BHARATI VIDYAPEETH PUNE

Algorithm Analysis and Design

Bharati Vidyapeeth's Institute of Computer Applications and Management (GGS IP University) New Delhi, India by Dr. Saumya Bansal

Pre-Requisites & Course Outcomes

PRE-REQUISITES:

1. Programming Skills
2. Discrete Structures
3. Data Structures

COURSE OUTCOMES (COs):

After completion of this course, the learners will be able to:-

CO #	Detailed Statement of the CO	BT Level	Mapping to PO #
CO1	Demonstrate P and NP complexity classes of the problem.	BTL2	PO1, PO2, PO3
CO2	Apply the concepts of asymptotic notations to analyze the complexities of various algorithms.	BTL4	PO1, PO2, PO3, PO4
CO3	Analyze and evaluate the searching, sorting and tree-based algorithms.	BTL5	PO1, PO2, PO3, PO4, PO5
CO4	Design efficient solutions using various algorithms for given problems.	BTL6	PO1, PO2, PO3, PO4, PO5, PO6, PO10
CO5	Develop innovative solutions for real-world problems using different paradigms.	BTL6	PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO9, PO10,

 U3.2

Syllabus (Unit-III)

- **Greedy Algorithms:** General Concept, Applications, Activity Selection Problem, Fractional Knapsack problem, Job Sequencing with Deadlines, Huffman Coding, Analysis and Correctness of Prim's, Kruskal Algorithm and Dijkstra Algorithm.
- **Dynamic Programming:** General Concept, Matrix-Chain Multiplication Problem, Longest Common Subsequence Problem, Bellman-Ford Algorithm, Analysis and Correctness of Floyd-Warshall Algorithm, Optimal Binary Search Trees, 0/1 Knapsack Problem, Network Flow Problem.
- **No. of Hours:** 12
- **Books:**
  - T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to Algorithms", PHI, 2nd Edition, 2006. **Chapters[15-16 & 23-25]**
  - S. Dasgupta, C. Papadimitriou and U.Vazirani, "Algorithms", McGraw Hill Higher Education, 1st Edition, 2017. **Chapters[4-6]**
  - J. Kleinberg and E. Tardos, "Algorithm Design", Pearson Education, 2nd Edition, 2009. **Chapters[4-6]**

 U3.3

Greedy Algorithm: Overview

- Most of the problem in greedy algorithm contains n inputs and we have to find the subset of given input which gives the maximum profit and minimum cost.
- All possible solutions for given input n are solution space.
- One of the solution, from the solution space, which satisfies the problem condition is called feasible solutions.
- An optimization problem is one in which you want to find, not just a feasible solution, but the best solution.

Greedy Algorithm: Overview

- Being greedy for local optimization with the hope it will lead to a global optimal solution, not always, but in many situations, it works.
- A greedy algorithm works in phases. At each phase:
  - We take the best you can get right now, without regard for future consequences
  - We hope that by choosing a local optimum at each step, you will end up at a global optimum.

Greedy Algorithm: Overview

- For example: Suppose we have to pay bill of Rs. 67.50 with minimum number of notes.
- With greedy algorithm, we start with higher value of note.
  - We take Rs. 50 note. [67.50 – 50 = 17.50]
  - Then we take, Rs. 10 note. [17.50 – 10 =7.50]
  - Then we take, Rs. 5 note. [7.50 – 5 =2.50]
  - Then we take, Rs. 2 note. [2.50 – 2 =0.50]
  - Then we take, Ps. 50 coin. [0.50 – 0 =0.50]

Greedy Algorithm: Overview

- In some monetary system, notes come in 1, 7, and 10.
  - Using a greedy algorithm to count out 15, you would get
    - ✓A 10 piece
    - ✓Five 1 kron pieces, for a total of 15 krons
  - This requires six coins
- A better solution would be to use two 7 pieces and one 1 piece
  - This only requires three coins

The greedy algorithm results in a solution, but not in an optimal solution always.

