HKN CS 374 Review Session Midterm 2

Recurrences/ Divide Conquer

- Solving Problems by reducing them to smaller cases of themselves
- Base Case- Cases where we can solve it directly (typically the simplest case)
- Inductive Case- Simpler Instances of the same problem
- Prove using a proof by induction
- Divide and Conquer(Divide, Delegate, Combine)
- Remember Important Divide Conquer Algorithms: Binary Search, Merge Sort, Quick Sort, Karatsuba's Algorithm, Linear Selection

Recurrences- Run Time

- Draw out the recursion tree
- Look at level by level sum
- $T(n)=r*T(n/c)+f(n)$
- Three Cases:
  - Decreasing then $T(n)=O(f(n))$
  - Equal then $T(n)=O(f(n)*L)=O(f(n)*\log(n))$
  - Increasing then $T(n)= O(n^{\log c(r)})$
- You can ignore floor and ceiling
- Guess and Check
- Master Theorem

Backtracking

- Sub-family of recursion
- Making a sequence of decision to satisfy some constraint
- Each recursive call corresponds to one of these decisions
- For each call need to pass the data yet to be processed as well as the decisions also made
- A lot of these problems will also have DP solutions
- Know Common Backtracking Problems: SubsetSum, Longest Increasing Subsequence, N Queens, etc.

Dynamic Programming

- Backtracking can be slow
- Use memoization to “remember” previous
- You need to come up with a recurrence first
- Think bottom up
    - Good data structure
    - Sub problems
    - Evaluation Order

Graphs

- Graph is a set (V,E)
    - V is non empty set of vertices
    - E is (possibly empty) set of edges
- Two Type:
    - Undirected (Unordered Edges)
    - Directed (Ordered Edges)
- Walk vs Path
    - Walk sequence of vertices in the graph
    - Path, Walk with no repeating vertices

Graph Search

- Search is a test of reachability
- Every search algorithm puts edges into a “bag” and takes them out one by one
- Depth First Search where bag is a stack (Runtime: O(V+E))
- Breadth First Search where bag is queue (Runtime: O(V+E))
- Best First Search where bag is a priority Queue (Runtime: O(V+ElogE))

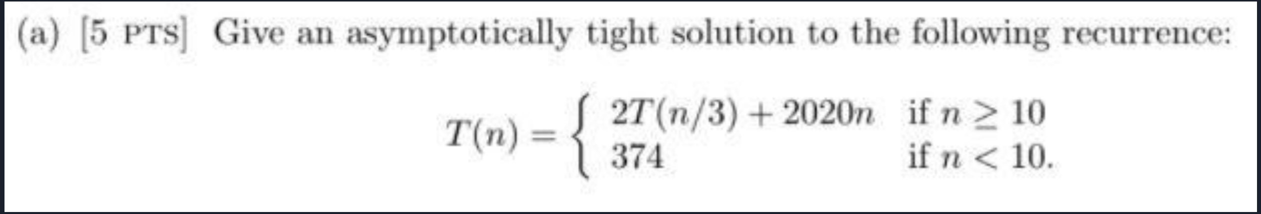
Directed Acyclic Graphs

- Directed Graph with no Cycles
- One Source and One Sink
- Topological Sort is a ordering a graph where all edges point left to right
- Topological Sort is a modified DFS so linear runtime
- Strongly Connected if any two vertices can reach each other
- Strongly Connected Component Graph can be done in linear time

Shortest Path

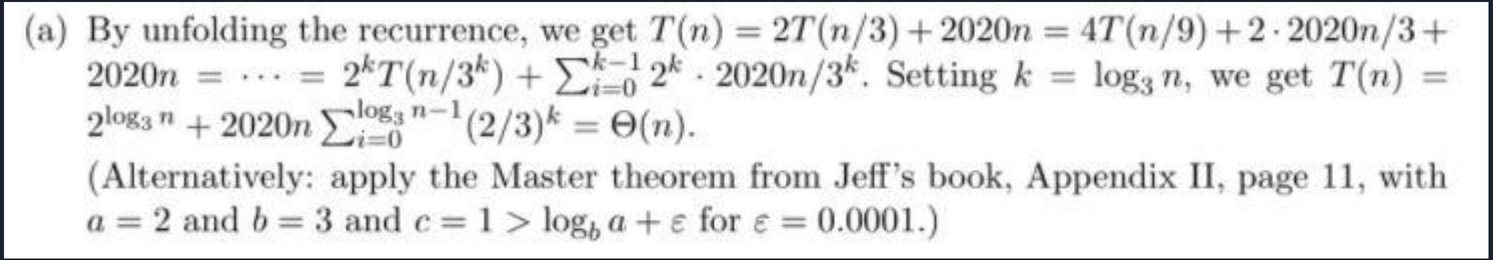
- Connect shortest path between nodes
- Generic Shortest Path Algorithm relax all edges till no more tense edges
- Unweighted Graphs: BFS (Linear Time)
- DAG: DFS (Linear Time)
- No Negative Edges : Dijkstra’s O(E+Vlog(V))
- Else: Bellman-Ford O(EV)
- Can use Bellman-Ford to detect negative Cycles

