GREEDY METHOD

ACTIVITY SELECTION PROBLEM

Suppose you have a series of lectures, all scheduled in one room.

Each Lecture is denoted by a start time and an end time.

You cannot have two lectures in the room at the same time

- Two lectures are compatible if their start and finish time do not overlap

What is the maximum number of lectures you can schedule in a day ?

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

NAÏVE SOLUTION

Select all the possible subsets of the activities

Find the most non-overlapping ones to fit within the time frame.

Time Complexity $O(2^n)$

RECURSIVE SOLUTION

Arrange the events by the finishing time

- Not start time, because the maximum time allowed is defined by the latest finishing time

Let i be the activity that can be finished before any other activities in the set;

- fi is the minimum among all finishing times

Let j be the activity that will be finished after any other activities in the set;

- fj is the maximum among all finishing times

Let the maximum number of events that can be scheduled between start of event i and end of event j be c[i,j]

RECURSIVE SOLUTION

Let ak be an activity between activities i and j

- fi <=sk <fk<=fj

Let the maximum number of events that can be scheduled between start of event i and end of event j be c[i,j]

Sij=set of events from which the optimum set is to be selected. All events with finishing time between fi and fj

This is also very expensive
Can we do better ?

Acitivity_Selection(Sij, c[i,j])

- If Sij is the null set ; c[i,j]=0
- Else
- T[k]=0;
- For every ak ϵ Sij
  - X=Acitvity_Selection (Sik, c[i,k])
  - Y=Acitvity_Selection (Skj, c[k,j])
  - T[k]=X+Y+1
- C[i,j]=max(T[k]) for all ak ϵ Sij

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset\,, \\ \max\limits_{\substack{i<k<j \\ a_k \in S_{ij}}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset\,. \end{cases}$$

REDUCING RECURSION

Activity_Selection(Sij, c[i,j])

