EECS 281, Week 8: Wednesday

Hash Tables

1. Describe hashing as a two-step process.

   key (any object) ⟶ {0, 1, .., n-1}

   ① → hash code (any integer)

   ② → compress

2. Consider the hash function for a `string` below:

```
int hash(const string &s) {
   return toupper(s[0]) - 'A';
}
```

   ignoring length + rest of chars AT ALL

   Critique this hash function, identifying and explaining its downsides.

   Only 26 possible hash codes

   Case insensitive

3. What is a collision?

   insert, try to use same index in table

4. When can collisions occur?

   ① Get same hash code

   ② After compression

   ③ (During probing)

5. How are collisions different from clusters?

   Long runs of filled elts
   Made by collision resolution

6. What are the properties of a good hash function?

   Same input → same output
   Uniform distribution of output
   Generate different output for similar input

   Quick

7. What is a good way to generate hash codes for a string? $a = 31, 33$

   $h(s) = s_0 \cdot a^{n-1} + s_1 \cdot a^{n-2} + s_2 \cdot a^{n-3} + \ldots + s_{n-1} \cdot a^0$

8. What is load factor α?

   $\alpha = \frac{n}{m}$

EECS 281, Week 8: Wednesday

9. What is dynamic hashing?

$> \alpha_{max} \rightarrow$ grow + rehash ($< \alpha_{min}$ shrink + rehash)

10. What is simple uniform hashing?

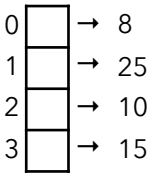
$Pr(\text{elt } i \text{ hashes to slot } k) = \frac{1}{m}$

11. Suppose we use a hash function *h* to hash *n* distinct keys into an array of length *m*. Assuming simple uniform hashing, what is the expected number of collisions?

$$E(\#\text{ of colliding pairs}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{m} = \Theta\left(\frac{n^2}{m}\right)$$

12. Demonstrate what happens when we insert the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \mod 9$.

13. Suppose we have a hash table that stores integer keys and uses chaining as its collision resolution technique. Assume that the hash code for an integer is that integer itself.



Demonstrate what happens when we insert the keys 18 and 5. Be sure to resize the hash table by doubling the array when the load factor reaches 1.5.

14. When you modify a key that has been inserted into a hash table, will you be able to retrieve that entry again? Explain.

A. Always
B. Sometimes
C. Never

→ Yes if hash code remains the same
↳ No if hash code is different

15. When you modify a value that has been inserted into a hash table, will you be able to retrieve that entry again? Explain.

A. Always
B. Sometimes
C. Never

EECS 281, Week 8: Wednesday

16. (T/F) While a good hash table that contains *N* elements and uses separate chaining has average-case constant time for look-up, the worst case running time for a lookup is $\Theta(\log N)$.

17. (T/F) `std::vector` can be used as a key in `std::unordered_map`.

18. What do we need to do in order to use `std::unordered_map` with custom types?

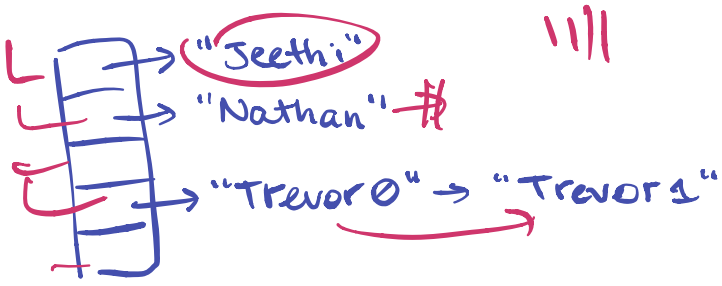
1. Operator ==
2. Hash function / functor
   or specialize std::hash

19. (T/F) A hash table can be used to store a dictionary in a spell-checker program.

