Greedy Design Technique

Adnan YAZICI
Dept. of Computer Engineering
Middle East Technical Univ.
Ankara - TURKEY

Greedy Design Technique

The Greedy Approach can be described in general terms as follows.

- The Greedy technique suggests that one can device an algorithm which works in stages, considering one input at a time.
- At each stage, a decision is made regarding whether or not a particular input is an optimal solution.
- This is done by considering the inputs in an order determined by some selection procedure.
- The selection procedure itself is based on some optimization measure.

Greedy Design Technique

The Greedy Approach can be described in general terms as follows.

- It is clear why such algorithms are called "greedy": *at every step, the procedure chooses the best morsel it can swallow, without worrying about the future. It never changes its mind: once a candidate is included in the solution, it is there for good; once a candidate is excluded from the solution, it is never reconsidered.*
- The resulting solution may or may not be optimal, depending on the problem being solved (in other words, the greedy method does not work for all problems.)

Greedy Design Technique

- For convenience, we formulate Greedy Approach as a procedure.

```
Procedure Greedy (A,n)
   //  A(1:n) contains n inputs   //
   solution ← ∅              // initializes the solution to
   empty //
   for i ←1 to n do
         x ← Select (A)
         If Feasible (solution, x) Then
                  solution ← Union (solution,x)
   repeat
   return(solution)
End Greedy
```

Graphs (review)

Definition. A ***directed graph (digraph)*** $G = (V, E)$ is an ordered pair consisting of

- a set V of ***vertices*** (singular: ***vertex***),
- a set $E \subseteq V \times V$ of ***edges***.

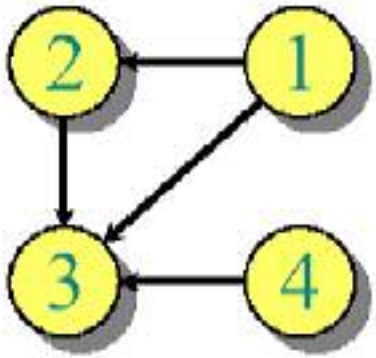
In an ***undirected graph*** $G = (V, E)$, the edge set E consists of *unordered* pairs of vertices.

In either case, we have $|E| = O(V^2)$. Moreover, if G is connected, then $|E| \geq |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$.

Greedy Design Technique

The ***adjacency matrix*** of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$

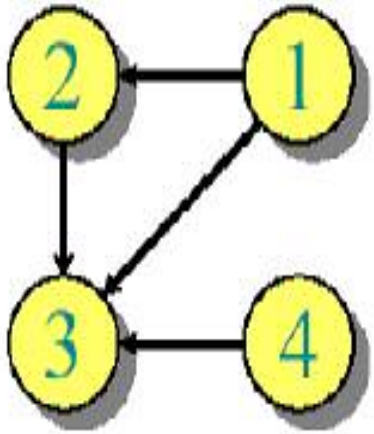


A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$\Theta(V^2)$ storage $\Rightarrow$ ***dense*** representation.

Greedy Design Technique

An ***adjacency list*** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v.



$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

For undirected graphs, $|Adj[v]| = degree(v)$.
For digraphs, $|Adj[v]| = out\text{-}degree(v)$.

Geedy Design Technique

MST (Minimum Spanning Tree):

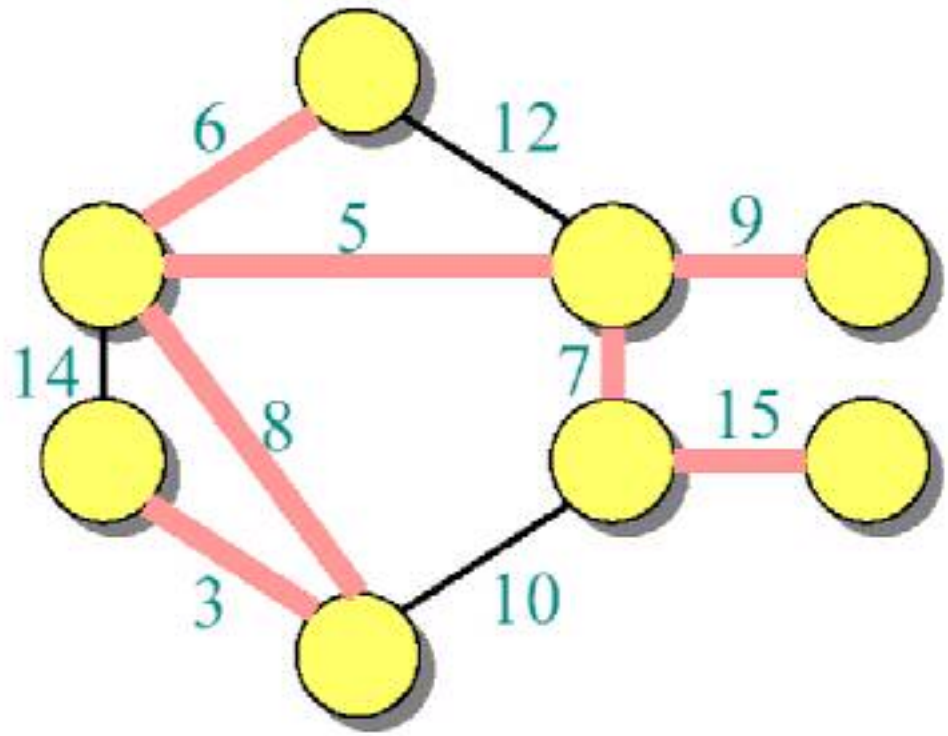
Problem Instance: a graph G in which each edge has a non-negative weight (or cost).