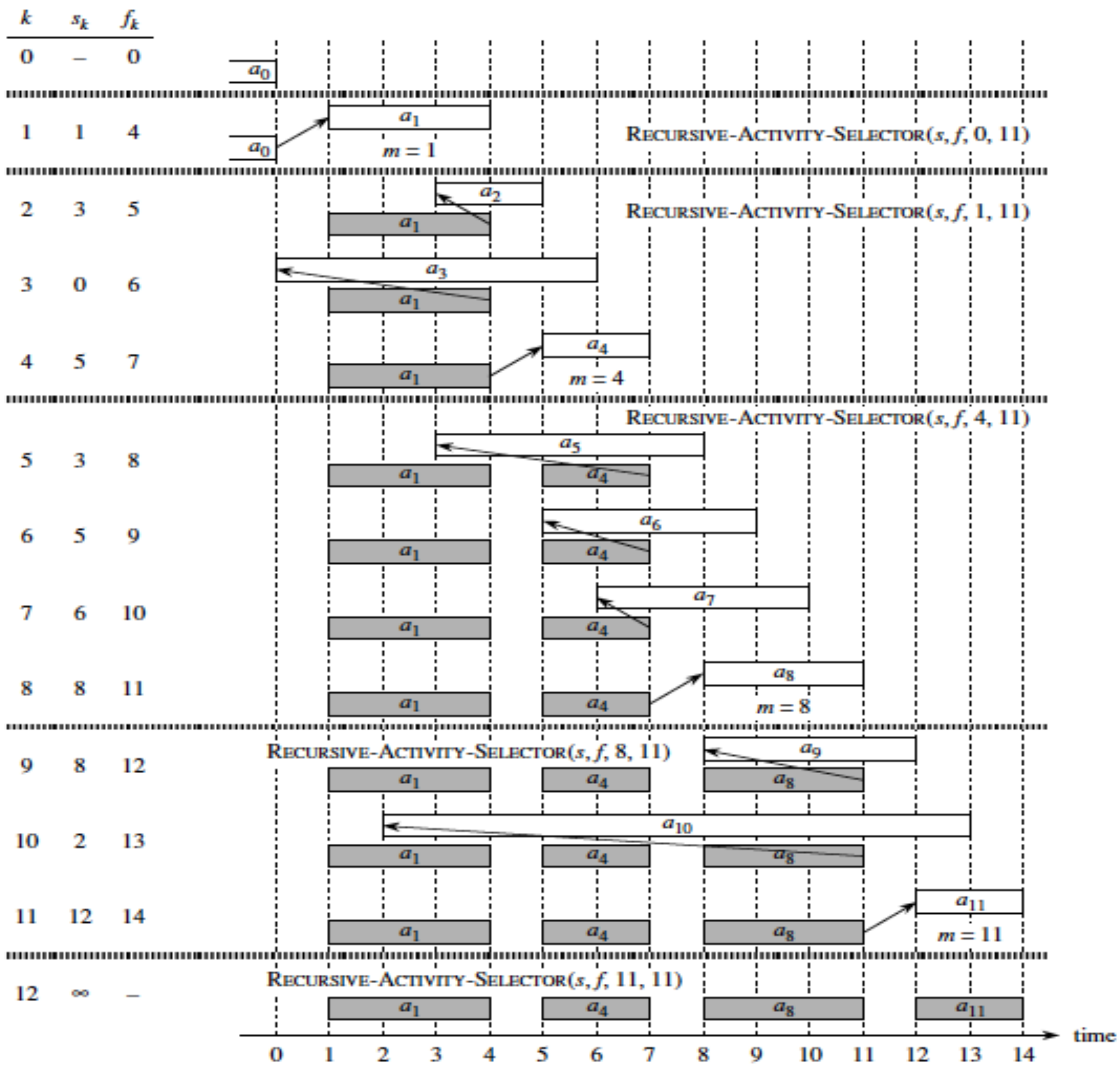
- If Sij is the null set ; c[i,j]=0
- Else
- T[k]=0;
- For every ak ϵ Sij
  - X=Acitvity_Selection (Sik, c[i,k])
  - Y=Acitvity_Selection (Skj, c[k,j])
  - T[k]=X+Y+1
- C[i,j]=max(T[k]) for all ak ϵ Sij

Activity_SelectionX(Sij, c[i,j])

- If Sij is the null set ; c[i,j]=0
- Else
- T[k]=0;
- Select ak with minimum finish time
- X=Acitvity_SelectionX(Sik, c[i,k])
- C[i,j]=X+1

Take ak as the activity with the earliest finishing time (greedy selection)
Then recursion reduces to a linear problem

Does this guarantee optimum set, i.e. most number of activities ?



k s_k f_k
0 – 0
1 1 4
2 3 5
3 0 6
4 5 7
5 3 8
6 5 9
7 6 10
8 8 11
9 8 12
10 2 13
11 12 14
12 ∞ –
RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, 11)
m = 1
RECURSIVE-ACTIVITY-SELECTOR(s, f, 1, 11)
m = 4
RECURSIVE-ACTIVITY-SELECTOR(s, f, 4, 11)
m = 8
RECURSIVE-ACTIVITY-SELECTOR(s, f, 8, 11)
m = 11
RECURSIVE-ACTIVITY-SELECTOR(s, f, 11, 11)
time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

ELEMENTS OF GREEDY METHOD

Determine optimal substructure of the problem

Develop a recursive solution

Show that at stage of the recursion, only one of the choices (the greedy choice) is best.

Show that if this greedy choice is made, the correct answer is obtained

PROOF OF CORRECTNESS OF GREEDY SELECTION:I

S1:If am is the activity with the earliest finishing time finishing time, then Sim will be empty

- Recall recusive formula
  - C[i,j]=c[i,m] +c[m,j]+1 (when am is selected)
- If am is the activity with earliest finishing time, then there can be no activity scheduled before it.