20. Consider this definition of a hash table that stores English words and uses separate chaining as its collision resolution technique:

strings are keys, no values

```
struct HashTableNode {
    string word;
    HashTableNode* next = nullptr;
};

class HashTable {
public:
    size_t size() const;
private:
    vector<HashTableNode*> slots;
}
```



Suppose that `HashTable` has been implemented without an instance variable to keep track of the number elements. Complete the implementation of `size` method below in such a way that it returns the number of words stored in the hash table. Feel free to add other helper (private) functions.

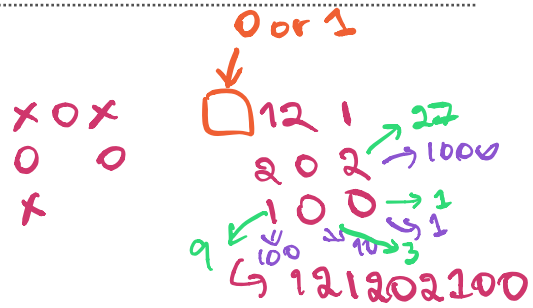
```
size_t HashTable::size() const {

}
```

for (auto ptr : slots) {
...
ptr = ptr->next

EECS 281, Week 8: Wednesday

21. Consider the definition of a `TicTacToeGame` class.

```
class TicTacToeGame {
    public:
        // always 3-by-3
        const size_t size = 3;
        // hash code to use for std::hash()
        int hashCode() const;
    private:
        // either 'X', 'O' or ' '
        char tiles[size][size];
        // keep track of turns
        bool isXTurn = true;
};
```



Implement the `hashCode` methods that computes the hash code to specialize `std::hash` for the Tic-Tac-Toe game. Ensure that different board configurations have different hash codes.

```
int TicTacToeGame::hashCode() const {

}
```

22. How many different configurations of the Tic-Tac-Toe board are there?

$3 \times 3 \times 3 \times \ldots \times 2$ (9 factors) $= 2 \times 3^9$

EECS 281, Week 8: Wednesday

Tries

23. (T/F) A trie is a binary tree.

24. To use objects of type T as keys in a trie, what property must the type T have?

made of components (ex.: chars)
↳ from a fixed Alphabet

25. Suppose we have a trie that stores *n* English words, and each of trie's nodes contains an array of *p* pointers to other nodes. What is the time time complexity of looking up a word of size *k* in this trie?

$O(k)$

best case → $\Theta(1)$
worst case → $\Theta(k)$

or constant if $k \leq 45$

26. (T/F) Tries can be used to store valid words (i.e., if they are in the dictionary), but cannot be used to store the words' definition.

27. (T/F) Unlike hash tables, we cannot implement tries in a way to store duplicate keys.

chain / vector

28. (T/F) Let *T* be a trie that stores *N* ~~strings~~ English words. We can traverse *T* in sorted order in $\Theta(N)$ time.

if $k \leq C$ (constant)

29. (T/F) Let *T* be a trie that stores strings backwards, with the top of trie corresponding to the last, rather than the first character of the string. We have traverse *T* in sorted order in $\Theta(N)$ time.

30. (T/F) We can ~~you~~ use a trie to do a range query, e.g., to get all strings between "maxim" and "pizza" in the dictionary in $\Theta(N)$ time.

31. If we traverse the same node while searching for two strings, S_1 and S_2, in a trie and that node is at depth *k* in the trie (where the root is depth 0), what can we say about S_1 and S_2?

(At least) first k chars of S_1 and S_2 are the same

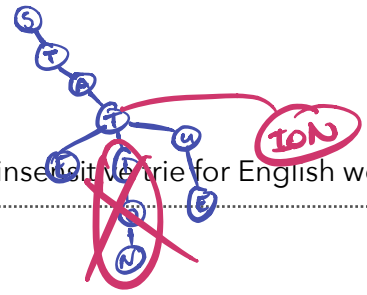
32. Describe one disadvantage of using a trie to store a dictionary of English words?

