Discussion 7- Midterm Review Sol.

1 Potential Method (WN22 MCQ1)

Consider the following code. Note that the variables x and y are *real* numbers.

Input: $x > y > 0$ are *real* numbers // Hint: This actually matters...

```
1: function Foo376(x, y)
2:     if x ≤ 0 then return 1;
3:     z ← Foo376(x − log y, y)
4:     return (z + 1)
```

Which of the following is a valid potential function for the algorithm Foo376 (above)?

- ○ $s = x + y$
- ○ $s = e^y - x$
- ○ $s = x$
- ✓ None of the above

$y < 1 \Rightarrow x - \log y$ strictly increasing
$\Rightarrow$ will never reach $x \leq 0$
$\Rightarrow$ will not halt.

2 Divide and Conquer (WS7 Review 2)

Describe an efficient divide and conquer algorithm to computer the value of 376^k. For simplicity, you may assume that k is a power of 2. Your solution should include a correctness and runtime analysis in terms of n (assuming multiplication takes constant time).

```
Func(k):
If k=1 then return 376
Return Func(k/2) · Func(k/2)
```

same recursive call — only have to call once and multiply by itself.

Correctness

Base: $k=1$: $376^1 = 376$

Inductive: suppose $Func(k) = 376^k$

Next power of 2: $2k$.

By construction,

$Func(2k) = Func(k) \cdot Func(k)$

$= 376^k \cdot 376^k$

$= 376^{2k}$ Master's Thm

Runtime $T(k) = 1 \cdot T(\frac{k}{2}) + O(1) = O(\log k)$

Input size: $n = \log k \Rightarrow$ Runtime $= O(n)$

3 Dynamic Programming (WS7 Review 3)

Give a recurrence relation (including base cases) that is suitable for dynamic programming solutions to the following problem. You do not need to prove your correctness.

Can jump.

LONGEST-ARITHMETIC-SUBSEQUENCE(A, d): Given an array of integers A and a difference d, return the length of the longest arithmetic subsequence in A with difference d. That is, return the longest subsequence S such that $S[i+1] - S[i] = d$ for each i.

Step 0: Determine dimension of DP

1D

Reduce in 1 direction.

Step 1: Define recurrence in English.

Let $L(i)$ = length of $LAS(A[1, \ldots, i], d)$ ending at i

Step 2: Base case

$i = 1 \Rightarrow$ length $= 1$

Step 3a: Sub-solution:

[sub] Reduce from $A[1, \ldots, i]$ to $A[1, \ldots, i-1]$

[solution] Find best LAS in $A[1, \ldots, i-1]$, say j such that $A[i] - A[j] = d$

Step 3b: Optimization

maximization problem: use max

"best"

over: All $j < i$ s.t. $A[i] - A[j] = d$

Obj. func: $L(j)$

Recurrence Relation:

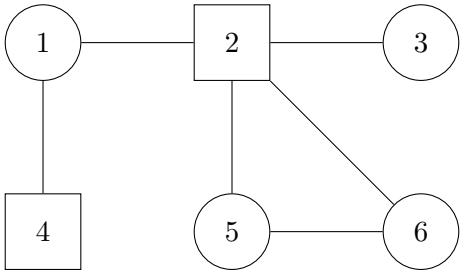
$$L(i) = \begin{cases} 1 & \text{, if } i = 1 \\ 1 + \max_{\substack{j < i : \\ A[i] - A[j] = d}} L(j) & \text{, otherwise} \end{cases}$$

4 Greedy Algorithms (WN23 Short3)

A *dominating set* S in a graph G is a set of vertices for which every vertex of G either is in S, or is adjacent to some vertex in S.

We are interested in a *smallest* dominating set of a given graph, i.e., one that has the fewest possible vertices. (There may be more than one smallest dominating set.)

For example, the following graph has a smallest dominating set $S^* = \{2, 4\}$: every vertex other than 2 and 4 is adjacent to 2 or 4 (or both), and there is no dominating set consisting of a single vertex.



Consider the following greedy algorithm for finding a dominating set in a graph.

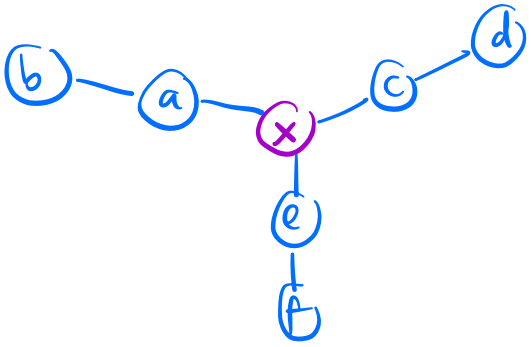
```
1: function GreedyDS(G)
2:     S ← ∅
3:     while G has at least one vertex do
4:         Select any vertex v in G that has largest degree (i.e., the most neighbors)
5:         Add v to S
6:         Remove v and all its neighbors, including all incident edges, from G
7:     return S
```

Give a small graph G on which the algorithm **might not return a *smallest* dominating set**.