PROOF OF CORRECTNESS OF GREEDY SELECTION:II

S2: There is an optimum set of activities that start with am (the activity with earliest finishing time)

- Suppose Pij is an optimum set of activities that does not include am
- Let the first activity in Pij be ap
- We can exchange ap with am
  - Because fp > fm, exchanging am will not conflict with other choices
  - Thus we have an optimal set that begins with am

PROOF OF CORRECTNESS OF GREEDY SELECTION:III

Assume that the set of activities given by the greedy algorithms is Y={y1, y2,….,yn}

Assume that the set of activities given by an optimal algorithm is O={o1, o2,…ok}

We will show |O| (size of set O)=|Y|(size of set Y)

We will do this by showing that all events in O can be replaced with the events in Y

PROOF OF CORRECTNESS OF GREEDY SELECTION:IV

In both the sets, the events are arranged in increasing order of their finishing time

Let for events 1 to l oi=yi, so we do not have to replace

So we can consider the set from {yi+1,…yn} and {oi+1,…,ok} (as per S1)

PROOF OF CORRECTNESS OF GREEDY SELECTION:V

Let the jth event be the first where oi ≠ yi

As per S2 we can replace yi with oi

- This is because finishing time of yi has the earliest finishing time among the remaining events
- Thus the finishing time is less than equal to oi and will not conflict with the remaining events in O
- …Thus we have reduced the sets to be compared to {y(j+1)…yn} and {o(j+1)…ok}
- If we continue we will eventually replace on with yn

PROOF OF CORRECTNESS OF GREEDY SELECTION:VI

If there are any events left in O={o(n+1),…ok}, they should have been picked up by the greedy algorithm

This is because they do not conflict with yn, and are part of the set of activities

Since yn is the last event in Y, thus no other activities were selected

Thus on is the last element in O

Hence all events of O can be replaced in all events in Y

|O|=|Y|

EXAMPLE

Y={1,4,8,11}

O={2,4,9,11}

Step 1; replace 2 with 1

- {1,4,8,11}; {1,4,9,11}

Step 2 ; to compare {8,11} and{9,11}; replace 9 with 8

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

RECAP

Can be expressed as a recursive problem similar to divide and conquer and dynamic programming

However, only one subproblem need to be solved

The subproblem selected is the one that gives the optimal value in the current situation

Algorithms must be accompanied with proofs to show why greedy selection works

FRACTIONAL KNAPSACK PROBLEM

- You have a bag that can hold upto W pounds. You have n items , where the value of the ith item is v_i and the weight is w_i. How will you fill your bag such that you maximize your value
- In this case the items can be divided divided---rice, coffee beans, gold dust, etc.

FRACTIONAL KNAPSACK PROBLEM

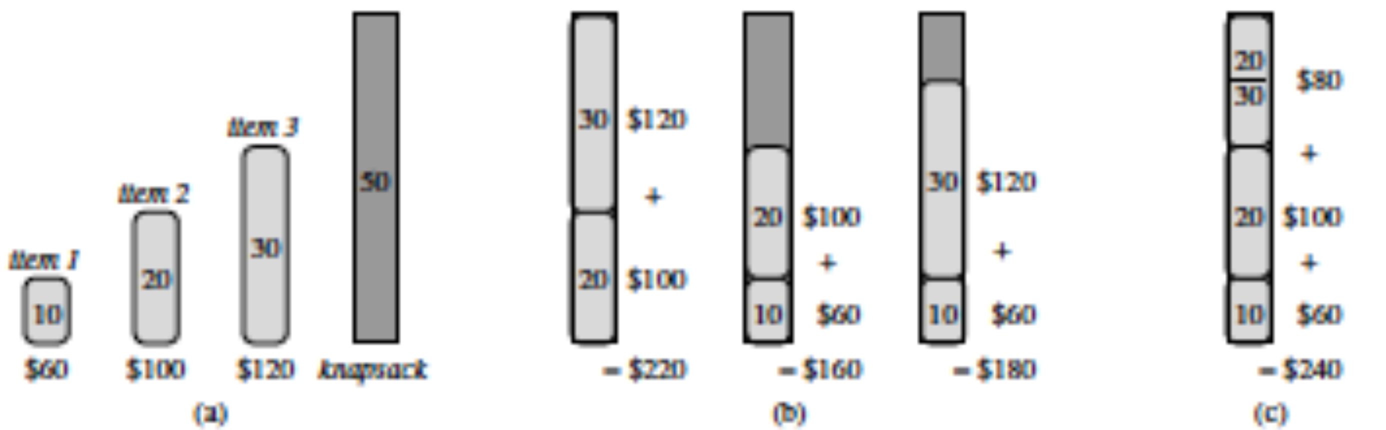
We can try the dynamic programming route and recursively select an item or not per subset