Feasible Solution: any spanning tree (a set of (n-1) edges, and consequently all vertices, containing no cycles).

Objective Function: $c(T) = \sum_{e \in T} c(e)$, where T is a spanning tree of G and c(e) is the cost of edge e.

Optimal Solution: the minimum cost spanning tree.

Geedy Design Technique



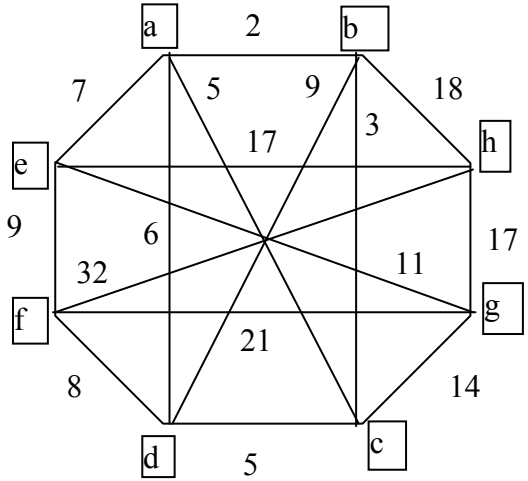
Geedy Design Technique

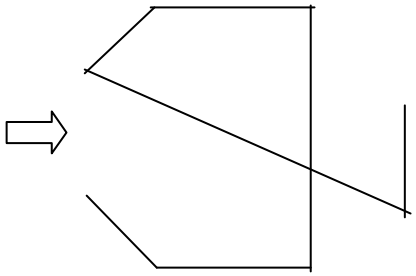
function kruskal (G = <N,E>:graph; length: E → ℜ+): set of edges

{initialization}
Sort the edges, E, of G in ascending order of the costs
n ← the number of nodes in N
T ← ∅ // initializes the solution to empty, will contain the edges of MST //

{greedy loop}
repeat
  e ← {u,v} //e be the next shortest edge of G in order of cost not yet considered//
  ucomp ← find(u) // which tells us in which connected node u is to be found//
  vcomp ← find(v) // which tells us in which connected node u is to be found//
// Check the feasibility, i.e., if T ∪ {e} does not contain a cycle //
if ucomp ≠ vcomp then
        union(ucomp,vcomp)   //merges two connected components//
        T ← T ∪ {e}
until T contains n-1 edges
return T

Geedy Design Technique





Step	Edge considered	Cost of edge	Cycle Created	Connected Components
Initial.	-	-	-	{a} {b} {c} {d} {e} {f} {g} {h}
1	{a,b}	2	none	{a, b} {c} {d} {e} {f} {g} {h}
2	{b,c}	3	none	{a, b, c} {d} {e} {f} {g} {h}
3	{c,a}	5	{a,b,c,a}	rejected
4	{c,d}	5	none	{a, b, c, d} {e} {f} {g} {h}
5	{d,a}	6	{a,b,c,d,a}	rejected
6	{a,e}	7	none	{a, b, c, d, e} {f} {g} {h}
7	{d,f}	8	none	{a, b, c, d, e, f} {g} {h}
8	{e,f}	9	{a,b,d,f,e,a}	rejected
9	{d,b}	9	{b,c,d,b}	rejected
10	{e,g}	11	none	{a, b, c, d, e, f, g} {h}
11	{c,g}	14	{e,a,b,c,g,e}	rejected
12	{g,h}	17	none	{a, b, c, d, e, f, g, h}
				c(T) = 17+11+8+7+5+3+2 = 53

Geedy Design Technique

We can evaluate the execution time of the algorithm as follows: On a graph with n nodes and E edges, the number of operations is in

• $\Theta(ElgE)$ to sort the edges, which is equivalent to $\Theta(Elgn)$ because $n-1 \leq E \leq n(n-1)/2$;

• $\Theta(n)$ to initialize the n disjoint sets

• There are at most 2E find operations and (n-1) merge operations on a universe containing n elements.

• At worst O(E) for the remaining operations

•We conclude that the total time for the algorithm is in $\Theta(ElgE)$.

Geedy Design Technique

To prove that Kruskal's algorithm produces a MST, we must first enumerate some properties of spanning trees:

1. Any spanning tree in a graph G of n vertices has n-1 edges.

2. If T is a spanning tree in a graph G, and e is any edge in G-T, then T+e contains a unique cycle C.

3. The removal of any edge e' of C from T+e (as in 2 above) produces a spanning tree T' = T + e – e'.

Geedy Design Technique

Theorem: The spanning tree T produced by Kruskal's algorithm is a MST. Any other MST, T', can be transformed to T by successively using the property (3).

Proof: Assume that $e_1,\ldots e_n$, the costs of the edges, are in increasing order. That is, $c(e_1) \leq \ldots \leq c(e_n)$.

