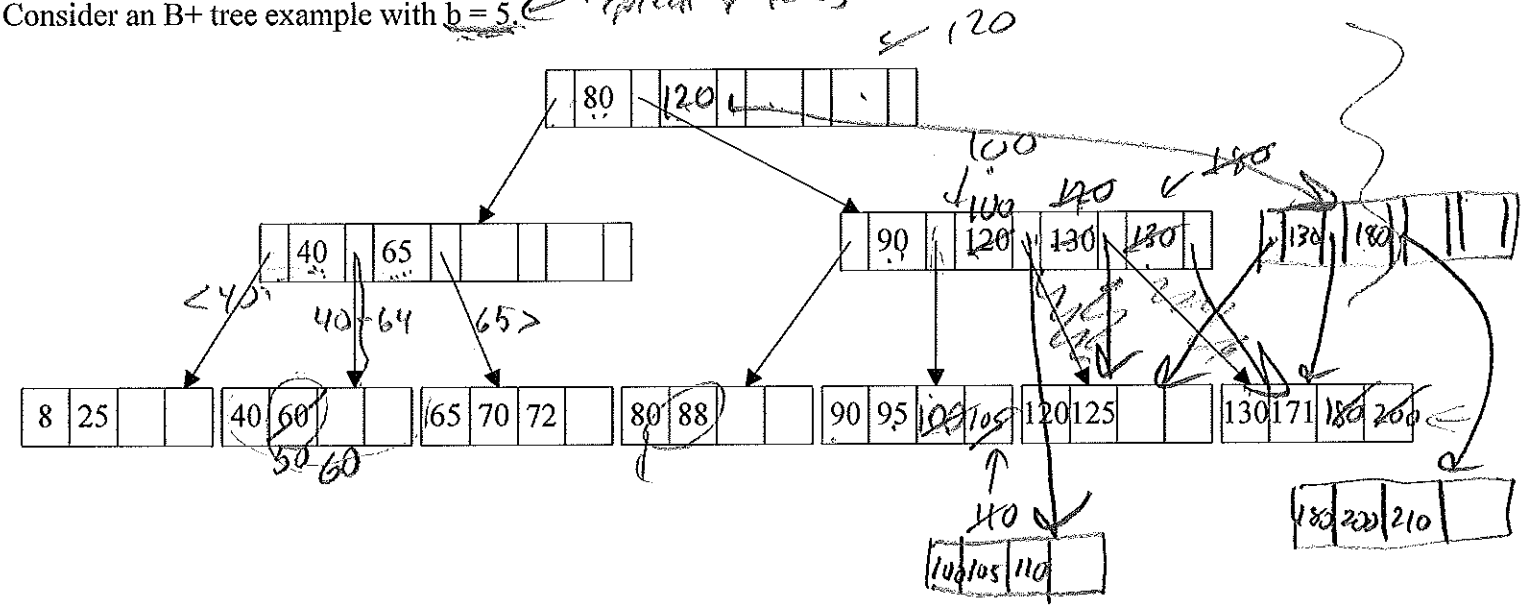
9. A B+ Tree is a multi-way tree (typically in the order of 100s children per node) used primarily as a file-index structure to allow fast search (as well as insertions and deletions) for a target key on disk. Two types of *pages* (B+ tree "nodes") exist:

- Data pages - which always appear as leaves on the same level of a B+ tree (usually a doubly-linked list too)
- Index pages - the root and other interior nodes above the data page leaves. Index nodes contain some minimum and maximum number of keys and pointers bases on the B+ tree's *branching factor (b)* and *fill factor*. A 50% fill factor would be the minimum for any B+ tree. All index pages must have $\lceil b/2 \rceil \leq$ # child $\leq b$, except the root which must have at least two children.