- However, since the items can be subdivided we have to consider all possible fractions
- That will be enormously expensive

Algorithm

- Find the cost per weight of each item v_i/w_i
- Sort them in decreasing order—highest first (Onlogn)
- Pick the most valuable item. If all of it is taken and there is space left, pick the next valuable item and so on….

FRACTIONAL KNAPSACK PROBLEM



PROOF OF CORRECTNESS-I

Assume the items are numbered 1 to n in decreasing order of their value by weight
- (x1,x2,x3,….,xn)

Let Y be the solution obtained by the greedy algorithm
- (y1,y2,y3,…yk,…,yn)
- Weight yi of item xi is selected
- Considering weight as a percentage, the value of yi would be
  - 1,1,1,…,f,(0,0,..0); where f=yk
  - Everything before yk is taken completely, then f percentage of yk is taken which fills up the weight.

Let O be the solution obtained by an algorithm that gives the most value
- (o1,o2,o3,…..ok,..,on)
- Weight oi of item xi is selected

PROOF OF CORRECTNESS-II

If for all items i oi=yi then we are done

Otherwise let us assume that the first item where the values do not match is the jth item

Case I: this is not the kth item then, yj>oj

Let oj+dj=yj

So we add dj to oj to make the weight come up to yj

We delete weight d from the remaining items of oj+1…oj to make up the extra weight

Since we are deecreasing weights of more expensive items and increasing weights of less expensive items, the value is decreased or stays the same.

It should stay the same since we already had optimal value for the max weight

Now weight of items 1 to j, and we can continue the same way.

PROOF OF CORRECTNESS-III

The first mismatch is when the item is the kth item

If yk > ok, it is case 1

Case 2: If yk<ok

Since from 1 to k-1 the weights are equal, therefore the amount of weight left is yk. Thus ok has to be equal to yk

EXAMPLE

Maximum Weight

- 6 pounds

Items :

- 1: 1 pounds=>$6 ; v/w=6
- 2: 2 pounds=>$10; v/w=5
- 3: 5 pounds =>$25; v/w=5

Solution 1 (Greedy)

- Item 1(1), 2(2), 3(3) =$6+$10+$3*5=$31

Optimal Solution

- Item 1 (1),2(1), 3(4)=$6+$5+$20=$31

EXAMPLE

List of items in decreasing order of v/w {1,2,3}

Y={1,2,3}

O={1,1,4}

Step 1; add 1 to item 2 in O and remove 1 from item 3

- O={1,2,3}=$31

HUFFMAN CODING

Consider a string that has only six letters, with varying frequencies

- A(45%); B(13%); C(12%); D(16%); E(9%); F(5%)

How would you encode the letters using binary digits?

