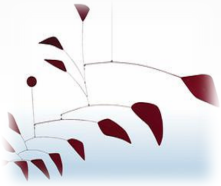
EECS 3101 - Design and Analysis of Algorithms



Shahin Kamali

Tutorial

York University

Picture is from the cover of the textbook CLRS.

Partition Problem

- Given an array A of n distinct positive integers, we want to investigate whether items in A can be divided into two subsets A_1 and A_2 such that the sum of the numbers in A_1 equals the sum of the numbers in A_2.
- For example, for $A = [1, 2, 4, 5, 10, 18]$, the answer is 'true' because items in A can be divided into $A_1 = \{2, 18\}$ and $A_2 = \{1, 4, 5, 10\}$, and numbers in both A_1 and A_2 sum to 20. On the other hand, for subset $A' = [2, 3, 4, 5, 7, 19]$, the answer is 'false.'
- Use a DP approach to answer the Partition problem for any array A of size n. Assume indices start at 1.

Partition Problem

- Let S denote the total sum of the items. We consider the decision problem that asks whether the first i items of A contains a subset of size M. There is a 'yes' or 'no' in each cell of the DP table T. The final solution is stored in $P[n, S/2]$.

Partition Problem

- Let S denote the total sum of the items. We consider the decision problem that asks whether the first i items of A contains a subset of size M. There is a 'yes' or 'no' in each cell of the DP table T. The final solution is stored in $P[n, S/2]$.
- In the base case, $P[i, 0] = \mathit{true}$, $P[0, M] = \mathit{false}$ if $M \neq 0$.

Partition Problem

- Let S denote the total sum of the items. We consider the decision problem that asks whether the first i items of A contains a subset of size M. There is a 'yes' or 'no' in each cell of the DP table T. The final solution is stored in $P[n, S/2]$.
- In the base case, $P[i, 0] = \mathit{true}$, $P[0, M] = \mathit{false}$ if $M \neq 0$.
- We can write $P[i, M] = P[i-1, M] \vee P[i-1, M - A[i]]$.

Subset Sum Problem

- In the **subset sum problem**, the goal is to find a subset of S of A whose sum is a certain target number t given as input.
- The partition problem is the special case in which t is half the sum of S.

Subset Sum Problem

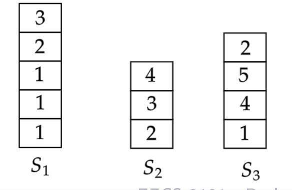
- In the **subset sum problem**, the goal is to find a subset of S of A whose sum is a certain target number t given as input.
- The partition problem is the special case in which t is half the sum of S.
- The same approach that we used can be applied to solve subset sum problem.

3-Partition Problem

- Given an array A of n distinct positive integers, we want to investigate whether items in A can be divided into **three** subsets A_1 and A_2 such that the sum of the numbers in A_1 equals the sum of the numbers in A_2 and also the sume of the numbers in A_3.
- For example, for $A = [1, 2, 3, 5, 8, 10, 11, 17]$, the answer is 'true' because items in A can be divided into $A_1 = \{2, 17\}$, $A_2 = \{1, 3, 5, 10\}$, and $A_3 = \{8, 11\}$ and numbers in A_1, A_2 and A_3 sum to 19.
- Use a DP approach to answer the 3-Partition problem for any array A of size n. Assume indices start at 1.

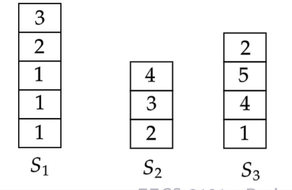
Three Stack Question

- Given three stacks of the positive numbers, the task is to find the possible equal maximum sum of the stacks with the removal of top elements allowed.
- Stacks are represented as an array of lengths n_1, n_2, n_3, and the last index of the array represent the top element of the stack.
  - E.g., $S_1 = [1, 1, 1, 2, 3]$, $S_2 = [2, 3, 4]$, and $S_3 = [1, 4, 5, 2]$.



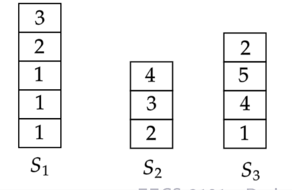
Three Stack Question

- Given three stacks of the positive numbers, the task is to find the possible equal maximum sum of the stacks with the removal of top elements allowed.
- Stacks are represented as an array of lengths n_1, n_2, n_3, and the last index of the array represent the top element of the stack.
  - E.g., $S_1 = [1, 1, 1, 2, 3]$, $S_2 = [2, 3, 4]$, and $S_3 = [1, 4, 5, 2]$.
  - Popping all elements result in an equal sum of 0.