- Let also e_j be the first edge in T but not in T'. Then by property (2) above, T' + e_j contains a unique cycle C.
- There must be some edge e_i of C that is not in T, for otherwise T would contain the cycle C, which is impossiblity.
- Then T" = T' + e_j – e_i is a spanning tree of G by property (3). The following cost relationship holds:
  $c(T") = c(T') + c(e_j) – c(e_i)$

Geedy Design Technique

The following cost relationship holds: $c(T'') = c(T') + c(e_j) - c(e_i)$

- $c(T'') \leq c(T')$ iff $j < i$, since the edges are sorted by cost in increasing order. Up to edge j, every edge in T is in T'.

- If $i < j$ and $e_i \notin T$, then adding e_i to T would have created a cycle C, all of whose edges precede e_j . However, if this were the case, C would be in T', which is impossible (since T' is assumed to be a MST). Therefore, $c(T'') \leq c(T')$, but T' is a MST by our original assumption. Thus, $c(T'') = c(T')$, which can happen only if $c(e_j) = c(e_i)$.
- Now, T'' must be a MST with one more edge in common with T than with T'. We can repeat the above process starting with T'' and repeat this as often as necessary (at most n-1 times) to obtain T, which proves that T is a MST.

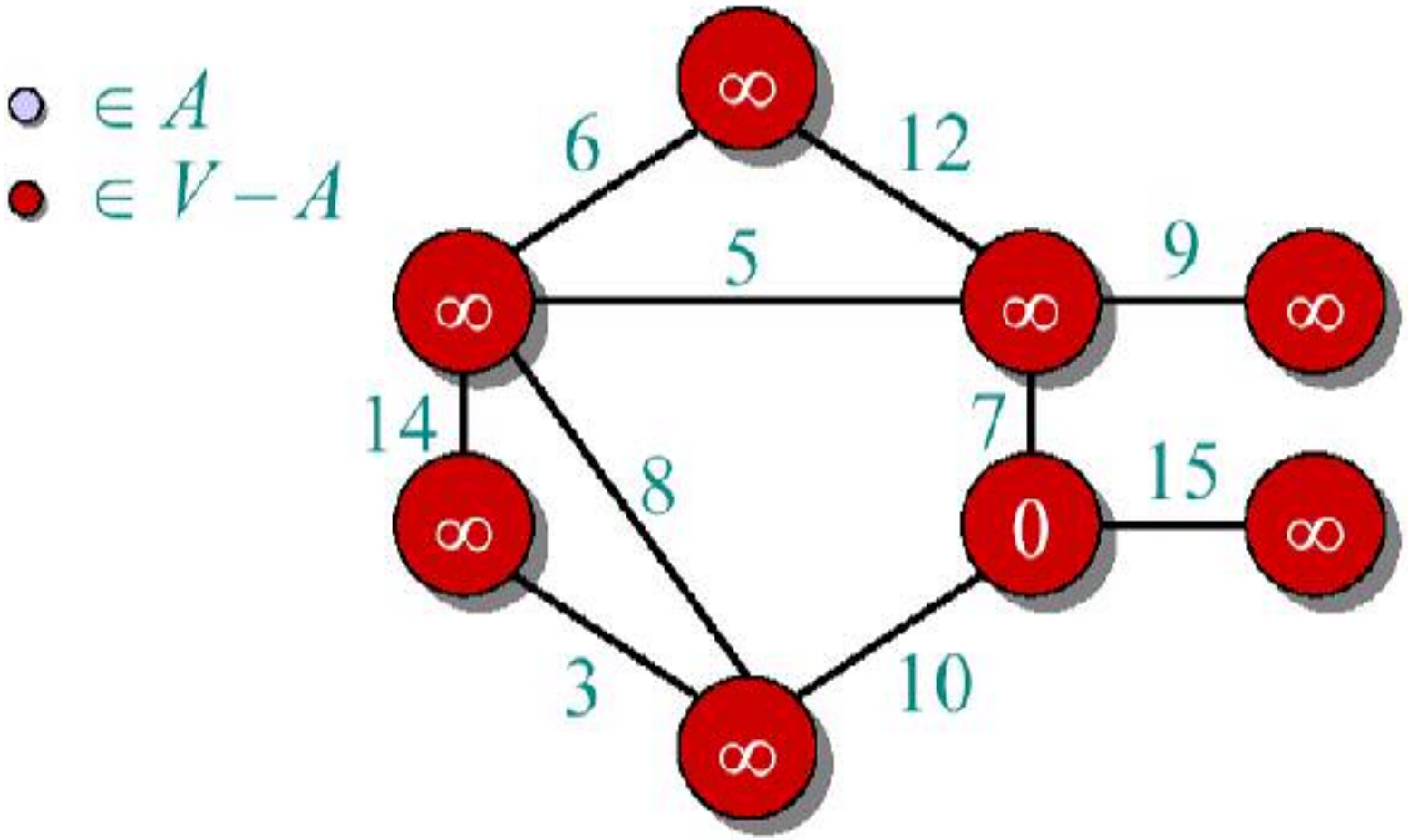
Prim's algorithm

IDEA: Maintain $V - A$ as a priority queue Q. Key each vertex in Q with the weight of the least-weight edge connecting it to a vertex in A.