 U3.7

Greedy Algorithm: Overview

- Greed Advantages:
  - Greedy approach is easy to implement: Always taking the best available choice is usually easy.
    - ✓It usually requires sorting the choices
  - Less time complexity: Repeatedly taking the next available best choice is usually linear work.
    - ✓But don't forget the cost of sorting the choices.
  - Much cheaper than exhaustive search.
    - ✓Much cheaper than most other algorithms.

 U3.8

Greedy Algorithm: Overview

- Greedy Disadvantages:
  - Greedy algorithms don't work for some problems.
  - Despite their simplicty, correct greedy algorithms can be complex.
- Greedy Conditions:
  - There's no guaranteed way to recognize problems that can be solved by a greedy algorithm. but,
  - A problem in which, a locally optimal choice leads to a global optimum, and each remaining subproblem also leads to an optimal choice can be solved with a greedy algorithm.

 U3.9

Greedy Algorithm: Overview

- Greedy Method Applications:
  - Activity Selection Problem
  - Fractional Knapsack problem
  - Job Sequencing with Deadlines
  - Huffman Coding
  - Prim's Algorithm
  - Kruskal Algorithm
  - Dijkstra Algorithm
  - CPU Scheduling algorithms

Greedy Algorithm: Activity Selection Problem

- Problem: Let us consider we have n activities, Say A={a1, a2,…, an). Each activity has start time and finish time.
- Objective: Find solution set having maximum number of non-conflicting activities that can be executed in a single time frame, assuming that only one person or machine is available for execution.
  - Two activities, say i and j, are said to be non-conflicting if si >= fj or sj >= fi where si and sj denote the starting time of activities i and j respectively, and fi and fj refer to the finishing time of the activities i and j respectively.

Greedy Algorithm: Activity Selection Problem

Example1: Consider the following 3 activities sorted by finish time.

start[] = {10, 12, 20};

finish[] = {20, 25, 30};

- A person can perform at most two activities. The maximum set of activities that can be executed is {0, 2} [Here, 0 and 2 are indexes of activity]

Greedy Algorithm: Activity Selection Problem

Example2: Consider the following 3 activities sorted by finish time.

start[] = {1, 3, 0, 5, 8, 5};

finish[] = {2, 4, 6, 7, 9, 9};

- A person can perform at most four activities. The maximum set of activities that can be executed is {0, 1, 3, 4} [Here, 0,1,3 and 4 are indexes of activity]

 U3.13

Greedy Algorithm: Activity Selection Problem

- Solution:
  - Step 1: Sort the activities according to finish time.
  - Step 2:Select the first activity from the sorted array and print it.
  - Step 3: Do the following for the remaining activities in the sorted array.

    If the start time of this activity is greater than or equal to the finish time of the previously selected activity, then

    select this activity and print it.

 U3.14

Greedy Algorithm: Activity Selection Problem

- Solution:
  - Step 1: Sort the activities according to finish time. O(nlogn)
  - Step 2:Select the first activity from the sorted array and print it. O(1)`
  - Step 3: Do the following for the remaining activities in the sorted array. O(n-1)

    If the start time of this activity is greater than or equal to the finish time of the previously selected activity, then

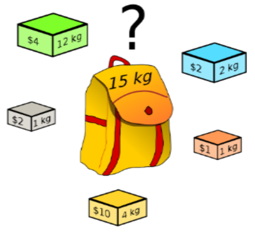
    Select this activity and print it.

Total Time (Worst case)=O(nlogn) Best Case=O(n)

 U3.15

Greedy Algorithm: Fractional Knapsack Problem

- Problem: Given a set of items, each with a weight and a value,
- Objective: Determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.



