Algorithm-Analysis-Module-5-Important-Topics

1. Randomized quick sort

What is a randomized Algorithm?

- First we need to know what is deterministic algorithm
  - Algorithm is executed based only on the input
- In randomized algorithm, the algorithm is executed not only based on an input, but also a random number
- An algorithm that uses random numbers to decide what to do next anywhere in logic is called randomized algorithm
- This randomness is used to reduce time complexity or space complexity in other standard algorithms

Advantages of randomized algorithms?

- For many problems, a randomized algorithm is the simplest and fastest
- Many NP-hard/NP complete problems can be easily solvable

Types of Randomized algorithm

- Randomized Las Vegas Algorithms
- Randomized Monte Carlo Algorithms

Deterministic Quicksort Algorithm

Algorithm QuickSort(A[],low,high)

1. If low >=high, then EXIT
2. Let the first element of S as the pivot element, say x

3. Partition A[low..high] into two subarrays. The first subarray has all the elements of A that are less than x and the second subarray has all those that are freater than x. Now the index of x be pos
4. QuickSort(A,low,pos-1)
5. QuickSort(A,pos+1,high)

Worst case

- The Array elements are sorted, the running time = $O(n^2)$
- In average case, the expected running time = $O(nlogn)$

Randomized Quicksort Algorithm

Algorithm randQuickSort(A[],low,high)

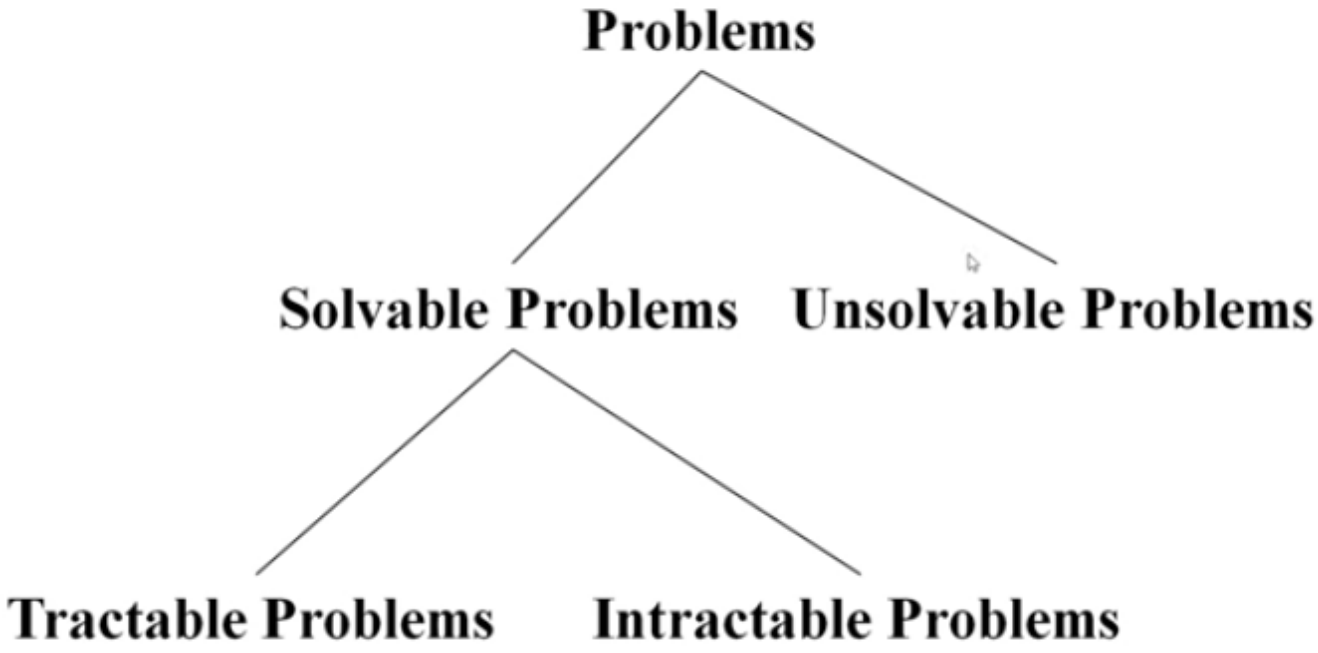
1. If low >= high, then EXIT
2. While pivot x is not a Central pivot
    1. Choose uniformly at a random a number from [low..high]. Let the randomly picked number be x
    2. Count elements in A[low..high] that are smaller than A[x]. Let this count be sc
    3. Count elements in A[low..high] that are greater than A[x]. Let this count be gc
    4. Let n = (high-low+1)
        1. If sc >=n/4 and gc >=n/4 then x is the central pivot
3. Partition A[low..high] into two subarrays. The subarray has all the elements of A that are less than x and the second subarray has all those that are greater than x. Now the index of x be pos
4. randQuickSort(A,low,pos-1)
5. randQuickSort(A,pos+1,high)

Time complexity

- The expected time complexity of step 2 is $O(n)$
    - The probability that the randomly chosen element is central pivot is 1/n
    - Therefore the expected number of times the while loop runs is n
- Worst case time complexity of Randomized quick sort = $O(nlogn)$

2. Complexity Classes (P,NP,NPH,NPC)

Problems can be classified as



Tractable Problems

- Algorithm complexity is polynomial
- Tractable problem solutions are implemented in practice
- Example Path Problem
  - Given directed Graph G. determine where a directed path exists from vertex s to vertex t
  - Time complexity = $O(n)$
  - Here n = total number of vertices

Intractable Problems

- Algorithm complexity is exponential
- An intractable problem has a faster complexity growth compared to tractable problems
- Example
  - Knacksack problem

        - Time Complexity = $O(2^n)$
    - Travelling salesman problem
        - Time Complexity = $O(n^2 2^n)$

Complexity classes

- There are 4 complexity classes
    - P
    - NP
    - NP-Hard
    - NP-Complete

Class P

What is Class P?

- Class P problems are solvable in Polynomial time
- Time Complexity = $O(n^k)$
    - n = size of input
    - k = constant
- Example
    - Path Problem: Given directed graph G, determine where a directed path exists from s to t

PATH - Problem - Algorithm

- **Inputs: <G,s,t> G - Directed graph s,t - 2 nodes**

1. Place a mark on node s and enqueue it into an empty queue
2. Repeat step 3 until the queue is empty
3. Dequeue the front element a. Mark all unvisited neighbors of a and enqueue those into the queue.
4. If t is marked, then accept. Otherwise reject
   Complexity calculation

- Step 1 and 4 will execute exactly once
- Step 3 and 4 will execute at most n times

- Time complexity $O(n)$
- This is a polynomial time algorithm, so PATH problem belongs to class P

More Examples of P Problem

- Single source shortest Path problem using Dijkstras Greedy method
- Multistage Graph problem implemented using forward or backward dynamic programming.
- Minimum cost spanning tree using Prims or Kruskals method
- Network flow problem using Ford-Fulkerson algorithm

Class NP

- Some problems can only be solved in **exponential or factorial time**
- And it can be verified in polynomial time
- Then these problems are called NP problems