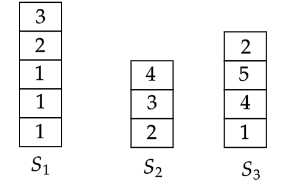
Three Stack Question

- Given three stacks of the positive numbers, the task is to find the possible equal maximum sum of the stacks with the removal of top elements allowed.
- Stacks are represented as an array of lengths n_1, n_2, n_3, and the last index of the array represent the top element of the stack.
  - E.g., $S_1 = [1, 1, 1, 2, 3]$, $S_2 = [2, 3, 4]$, and $S_3 = [1, 4, 5, 2]$.
  - Popping all elements result in an equal sum of 0.
  - A better solution is popping one element from S_1, one element from S_2 and two elements from S_3 results in the three stacks having an equal value of 5.



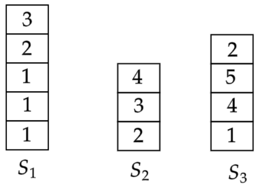
Three Stack Question

- Optimal subproblem property holds.
  - Let $Val[i, j, k]$ denote the optimal sum for the input formed by the first i items of S_1, the first j items of S_2, and the first k items of S_3. We want $Val[n_1, n_2, n_3]$.



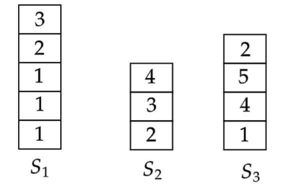
Three Stack Question

- Optimal subproblem property holds.
  - Let $Val[i, j, k]$ denote the optimal sum for the input formed by the first i items of S_1, the first j items of S_2, and the first k items of S_3. We want $Val[n_1, n_2, n_3]$.
  - To set $Val[i, j, k]$, we check if total sum of the three stacks (up to indices i, j, k respectively) is equal. If it is, return the equal sum.
    - E.g., here the three sums $S_1[1..4]$, $S_2[2..3]$, and $S[1..2]$ are equal to 5. Thus, $Val[4, 2, 2] = 5$.



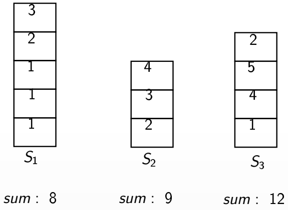
Three Stack Question

- Optimal subproblem property holds.
  - Let $Val[i, j, k]$ denote the optimal sum for the input formed by the first i items of S_1, the first j items of S_2, and the first k items of S_3. We want $Val[n_1, n_2, n_3]$.
  - To set $Val[i, j, k]$, we check if total sum of the three stacks (up to indices i, j, k respectively) is equal. If it is, return the equal sum.
    - E.g., here the three sums $S_1[1..4]$, $S_2[2..3]$, and $S[1..2]$ are equal to 5. Thus, $Val[4, 2, 2] = 5$.
  - If the three sums are not equal, we must pop from one of the stacks. We can write $Val[i, j, k] = \max\{Val[i-1, j, k], Val[i, j-1, k], Val[i, j, k-1]\}$



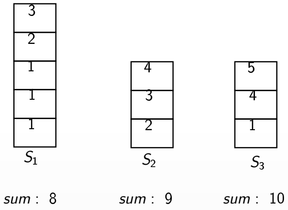
Three Stack Question

- Greedy-Choice property holds.
  - The greedy choice is to pop from the stack with the largest sum and repeat.



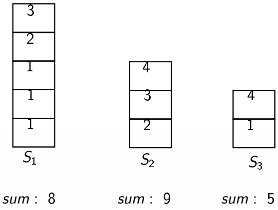
Three Stack Question

- Greedy-Choice property holds.
  - The greedy choice is to pop from the stack with the largest sum and repeat.



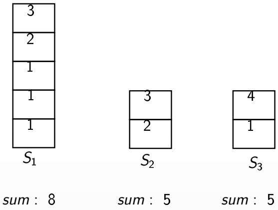
Three Stack Question

- Greedy-Choice property holds.
  - The greedy choice is to pop from the stack with the largest sum and repeat.



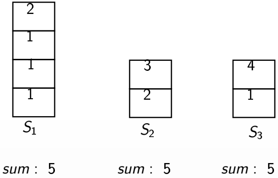
Three Stack Question

- Greedy-Choice property holds.
  - The greedy choice is to pop from the stack with the largest sum and repeat.



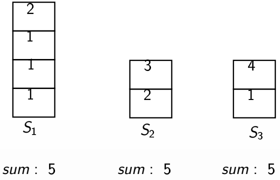
Three Stack Question

- Greedy-Choice property holds.
  - The greedy choice is to pop from the stack with the largest sum and repeat.



Three Stack Question

- Greedy-Choice property holds.
  - The greedy choice is to pop from the stack with the largest sum and repeat.
  - **Greedy Choice property:** there is an optimal solution which starts by popping from the stack with the larges sum (why?)