Knapsack Problem

Greedy Algorithm: Fractional Knapsack Problem

- Example 1: Items as (value, weight) pairs

  arr[] = {{60, 10}, {100, 20}, {120, 30}}

  Knapsack Capacity, W = 50;
- Calculate ratio= Value/Weight;
  - ratio={6,5,4} // it is already in decreasing order otherwise we have to sort
  - Apply greedy,
    - ✓ Pick item 1, weight=10, [50-10=40], now pick the next maximum profit
    - ✓ Pick item 2, weight=20, [40-20=20], Now we cannot take all item3 because remaining weight i.e. 20 < item 3 weight 30.
    - ✓ So, we will take 2/3 rd of item3. Hence total value =60+100+(2/3)(120)=240

Greedy Algorithm: Fractional Knapsack Problem

- Example 2: Items as (value, weight) pairs

  arr[] = {{5, 5}, {2, 4}, {2, 6} , {4, 2} , {5, 1}}

  Knapsack Capacity, W = 12;
- Calculate ratio= Value/Weight;
  - ratio={1,0.5,0.33,2,5} // Sort item into decreasing order

  Total Value=16

Greedy Algorithm: Fractional Knapsack Problem

- Solution:
  - Step 1: For each item, compute its value / weight ratio.
  - Step 2: Arrange all the items in decreasing order of their value / weight ratio.
  - Step 3: Start putting the items into the knapsack beginning from the item with the highest ratio util we get the Knapsack Capacity.

Greedy Algorithm: Fractional Knapsack Problem

- Time Complexity:
  - Step 1: For each item, compute its value / weight ratio. O(n)
  - Step 2: Arrange all the items in decreasing order of their value / weight ratio. O(nlogn)
  - Step 3: Start putting the items into the knapsack beginning from the item with the highest ratio until we get the Knapsack Capacity becomes 0. O(n)

Total Time= O(nlogn)

Greedy Algorithm: Job Sequencing with Deadlines

- Problem: A set of n given jobs which are associated with deadlines and profit is earned, if a job is completed by its deadline.
  - Only one processor is available for processing all the jobs.
  - Processor takes one unit of time to complete a job.
- Objective: Find a sequence of jobs, which is completed within their deadlines and gives maximum profit.

Greedy Algorithm: Job Sequencing with Deadlines

- Example 1:Four Jobs with following deadlines and profits

Job ID	Deadline	Profit
A	4	20
B	1	10
C	1	40
D	1	30

- Job Sequence: C, A (How?)

Greedy Algorithm: Job Sequencing with Deadlines

- Example 1:Four Jobs with following deadlines and profits. Maximum Deadline =4.

Job ID	Deadline	Profit
A	4	20
B	1	10
C	1	40
D	1	30

Deadline	1	2	3	4
Job	C	-	-	A

- Job Sequence: C, A

Greedy Algorithm: Job Sequencing with Deadlines

- Example 1:Six Jobs with following deadlines and profits. Maximum Deadline =5.

Job ID	Deadline	Profit
A	5	24
B	3	18
C	3	22
D	2	30
E	4	12
F	2	10

Deadline	1	2	3	4	5
Job	B	D	C	E	A

- Job Sequence: B,D,C,E,A Profit=18+30+22+12+30=112

Greedy Algorithm: Job Sequencing with Deadlines

- Solution:
  - Step 1: Sort all the given jobs in decreasing order of their profit.
  - Step 2: Check the value of maximum deadline and Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline.
  - Step 3: *Iterate on jobs in decreasing order of profit.For each job , do the following :*
    - ✓ *Find a time slot i, such that slot is empty and i < deadline and i is greatest. Put the job in this slot and mark this slot filled.*
    - ✓ *If no such i exists, then ignore the job.*

Greedy Algorithm: Job Sequencing with Deadlines

- Time Complexity: O(nlogn) using Max heap
  - Step 1: Sort all the given jobs in decreasing order of their profit. O(nlogn)
  - Step 2: Check the value of maximum deadline and Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline. O(n)
  - Step 3: *Iterate on jobs in decreasing order of profit. For each job , do the following : (nlogn) using heap or linear search O(n^2)*
    - ✓ *Find a time slot i, such that slot is empty and i < deadline and i is greatest. Put the job in this slot and mark this slot filled.*
    - ✓ *If no such i exists, then ignore the job.*

Greedy Algorithm: Huffman Coding

- Overview
  - Suppose we have to send data “AABBBCCDDDEEEEE”.
  - If we send it as it is, we require 15 (length of the message) * 8 (bits to represent single character) =120 bits.
  - But We are sending only 5 characters i.e. A, B, C, D and E.
  - We can choose another method to send the data with our own code.