Class NP Example (HAMPATH Problem)

- A Hamiltonian path in a directed graph G is a **directed path that goes through each node exactly once**
- The HAMPATH problem is to test whether a graph a graph contains a hamiltonian path connecting 2 specified nodes
- There is no polynomial solution
- HAMPATH Problem have a feature called polynomial verifiability

Algorithm

- Inputs G,s,t,P
- P: Is the path P1,P2,P3 .... Pm
- m number of nodes in G
- s and t are the two vertices

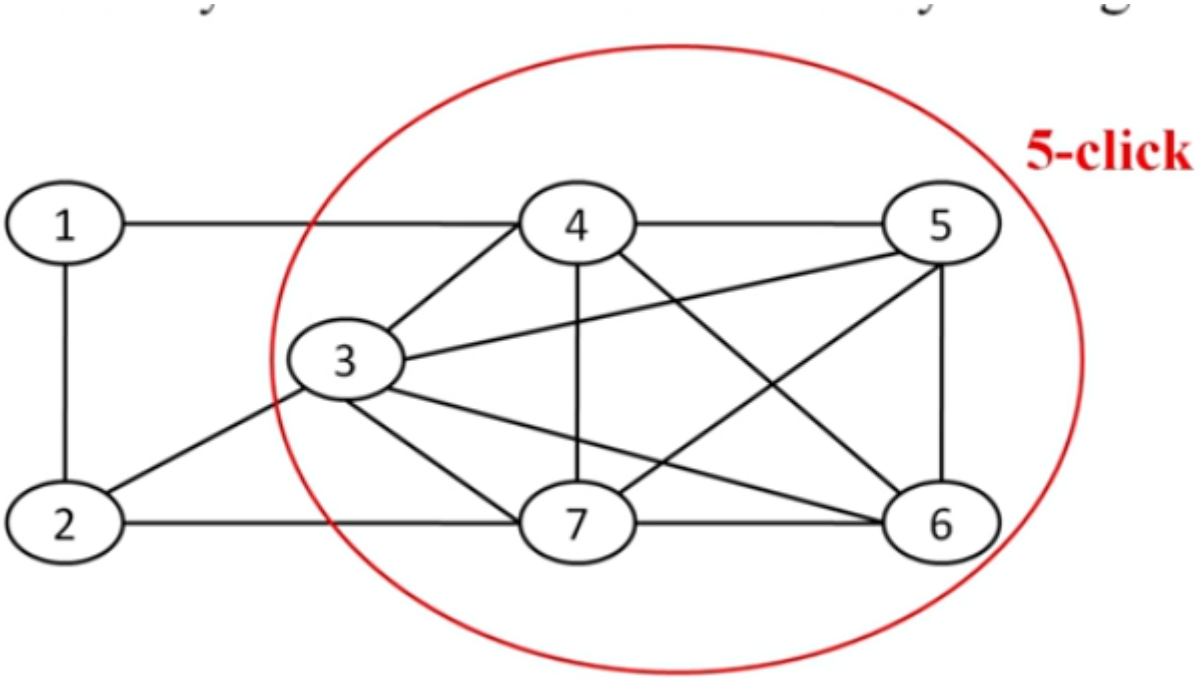
1. Check whether $s = P_1$ and $t = P_m$ if either fails reject
    1. Check whether first and last are s and t or not
2. Check for the repetition of the nodes in the list P. If any are found, reject
3. For each i, check whether (P_i,P_{i+1}) is an edge in G. Here i varies from 1 to m-1, If any are not, reject
4. If all test have been passed, then accept it

    1. This means its a Hamiltonian path

- This is hamiltonian path verifier algorithm, this algorithm complexity is in polynomial time
- For checking if Hamiltonian path is existing, a polynomial time algorithm doesnt exist
- But, for verification, polynomial time algorithm exists

Class NP Example (CLIQUE)

- A clique in an undirected graph is a subgraph where every two nodes are connected by an edge



- Inside the circle, you can see, every two nodes are connected, which forms a Clique

Clique Problem: To Determine whether a graph contains a clique of specified size

- There is no polynomial time algorithm
- But we can verify this in polynomial time

Algorithm: Clique Verifier Algorithm

- Input <G,k,V>

1. Test whether V' is a set of k vertices in the graph G
    1. Checking if there are k distinct vertices
2. Check whether for each pair (u,v) ∈ V'. the edge (u,v) belongs to E
3. If both steps pass then accept. Otherwise reject

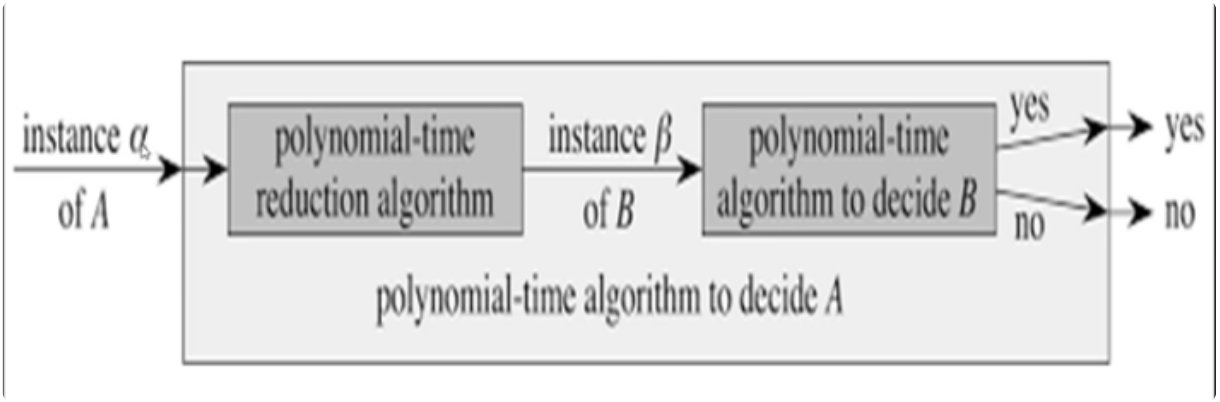
- This algorithm will execute in polynomial time
- Therefore CLIQUE problem is a NP Problem

More Examples

- Circuit-SAT problem
- 3CNF-SAT Problem
- Vertex Cover Problem
- Independence set problem
- Travelling salesman problem
- 3-coloring problem

Polynomial time reduction

Where is polynomial time reduction being used?

