Algorithm: topological sort:</u>

- For each node of the graph (in any order), recursively visit in a depth-first manner. After visiting each node, add it to the head of the list.
- When visiting, return (do not recurse) when:
  - A node has already been visited, or
  - the node has no outgoing edges.

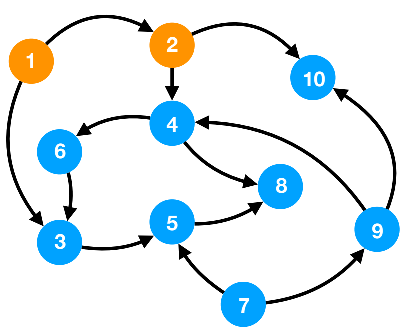
Topological sort



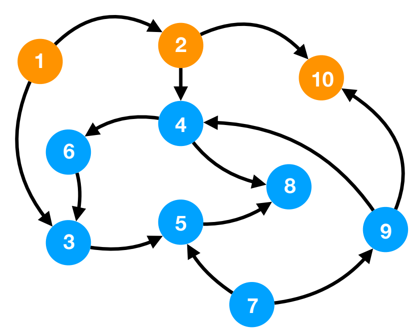
Topological sort



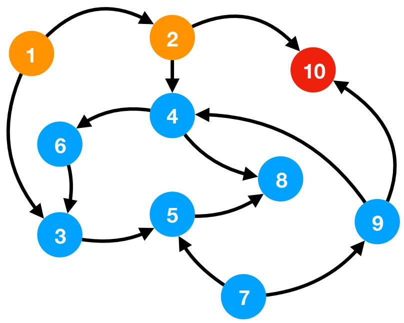
Topological sort



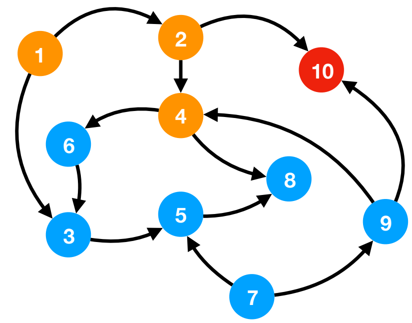
Topological sort



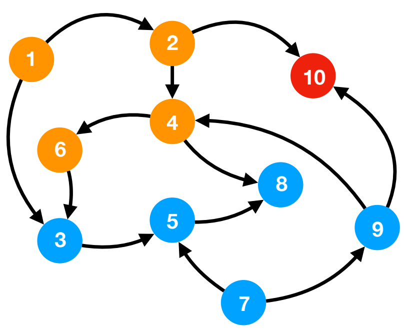
Topological sort



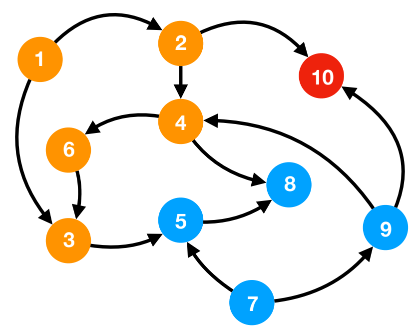
Topological sort



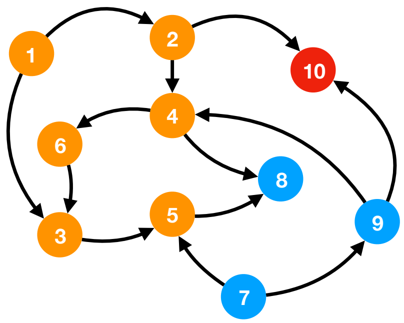