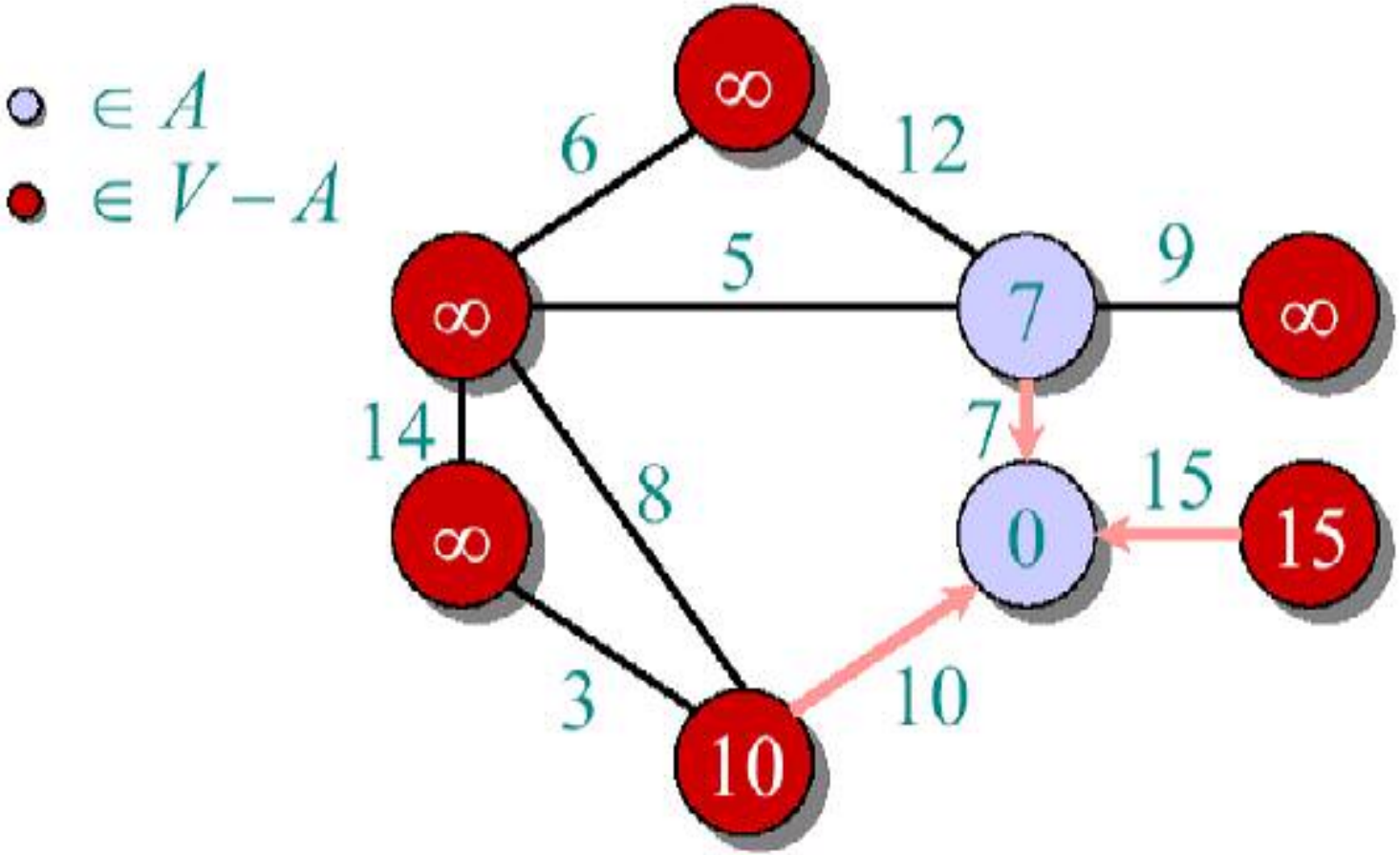
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        for each $v \in Adj[u]$
            do if $v \in Q$ and $w(u, v) < key[v]$
                then $key[v] \leftarrow w(u, v)$ ▷ DECREASE-KEY
                $\pi[v] \leftarrow u$

At the end, $\{(v, \pi[v])\}$ forms the MST.