- Suppose we have 2 decision problems
- A
  - We like to solve in polynomial time
  - Instance of A is alpha
- B
  - It having a polynomial time algorithm
  - Instance of B is beta
- A is the problem we want to solve in polynomial time, but its not possible
  - Whereas B is a problem which is solvable in polynomial time.
  - We can try converting Instance Alpha of A to Instance beta of B
- The below is the Polynomial time reduction
- 

What is Polynomial time reduction

- Suppose that we have a procedure that transforms alpha to beta with the following characteristics
  - The transformation takes polynomial time
  - The answers are the same, That is the answer for alpha is yes if the answer of beta is also yes
- Such a procedure is called polynomial time reduction
-

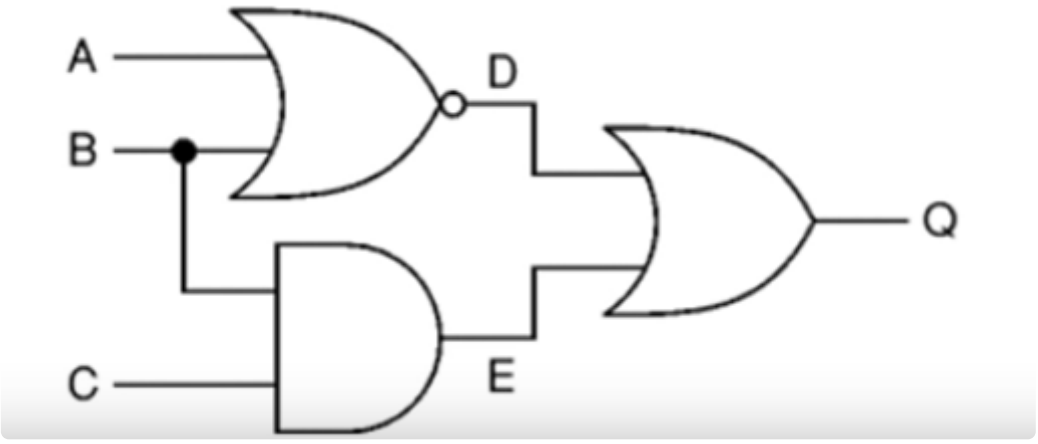
Class NP Hard

- If a decision problem X is NP-Hard if every problem in NP is polynomial time reducible to X
  - Y <= p X, Where Y is all NP Problems
    - Y <= p X
      - The above statement means Y is polynomial time reducible to X
- X is hard as all problems in NP
- If X can be solved in polynomial time, then all problems in NP can be solved in polynomial time

Class NP Complete

- If the problem is NP as well as NP-Hard then that problem is NP Complete

Example: Circuit-SAT problem

- Given a Boolean circuit C, is there an assignment to the variables that causes the citcuits to output 1?
  - Consider the below
  - 

  - Is there any Combination of A, B and C which will make Q as 1?

- This is a yes or no question
- It a combination exists, then the result is yes, otherwise its no

Other examples

- Clique Problem
  - Problem is to determine whether the graph contains clique of size k (Yes/No)
- Vertex Cover problem
  - GIven a graph G(V,E)
  - Vertices V
  - Subset of V is V'
  - assume V' size = k
  - Here the question is whether we can use k nodes to cover all edges in a graph or not? (Yes/No)
- 3-CNF SAT

3-CNF-SAT

- Literal: Variables and its negation
- Clause:OR of one or more literals[Ex: $(x_1 V \neg x_2 V x_3)$]
- Conjunctive Normal Form(CNF): AND of clauses
  - Ex: $(x_1 V \neg x_2 V x_3) \wedge (x_1 V x_2) \wedge (\neg x_1 V x_2 V x_3)$
- 3-CNF: Each clause has exactly 3 distinct literals.
  - Ex: $(x_1 V \neg x_2 V x_3) \wedge (x_1 V x_2 V x_3) \wedge (\neg x_1 V x_2 V x_3)$
- **3-CNF-SAT Problem**: Given a 3-CNF expression ϕ, is there an assignment to the variables that causes the expression to output 1?

3. Approximation Algorithm

- **Approximate solution**

    - A feasible solution with value close to the value of optimal solution is called an approximate solution.
- **Approximation Algorithms:**
    - An algorithm that returns near optimal solution is called Approximation Algorithm
- **Approximation algorithms have two main properties**
    - They run in polynomial time
    - They produce solutions close to optimal solutions
- Approximation algorithms are useful to give approximate solutions to NP complete optimization problems
- Its also useful to give fast approximations to problems that run in polynomial time
- **Approximation Ratio/Approximation Factor**
    - For a given problem assume C is the result obtained by the algorithm and C* is the optimal result
    - The approximation ratio of an algorithm is the ratio between the result obtained by the algorithm and the optimial result
    - **For maximization problem** 0 < C <= C *
        - Approximation Ratio = C*/ C
    - For **minimization problem** 0 <= C * <= C
        - Approximation Ratio = C / C *
    - Approximation ratio of an approximation algorithm is never less than 1

Bin Packing approximation Algorithm

- Given n items of different weights and bins each of capcity c, assign each item to a bin such that the number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity
- The lower bound on minimum number of bins required can be given as
- Min no of bins >= Ceil((Total Weight) / (Bin Capacity))

Applications

- Loading of containers like trucks
- Placing data on multiple disks
- Job scheduling
- Packing ads on fixed length radio/TV station breaks

- Storing a large collection of music onto tapes /CD's etc

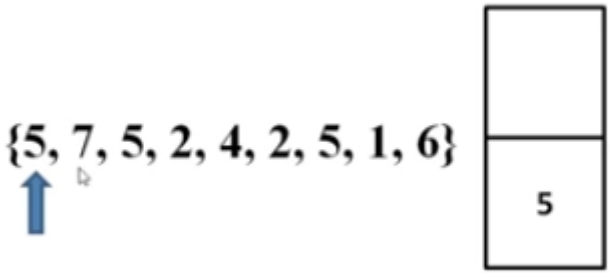
Different Bin Packing Approximation Algorithms

- **Online Algorithm**
  - n items and corresponding weights are not initially available, will be available in realtime
  - These are divided into
    - Next Fit Algorithm
    - First Fit Algorithm
    - Best Fit Algorithm
    - Worst Fit Algorithm
- **Offline Algorithm**
  - n items and corresponding weights will be initially available
  - First Fit Decreasing Algorithm
  - Best Fit Decreasing Algorithm

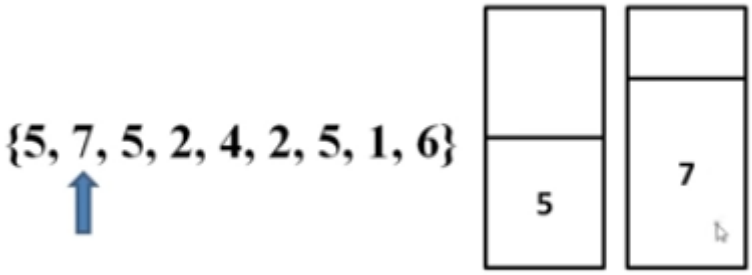
Next Fit Algorithm

Example

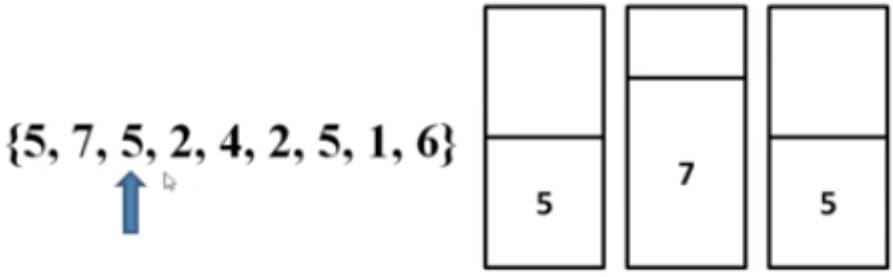
- Apply Next Fit Bin packing approximation algorithms on the following items with bin capacity = 10. Assuming the sizes of items be {5,7,5,2,4,2,5,1,6}
- Minimum number of bins >= Ceil(Total Weight / Bin Capacity)
  - = Ceil(37/10) = 4
- We got the minimum number of bits, now lets start
- We have the following array
  - {5,7,5,2,4,2,5,1,6}
- Lets take each one by one
  - First we have 5
    - Total capacity of the bin = 10, So it can be fitted in a bin