Topological sort



Topological sort

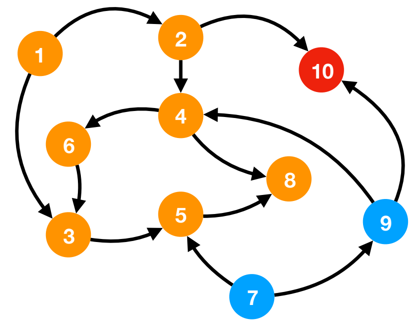


Topological sort



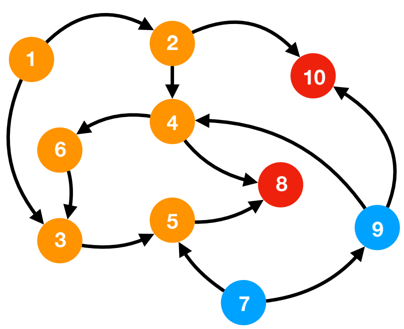
10

Topological sort



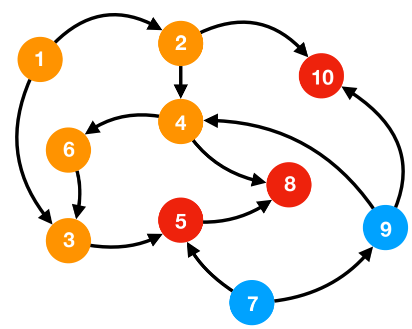
10

Topological sort



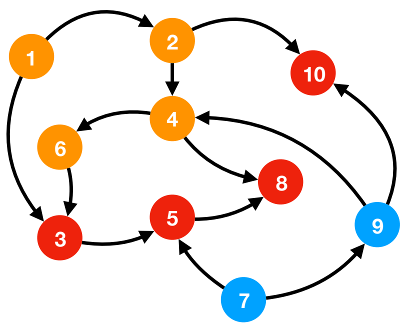
8, 10

Topological sort



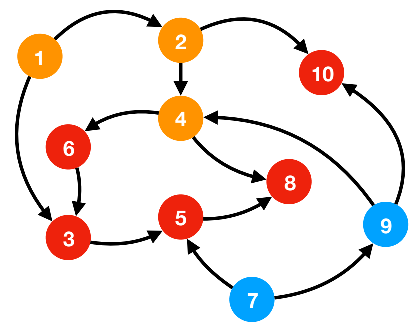
5, 8, 10

Topological sort



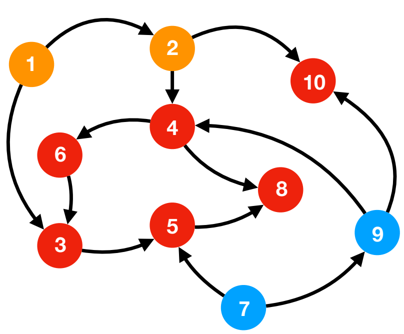
3, 5, 8, 10

Topological sort



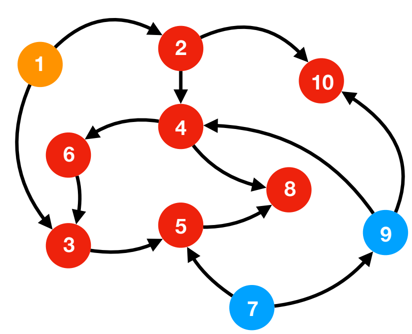
6, 3, 5, 8, 10

Topological sort



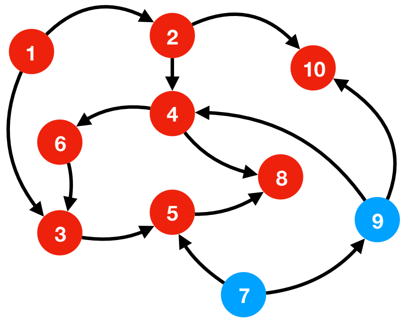
4, 6, 3, 5, 8, 10

Topological sort



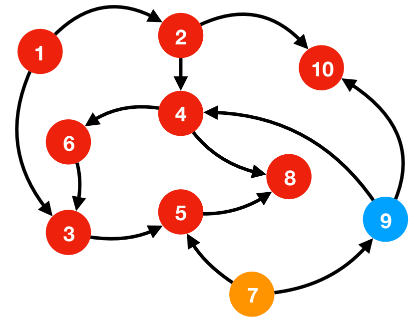
2, 4, 6, 3, 5, 8, 10

Topological sort