Exam Questions

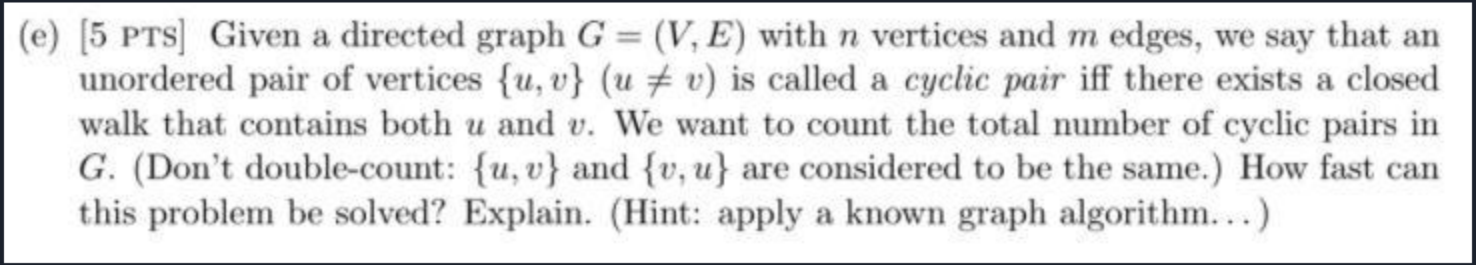


(a) [5 PTS] Give an asymptotically tight solution to the following recurrence:

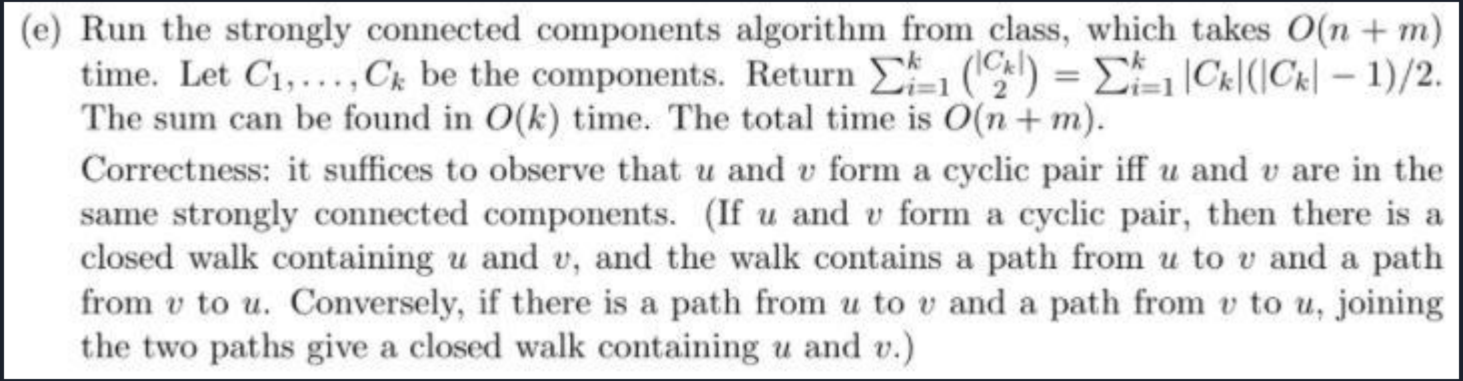
$$T(n) = \begin{cases} 2T(n/3) + 2020n & \text{if } n \geq 10 \\ 374 & \text{if } n < 10. \end{cases}$$



(a) By unfolding the recurrence, we get $T(n) = 2T(n/3) + 2020n = 4T(n/9) + 2 \cdot 2020n/3 + 2020n = \cdots = 2^k T(n/3^k) + \sum_{i=0}^{k-1} 2^k \cdot 2020n/3^k$. Setting $k = \log_3 n$, we get $T(n) = 2^{\log_3 n} + 2020n \sum_{i=0}^{\log_3 n - 1} (2/3)^k = \Theta(n)$.
(Alternatively: apply the Master theorem from Jeff's book, Appendix II, page 11, with $a = 2$ and $b = 3$ and $c = 1 > \log_b a + \varepsilon$ for $\varepsilon = 0.0001$.)