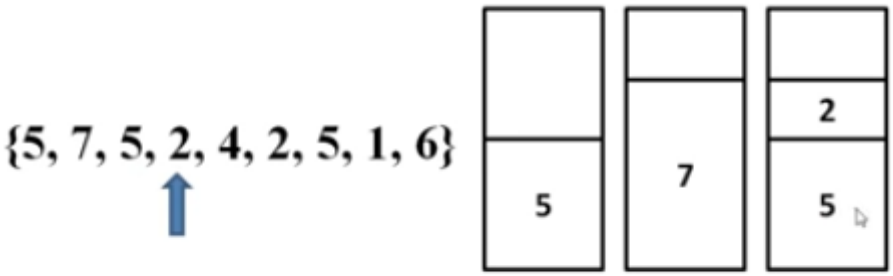
- Next value is 7
  - We cant place in first bin because 5 + 7 = 12 which is bigger than max capacity 10



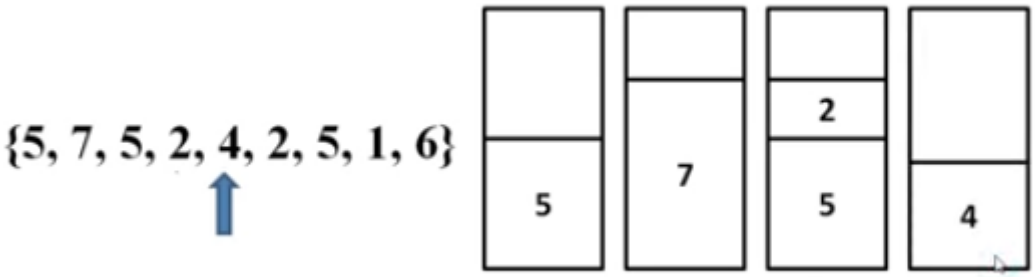
- Next value is 5,
  - Our current Bin is 7
  - 7 + 5 = 12, not possible, since the max capacity is 10



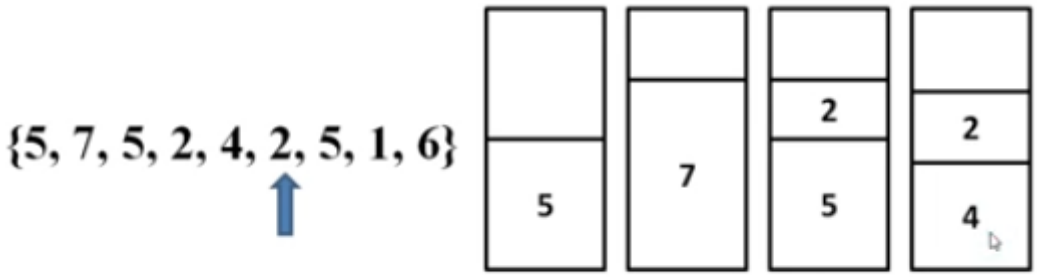
- Next value is 2
  - Current bin in 5
  - 5 + 2 = 7, less than 10, so possible



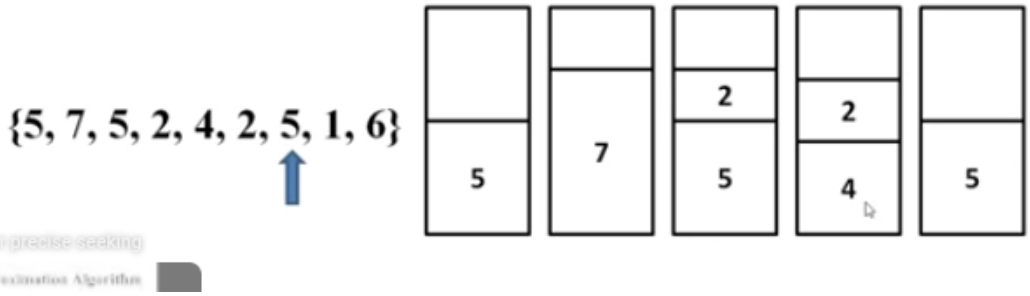
- Next value is 4
  - current bin is 5+2
  - 5+2+4 = 11 not possible



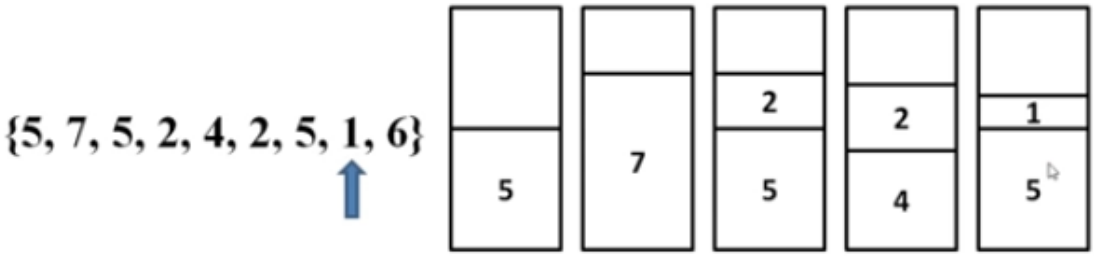
- Next value 2
  - Current bin is 4
  - 4+2 = 6, possible



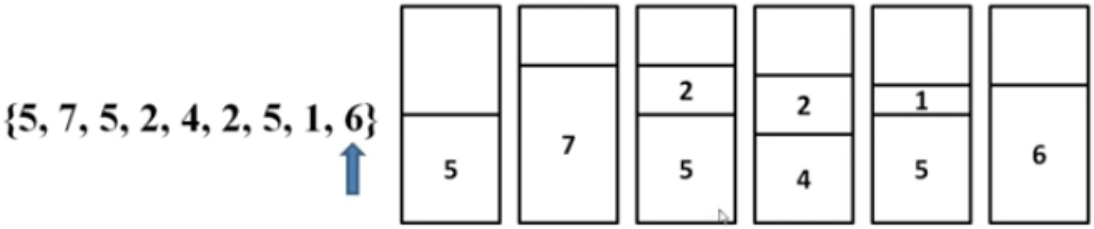
- Next value 5
  - New bin



- Next value 1



- Next value 6



- The number of bins required = 6

Definition

- If the current item is fit in the same bin as the last item. then insert it in the same bin
- Otherwise use the new bin
- Time complexity
  - Best case time complexity = θ(n)
  - Average case = θ(n)
  - Worst case = θ(n)