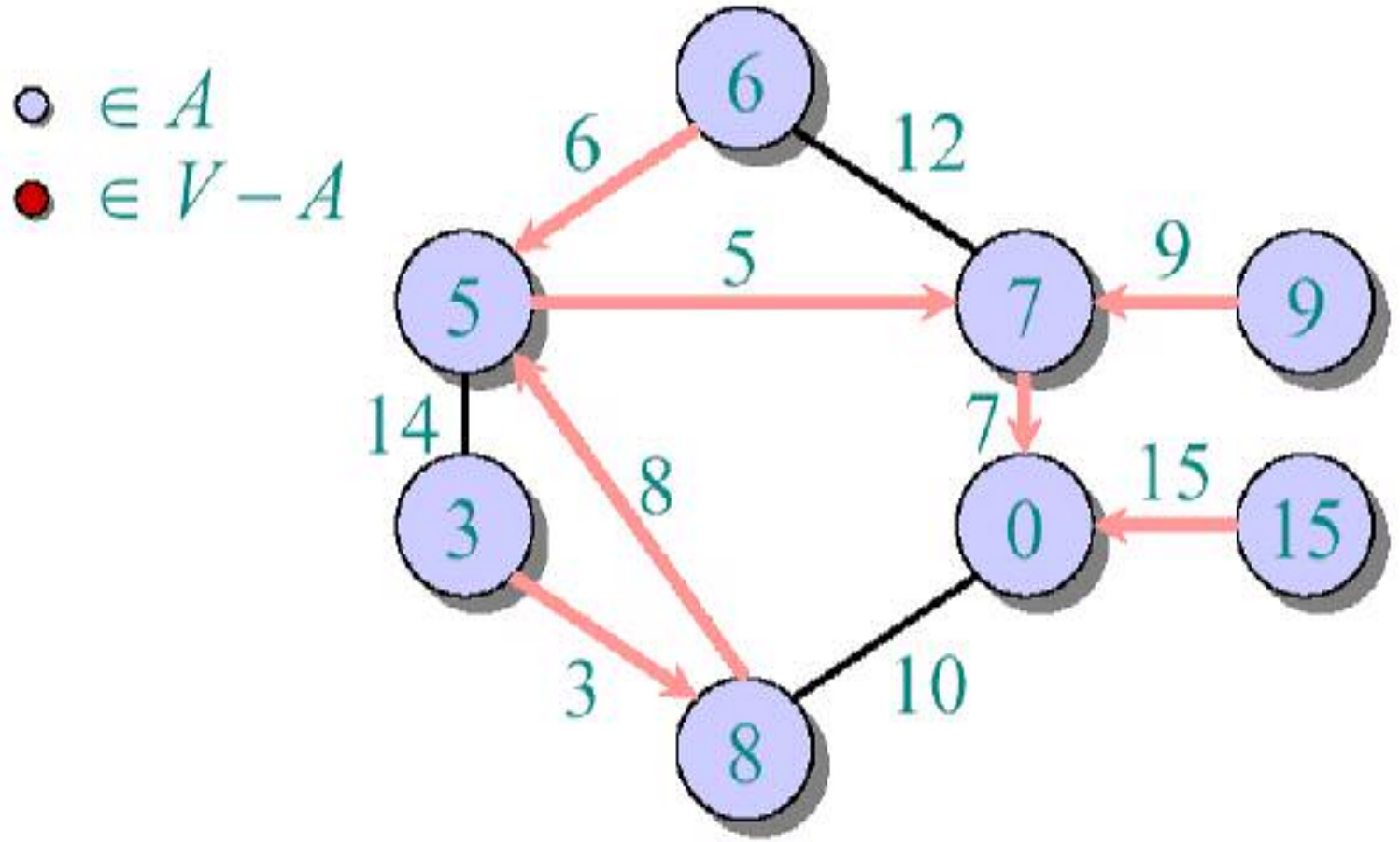
Example of Prim's algorithm



Example of Prim's algorithm



Example of Prim's algorithm



Analysis of Prim (continued)

$$\text{Time} = \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case

Geedy Design Technique

RKP(Rational Knapsack Problem):

Problem Instance: positive integers w_1, …, w_n, p_1, …, p_n, M (we have n objects, where object i has weight w_i and profit p_i, and M is the capacity of the knapsack.)

Feasible Solution: a vector $(x_1,\ldots x_n)$, where $0\le x_i\le 1$ and $\sum_{0\le i\le n} w_i x_i \le M$. In other words, the knapsack capacity cannot be exceeded. Note that we allow a fraction of an object to be placed into the knapsack.

Objective Function: $P = \sum_{0\le i\le n} p_i x_i$, where P is the profit associated with $(x_1,\ldots x_n)$

Optimal Solution: the maximum profit feasible solution.

Geedy Design Technique

Object Order	Profit
1, 2, 3	30 (highest profit 1st)
1, 3, 2	30
3, 2, 1	34 (lowest weight 1st)
3, 1, 2	29
2, 1, 3	**35** (highest profit density 1st)
2, 3, 1	34

Item	Profit	Weight	Density
1	30	10	3
2	20	5	4
3	2	1	2

Geedy Design Technique

```
function knapsack (w[1..n],p[1..n]): array [1..n]

{initialization}
for i = 1 to n do x[i] ← 0
weight ← 0

{greedy loop}
while weight < M do
        i ← the best remaining object (the objects are sorted in
                        terms of the profit density)
        if weight + w[i] ≥ M then x[i] ← 1
                weight ← weight + w[i]
        else x[i] ← (M – weight) / w[i]
                weight ← M
return x
```

Geedy Design Technique

Theorem: If objects are selected in order of decreasing pi/wi, then algorithm knapsack finds an optimal solution.

Proof: Suppose that $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$. Let $X = (x_1 \ \ldots \ x_n)$ be solution found by the greedy algorithm.

- If all the x_i are equal to 1, this solution is clearly optimal.
- Otherwise, let j be the smallest index such that $x_j \neq 1$. Looking at the way the algorithm works, it is clear that $x_i = 1$ for $1 \leq i < j$, $x_i = 0$ for $j < i \leq n$, and that $0 \leq x_i < 1$ and $\sum_{1 \leq i \leq n} w_i x_i = M$ and the value of the solution X be $P(X) = \sum_{1 \leq i \leq n} p_i x_i$.
- Now let $Y = (y_1 \ \ldots \ y_n)$ be an optimal solution. Since Y is feasible, $\sum_{0 \leq i \leq n} w_i y_i = M$ and the value of the solution Y be $P(Y) = \sum_{1 \leq i \leq n} p_i y_i$.
- Let k be the least index such that $y_k \neq x_k$. Clearly, such a k must exist. It also follows that $y_k < x_k$. To see this, consider the three possibilities: $k < j$, $k=j$, or $k > j$.