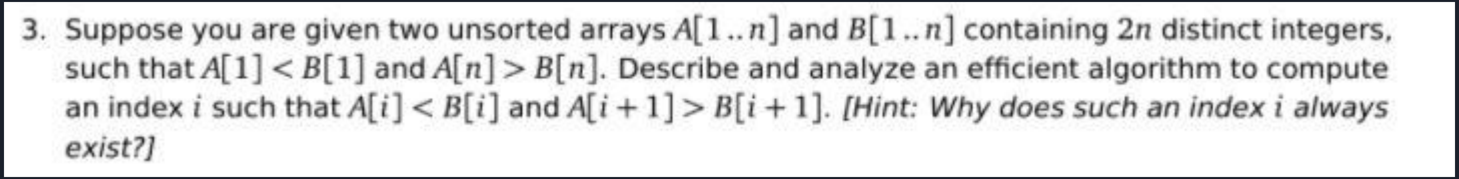


(e) [5 PTS] Given a directed graph $G = (V, E)$ with n vertices and m edges, we say that an unordered pair of vertices $\{u, v\}$ $(u \neq v)$ is called a *cyclic pair* iff there exists a closed walk that contains both u and v. We want to count the total number of cyclic pairs in G. (Don't double-count: $\{u, v\}$ and $\{v, u\}$ are considered to be the same.) How fast can this problem be solved? Explain. (Hint: apply a known graph algorithm...)

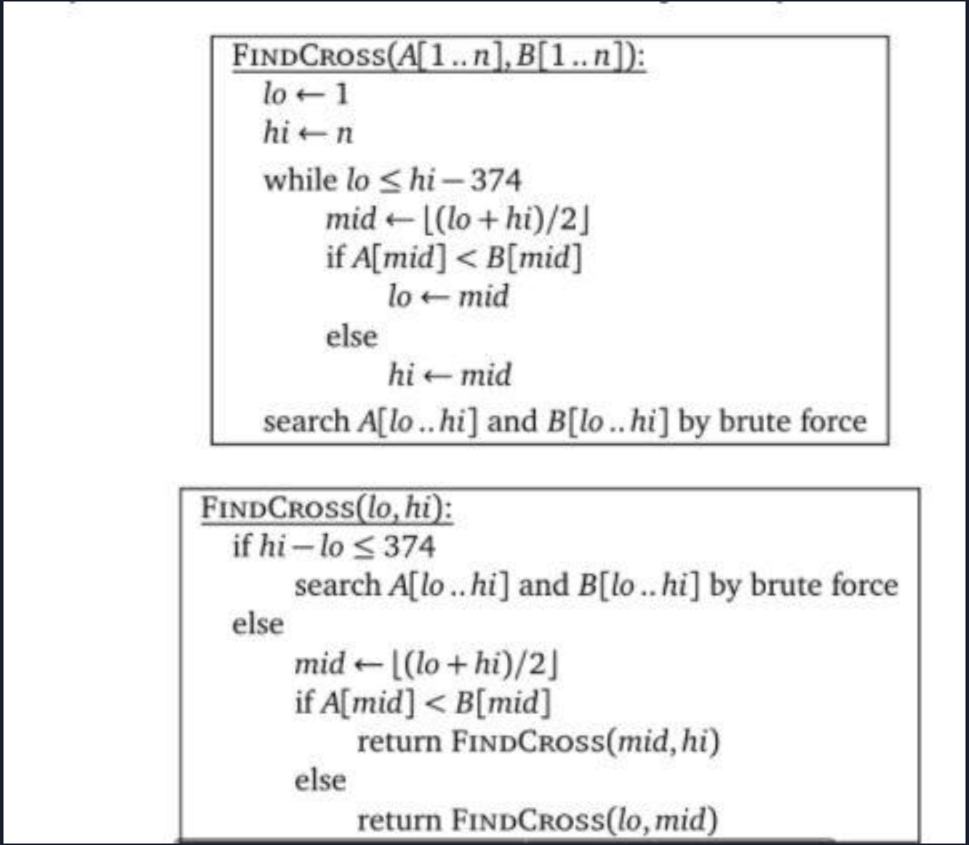


(e) Run the strongly connected components algorithm from class, which takes $O(n+m)$ time. Let $C_1, \ldots, C_k$ be the components. Return $\sum_{i=1}^{k} \binom{|C_k|}{2} = \sum_{i=1}^{k} |C_k|(|C_k|-1)/2$. The sum can be found in $O(k)$ time. The total time is $O(n+m)$.