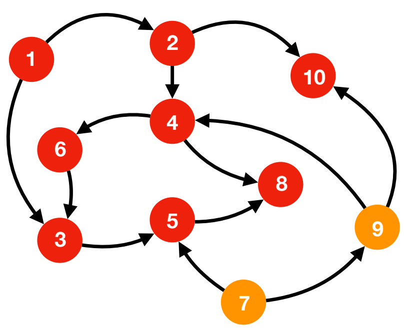
1, 2, 4, 6, 3, 5, 8, 10

Topological sort



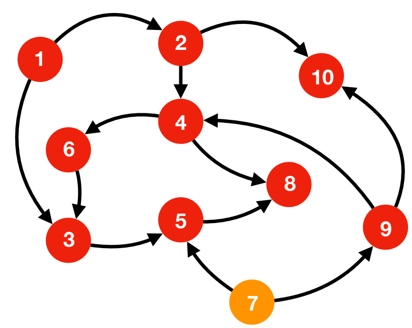
1, 2, 4, 6, 3, 5, 8, 10

Topological sort



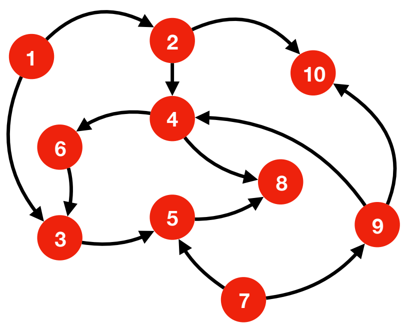
1, 2, 4, 6, 3, 5, 8, 10

Topological sort



9, 1, 2, 4, 6, 3, 5, 8, 10

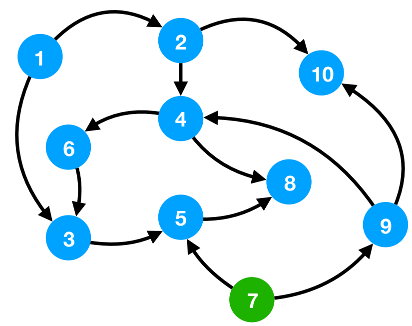
Topological sort



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

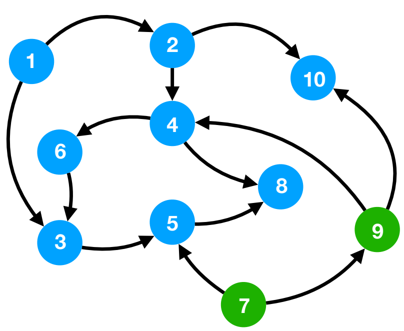
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

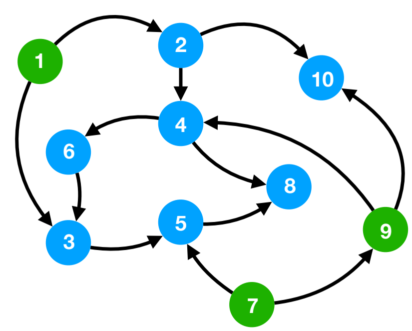
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

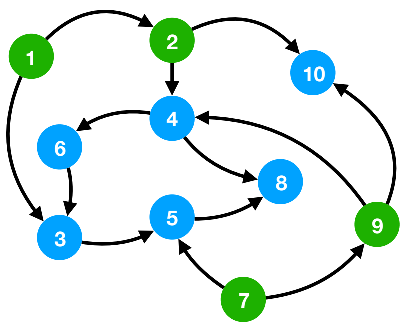
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