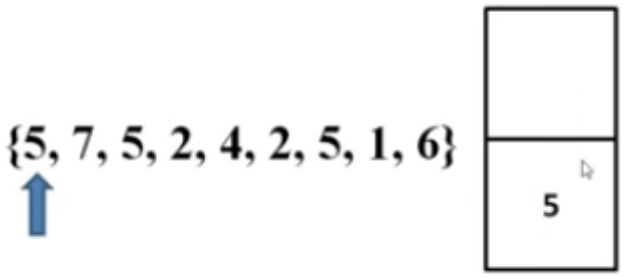
First Fit Algorithm

Example

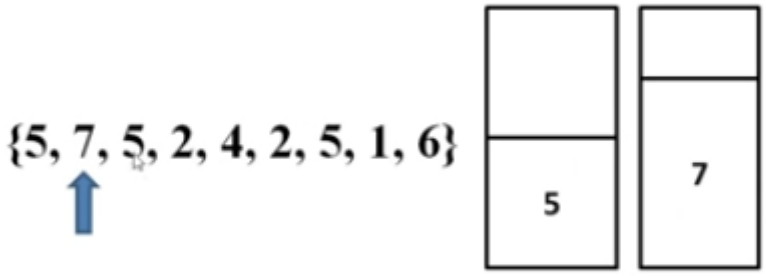
- Apply First Fit Bin packing approximation algorithms on the following items with bin

capacity = 10. Assuming the sizes of items be {5,7,5,2,4,2,5,1,6}

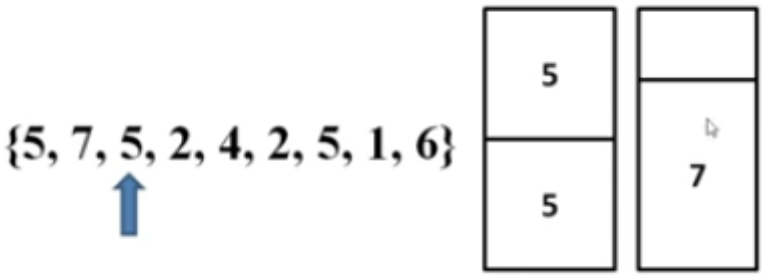
- Considering the array {5,7,5,2,4,2,5,1,6}, taking each element one by one
- Taking 5
  - Putting in first bin



- Next, 7
  - Check 5, 5+7 = 12, not possible
  - new bin

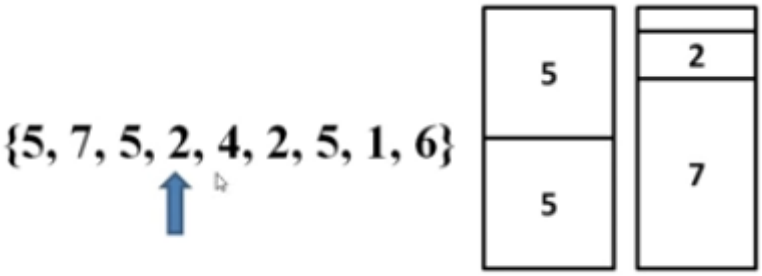


- Next 5
  - We check the first bin for space, instead of the current bin
  - 5 + 5 = 10, possible

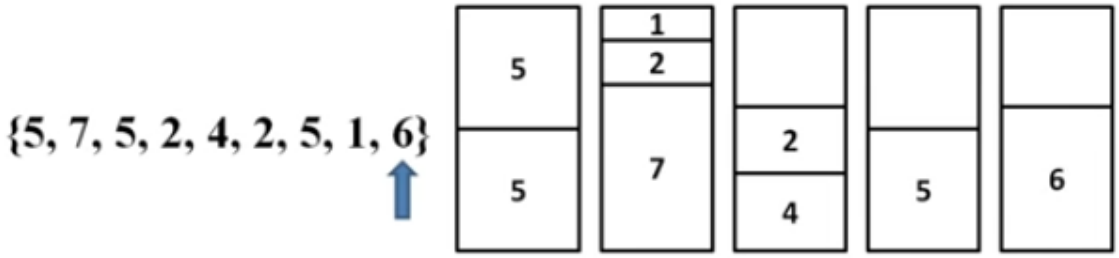


- Next 2
  - First bin space?

    - 5 +5 + 2 = 12, no
  - 2nd bin? = 7 + 2 = 9, yes



- Repeat the steps and we get



  - No of bins required = 5

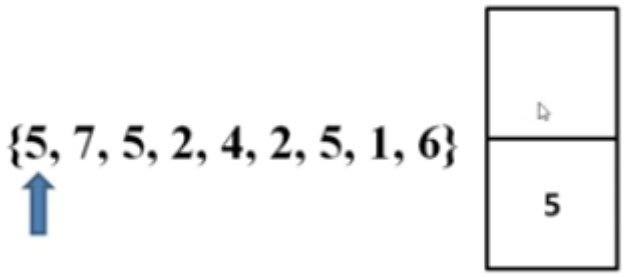
Definition

- Scan the previous bins in order and find the first bin that it fits
- It such bin exits, place the item in that bin, otherwise use a new bin
- Time complexity
  - Best case time complexity = $\theta(nlogn)$
  - Average case time complexity = $\theta(n^2)$
  - Worst case Time complexity = $\theta(n^2)$

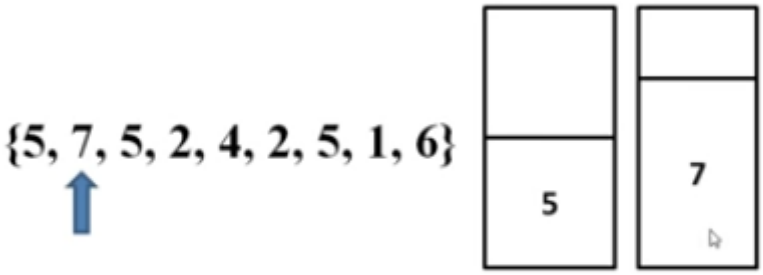
Best Fit Algorithm

Algorithm

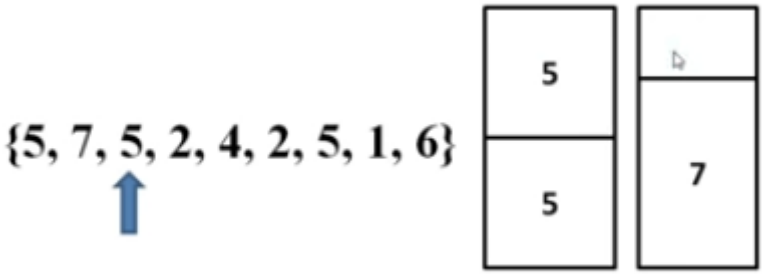
- Apply Best Fit Bin packing approximation algorithms on the following items with bin capacity = 10. Assuming the sizes of items be {5,7,5,2,4,2,5,1,6}
- Starting with 5