Memory

33. Suppose we have a trie that stores n English words, and each of trie's nodes contains an array of p pointers to other nodes. Describe two ways to improve memory usage of this trie.

① Use a hash table instead of fixed-size array

② Compression



34. Consider this definition of a case-insensitive trie for English words:

```
struct TrieNode {
    bool isWord;
    TrieNode* children[26];
};

class Trie {
public:
    size_t size() const;
private:
    TrieNode* root;
}
```

BREAK UNTIL 1:45 PM

Suppose that `Trie` has been implemented without an instance variable to keep track of the number of words in trie. Complete the implementation of `size` method below in such a way that it returns the number of words stored in the trie. Feel free to add other helper (private) functions.

```
size_t Trie::size() const {

}
```

35. When should we use a hash table over a trie? Trie over a hash table?

EECS 281, Week 8: Wednesday

Trees

36. What is the depth of a node?

length of path from root

37. Implement depth, a function that takes a `BinaryTreeNode*` and returns the depth of that node. Assume that `BinaryTreeNode` struct has a `parent` pointer.

```
int depth(BinaryTreeNode* node) {
    if parent is null ptr
        → 0
    else
        → depth of parent + 1
```

38. What is the height of a node?

max dist to a leaf

39. Implement height, a function that takes a `BinaryTreeNode*` and returns the height of the tree rooted that node. Assume that `BinaryTreeNode` struct has a `left` pointer and a `right` pointer, and that the height of an empty tree is -1 and that the height of a tree with one node is 0.

```
int height(BinaryTreeNode* node) {
    if node == nullptr
        → -1
    else
        → 1 + max(height(node → left),
                  height(node → right))
```

40. Complete the table below with amortized time complexities and the data structure generally used by the STL for these containers.

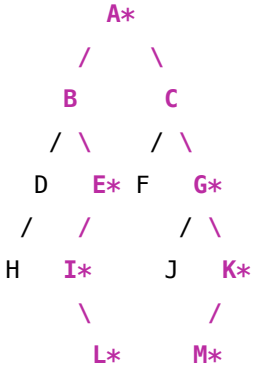
Operation	set	map	unordered_set	unordered_map
insert	$\Theta(\log N)$	$\Theta(\log N)$	$\Theta(1)$	$\Theta(1)$
find	$O(\log N)$	$O(\log N)$	$\Theta(1)$	$\Theta(1)$
remove	$O(\log N)$	$O(\log N)$	$\Theta(1)$	$\Theta(1)$
Data structure	Red-black tree	Red-black tree	hash table	hash table

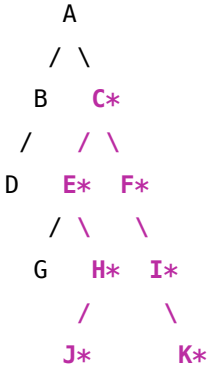
order usually based on <

avg amortized if hash fn is good

EECS 281, Week 8: Wednesday

41. The *diameter* of a tree is the maximum number of edges on any path connecting two nodes of the tree. For example, here are two sample trees and their diameters. In each case the longest path is shown with by circling the codes and shown in purple. Note that there can be more than one longest path.





Implement the function `diameter` that computes the diameter of a binary tree represented by a pointer to an object of `BinaryTreeNode` class.

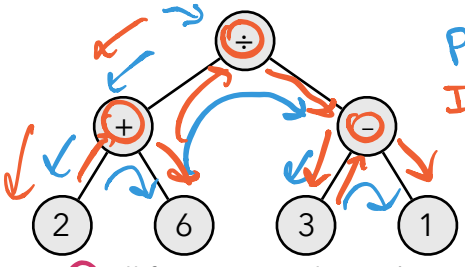
```
int diameter(BinaryTreeNode* node) {

}
```

42. (T/F) The height of a binary tree with *N* elements is log *N*.