Correctness: it suffices to observe that u and v form a cyclic pair iff u and v are in the same strongly connected components. (If u and v form a cyclic pair, then there is a closed walk containing u and v, and the walk contains a path from u to v and a path from v to u. Conversely, if there is a path from u to v and a path from v to u, joining the two paths give a closed walk containing u and v.)

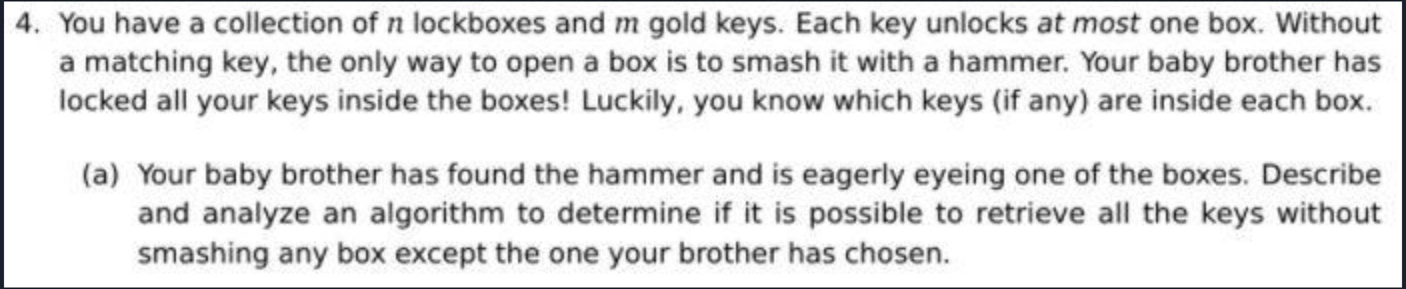


3. Suppose you are given two unsorted arrays $A[1..n]$ and $B[1..n]$ containing $2n$ distinct integers, such that $A[1] < B[1]$ and $A[n] > B[n]$. Describe and analyze an efficient algorithm to compute an index i such that $A[i] < B[i]$ and $A[i+1] > B[i+1]$. *[Hint: Why does such an index i always exist?]*



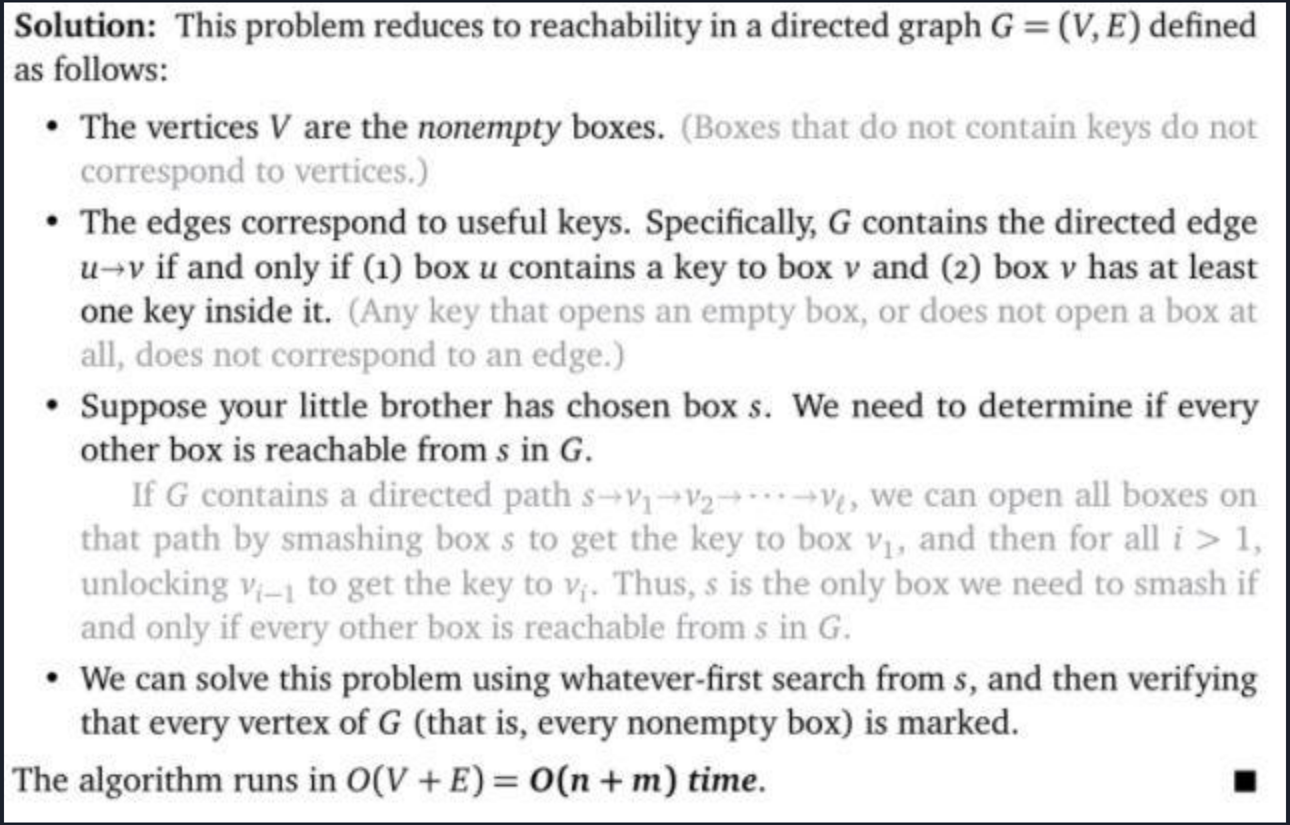
```
FindCross($A[1..n], B[1..n]$):
    $lo \leftarrow 1$
    $hi \leftarrow n$
    while $lo \leq hi - 374$
        $mid \leftarrow \lfloor (lo + hi)/2 \rfloor$
        if $A[mid] < B[mid]$
            $lo \leftarrow mid$
        else
            $hi \leftarrow mid$
    search $A[lo..hi]$ and $B[lo..hi]$ by brute force
```

```
FindCross($lo, hi$):
    if $hi - lo \leq 374$
        search $A[lo..hi]$ and $B[lo..hi]$ by brute force
    else
        $mid \leftarrow \lfloor (lo + hi)/2 \rfloor$
        if $A[mid] < B[mid]$
            return FindCross($mid, hi$)
        else
            return FindCross($lo, mid$)
```