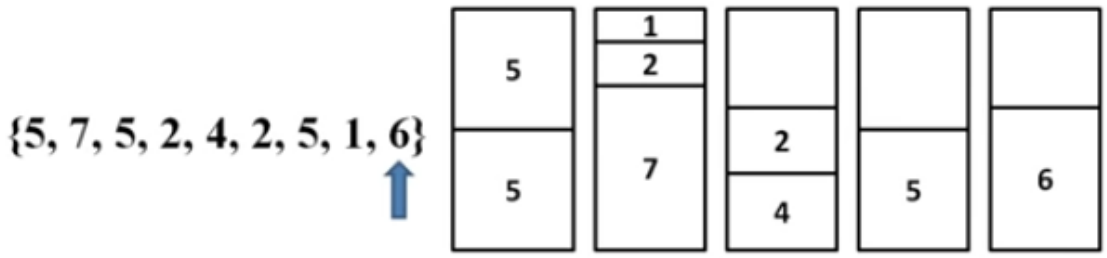
- Next 7



- Next 5
  - Check both bins, and checks which is the best fitting of both



- Repeating steps



Number of bins required = 5

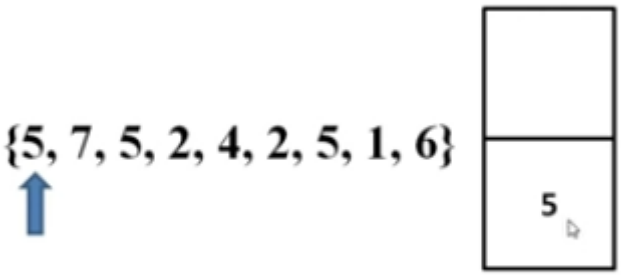
Definition

- Scan the previous bins and find a bin that is having the minimum remaining capacity that can accommodate this item
- If such bin exists, place the item in that bin
- Otherwise use a new bin
- Time complexity
  - Best case time complexity = $\theta(nlogn)$
  - Average case = $\theta(n^2)$
  - Worst case = $\theta(n^2)$

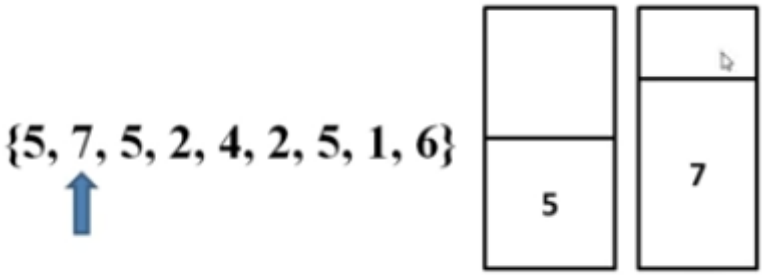
Worst Fit Algorithm

Algorithm

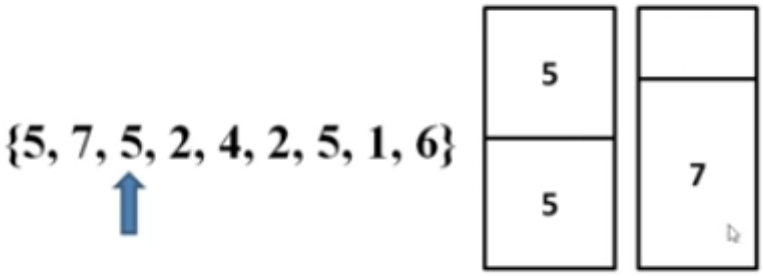
- Apply Worst Fit Bin packing approximation algorithms on the following items with bin capacity = 10. Assuming the sizes of items be {5,7,5,2,4,2,5,1,6}
- Starting with 5



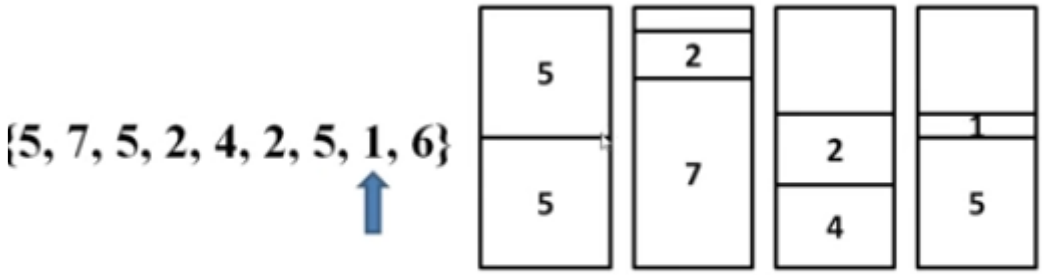
- Next 7



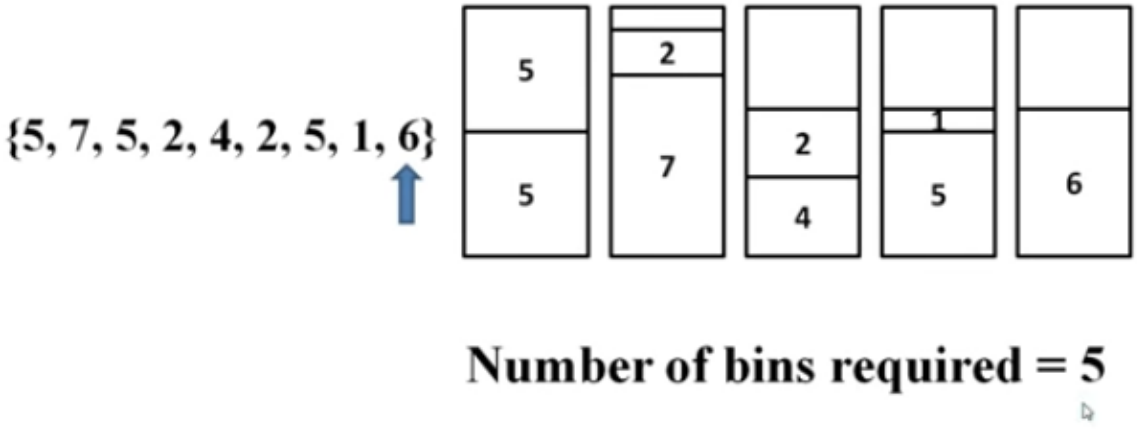
- Next 5
  - Take both bins,choose one with the larger balance space



- Repeat the steps
- For 1



  - Here the largest space was with 5, so placing it there
- Repeating



Definition

- Scan the previous bins and find a bin that has the maximum remaining capacity that can accomodate this item
- If such bin exists, place the item in that bin

- Otherwise use a new bin
- Time complexity
  - Best case time complexity = $\theta(nlogn)$
  - Average case = $\theta(n^2)$
  - Worst case = $\theta(n^2)$