Geedy Design Technique

1. When $k < j$, $x_k = 1$. But $y_k \neq x_k$ and so $y_k < x_k$.
2. When $k = j$, then since $\sum_{1 \leq i \leq n} w_i y_i = M$ and $y_i = x_i$ for $1 \leq i < j$, it follows that either $y_k < x_k$ or $\sum_{1 \leq i \leq n} w_i y_i > M$.
3. When $k > j$, then $\sum_{1 \leq i \leq n} w_i y_i > M$ which is not possible.

- Now suppose we increase y_k to x_k and decrease as many of $(y_{k+1}, \ldots, y_n)$ as is necessary so that the total capacity used is still M. This results in a new solution $Z = (z_1, \ldots, z_n)$ with $z_i = x_i$, $1 \leq i \leq k$ and $\sum_{k<i\leq n} w_i(y_i - z_i) = w_k(z_k - y_k)$. Then for Z we have
- $\sum_{1 \leq i \leq n} p_i z_i = \sum_{1 \leq i \leq n} p_i y_i + (z_k - y_k)\, w_k p_k / w_k - \sum_{k<i\leq n} (y_i - z_i)\, w_i(p_j/w_j)$
  $\geq \sum_{1 \leq i \leq n} p_i y_i + [(z_k - y_k)\, w_k - \sum_{k<i\leq n} (y_i - z_i) wi]\, p_k / w_k$
  $= \sum_{1 \leq i \leq n} p_i y_i$

Geedy Design Technique

- If $\sum_{1 \le i \le n} p_i zi > \sum_{1 \le i \le n} p_i y_i$ then Y could not have been an optimal solution. If these sums are equal then either Z = X and X is optimal or $Z \neq X$. In this latter case, repeated use of the above argument will either show that Y is not optimal or will transform Y into X, showing that X too is optimal.

Geedy Design Technique

Example (The Task Sequencing Problem):

Problem Instance: a set of T of n tasks to be performed. For each task $t \in T$, there is a length l(t), a weight (or penalty) w(t) and a deadline d(t), all entries are positive integers.

Feasible Solution: any schedule of a subset T1 of T, in which at most one task is executed at a time, and such that each scheduled task finishes before its deadline. Scheduling function, s(t) satisfies the following properties:

if $s(t) < s(t')$, then $s(t) + l(t) \leq s(t')$, $\forall t,t' \in T1$

(each task terminates before any other one begins)

$s(t) + l(t) \leq d(t)$ $\forall t \in T1$

(each task terminates before its deadline)

Objective Function: $W = \sum_{t \in T\text{-}T1} w(t)$ = the sum of weights of unscheduled (tardy) tasks, i.e., the tardy task weight (T- T1 are the ones not able to schedule).

Optimal Solution: the schedule with the minimum tardy task weight.

Geedy Design Technique

Example (The Task Sequencing Problem):

A *pseudo-code* description of the algorithm is as follows:

Sort the tasks so that $w_1 \geq w_2 \geq \ldots \geq w_n$
T1 ← {}
For i ← 1 to n do
    If T1 $\cup$ $\{t_i\}$ is feasible then
        T1 ← T1 $\cup \{t_i\}$
$W = \sum_{t \in T-T1} w(t)$

Geedy Design Technique

Example: T = $\{t_1, t_2, t_3, t_4, t_5, t_6\}$
$w_1 = 7, w_2 = 5, w_3 = 4, w_4 = 3, w_5 = 2, w_6 = 1,$
$d_1 = 3, d_2 = 2, d_3 = 1, d_4 = 2, d_5 = 4, d_6 = 3,$
All lengths = 1.

Tasks considered (in increasing order of weights)	The schedule (based on the decreasing order of deadlines)
$T_1 \leftarrow \{\}$	$\{t_1\}$
$T_1 \leftarrow \{ t_1\}$	$\{t_2, t_1\}$
$T_1 \leftarrow \{ t_1 , t_2\}$	$\{t_3, t_2, t_1\}$
$T_1 \leftarrow \{ t_1 , t_2 , t_3\}$	$\{t_3, t_2, t_1\}$
$T_1 \leftarrow \{ t_1 , t_2 , t_3 , t_4\}$	not feasible
$T_1 \leftarrow \{ t_1 , t_2 , t_3 , t_5\}$	$\{t_3, t_2, t_1, t_5\}$
$T_1 \leftarrow \{ t_1 , t_2 , t_3 , t_5 , t_6\}$	not feasible

- Since t_4 and t_3 have the same deadlines, 2, we could not add task t_4 to the list, $\{t_3, t_2, t_1, t_4\}$
- Is not feasible. Similarly, t_6 and t_1 have the same deadlines, 3.

Therefore, $W = w_4 + w_6 = 4$.

Geedy Design Technique

Theorem: If all tasks have length = 1, then the greedy algorithm determines the minimum tardy task weight.

Proof: We give only an outline of the proof. Let T(G) denote the non-tardy tasks in the schedule found by the greedy algorithm and T(O) be the non-tardy tasks in an optimal solution. Also, let s(G) and s(O) denote feasible schedules for the tasks in T(G) and T(O), respectively. We also assume that s(G) schedules the tasks in T(G) in increasing order by deadline, so s(G) is feasible. The proof proceeds along the same lines as all previous proofs for greedy algorithms. That is, s(O) is transformed into another optimal schedule, s(O'), which has one more task in common with s(G) than does s(O). This process is repeated as often as necessary and eventually it is shown that s(G) = s(O).