4. You have a collection of n lockboxes and m gold keys. Each key unlocks *at most* one box. Without a matching key, the only way to open a box is to smash it with a hammer. Your baby brother has locked all your keys inside the boxes! Luckily, you know which keys (if any) are inside each box.

   (a) Your baby brother has found the hammer and is eagerly eyeing one of the boxes. Describe and analyze an algorithm to determine if it is possible to retrieve all the keys without smashing any box except the one your brother has chosen.

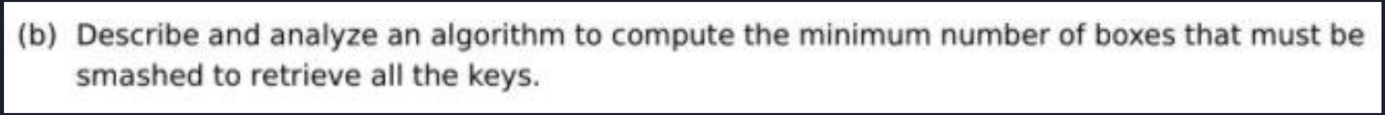


Solution: This problem reduces to reachability in a directed graph $G = (V, E)$ defined as follows:

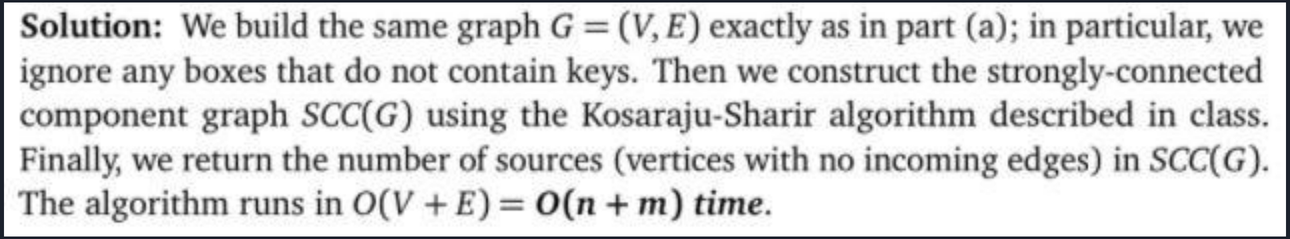
- The vertices V are the *nonempty* boxes. (Boxes that do not contain keys do not correspond to vertices.)
- The edges correspond to useful keys. Specifically, G contains the directed edge $u \rightarrow v$ if and only if (1) box u contains a key to box v and (2) box v has at least one key inside it. (Any key that opens an empty box, or does not open a box at all, does not correspond to an edge.)
- Suppose your little brother has chosen box s. We need to determine if every other box is reachable from s in G.