ENCODING THE LETTERS

A=000; b=001, C=010, D=011, E=100, F=101

If there are 100,000 characters in the string then space taken= 3*100,000=300,000 bits

A more efficient coding

A=0, B=101, C=100, D=111, E=1101, F=1100

Going by the frequencies space taken

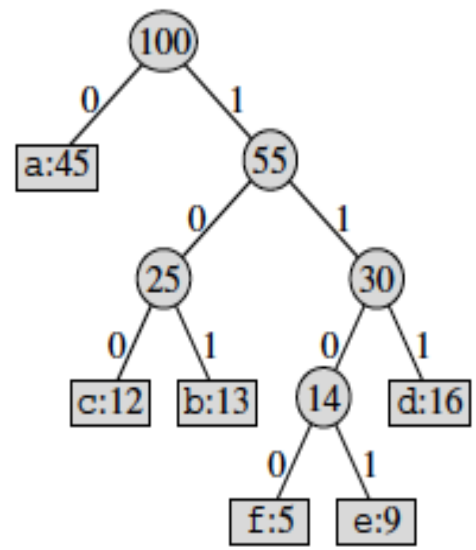
- (45*1+13*3+12*3+16*3+9*4+5*4)*1000=224,000

Would this coding be easy to decode ? How would you do it ?

PREFIX CODES

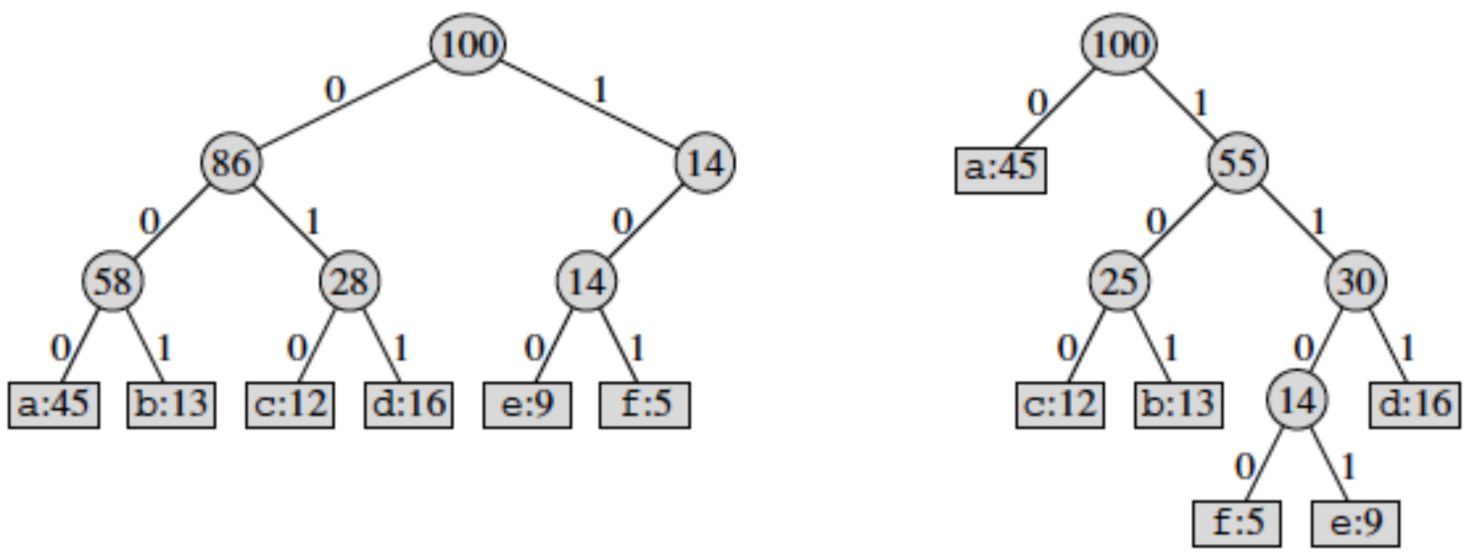
Prefix codes are codes in which no code word is also a prefix of some other code word.

