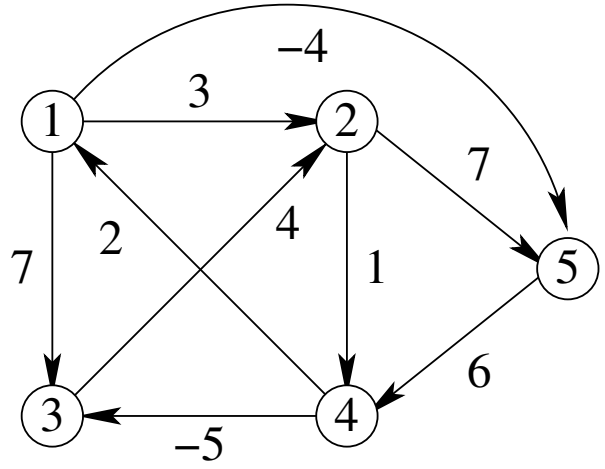
Sample run of Floyd-Warshall's algorithm for APSP



$D^{(0)}$

0	3	7	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

$D^{(1)}$

0	3	7	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	**5**	-5	0	**-2**
∞	∞	∞	6	0

$D^{(2)}$

0	3	7	**4**	-4
∞	0	∞	1	7
∞	4	0	**5**	**11**
2	5	-5	0	-2
∞	∞	∞	6	0

$D^{(3)}$

0	3	7	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	**-1**	-5	0	-2
∞	∞	∞	6	0

$D^{(4)}$

0	3	**-1**	4	-4
3	0	**-4**	1	**-1**
7	4	0	5	**3**
2	-1	-5	0	-2
8	**5**	**1**	6	0

$D^{(5)}$

0	**1**	**-3**	**2**	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

Project 4 feedback.

See notes for Lecture 22 on How to use Graph class.

1. Use the public methods of Graph (only).

List of problems & solutions

1. ~~store.get(u)~~ get(u)

2. ~~store.put(u, ...)~~ ~~put(u, ...)~~ — never needed

3. ~~g.directed~~ g.isDirected()

4. ~~e.to~~ e.toVertex()
   ~~e.from~~ e.fromVertex()

5. ~~HashMap<Vertex, PartVertex>~~ (Integer) — not needed

   get(u)

   ~~HashMap<MstVertex, Vertex>~~

   ```
   MstVertex {
       Vertex vertex;

   MstVertex (u) {
       vertex = u;
       :
   }
   getVertex() {
       return vertex;
   }
   ```

6. u.name — not needed

   u.getName()

7. g.adj(u) ~~.inEdges~~ ~~.outEdges~~ = { g.inEdges(u), g.outEdges(u), g.incident(u) }

Do not use explicit iterators, unless needed.

Color : public enum Color { WHITE, GRAY, BLACK };

```
DFSVertex {
    Color color;
}
.
.
get(u).color = Color.WHITE;
```

Do not add / remove / modify parameters or return types of public methods.

```
static MST kruskal(Graph g) {
    MST m = new MST(g);
    m.kruskal();
    return m;
}

void kruskal() {
    code ..
}
```

(m.get(u) — MST vertex of u in instance m of MST.)

Graph is g.

Variety of topics

1. Trie data structure: parse strings efficiently.
   Professional code: try to avoid string.equals("else")
   or ~~case~~ switch(str) { case "Add"
   :
   }

   e.g. Address book,,
   Keywords

   Entry { HashMap<Character, Entry> child;
   int depth; Value value; }

2. Suffix trees: compressed trie that stores all suffixes of a piece of text (string)

3. Counting sort. A[1..n] elements are from 1..k
   A has a field key ∈ [1..k] for some k = O(n).

```
for (x: A) {
    list[x.key].add(x)
} j ← 1
for i ← 1 to k do
    for (x: list[i]) { do
        A[j++] ← x
```

RT = O(n+k)
= O(n).