42.1. FALSE — BST — log N

insert 1, 2, 3, 4

43. What are pre-order, in-order, post-order and level-order traversals of this tree?



Pre-order: ÷ + 2 6 - 3 1
In-order: 2 + 6 ÷ 3 - 1
Post-order: 2 6 + 3 1 - ÷
Level-order: ÷ + - 2 6 3 1

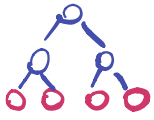
44. (T/F) All four traversals can be performed on all trees.

in-order only for binary trees

45. (T/F) For any non-empty binary tree with *A* leaves and *B* nodes of degree 2, *A* = *B* + 1.

4

4 = 3 + 1



46. (T/F) For any non-empty binary tree with *A* nodes and *B* nodes of degree 2, *A* = *B* + 1.

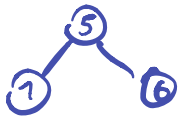
7

3

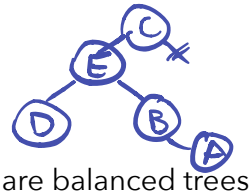
7 ≠ 3 + 1

EECS 281, Week 8: Wednesday

47. (T/F) In-order traversal of a binary search tree will visit the nodes in sorted order.



48. A binary tree has a post-order traversal D A B A C and an in-order traversal D C B A C. Draw the binary tree and give its pre-order traversal.

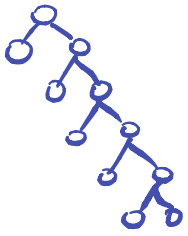


C E D B A

49. (T/F) BSTs are balanced trees.

50. (T/F) Searching for an element in a binary search tree with *N* nodes where each node has either 0, 1 or 2 children takes Θ(*N*) time in the worst case.

51. (T/F) Searching for an element in a binary search tree with *N* nodes where each node has either 0 or 2 children takes Θ(log *N*) time in the worst case.



52. (T/F) In a binary search tree, all internal nodes, except the ones at the last level, have two children.

NOT A LEAF

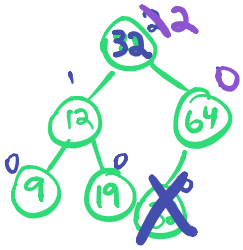
53. What is the purpose of AVL trees?

Guaranteed to be balanced → O(log N)

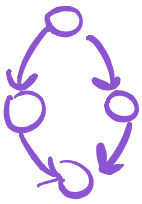
54. Insert the following values into an empty AVL tree: 9, 21, 64, 12, 1, 19, 32. Then remove 1 and 21.

AVL tree property: height of children differs by ≥ 1

55. Insert the following values into an empty AVL tree: 21, 32, 64, 72, 17. Then remove 21, 32, 64, 72 and 17.



EECS 281, Week 8: Wednesday



Graphs and Graph Algorithms

56. Describe the differences between trees and graphs.

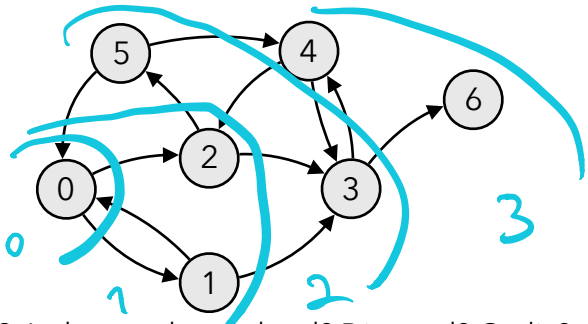
No cycles ↗ ↖ can have more than one path between any two nodes

57. Breadth-first search traversal of a graph is equivalent to which traversal of a tree?

Level-order

58. Describe an algorithm for finding connected components in a graph.

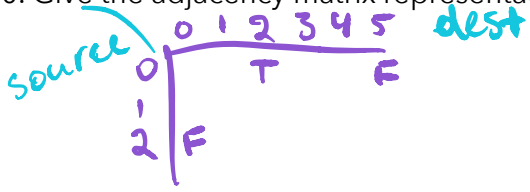
Consider the following graph for the next three questions:



59. Is the graph weighted? Directed? Cyclic?

NO yes yes

60. Give the adjacency matrix representation for the graph shown above.



61. Give the adjacency list representation for the graph shown above.

0 → 1 → 2

62. Give the (pre-order) DFS traversal for the graph shown above, starting at node 0.

0 1 3 4 2 5 6

63. Give the post-order DFS traversal for the graph shown above.

5 2 4 6 3 1 0

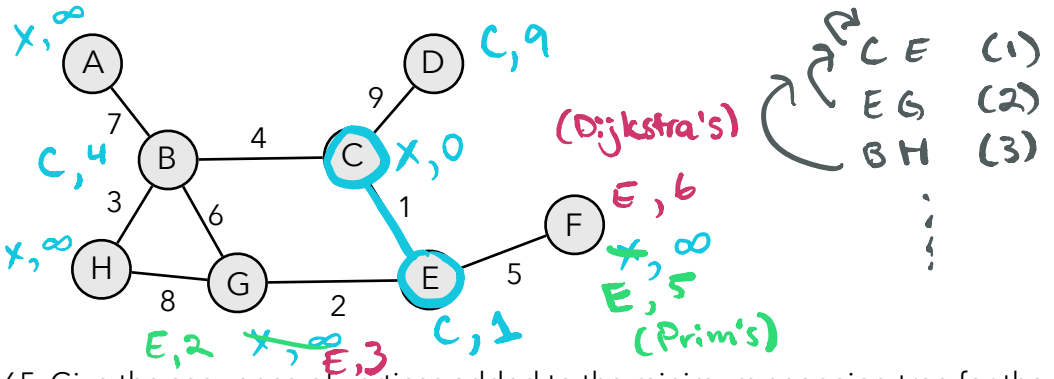
64. Give the BFS traversal for the graph shown above.

BFS:

0 1 2 3 5 4 6

EECS 281, Week 8: Wednesday

Consider the following undirected graph for the next five questions:



65. Give the sequence of vertices added to the minimum spanning tree for the above graph when running Prim's algorithm. Start with vertex C.

C E G B H F A D

66. Give the sequence of vertices added to the maximum spanning tree for the above graph when running Prim's algorithm. Start with vertex C.

67. Give the sequence of edges added to the minimum spanning tree for the above graph when running Kruskal's algorithm.

C-E, E-G

68. Give the sequence of edges added to the maximum spanning tree for the above graph when running Kruskal's algorithm.

69. Give the sequence of edges that would be "explored" when running Dijkstra's algorithm to find the shortest path from node D to node H. D-C, C-E, E-G, B-C, E-F, B-H

Same as Prim's, keep track of dist to source (first node)

70. Prim's, Kruskal's and Dijkstra's are considered greedy algorithms because they solve the problems by progressively making the locally optimal choice at each stage or iteration.

71. (T/F) Dijkstra's algorithm always finds the shortest path on any graph.

does not work on neg. weight

72. An airline is attempting to restructure its services so that that every city in its network can connect to any other city while minimizing the total length of its routes. What problem is the airline trying to solve?

Min span tree

73. An airline's network is set up so that most cities have flights to only a few destinations, and a few cities are hubs, with connections to many destinations. Given this setup, which algorithm should the airline use to create its new network?