  If G contains a directed path $s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\ell$, we can open all boxes on that path by smashing box s to get the key to box v_1, and then for all $i > 1$, unlocking v_{i-1} to get the key to v_i. Thus, s is the only box we need to smash if and only if every other box is reachable from s in G.
- We can solve this problem using whatever-first search from s, and then verifying that every vertex of G (that is, every nonempty box) is marked.

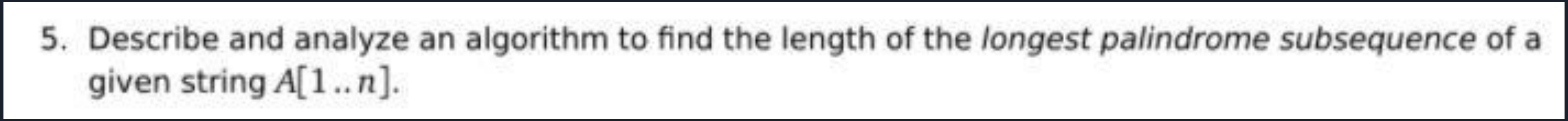
The algorithm runs in $O(V + E) = \boldsymbol{O(n + m)}$ ***time***. ■

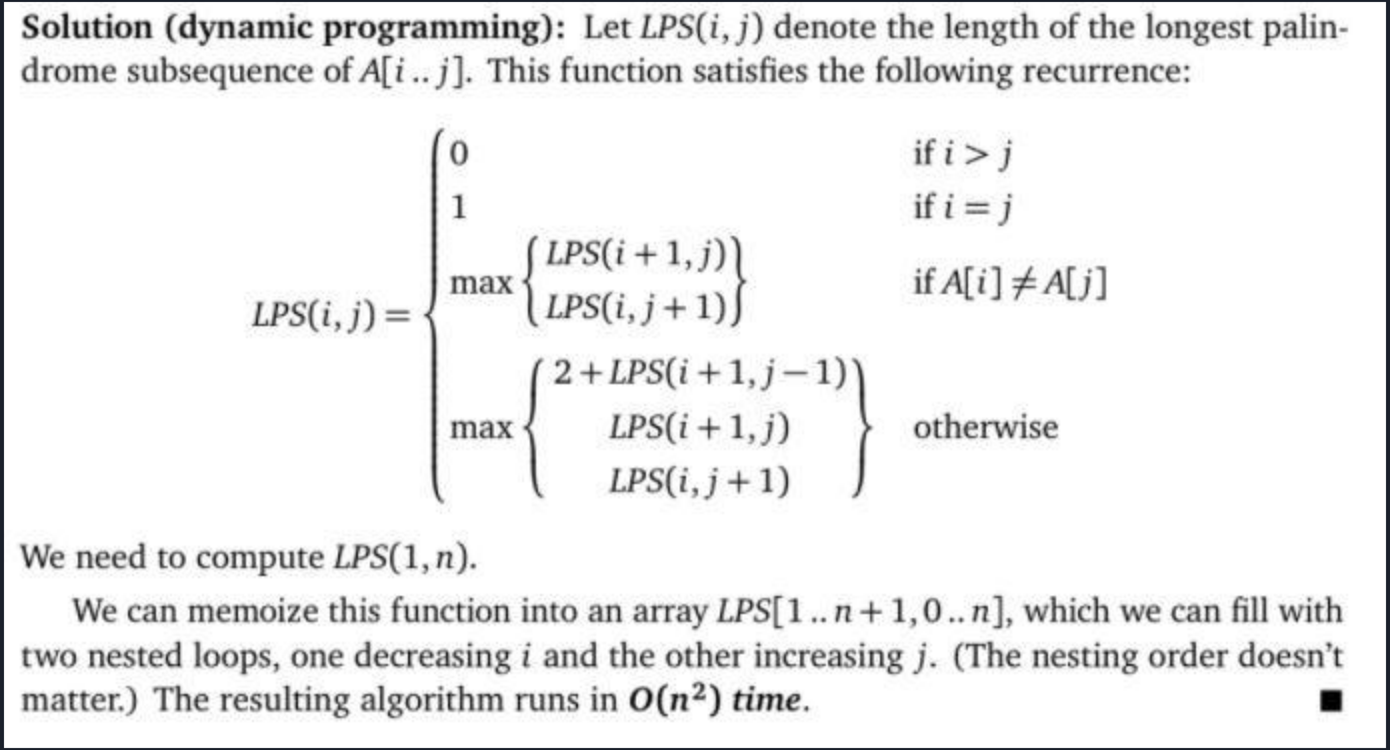


(b) Describe and analyze an algorithm to compute the minimum number of boxes that must be smashed to retrieve all the keys.