Character	Code	Frequency	Total bits
A	000	2	6
B	001	3	9
C	010	2	6
D	011	3	9
E	110	5	15

Greedy Algorithm: Huffman Coding

- If we represent our message with the code, it will require 45 bits.
- But to decode this , we have to send the code table also to receiver.
- Size of the table= 5 (A-E)*8(ASCII for single character) +5*3 (3bits each code)=40+15=55
- Therefore, total bits to send the message = Table +Message
  =55+45 =100 bits
- It is less than the previous message encoding scheme.
- It can be further reduced, if we could code the character in variable length bits.

 U3.28

Greedy Algorithm: Huffman Coding

- Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1.
- If the compressed bit stream is 0001 (ab) , the de-compressed output may be "cccd" or "ccb" or "acd" or "ab".
- The above problem occurs because the code of c is the prefix of a and b.
- Therefore, The variable-length codes should be assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character.
- This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

 U3.29

Greedy Algorithm: Huffman Coding

- Huffman Tree Design:
  Let's take the message "AABBBCCDDDEEEEE".

Character	Frequency
A	2
B	3
C	2
D	3
E	5

- Sort the characters according to their frequency

2	2	3	3	5
A	C	B	D	E

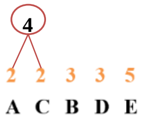
 U3.30

Greedy Algorithm: Huffman Coding

- Huffman Tree Design:

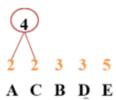
2	2	3	3	5
A	C	B	D	E

- Take two minimum from the list.

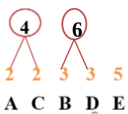


Greedy Algorithm: Huffman Coding

- Huffman Tree Design:



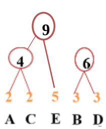
- Take two minimum from the list. Now, list is [4,3,3,5]



Greedy Algorithm: Huffman Coding

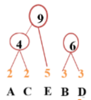
- Huffman Tree Design:

- Take two minimum from the list. Now, list is [4, 5, 6]

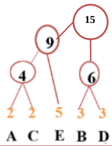


Greedy Algorithm: Huffman Coding

- Huffman Tree Design:

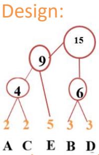


- Take two minimum from the list. Now, list is [9, 6]



Greedy Algorithm: Huffman Coding

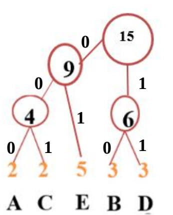
- Huffman Tree Design:



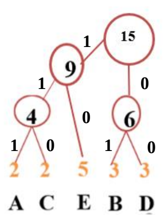
- Now, We have only one element in the list i.e. 15. Now assign the bits.
- We can assign left side edge as 0 and right side edge with 1 or vice-versa. But, don't mix both strategies.

Greedy Algorithm: Huffman Coding

- Huffman Tree Design:



Left=0 Right=1



Left=1 Right=0

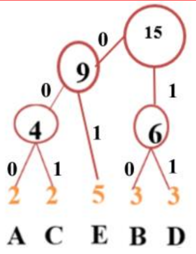
Greedy Algorithm: Huffman Coding

- Huffman Tree Design:
  - Now Build the code
  - We ill start from the root,
  - Lets Make code for A
  - Traverse from the root i.e. 15 to leaf node A , 15->9->4->2 = 000

  Similarly, we can code

  B=10, C=001, D=11, E=01

  Observe: No code is prefix of other



Greedy Algorithm: Huffman Coding

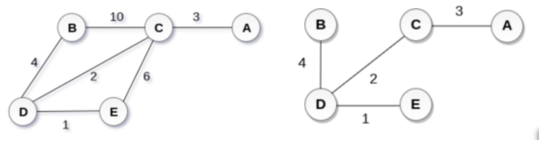
- Problem: Create a Huffman tree
- Solution:
  - Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
  - Extract two nodes with the minimum frequency from the min heap.
  - Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
  - Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Greedy Algorithm: Huffman Coding

