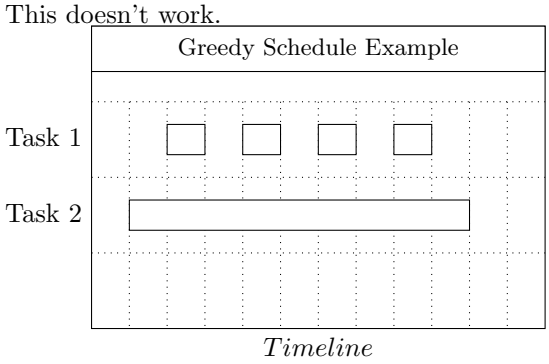
Problem 1
[50 pts] The greedy algorithm we described for the class scheduling problem is not the only greedy strategy we could have tried. For each of the following alternative greedy strategies, either prove that the resulting algorithm always constructs an optimal schedule, or describe a small input example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control).

1. Choose the course x that ends last, discard classes that conflict with x, and recurse.
2. Choose the course x that starts first, discard all classes that conflict with x, and recurse.
3. Choose the course x that starts last, discard all classes that conflict with x, and recurse.
4. Choose the course x with shortest duration, discard all classes that conflict with x, and recurse.
5. If no classes conflict, choose them all. Otherwise, discard the course with longest duration and recurse.
6. If no classes conflict, choose them all. Otherwise, discard a course that conflicts with the most other courses and recurse.
7. If any course x completely contains another course, discard x and recurse. Otherwise, choose the course y that ends last, discard all classes that conflict with y, and recurse.

Solution:

1. This doesn't work.

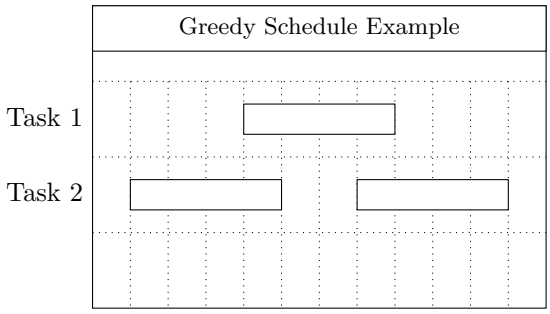


Given the input above, the greedy algorithm chooses the single long course, but the optimal schedule contains all the short courses.

2. This doesn't work, same as part (1).

3. This greedy algorithm works.
Let x be the course that starts last. Let S be any schedule that does not contain x, and let z be the last course in S. Because x starts last, we have $S[z] < S[x]$. Thus $F[i] < S[z] < S[x]$ for every other class i in S, which implies that $S' = S - z + x$ is still a valid schedule, containing the same number of classes as S. In particular, if S is an optimal schedule, then S' is an optimal schedule containing x.

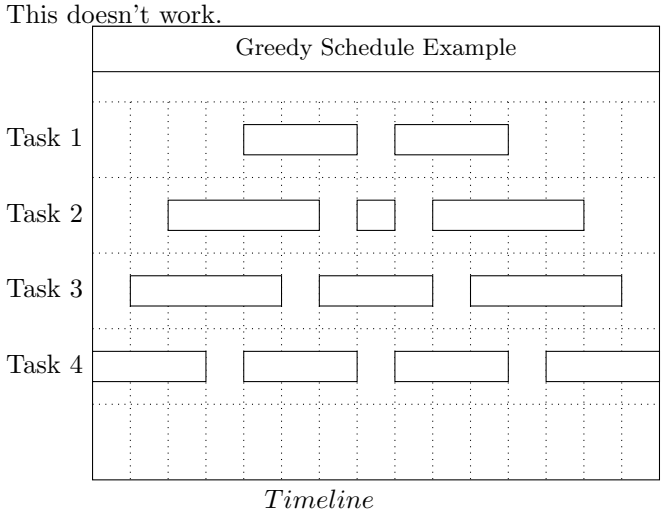
4. This doesn't work.



Given the input above, this greedy algorithm chooses the single course in the middle, but the optimal schedule contains the other two courses.