Solution: We build the same graph $G = (V, E)$ exactly as in part (a); in particular, we ignore any boxes that do not contain keys. Then we construct the strongly-connected component graph $SCC(G)$ using the Kosaraju-Sharir algorithm described in class. Finally, we return the number of sources (vertices with no incoming edges) in $SCC(G)$. The algorithm runs in $O(V + E) = \boldsymbol{O(n+m)}$ ***time***.


5. Describe and analyze an algorithm to find the length of the *longest palindrome subsequence* of a given string $A[1..n]$.

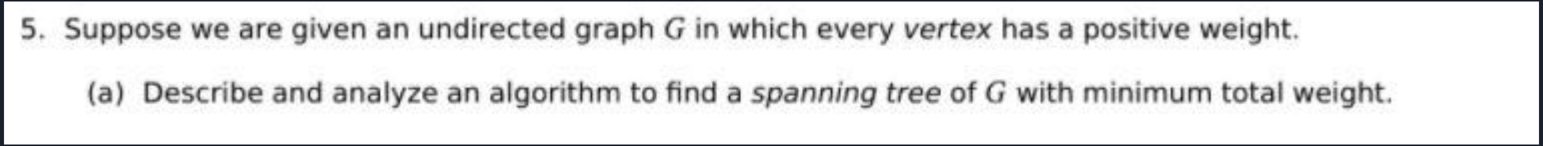


Solution (dynamic programming): Let $LPS(i,j)$ denote the length of the longest palindrome subsequence of $A[i\,..\,j]$. This function satisfies the following recurrence:

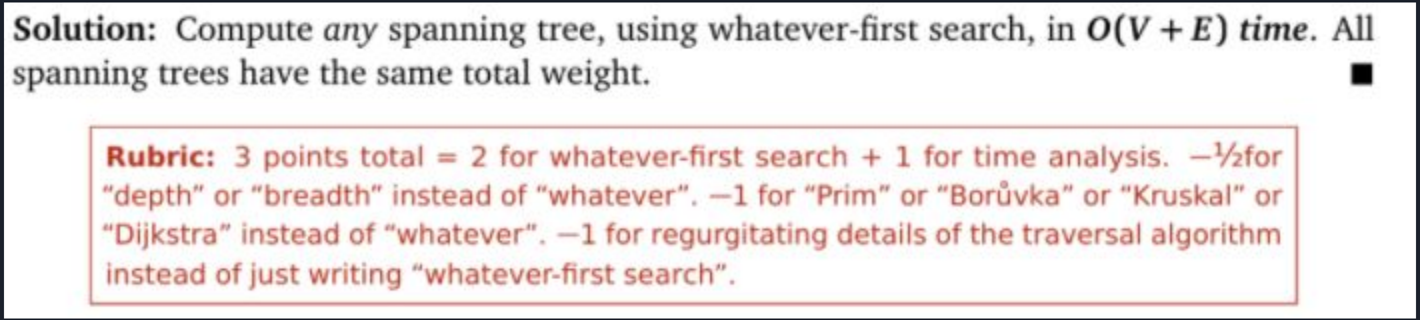
$$LPS(i,j) = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ \max\left\{\begin{array}{l} LPS(i+1,j) \\ LPS(i,j+1) \end{array}\right\} & \text{if } A[i] \neq A[j] \\ \max\left\{\begin{array}{c} 2 + LPS(i+1,j-1) \\ LPS(i+1,j) \\ LPS(i,j+1) \end{array}\right\} & \text{otherwise} \end{cases}$$

We need to compute $LPS(1,n)$.

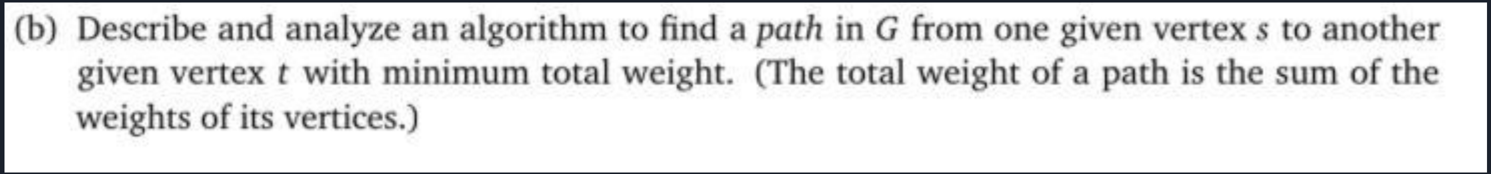
We can memoize this function into an array $LPS[1\,..\,n+1, 0\,..\,n]$, which we can fill with two nested loops, one decreasing i and the other increasing j. (The nesting order doesn't matter.) The resulting algorithm runs in ***$O(n^2)$ time***. ■