- Problem: Create a Huffman tree
- Time Complexity: **O(nlogn) //n=number of unique character**
  - Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root) **O(nlogn)**
  - Extract two nodes with the minimum frequency from the min heap. **O(1)**
  - Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap. **O(n)**
  - Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Greedy Algorithm: Minimum Spanning Tree

- Minimum Spanning Tree: A minimum spanning tree is a least-cost subset of the edges of a graph that connects all the nodes.



Greedy Algorithm: Prim's Algorithm

Minimum Spanning Tree

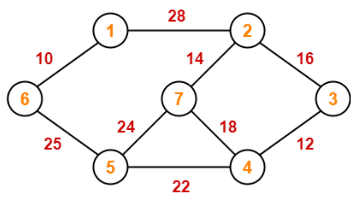
Prim's Algorithm | Kruskal's Algorithm

Greedy Algorithm: Kruskal's Algorithm

- Kruskal's algorithm selects an edge that has minimum weight and then adds that edge if it doesn't create a cycle.
- So, initially, there are | V | single-node trees in the forest. Adding an edge merges two trees into one.
- When the algorithm is completed, there will be only one tree, and that is the minimum spanning tree.
- There are two ways of implementing Kruskal's algorithm:
  - By using Disjoint Sets: Using UNION and FIND operations
  - By using Priority Queues: Maintains weights in priority queue
- The appropriate data structure is the UNION/FIND algorithm
- Animation: https://visualgo.net/en/mst

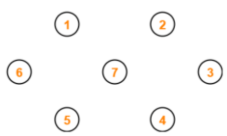
Greedy Algorithm: Kruskal's Algorithm

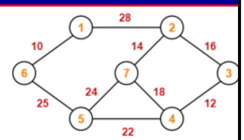
- Example 1: Lets consider the following graph.



Greedy Algorithm: Kruskal's Algorithm

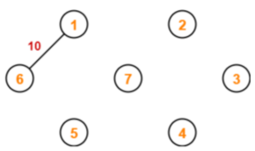
- Initially, Create all vertices as a single tree.

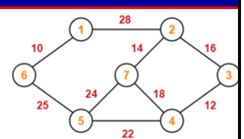




Greedy Algorithm: Kruskal's Algorithm

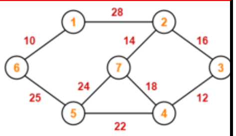
- Initially, Select the minimum edge i.e. 1-6 having weight 10.

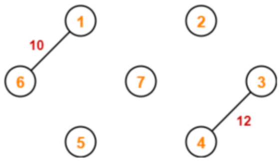




Greedy Algorithm: Kruskal's Algorithm

- Initially, Select the next minimum edge i.e. 3-4 having weight 12. Check whether the newly added edge makes cycle?

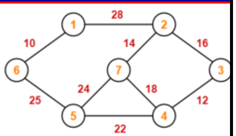


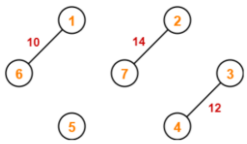


 U3.46

Greedy Algorithm: Kruskal's Algorithm

- Initially, Select the next minimum edge i.e. 2-7 having weight 14. Check whether the newly added edge makes cycle?

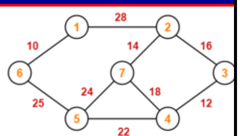


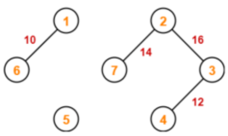


 U3.47

Greedy Algorithm: Kruskal's Algorithm

- Initially, Select the next minimum edge i.e. 2-3 having weight 16. Check whether the newly added edge makes cycle?