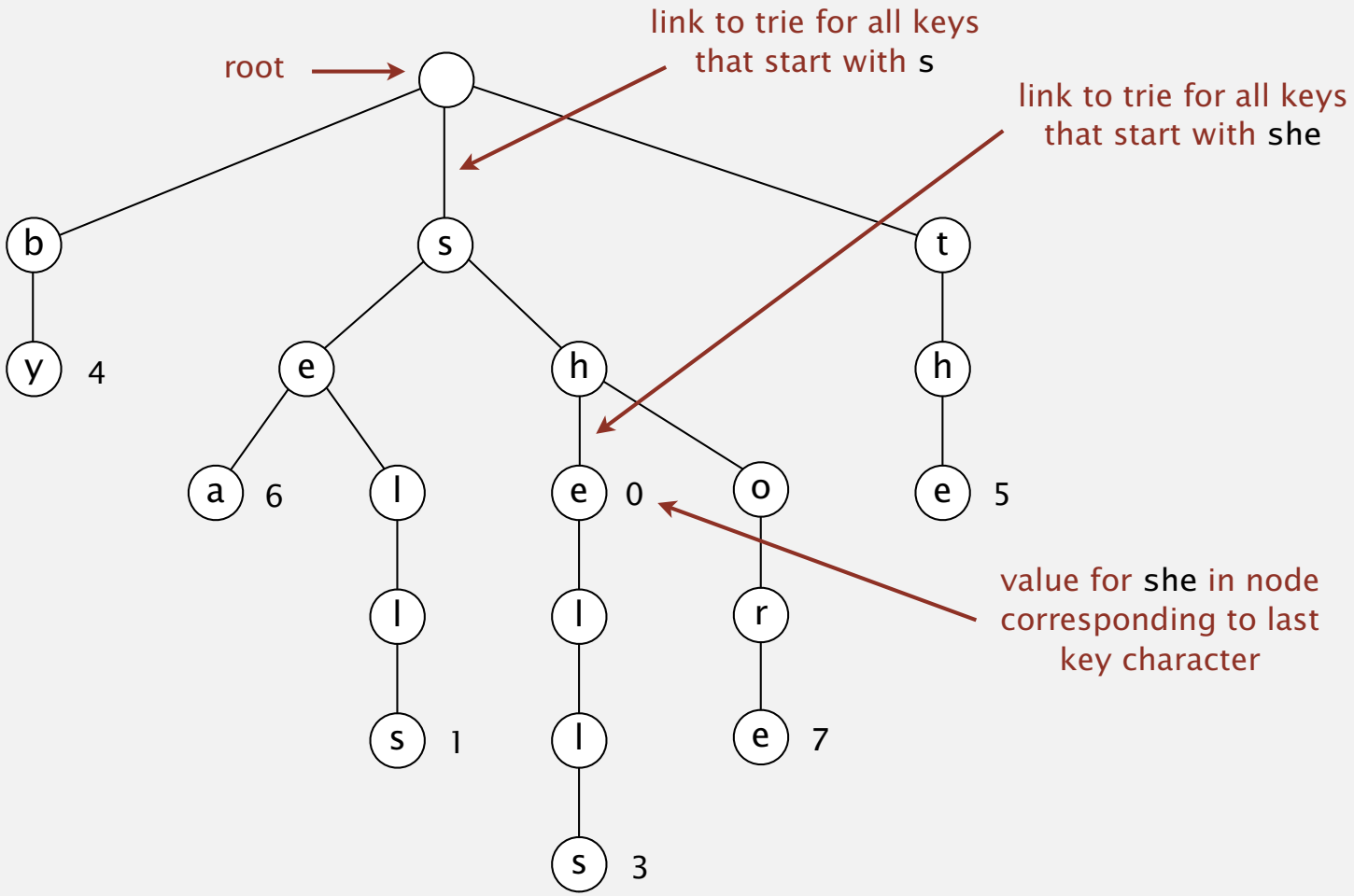
// Another version exists using just arrays instead of lists