Consider an B+ tree example with b = 5. ← typically 100's

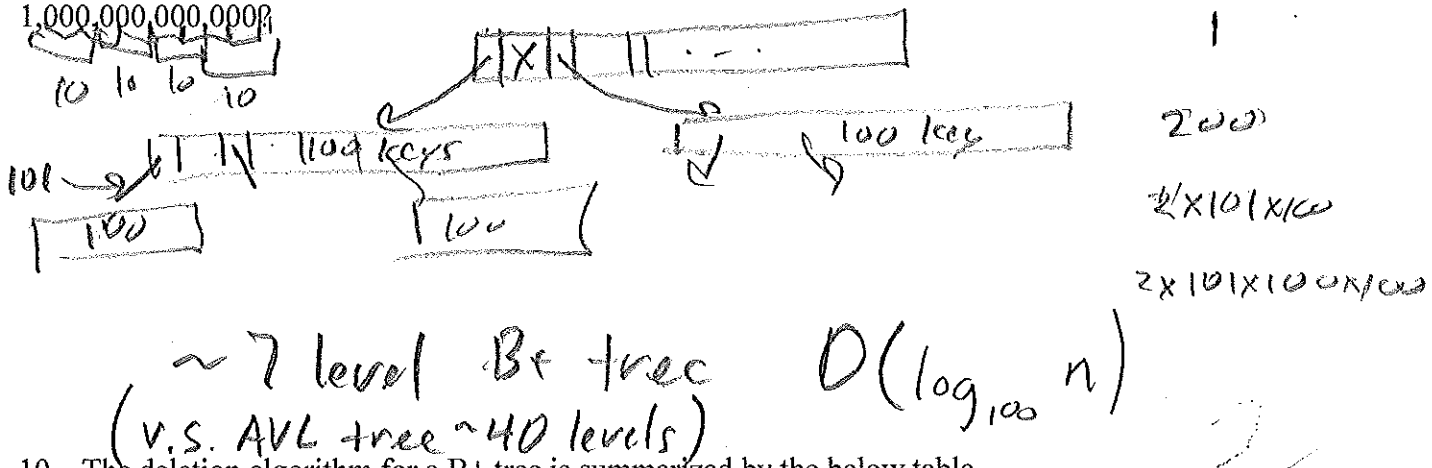


a) How would you find 88? Start at "root" index data and follow appropriate pointers down tree

b) The insert algorithm for a B+ tree is summarized by the below table. Where would you insert 50, 100, 105, 110, 180, 200, 210?

Situation		insertion Algorithm
Data Page Full?	Parent Index Page Full?	
No	No	Place record in sorted position in the appropriate data page.
Yes	No	1. Split data page with records < middle key going in left data page and records ≥ middle key going in right data page. 2. Place middle key in index page in sorted order with the pointer immediately to its left pointing to the left data page and the pointer immediately to its right pointing to the right data page.
Yes	Yes	1. Split data page with records < middle key going in left data page and records ≥ middle key going in right data page. 2. Adding middle key to parent index page causes it to split with keys < middle key going into the left index page, keys > middle key going in right index page, **and** the middle key inserted into the next higher level index page. If the next higher index page is full continue to splitting index pages up the B+ tree as necessary.

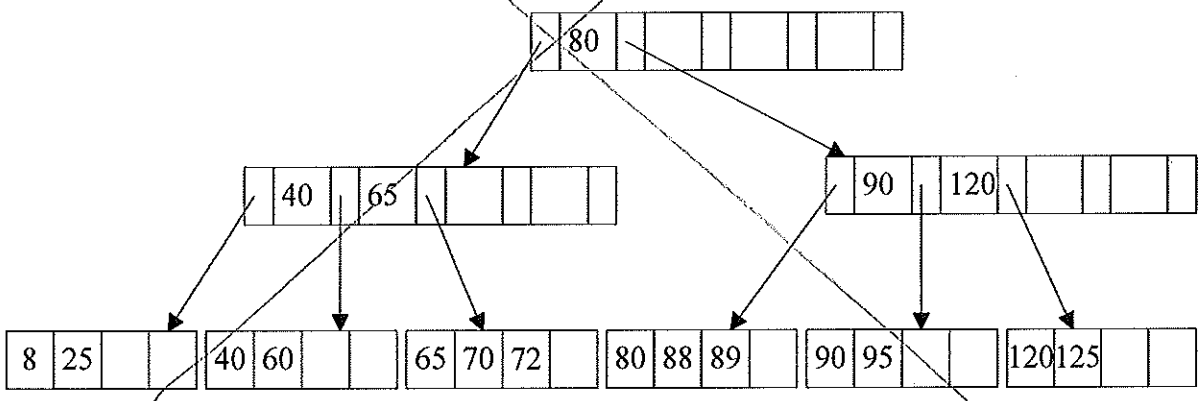
c) For a B+ tree with a branch factor 201, what would be the worst case height of the tree if the number of keys was 1,000,000,000,000?



10. The deletion algorithm for a B+ tree is summarized by the below table.

Situation		deletion Algorithm
Data Page Below Fill Factor?	Parent Index Page Below Fill Factor?	
No	No	Delete record from the data page. Shifting records with larger keys to left to fill in the hole. If the deleted key appears in the index page, use the next key to replace it.
Yes	No	1. Combine data page and its sibling. Change the index page to reflect the change.
Yes	Yes	1. Combine data page and its sibling. 2. Adjusting the index page to reflect the change causes it to drop below the fill factor, so combine the index page with its sibling. 3. Continue combining the next higher level index pages until you reach an index page with the correct fill factor or you reach the root index page.