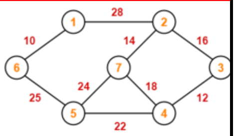


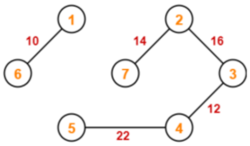


 U3.48

Greedy Algorithm: Kruskal's Algorithm

- Initially, Select the next minimum edge i.e. 4-5 having weight 22. Check whether the newly added edge makes cycle?



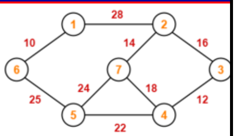


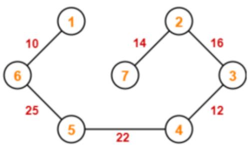
We did not take edge 7-4 (Weight =18), the next minimum edge because it forms cycle. Therefore, we go for the next minimum edge i.e. 4-5.

 U3.49

Greedy Algorithm: Kruskal's Algorithm

- Initially, Select the next minimum edge i.e. 5-6 having weight 25. Check whether the newly added edge makes cycle?



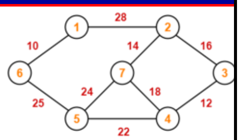


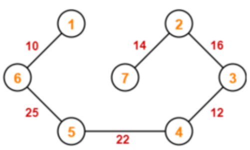
We did not take edge 7-5 (Weight =24), the next minimum edge because it forms cycle. Therefore, we go for the next minimum edge i.e. 5-6.

 U3.50

Greedy Algorithm: Kruskal's Algorithm

- Initially, Select the next minimum edge i.e. 1-2 having weight 28. But it forms cycle. So, We do not include this edge





Minimum Cost=10 + 25 + 22 + 12 + 16 + 14 =99 Units

 U3.51

Greedy Algorithm: Kruskal's Algorithm

- Time Complexity:
  - Let T = Ø.
  - Let S be a disjoint-set data structure.
  - For each v ∈ V: } O(V)
    - Call S.make-set(v)
  - For each edge (u, v) sorted by cost: } O(ElogE)
    - If S.find(u)≠ S.find(v):
      - Add (u, v) to T. } O(logE)
      - Call S.union(u, v).

Using Mean Heap O(ElogE)

Total runtime: O(E log E). [what about O(ElogV)?]

Greedy Algorithm: Kruskal's Algorithm

- Space Complexity: **O(|E| + |V|),**
  - Disjoint Set Data Structure takes O(|V|) space to keep track of the roots of all the vertices
  - Another O(|E|) space to store all edges in sorted manner.

Greedy Algorithm: Correctness of Kruskal's Algorithm

Theorem: : Kruskal's algorithm always produces an MST

Proof:

- Let T be the tree produced by Kruskal's algorithm and T* be an MST.
- We will prove c(T) = c(T*). If T = T*, we are done. Otherwise T ≠ T*, so T–T* ≠ Ø. Let (u, v) be an edge in T–T*.
- Let S be the cross cut containing u at the time (u, v) was added to T. We claim (u, v) is a least-cost edge crossing cut (S, V – S). First, (u, v) crosses the cut, since u and v were not connected when Kruskal's algorithm selected (u, v).
- Next, if there were a lower-cost edge e crossing the cut, e would connect two nodes that were not connected.
- Thus, Kruskal's algorithm would have selected e instead of (u, v), a contradiction. Since T* is an MST, there is a path from u to v in T*.

Greedy Algorithm: Correctness of Prim's Algorithm

- The path begins in S and ends in V – S, so it contains an edge (x, y) crossing the cut.
- Then $T^{*\prime} = T^* \cup \{(u, v)\} - \{(x, y)\}$ is a Spanning Tree of G and $c(T^{*\prime}) = c(T^*) + c(u, v) - c(x, y)$.
- Since $c(x, y) \geq c(u, v)$, we have $c(T^{*\prime}) \leq c(T^*)$.
- Since T^* is an MST, $c(T^{*\prime}) = c(T^*)$.
- Thus $c(T) = c(T^*)$.