5. Suppose we are given an undirected graph G in which every *vertex* has a positive weight.

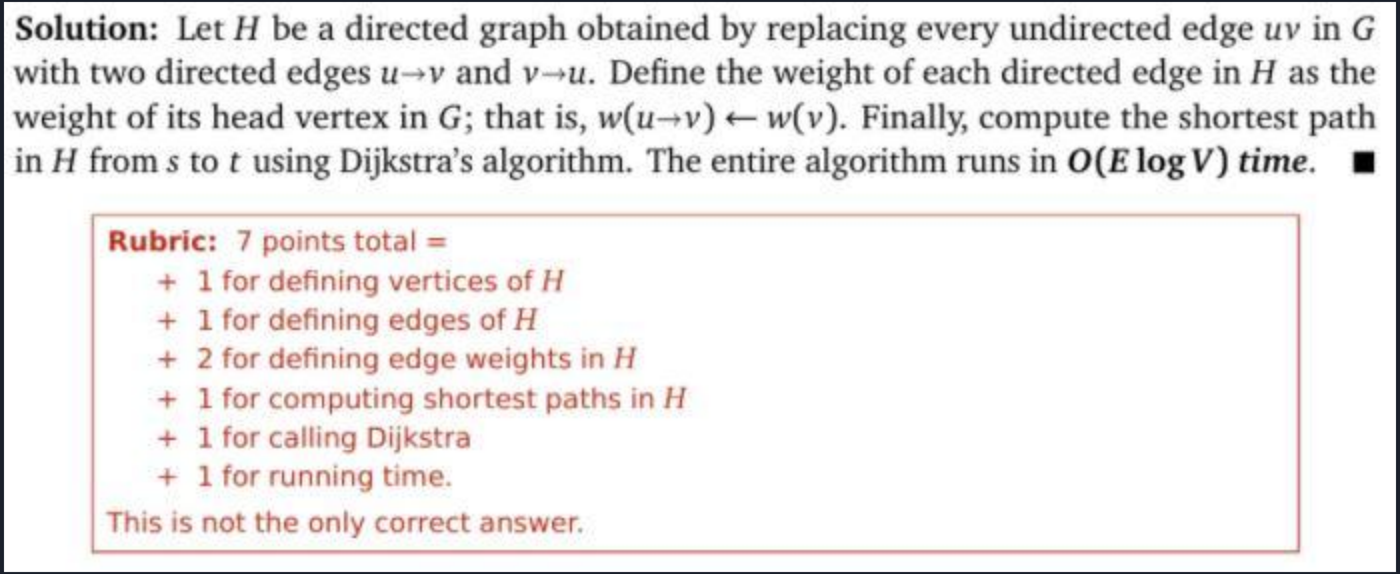
   (a) Describe and analyze an algorithm to find a *spanning tree* of G with minimum total weight.


Solution: Compute *any* spanning tree, using whatever-first search, in $O(V+E)$ *time*. All spanning trees have the same total weight. ∎

Rubric: 3 points total = 2 for whatever-first search + 1 for time analysis. −½for "depth" or "breadth" instead of "whatever". −1 for "Prim" or "Borůvka" or "Kruskal" or "Dijkstra" instead of "whatever". −1 for regurgitating details of the traversal algorithm instead of just writing "whatever-first search".



(b) Describe and analyze an algorithm to find a *path* in G from one given vertex s to another given vertex t with minimum total weight. (The total weight of a path is the sum of the weights of its vertices.)



Solution: Let H be a directed graph obtained by replacing every undirected edge uv in G with two directed edges $u{\rightarrow}v$ and $v{\rightarrow}u$. Define the weight of each directed edge in H as the weight of its head vertex in G; that is, $w(u{\rightarrow}v) \leftarrow w(v)$. Finally, compute the shortest path in H from s to t using Dijkstra's algorithm. The entire algorithm runs in $\boldsymbol{O(E \log V)}$ ***time***. ■

Rubric: 7 points total =

+ 1 for defining vertices of H
+ 1 for defining edges of H
+ 2 for defining edge weights in H
+ 1 for computing shortest paths in H
+ 1 for calling Dijkstra
+ 1 for running time.

This is not the only correct answer.