5. This doesn't work, same as part (4).

6. This doesn't work.

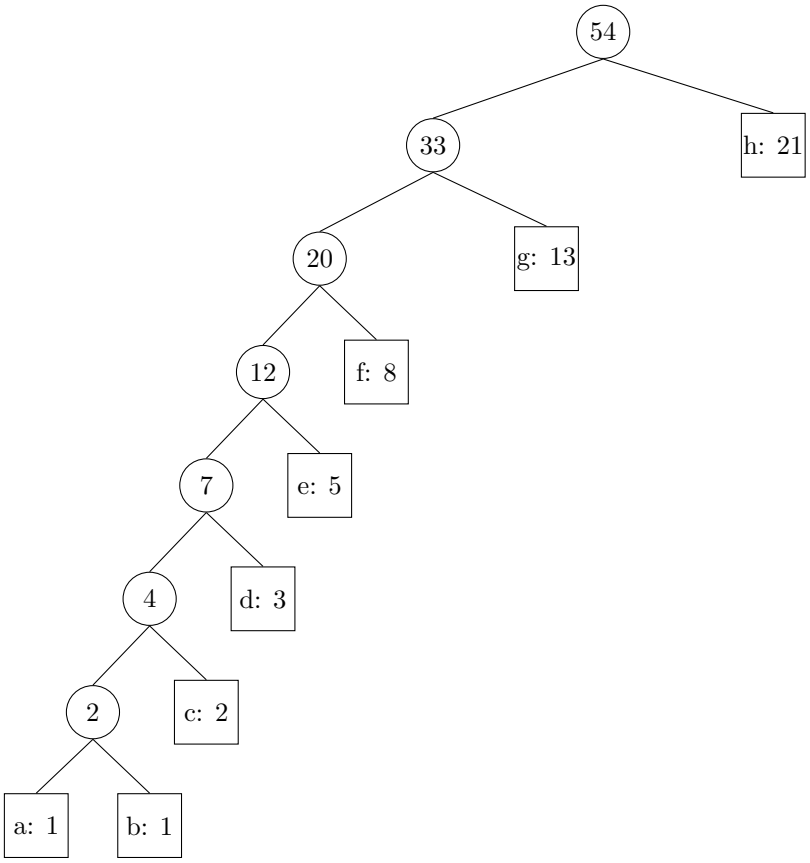


Given the input above, this greedy algorithm would discard one of the courses in the middle of the bottom row, which has only five conflicts, and thus would return a schedule containing only three courses. But the optimal schedule contains four courses.

7. This greedy algorithm works.
Let S be any valid schedule that contains x. Then $S - x + y$ is another valid schedule of the same size. If no course contains any other course, then the class that ends last is also the class that starts last.

Problem 2
[15 pts] Construct a Huffman code for the following set of characters and their frequencies that are based on the first 8 Fibonacci numbers:
a : 1; b : 1; c : 2; d : 3; e : 5; f : 8; g : 13; h : 21.

Solution:

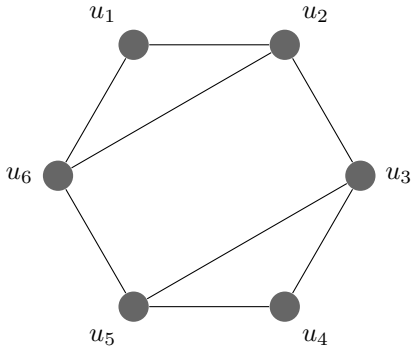


As we can see the tree is one long limb with leaves n = hanging off. This is true for Fibonacci weights in general, because the Fibonacci the recurrence is implies that

$$F_{n+2} = \sum_{i=0}^{n} F_i + 1 \tag{1}$$

Problem 3
[20 pts] An independent set in a graph G is a set of vertices I in G such that no two vertices in I are adjacent (neighbors). The maximum independent set problem is, given a graph G, to compute an independent set in G of maximum size (maximum number of vertices). Pinocchio claims that he has a greedy algorithm that solves the maximum independent set problem. Pinocchio's algorithm works as follows. The algorithm initializes the set I to the empty set, and repeats the following steps: Pick a vertex in the graph with the minimum degree, add it to the set I, and remove it and all the vertices adjacent to it from the graph. The algorithm stops when the graph is empty. Does Pinocchio's greedy algorithm always produces a maximum independent set? Prove your answer (if it does, give a proof; if it does not, give a counter example, that is, a graph on which Pinocchio's algorithm does not produce a maximum independent set).

Solution:
Pinocchio's algorithm is not guaranteed to find a maximum independent set.



It is not no hard to tell the maximum independent set would be $[u_1, u_3, u_5]$ or $[u_4, u_2, u_6]$. Follow by Pinocchio's algorithm we have to either add u_1 or u_4 into set I since they both have least degree of 2. Assume we start at vertex u_1 we have to remove u_6 and u_2, this result to a triangle graph among u_3, u_4 and u_5, due to Pinocchio's algorithm we can only keep one of them into set I. Consequently, Pinocchio's algorithm result $[u_1, u_3]$, $[u_1, u_4]$, $[u_1, u_5]$ are contradict to our actual result.

Problem 4
[15 pts] Professor Gekko has always dreamed of inline skating across North Dakota. He plans to cross the state on highway U.S. 2, which runs from Grand Forks, on the eastern border with Minnesota, to Williston, near the western border with Montana.
The professor can carry two liters of water, and he can skate m miles before running out of water. (Because North Dakota is relatively flat, the professor does not have to worry about drinking water at a greater rate on uphill sections than on flat or downhill sections.) The professor will start in Grand Forks with two full liters of water. His official North Dakota state map shows all the places along U.S. 2 at which he can refill his water and the distances between these locations.
The professor's goal is to minimize the number of water stops along his route across the state. Give an efficient algorithm by which he can determine which water stops he should make. Prove that your strategy yields an optimal solution, and give its running time.

Solution:

```
def MinWaterStop(availableSpot):
    distance ← 0;
    for each spot in order do
        if distance + dstanceToNextSpot > 2 mile then
            markCurrentSpot();
            numberOfWaterSpot++;
            distance = 0;
        end
        else
            distance += distanceToNextSpot;
        end
    end
    return markedSpot;
```

By following the algorithm shown above, the professor will not make a water stop if the distance from last refill to the upcoming refill is less than 2 miles. He will stop for refill only when he cannot reach the next spot at any point. That is, it holds suboptimal property all along, with m minimal stops in total. Assuming that there is a better solution with m - 1 stops, it is impossible to finish the section after the omitted spot. Therefore, the algorithm is optimal. And its running time is $O(n)$ where n is the number of available spots.