Specifically, **give a sequence of vertices that the algorithm might choose** to make up its final output set, and **give an optimal dominating set of G that is smaller** than this output set.

Idea: Have a vertex that will be selected by line 4 (largest deg) but not in opt. sol.

Example:



optimal: (3 vertices)
{a, c, e}

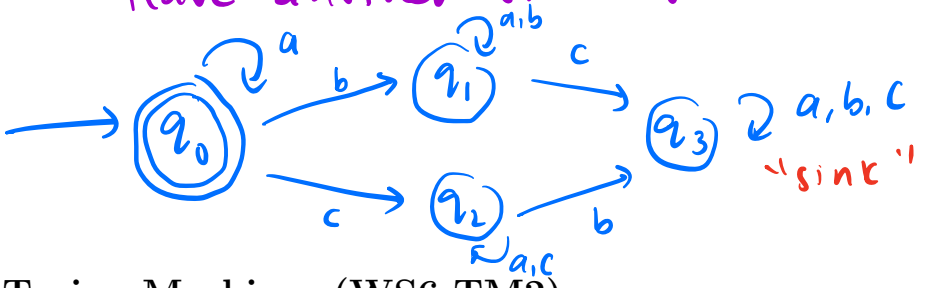
Greedy (4 vertices)
{x} ∪ {a ∨ b} ∪ {c ∨ d} ∪ {e ∨ f}

5 DFAs (WN22 9b)

Let $L \subseteq \{a, b, c\}^*$ be the set of all strings over the alphabet $\{a, b, c\}$ **except** those that contain both at least one b and at least one c. For example, aa, aba, cca are all in L, but abc is not as it contains both a b and a c.

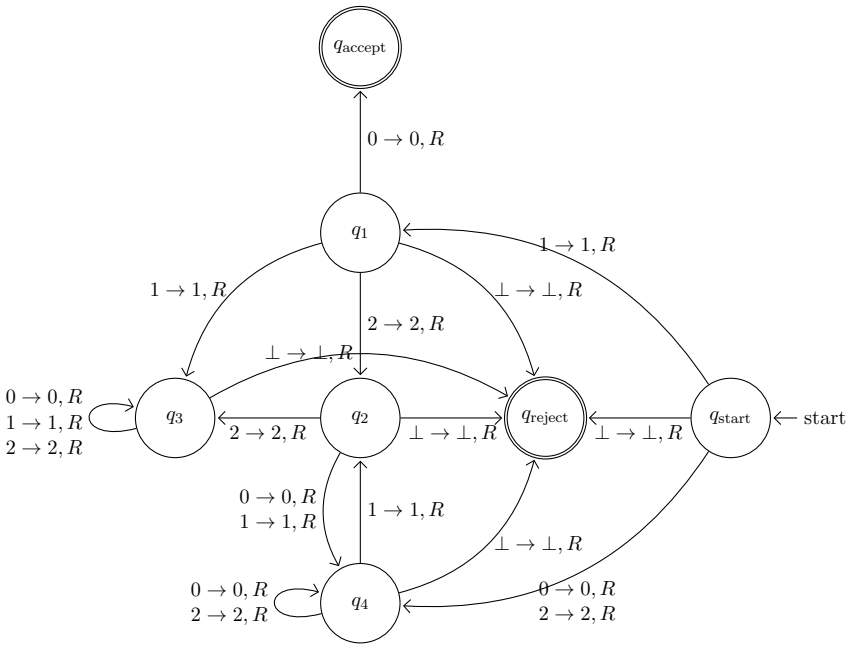
Write a DFA over the alphabet $\{a, b, c\}$ that decides the language L.

Idea: Have a state that keep track of "seen b" → Reject if see 'c'
Have another state for "seen c" → Reject if see 'b'



6 Turing Machines (WS6 TM2)

Consider the Turing Machine whose state diagram is given below:



Which of the following statements is true about this Turing Machine?

- ◯ It accepts all strings that contain the substring "10."
- ◯ It loops on any input string that contains only 2s.
- ◯ It loops on any string that contains the substring "11" until it reaches $\perp$.
- ⊘ None of the above

"210" → Reject
"2" → Reject
$\perp \notin \Sigma^*$!
Input string cannot contain '$\perp$' !