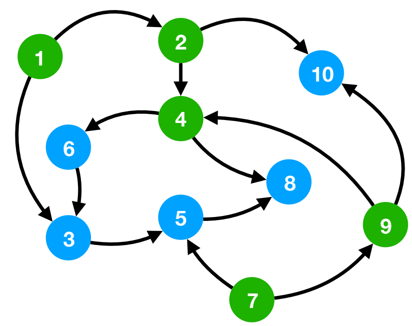
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

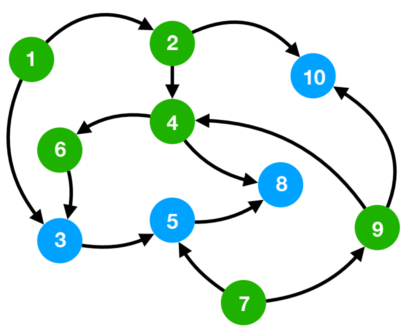
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

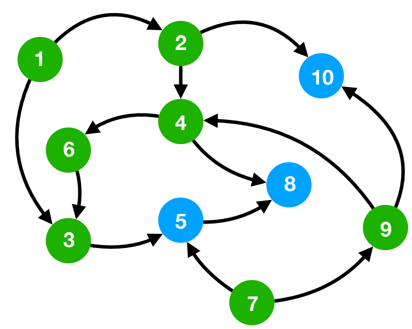
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

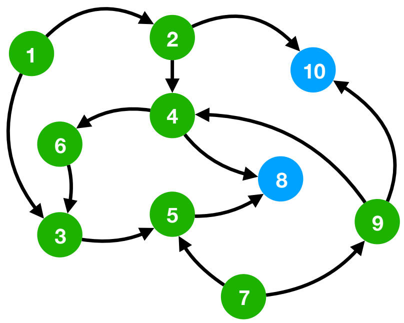
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

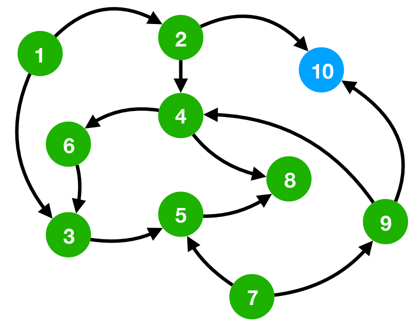
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

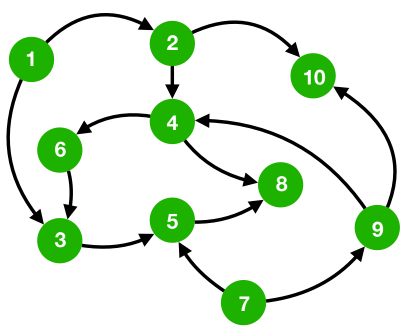
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Yes!

Topological sort (depth-first)

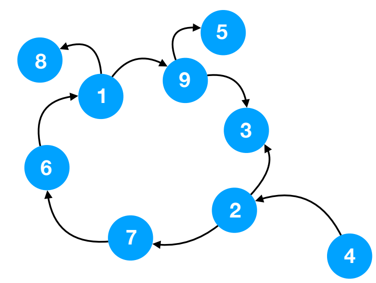
```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then return
    if n has a temporary mark then stop   (not a DAG)
    mark n with a temporary mark
    for each node m with an edge from n to m do
        visit(m)
    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

(from Wikipedia: topological sort)

Question

Why does revisiting a temporary mark (vs permanent or unmarked) mean that the graph is not a DAG?

Activity



Is this graph a DAG?

If so, produce a topological ordering of the vertices.

Hash tables

Recall: arrays

An **array** is a data structure consisting of a **sequential collection of elements**, each identified by an **index**.

A	13	2	451	42	9	6	-4	8
	0	1	2	3	4	5	6	7

Performance guarantees:

1. **read** using index: **O(1)**
2. **write** using index: **O(1)**

Can we capture some of this for a more general structure?

Generalization: associative array

An **associative array** or **key-value store** is a data structure consisting of a **sequential collection of elements**, each identified by a **key**.

A	13	2	451	42	9	6	-4	8
	Joe	Adam	Sue	Ed	Sam	Fay	Dan	Ted

Performance guarantees:

1. **read** using index: **O(1)?**
2. **write** using index: **O(1)?**

Need: function to map key to index

Suppose we had a function:

$$h(k) \rightarrow z$$

where k is a key of arbitrary value and $z \in \mathbb{Z}$,

then we could construct another function:

```
int index(K key) {
  return abs(h(key) % A.length);
}
```

A	13	2	451	42	9	6	-4	8
	Joe	Adam	Sue	Ed	Sam	Fay	Dan	Ted

Hash function

A **hash function** is any function that can be used to map data of **arbitrary size** onto data of a **fixed size**.

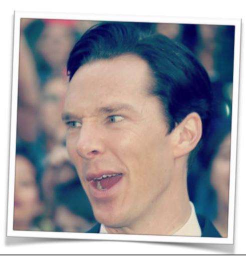
A	13	2	451	42	9	6	-4	8
	Joe	Adam	Sue	Ed	Sam	Fay	Dan	Ted

String of length 2.

String of length 3.

String of length 4.

Why not “`Benedict Cumberbatch`”?



Nerd rant

A.O. Scott in *The New York Times'* review deduced from the film that Turing was “a sentient robot, an empathetic space alien, a warm-blooded salamander with crazy sex appeal.”

“[C]olleagues at the time called him intensely shy and kindly.”

“... unfailingly generous with his time and expertise ...”

“... inspired loyalty and affection among those who appreciated his unusual gifts.”

See: http://blog.yalebooks.com/2015/01/07/alan-turing/

Hash function

Useful hash functions also provide the following guarantees:

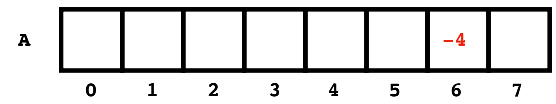
Determinism: a given input value must always generate the same hash value.

Uniformity: maps the expected inputs as evenly as possible over its output range.

Equivalence: any two values that are considered equivalent should produce the same hash value.

Hash table

A **hash table** is a data structure that implements an **associative array** abstract data type. A hash table uses a **hash function** to compute an index into an array of **buckets**, from which the desired value can be found.



`“Dan”, -4`

`index(“Dan”) → 6`

`A[index(“Dan”)] = -4`

Question

Is a function that **generates a random number** a **good hash function**?

No. Random numbers do tend to be uniform, but are not deterministic.

Activity

See if you can come up with a simple hash function for strings.

Determinism: a given input value must always generate the same hash value.

Uniformity: maps the expected inputs as evenly as possible over its output range.

Equivalence: any two values that are considered equivalent should produce the same hash value.

Code: let’s check the quality of hash

American Standard Code for Information Interchange (ASCII)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Hash codes

Hashing so important that every **`Object`** in Java has a built-in hash function.

hashCode

`public int hashCode()`

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)

Returns:

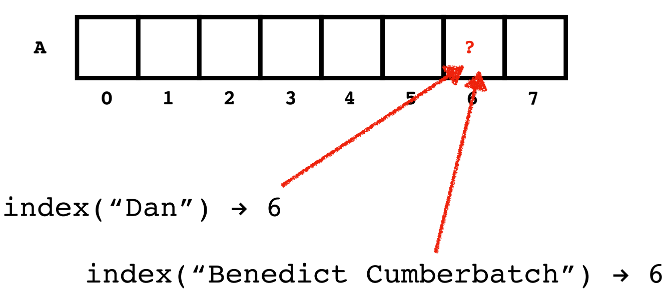
a hash code value for this object.

See Also:

`equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

Hash collisions

A **hash collision** is when **two or more distinct keys** have the **same hash value**.



Recap & Next Class

Today we learned:

Topological order

Hash tables

Next class:

Avoiding hash collisions

Collision-resistant hash tables