Geedy Design Technique

Example (Multiprocessor Scheduling)):

Problem Instance: a set of T of n tasks. For each task $t \in T$, there is a length l(t), a positive integer. Also given a positive integer m, the number of available processors.

Feasible Solution: a schedule for all the tasks, such that each processor executes only one task at a time. All m processors are identical. Also, each task must be scheduled on only one processor without interruption.

Objective Function: the time when the last task finishes executing (this is the finishing time, denoted by FT)

Optimal Solution: the feasible schedule with the minimum finishing time.

- This problem is an NP-Complete problem, so we do not expect to find a polynomial time algorithm to solve it. A greedy approach does not do too badly, however. It can be shown that it always produces a schedule that is at most 1/3 longer than optimal.

Geedy Design Technique

Sort tasks so that $l_1 \geq l_2 \geq \ldots \geq l_n$

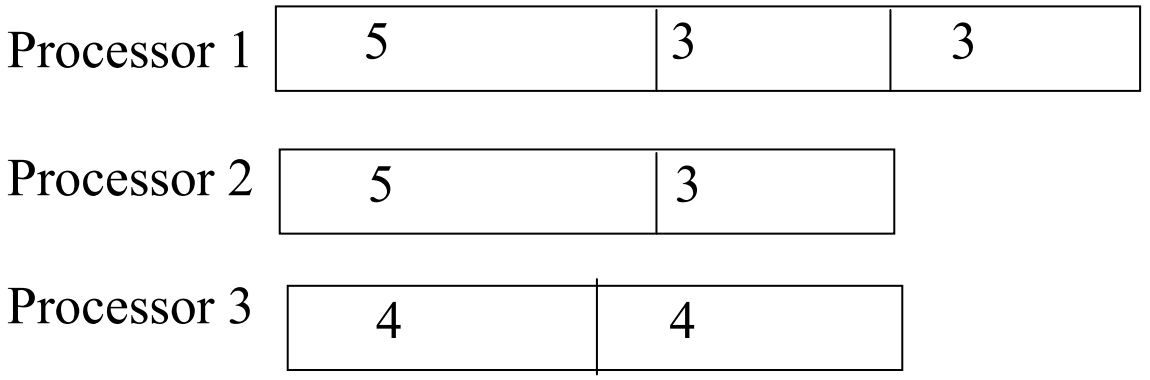
```
For i  = 1 to m do
    T(i) = 0; // T(i) keeps track of when processor i becomes free//
For j = 1 to n do
Begin
        Mini = 1;
        For i = 2 to m do
          If T(i) < T (mini) then
                  Mini = i; // this finds the next free processor//
        Assign t_j to processor mini at time T(mini)
        T(mini) = T(mini) + l_j;
End
        FT = max {T(1), T(2),…,T(m)}
```

--

The complexity of this algorithm is Θ (n.m).

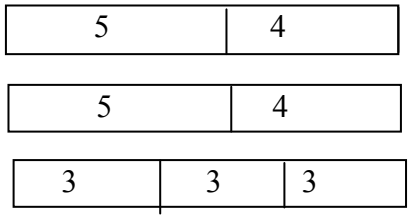
Geedy Design Technique

Example: Let us assume 7 jobs to be scheduled and their lengths are 5,5,4,4,3,3,3 and There are 3 processors available.
The greedy schedule is as follows:



FT = 11

The relative error is 2/9. It can be shown that the maximum relative error is 1/3–1/(3m). In the case of m = 3, this is 1/3 – 1/9 = 2/9.



Optimum Soln: FT = 9

Geedy Design Technique

Example (Compressing Data): Hoffman Coding

Problem Instance: a code, C, consists of a set of n characters. For each character $c \in C$, there is a frequency f(c), a positive integer. Also there is a priority queue Q, keyed on f, is used to identify the two least-frequent objects to merge together.

Feasible Solution: a tree, T, for n characters, each character has a codeword representing the frequency of a character, $c \in C$, is the sum of the frequencies of all characters in the code.

Objective Function: the total frequencies = $f(T) = \sum_{f \in T} f(c_i)$, where $1 \leq i \leq n$, T is a tree of C and f(c) is the frequency of c.

Optimal Solution: the minimum cost tree representing the Hoffman code.

Geedy Design Technique

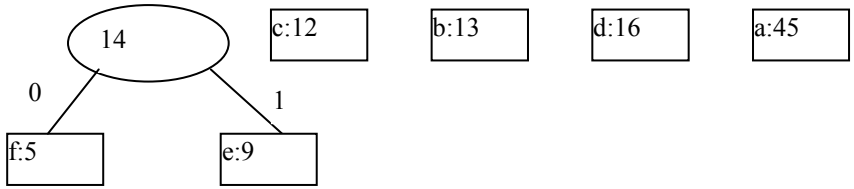
Hoffman invented a greedy algorithm that constructs an optimal prefix code called a Huffman code. The algorithm builds the tree T corresponding to the optimal code in a bottom up manner. It begins with a set of |C| leaves and performs a sequence of |C| + 1 “merging” operations to create the final tree.

```
HUFFMAN ( C )
n ← |C|
Q ← C
For i ← 1 to n-1
        do z ← Allocate-Node()
          x ← left[z] ← Extract-min (Q)
          y ← right[z] ← Extract-min (Q)
          f[z] ← f[x] + f[y]
          Insert (Q,z)
  Return Extract-Min (Q)
```