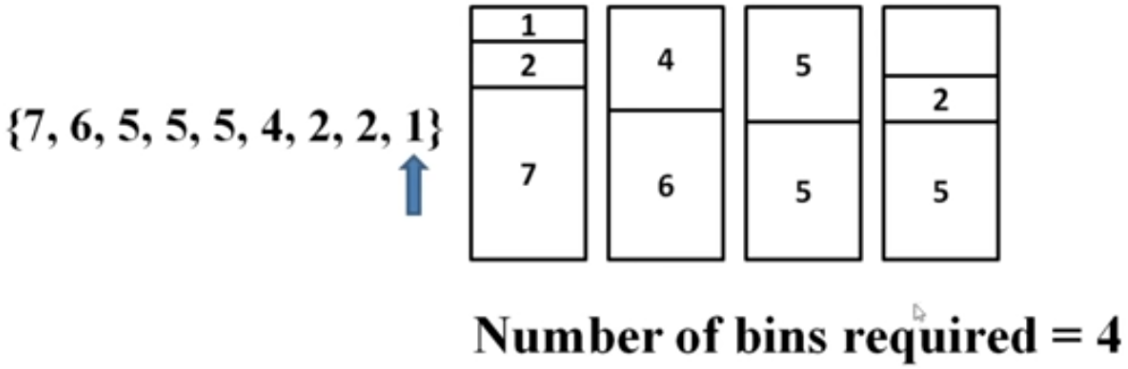
First Fit Decreasing Algorithm

Definition

- Sort items in the descending order of their size
- Apply first fit algorithm
- Time complexity
  - Best case time complexity = $\theta(nlogn)$
  - Average case = $\theta(n^2)$
  - Worst case = $\theta(n^2)$

Example

- Apply First Fit Bin packing approximation algorithms on the following items with bin capacity = 10. Assuming the sizes of items be {5,7,5,2,4,2,5,1,6}
- Arrange the items in the decreasing order of their size
  - {7,6,5,5,5,4,2,2,1}
  - Do the First fit as usual



-

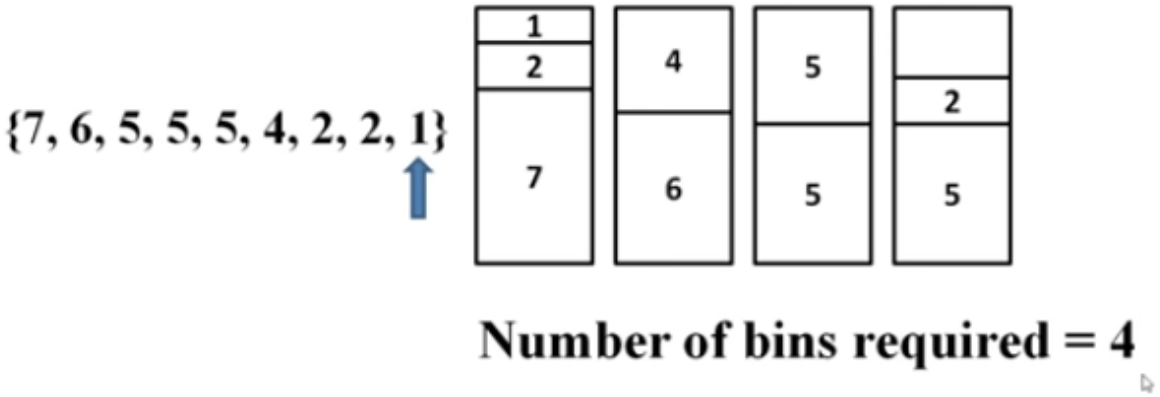
Best Fit Decreasing Algorithm

Definition

- Sort the items in the descending order of their size
- Apply Best fit algorithm
- Time complexity
  - Best case time complexity = $\theta(nlogn)$
  - Average case = $\theta(n^2)$
  - Worst case = $\theta(n^2)$

Example

- Apply Best Fit Bin packing approximation algorithms on the following items with bin capacity = 10. Assuming the sizes of items be {5,7,5,2,4,2,5,1,6}
- Arrange the items in the decreasing order of their size
  - {7,6,5,5,5,4,2,2,1}
- Do the Best fit as usual



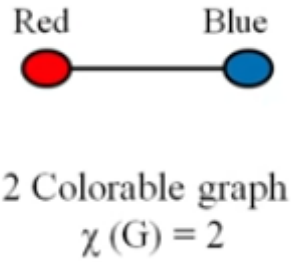
Graph Coloring

- This is another approximation algorithm
- There are different Graph Coloring Problems
  - Vertex Coloring
  - Edge Coloring
  - Face Coloring

Vertex Coloring

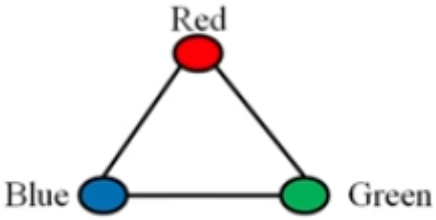
- Assignment of colors to vertices in a graph such that **no two adjacent vertices share the same color**
- A Graph is 0-colorable if no of verticies is empty (V = Φ)

- A Graph is 1-colorable if there are no edges(E = Φ)
- 2 colorable graph



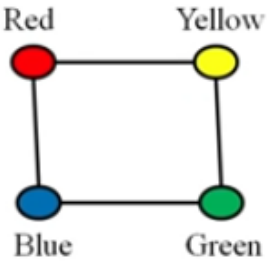
2 Colorable graph
$\chi(G) = 2$

- 3 Colorable Graph

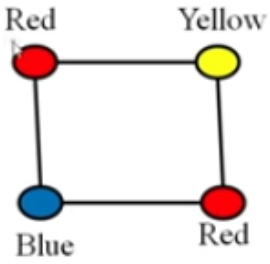


3 Colorable graph
$\chi(G) = 3$

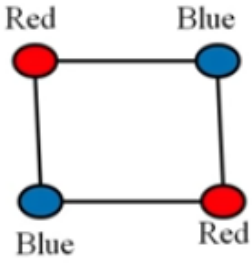
- The 4 colorable graph below can be colored in different ways



4 Colorable graph
$\chi(G) = 4$



3 Colorable graph
$\chi(G) = 3$



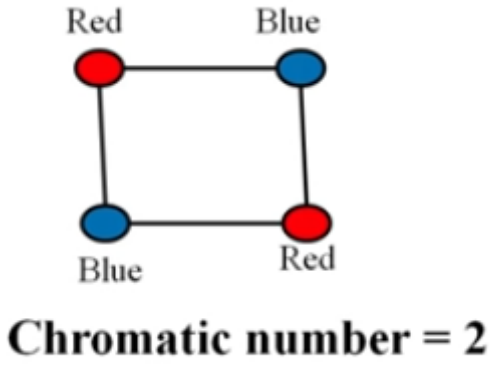
2 Colorable graph
$\chi(G) = 2$

- A null graph is a graph that contains vertices but no edge
  - In these cases only a single color is enough
- For all other graphs, no of colors are greater than or equal to 2

Four Color theorem

- For every Planar graph, the chromatic number is less than or equal to 4
- A Graph is k colorable if it has k colors