Greedy Algorithm: Prim's Algorithm

- Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.
- Prim's algorithm always gives the connected tree.
- Animation: https://visualgo.net/en/mst

Greedy Algorithm: Prim's Algorithm

- Algorithm:

1. **The edge queue is constructed**
2. **A predecessor list of predecessors for each node is constructed.**
3. **"Best" distances to each node are set to infinity.**
4. **Choose node 0 as the "root" of the MST (any node will do as the MST must contain all nodes),**
5. **While the edge queue is not empty,**
    1. **Extract the cheapest edge, *u*, from the queue,**
    2. **Relax all its neighbours - if the distance of this node from the closest node in the MST formed so far is larger than *d[u][v]*, then update *d[u][v]* and set *v*'s predecessor to *u*.**
6. **Return the predecessor list.**

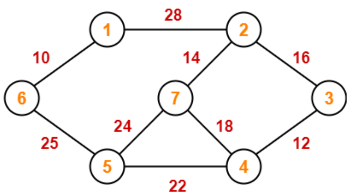
Greedy Algorithm: Prim's Algorithm

- Algorithm:

```
Prim's( Graph g, n, costs ) {
  Queue q;
  q = ConsEdgeQueue( g, costs );
  pl = ConsPredList( n );
  for(i=0;i<n;i++) {
    d[i] = INFINITY;
  }
/* Choose 0 as the "root" of the MST */
  d[0] = 0;
  pi[0] = 0;
while ( !Empty( q ) ) {
    u = ExtractMin( g );
    for each v in g->adj[u] {
      if ( (v in q) && costs[u][v] < d[v] ) {
        pl[v] = u;
        d[v] = costs[u][v];
      }
    }
  }
  return pl;
}
```

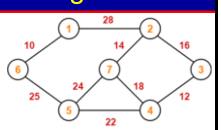

Greedy Algorithm: Prim's Algorithm

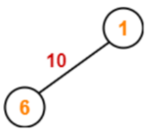
- Example 1:



Greedy Algorithm: Prim's Algorithm

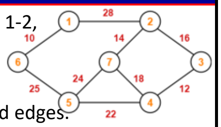
- Select any vertex, Say 1,
- Find all edges from 1, we have 1-6 and 1-2.
- Find the minimum weight among all edges.
- We have 1-6 with weight 10. Add 6 with 1.

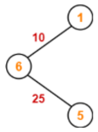




Greedy Algorithm: Prim's Algorithm

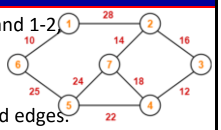
- Find all edges from 1 and 6, we have 1-6 and 1-2, and 6-5 but 1-6 is already visited, hence we shall not include again.
- Find the minimum weight among all unvisited edges.
- We have 5-6 with weight 25. Add 5 with 6.

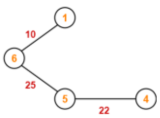




Greedy Algorithm: Prim's Algorithm

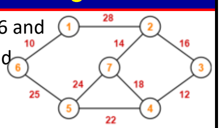
- Find all edges from 1, 6 and 5. We have 1-6 and 1-2, 6-5, 5-7, and 5-4 but 1-6 and 5-6 is already visited, hence we shall not include again.
- Find the minimum weight among all unvisited edges.
- We have 5-4 with weight 22. Add 7 with 5.

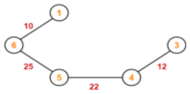




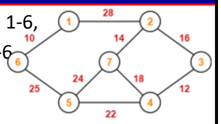
Greedy Algorithm: Prim's Algorithm

- Find all edges from 1, 6, 5 and 4. We have 1-6 and 1-2, 6-5, 5-7, 5-4, 4-7, and 4-3 but 1-6,5-6 and 5-4 are already visited, hence we shall not include again.
- Find the minimum weight among all unvisited edges.
- We have 4-3 with weight 12. Add 3 with 4.