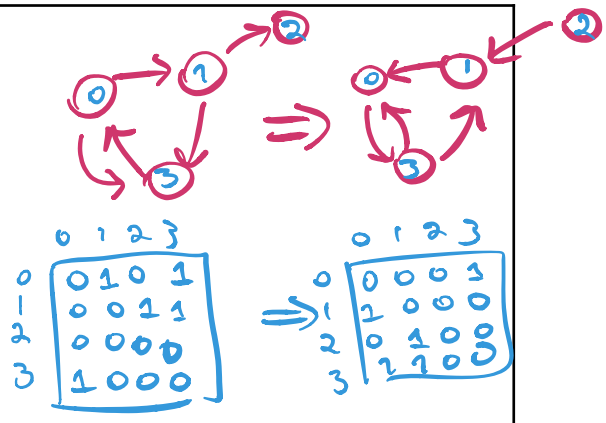
Kruskal's (sparse)

74. An airline wants to generate a minimum spanning tree of its network and represents it with the following distance matrix:
Could this be a valid solution to the airline's MST? Explain.

If we have a cycle ⇒ not a valid MST

75. Suppose that a directed weighted graph is represented with an adjacency matrix. Implement `reverse`, a function that would replace all edges (*v*, *w*) with (*w*, *v*).

```
void reverse(vector<vector<int>> &graph) {
```



Disjoint Sets

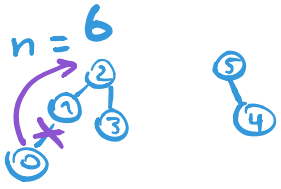
76. What operations can disjoint sets perform efficiently?

Find, Union

77. How can we represent disjoint set with an array?

store parent

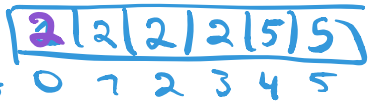
items in Universe are $\{0, 1, 2, \ldots, n-1\}$



78. What optimizations can we perform on disjoint sets?

Union by size: store size as neg. #, smaller set goes into larger set

Path compression: update to set's rep (or parent) (on union + find ops)



79. (T/F) Disjoint sets can be used with Prim's algorithm.

80. (T/F) Disjoint sets can be used with Kruskal's algorithm.

EECS 281, Week 8: Wednesday

Algorithm Design

81. What do C, L, R and S stand for in CLRS?

Authors: Cormen, Leiserson, Rivest and Stein

82. (T/F) Any divide-and-conquer algorithm is an example of a dynamic programming algorithm.

subproblems don't overlap

subproblems overlap

83. (T/F) Merge sort and Quicksort are considered combine-/divide-and-conquer algorithms rather than dynamic programming because they split the work into overlapping subproblems.

POOR WORDING

84. Top-down dynamic programming trades _____memory_____ for _____speed_____.

ex: knapsack

↳ (compared to bottom-up) solves fewer subproblems, use memory more

85. Suppose you wanted to know the number of possible subsets of *N* tasks that cost less than a certain amount in total. What type of algorithm should you use? Explain.

Brute force
Greedy
✓ Backtracking
Branch-and-bound
✓ DP
Divide + Conquer

always works

constraints: min cost or max benefit

optimize

optimize

X

86. What would be the worst case asymptotic time complexity of the algorithm in the previous question?

$\Theta(2^N)$

could be all subsets

87. Suppose you wanted to know the subset of *N* tasks that contains the most tasks while costing less than a certain amount. What type of algorithm should you use? Explain.

Greedy

88. What would be the worst case asymptotic time complexity of the algorithm in the previous question?

$\Theta(N \log N + N) \rightarrow \Theta(N \log N)$

or $\Theta(kN)$ — # of task in solution

89. What is the difference between backtracking and branch-and-bound?

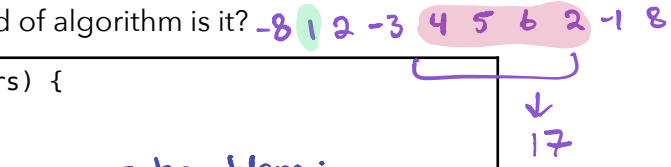
Constraint satisfaction (N queens)	optimization: min cost, max benefit
any single solution or all solutions	find the optimal solution