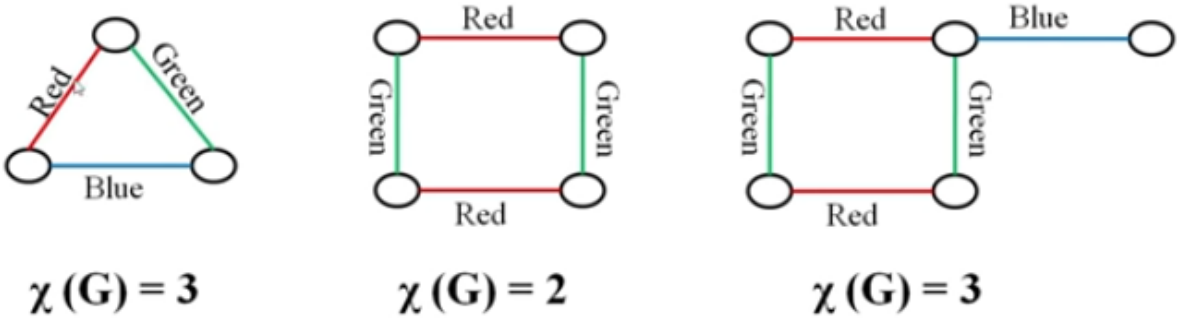
- **Chromatic number**: It is the minimum number of colors with which a graph can be colored.



  -
  - Here its 2 colors, so chromatic number is 2
- A Graph whose chromatic number is k, then it is called k-chromatic graph

Edge Coloring

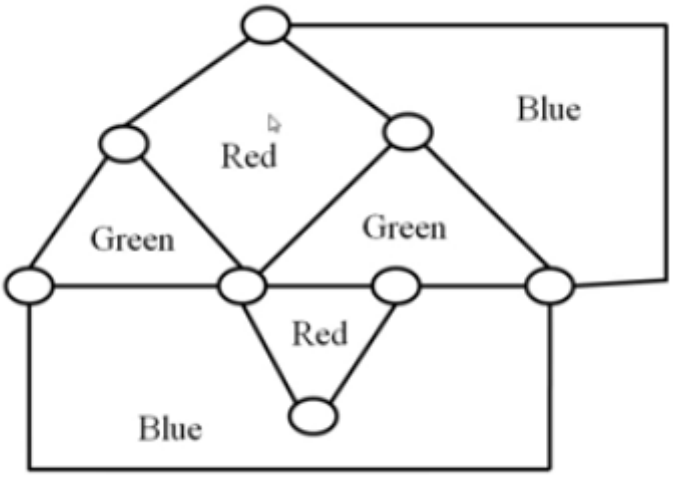
- Given a graph G=(V,E), assign a color to each edges so that no two adjacent edges share the same color



  -

Face Coloring

- For a planar graph, assign a color to each face/region so that no two faces that shares boundary have the same color

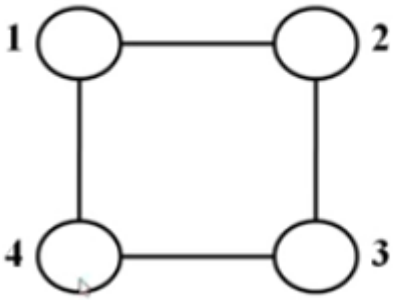


Graph coloring Approximation Algorithm

- Graph coloring problem is a NP-Complete Problem, but There are approximation Algorithms
- Important graph coloring problem is vertex coloring.

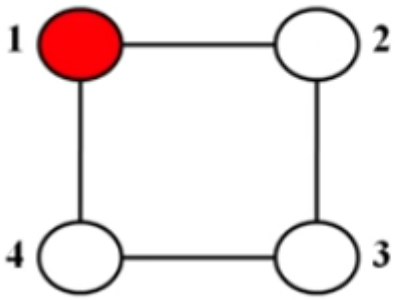
Lets see **Greedy Approximation Algorithm for Vertex Coloring**

- We are taking a graph of 4 vertex

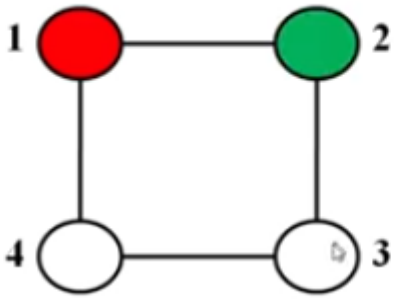


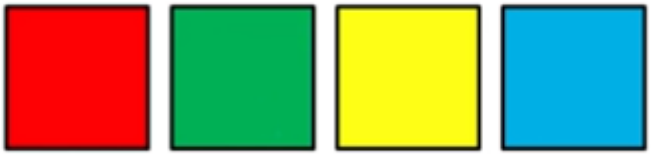
- We have 4 colors
- Vertex 1

- Here none of the vertices are colored, so we can place our first color



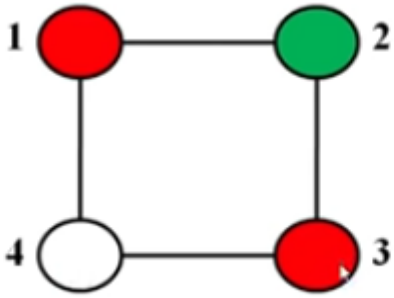
- Vertex 2
  - First color Red
    - Its not possible because there is an adjacent red in 1
  - Second color Green
    - No issues, we can place

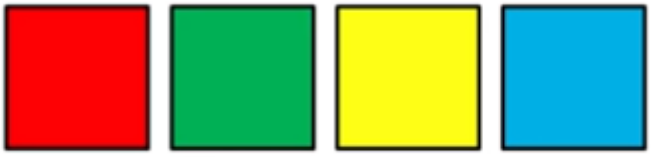




Colors 1 2 3 4

- Vertex 3
  - FIrst color red
    - We can place Red in 3, because neighbours are green

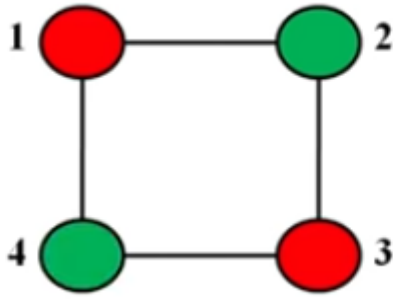


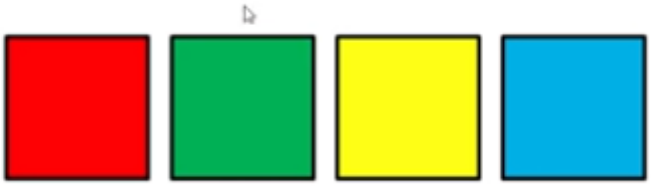


Colors 1 2 3 4

- Vertex 4
  - First color red
    - Not possible, 1, and 3 has red which are adjacent
  - 2nd color green

- Possible





Algorithm

Algorithm Approximate_Graph_Coloring(G,n)

1. for i=1 to n do // Each Vertex
    1. for c=1 to n do // Each color
        1. If no vertex adjacent to vi has color c
            1. Color vi with c
            2. Break

Time complexity

- $O(n^3)$

Applications of graph coloring

- Prepare time tabe
- Scheduling
- Register Allocation
- Mobile radio frequency assignment
- Map coloring