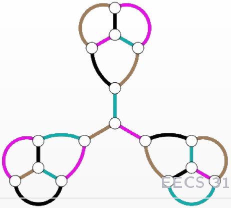


Problem Definition

- In edge coloring, the goal is to color edges of a graph with minimum number of colors
  - No two adjacent edges (edges sharing an endpoint) should have the same color
- The problem is **NP-hard**, that is, it is very unlikely that an algorithm which runs in polynomial time (in $O(n^c)$ for some constant c) can solve it optimally.

Problem Definition

- In edge coloring, the goal is to color edges of a graph with minimum number of colors
  - No two adjacent edges (edges sharing an endpoint) should have the same color
- The problem is **NP-hard**, that is, it is very unlikely that an algorithm which runs in polynomial time (in $O(n^c)$ for some constant c) can solve it optimally.
- For a graph of max-degree Δ, at least Δ and at most $\Delta + 1$ colors are required (Vizing theorem)
  - This implies that $cost(\text{Opt}) \approx \Delta$

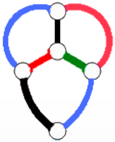


Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring

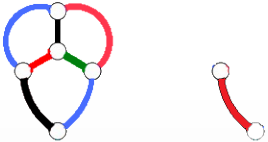
Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



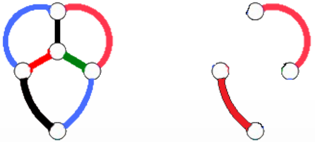
Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



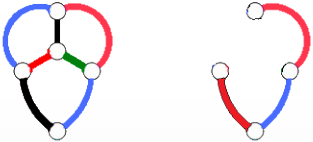
Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



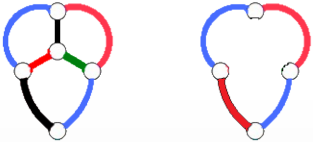
Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



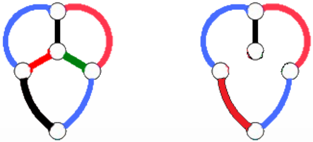
Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



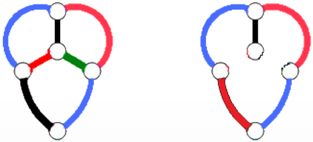
Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



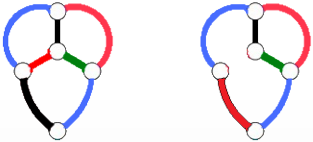
Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



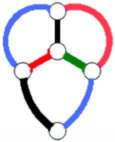
Problem Definition

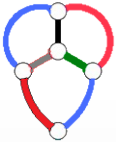
- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4



Problem Definition

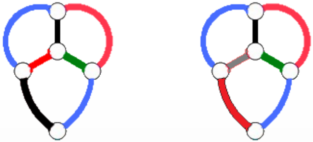
- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4





Problem Definition

- Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring
- cost of Opt is 4
- Cost of Greedy is 5, which is not optimal



Greedy algorithm

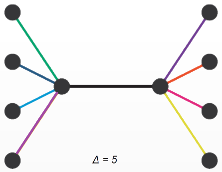
- The number of colors used by Greedy is at most 2 times worse than the optimal number of colors $\rightarrow$ **Greedy has an approximation factor of 2!**

Greedy algorithm

- The number of colors used by Greedy is at most 2 times worse than the optimal number of colors $\rightarrow$ **Greedy has an approximation factor of 2!**
- For any graph of degree Δ, cost of Opt is at least Δ.

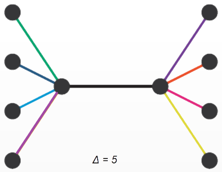
Greedy algorithm

- The number of colors used by Greedy is at most 2 times worse than the optimal number of colors $\rightarrow$ **Greedy has an approximation factor of 2!**
- For any graph of degree Δ, cost of Opt is at least Δ.
- Cost of greedy is at most $2\Delta - 1$.



Greedy algorithm

- The number of colors used by Greedy is at most 2 times worse than the optimal number of colors $\rightarrow$ **Greedy has an approximation factor of 2!**
- For any graph of degree Δ, cost of Opt is at least Δ.
- Cost of greedy is at most $2\Delta - 1$.
  - Consider the edge that demands the last color.
    - It is an edge between two vertices, each currently adjacent to at most $\Delta - 1$ edges.



Greedy algorithm

- The number of colors used by Greedy is at most 2 times worse than the optimal number of colors $\rightarrow$ **Greedy has an approximation factor of 2!**
- For any graph of degree Δ, cost of Opt is at least Δ.
- Cost of greedy is at most $2\Delta - 1$.
  - Consider the edge that demands the last color.
    - It is an edge between two vertices, each currently adjacent to at most $\Delta - 1$ edges.
    - The number of colors will be $2(\Delta - 1) + 1 = 2\Delta - 1$