EECS 281, Week 8: Wednesday

90. Implement `countDuplicates`, a function that counts the total number of duplicate numbers in a vector. For example, in a vector with elements {183, 490, 381, 281, 381, 281, 280, 281}, the total number of duplicates is 3 (2 for 281 and 1 for 381).

```
int countDuplicates(const vector<int> &numbers) {
```

91. Write an efficient function to find the sum of contiguous elements in a subarray within a one-dimensional vector of integers which has the largest sum. What is the time complexity? What is the space complexity? What kind of algorithm is it?




```
int findMaxSubarraySum(const vector<int> &numbers) {
```

subproblem:
maxSubArraySum up to i
max so far = 0
i = 0
max up to i = 0 -8 = -8
-8 < max so far
skip
max up to i = 0
i = 1
max up to i = 0 + 1 = 1
1 ≥ max so far
max so far = 1
i = 2
max up to i = 1 + 2 = 3
3 ≥ max so far
max so far = 3
i = 3
max up to i = 3 - 3 = 0
0 < max so far
skip
max up to i = 0

92. State the N-Queens problem. How to solve it?

93. State the Knapsack problem. Then solve it with backtracking and branch-and-bound.

94. Implement isSubsetSum, a function that solves the Subset Sum problem: given a set of *n* integers *S* and a value *W*, determine if there is a subset of integers that sum to *W*.

5 6 7 -1 3

10

```
bool isSubsetSum(const vector<int> &numbers, int sum) {
```

95. Given a text and a wildcard pattern, implement wildcard pattern matching algorithm that finds if wildcard pattern is matched with text. The matching should cover the entire text (not partial text). The wildcard pattern can include the characters ? (matches any single character) and * (matches any sequence of characters, including the empty sequence).

```
Text = "baaabab",
Pattern = "*****ba*****ab", output : true
Pattern = "baaa?ab", output : true
Pattern = "ba*a?", output : true
Pattern = "a*ab", output : false
```

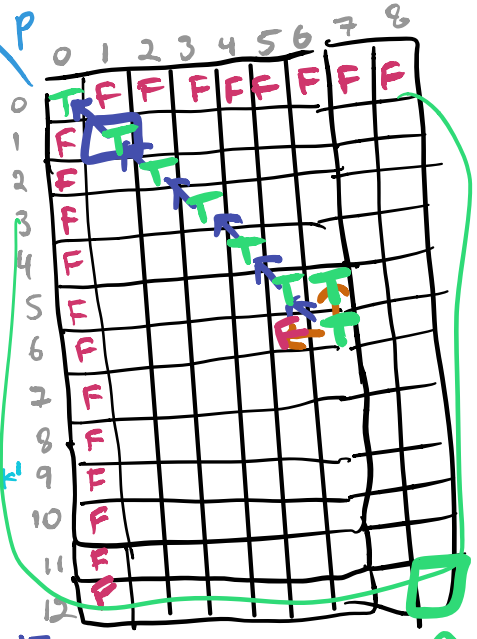
```
bool match(const string &text, const string &pattern) {
```

Subproblem:

M(t,p) do the first t characters in text match the first p characters in pattern?

text = "Hello world!"

pattern = "Hello *!"

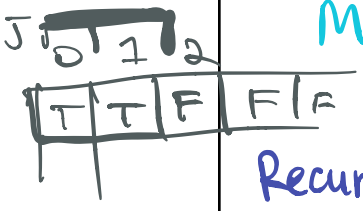


Base case:

M(i,0) where i≠0 = False

M(0,0) = True

M(0,j) = M(0,j-1) if pattern[j-1] is '*'



Recursive def:

if text[t-1] == pattern[p-1]
or pattern[p-1] = '?'

→ M(t-1, p-1)

else if pattern[p-1] == '*' ← ignore or match

→ M(t, p-1) or M(t-1, p)

use as empty string

match

else → FALSE

don't match

96. Given a weighted graph (← could be negative) represented with an adjancy matrix, find the lengths of the shortest paths from node 0 to all other nodes.

(Floyd-Warshall's alg.)