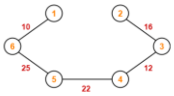




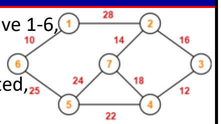
Greedy Algorithm: Prim's Algorithm



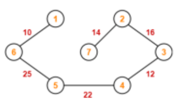
- Find all edges from 1, 6, 5, 4 and 3. We have 1-6, 1-2, 6-5, 5-7, 5-4, 4-7, 4-3, and 3-2 but 1-6,5-6, 5-4, and 4-3 are already visited, hence, we shall not include again.
- Find the minimum weight among all unvisited edges.
- We have 3-2 with weight 16. Add 2 with 3.



Greedy Algorithm: Prim's Algorithm



- Find all edges from 1, 6, 5, 4, 3 and 2. We have 1-6, 1-2, 6-5, 5-7, 5-4, 4-7, 4-3, 3-2, 2-1, and 2-7 but 1-6,5-6, 5-4, 4-3 and 3-2 are already visited, hence, we shall not include again.
- Find the minimum weight among all unvisited edges.
- We have 2-7 with weight 14. Add 7 with 2.



Now, all vertices have been included in the tree. Therefore, no need to check other edges.

Greedy Algorithm: Prim's Algorithm

- Time Complexity:
  - **Step 1:** Select a starting vertex O(1)
  - **Step 2:** Repeat Steps 3 to 5 until all the vertices are included. O(V+E) //Using BFS
  - **Step 3:** Find all the edges that connect the tree to new vertices.
  - **Step 4:** Find the least weight edge among those edges and include it in the existing tree. O(logE)/O(logV)
  - **Step 5:** If including that edge creates a cycle, then reject that edge and look for the next least weight edge. O(1)

So, overall time complexity = O(E + V) x O(logV) = O((E + V)logV) = O(ElogV)

Greedy Algorithm: Prim's Algorithm

- Space Complexity:
  - We need an array to know if a node is in MST or not. Space O(V).
  - We need an array to maintain Min-Heap. Space O(E).
  - So, Total space complexity is of order O(V+E).

Greedy Algorithm: Correctness of Prim's Algorithm

Theorem: If G is a connected, weighted graph with distinct edge weights, Prim's algorithm correctly finds an MST.

Proof:

- Let T be the spanning tree found by Prim's algorithm and T* be the MST of G.
- We will prove T = T* by contradiction. Assume T ≠ T*. Therefore, T – T* ≠ Ø.
- Let (u, v) be any edge in T – T*. When (u, v) was added to T, it was the least-cost edge crossing some cut (S, V – S).

Greedy Algorithm: Correctness of Prim's Algorithm

- Since T* is an MST, there must be a path from u to v in T*.
- This path begins in S and ends in V – S, so there must be some edge (x, y) along that path where x ∈ S and y ∈ V – S.
- Since (u, v) is the least cost edge crossing (S, V – S), we have c(u, v) < c(x, y).
- Let T*' = T* ∪ {(u, v)} – {(x, y)}.
- However, c(T*') = c(T*) + c(u, v) – c(x, y) < c(T*), contradicting that T* is an MST.
- We have reached a contradiction, so our assumption must have been wrong. Thus T = T*, so T is an MST.

Greedy Algorithm: Dijkstra Algorithm

- Function DIJKSTRA(G =< V, E, c, s >)

```
for (i = 1 to n) do
   d[i] = ∞
end for
d[s] = 0
Organize the vertices into a heap Q, based on their d values.
S ← φ.
while (Q ≠ φ)
do
 u ← EXTRACT-MIN(Q)
for all vertices v adjacent to u
do
 if (d[v] > d[u] + c(u, v)) then
       d[v] = d[u] + c(u, v))
 End if
 end for
S ← S ∪ {u}
end while
```


Topics We Have Learned So Far

The test can be scheduled in any lecture next week. Be Ready.

BHARATI VIDYAPEETH PUNE

Thank You

Bharati Vidyapeeth's Institute of Computer Applications and Management (GGS IP University) New Delhi, India by Dr. Saumya Bansal