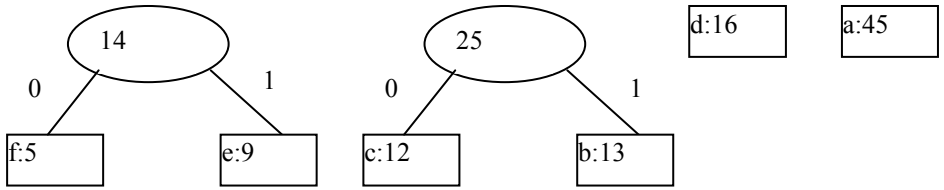
Geedy Design Technique

1. step:

2. step:

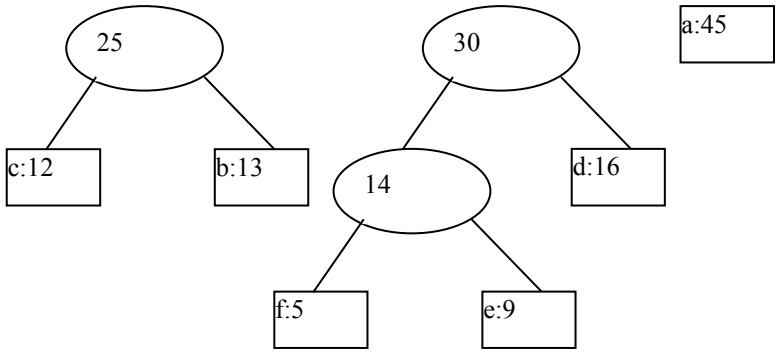


3.step:

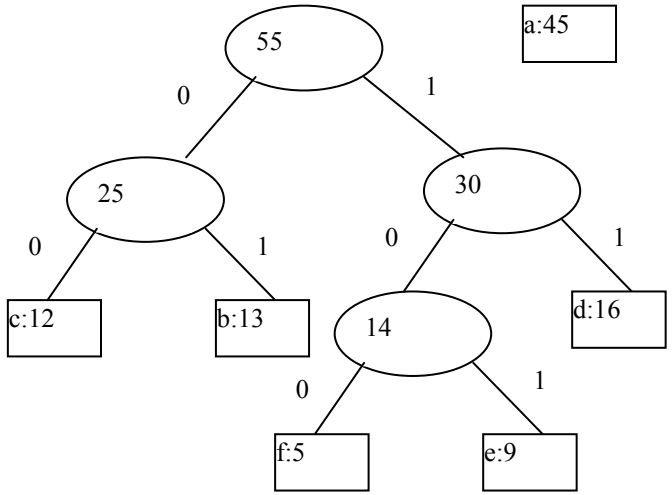


Geedy Design Technique

4. step:

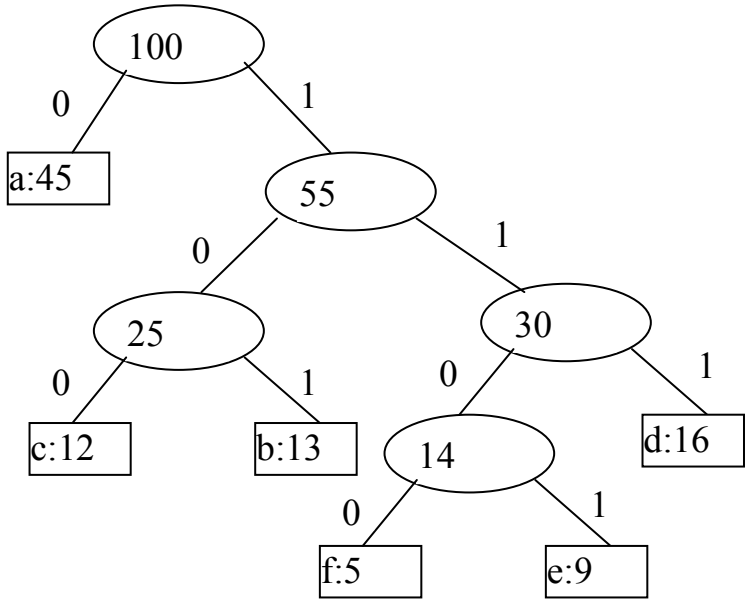


5.step:



Geedy Design Technique

6.step:



	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Variable-length codeword	0	101	100	111	1101	1100

This code requires 224,000 bits = ((45*1 + 13*3 + 12 *3 + 16*3 + 9*4 + 5*4) * 1000), which is an optimal character code for this file.

Geedy Design Technique

Complexity analysis:

• The analysis of the running time of Huffman's algorithm assumes that Q is implemented as a binary heap.

• For a set C of n characters, the initialization of Q can be performed in O(n) time using *Build-heap* procedure.

• The for loop is executed exactly n-1 times, and since each heap operation requires time O(lgn), the loop contributes O(nlgn) to the running time.

• Therefore, the total running time of Huffman's algorithm on a set of n characters is O(nlgn).

CONCLUSIONS

Greedy Design technique does not guarantee the best solutions for all the problems that are applicable.
Threrefore a proof must come with a solution.

- For some problems it finds the optimum soln.
- For some problems it is applicable to some specific instances of the problem.
- For some problems it finds a good solution, as an approximation algorithm.
- When it is applicable usually the computational complexity of the solution is quite good.