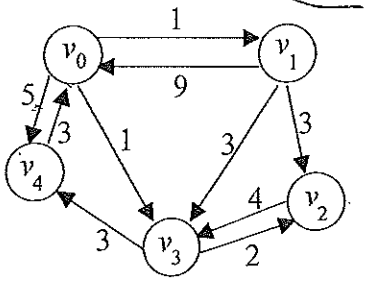
Consider an B+ tree example with b = 5 and 50% fill factor. Delete 89, 65, and 88. What is the resulting B+ tree?



NOTE: Delete from B+ tree not on test

Data Structures (CS 1520) Lecture 25 Name: _____

1. Consider the following directed graph (diagraph) $G = (V, E)$: set of edges, set vertices



a) What is the set of vertices? $V =$ $\{v_0, v_1, v_2, v_3, v_4\}$

b) An edge can be represented by a tuple (from vertex, to vertex [, weight]). What is the set of edges?

$E =$ $\{(v_0, v_1, 1), (v_0, v_3, 1), (v_0, v_4, 5), (v_1, v_0, 9), (v_1, v_2, 3), \ldots\}$

c) A path is a sequence of vertices that are connected by edges. In the graph G above, list two different pathes from v_0 to v_3. v_0, v_3 or v_0, v_1, v_3

d) A cycle in a directed graph is a path that starts and ends at the same vertex. Find a cycle in the above graph.

v_3, v_4, v_0, v_2

2. Like most data structures, a graph can be represented using an array, or as a linked list of nodes.

a) The array representation is called an *adjacency matrix* which consists of a two-dimensional array (matrix) whose elements contain information about the edges and the vertices corresponding to the indices.

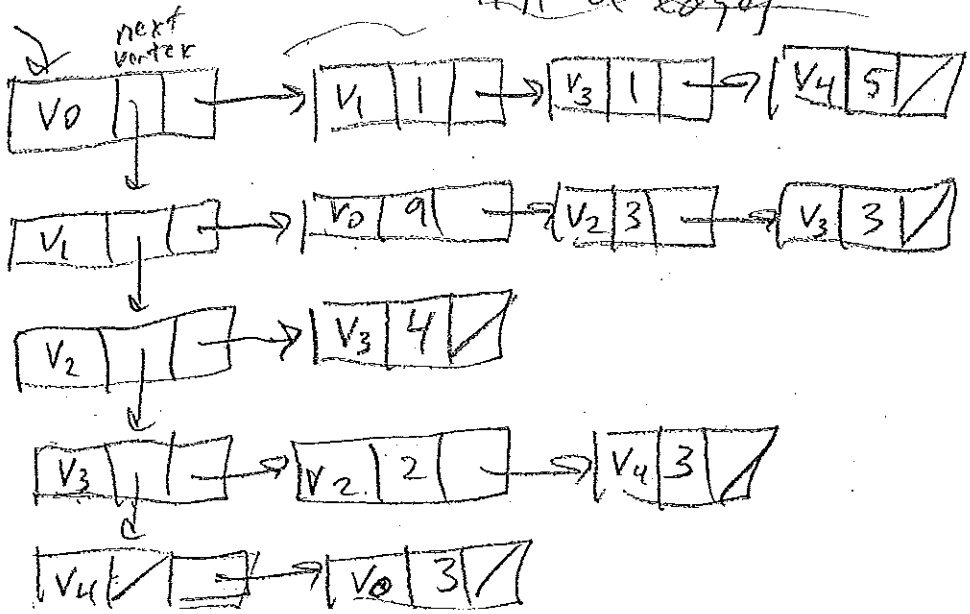
Complete the following adjacency matrix for the above graph.

Time to check an edge weight: $O(1)$

Storage: $O(|V|^2)$

(from vertex) \ (to vertex)	v_0	v_1	v_2	v_3	v_4
v_0	0	1		1	5
v_1	9		3	3	
v_2				4	
v_3			2		3
v_4	3				

3. The linked representation maintains a array/Python list (or Python dictionary) of vertices with each vertex maintaining a linked list of other vertices that it connects to. Draw the adjacency list representation below:



Time to find edge weight: $O(|V|+|V|)$

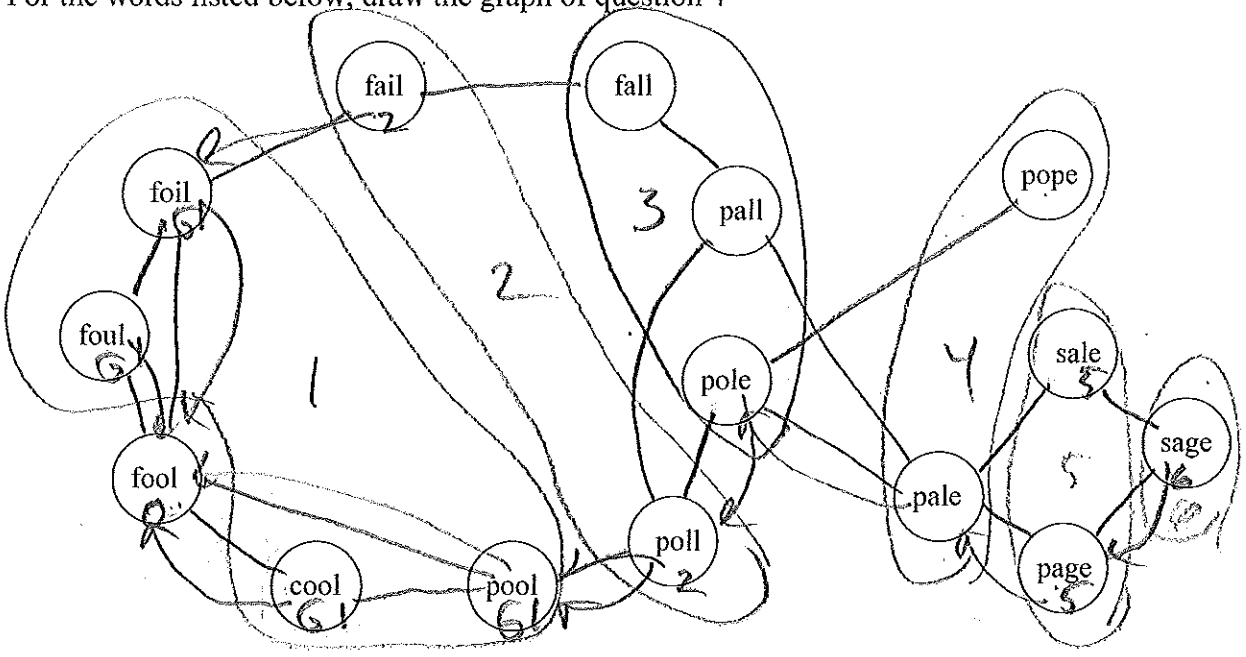
Storage $O(|V|+|E|)$

4. Graphs can be used to solve many problems by modeling the problem as a graph and using "known" graph algorithm(s). For example, consider the *word-ladder puzzle* where you tranform one word into another by changing one letter at a time, e.g., transform FOOL into SAGE by FOOL → FOIL → FAIL → FALL → PALL → PALE → SALE → SAGE.

We can use a graph algorithm to solve this problem by constructing a graph such that

- a word represents a vertex
- an edge represents? two words that differ at a single letter
- a word ladder transformation from one word to another represents? path from FOOL to SAGE

5. For the words listed below, draw the graph of question 4



a) List a different transformation from FOOL to SAGE

FOOL → COOL → POOL → PALL → PALE → SALE → SAGE

b) If we wanted to find the shortest transformation from FOOL to SAGE, what does that represent in the graph?

shortest path from FOOL to SAGE

c) There are two general approaches for traversing a graph from some starting vertex *s*:

- Breadth First Search (BFS) where you find all vertices a distance 1 (directly connected) from s, before finding all vertices a distance 2 from s, etc.
- Depth First Search (DFS) where you explore as deeply into the graph as possible. If you reach a "dead end," we backtrack to the deepest vertex that allows us to try a different path.

Which of these traversals would be helpful for finding the **shortest** solution to the word-ladder puzzle?

BFS - when SAGE is found we found it by shortest path

Data Structures (CS 1520) Lecture 26 Name:_____

1. There are two general approaches for traversing a graph from some starting vertex *s*:

- Depth First Search (DFS) where you explore as deeply into the graph as possible. If you reach a "dead end," we backtrack to the deepest vertex that allows us to try a different path.
- Breadth First Search (BFS) where you find all vertices a distance 1 (directly connected) from s, before finding all vertices a distance 2 from s, etc.

What data structure would be helpful in each type of search? Why?

a) Breadth First Search (BFS): queue

~~fail~~ ~~pool~~ | add | fail | fail | poll |

empty queue
enqueue start vertex with color gray
while queue is not empty
dequeue nextVertex
for each neighboring vertex
if color is white then
enqueue neighbor with color gray
make nextVertex as black

b) Depth First Search (DFS):

- use stack instead of queue to remember where to backtrack to

or

use recursion and the run-time stack for DFS.

2. On the next page is the textbook's edge, vertex, and graph implementations.

a) How does this graph implementation maintain its set of vertices?

b) How does this graph implementation maintain its set of edges?

3. Assuming a graph g containing the word-ladder graph from lecture 26, on the diagram trace the `bfs` algorithm by showing the value of each vertex's `color`, `predecessor`, and `distance` attributes?