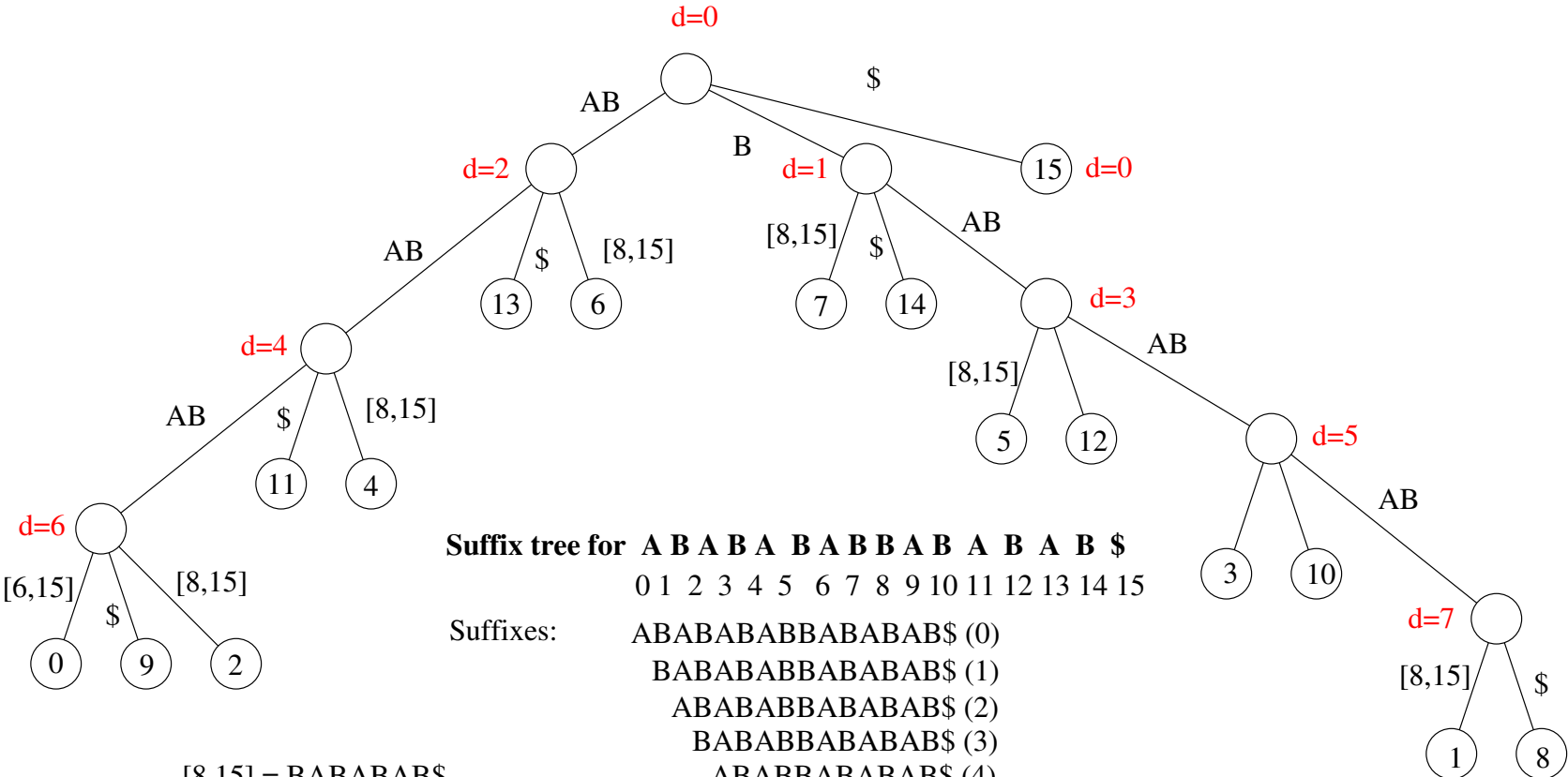
Tries

Tries. [from retrieval, but pronounced "try"]

- Store characters in nodes (not keys).
- Each node has R children, one for each possible character.
  (for now, we do not draw null links)

key	value
by	4
sea	6
sells	1
she	0
shells	3
shore	7
the	5





Suffix tree for A B A B A B A B B A B A B A B $

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Suffixes:

ABABABABBABABAB$ (0)
BABABABBABABAB$ (1)
ABABABBABABAB$ (2)
BABABBABABAB$ (3)
ABABBABABAB$ (4)
BABBABABAB$ (5)
ABBABABAB$ (6)
BBABABAB$ (7)
BABABAB$ (8)
ABABAB$ (9)
BABAB$ (10)
ABAB$ (11)
BAB$ (12)
AB$ (13)
B$ (14)
$ (15)

[8,15] = BABABAB$

[6,15] = ABBABABAB$