If the letters are code using prefix code, we can use a binary tree to decode

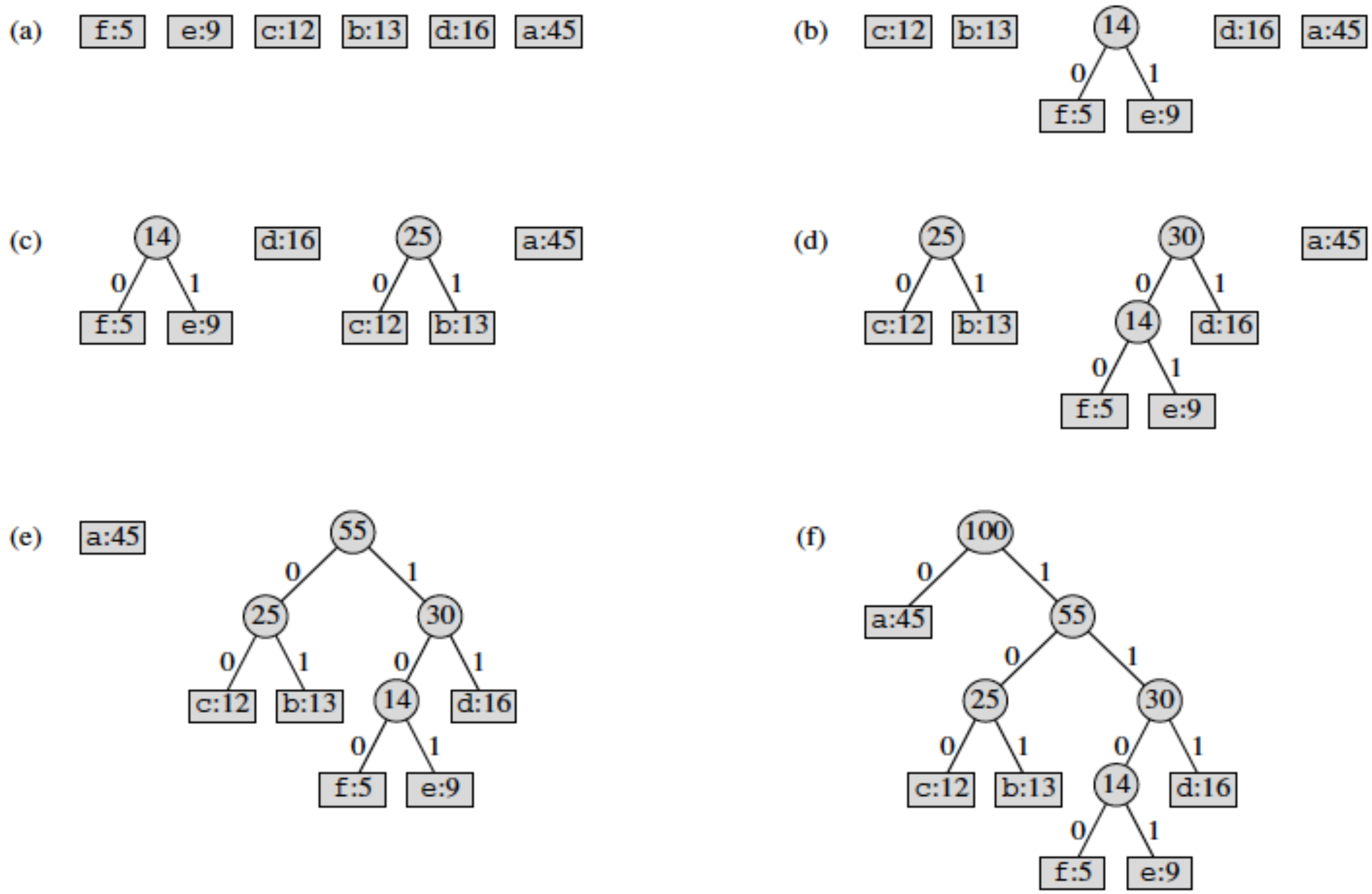


PREFIX CODES

Binary Trees corresponding to prefix codes are always full binary trees. Every non-leaf node has two children. Not so for fixed length code



HUFFMAN CODING



HUFFMAN CODING

```
HUFFMAN(C)
n ← |C|
Q ← C
for i ← 1 to n − 1
        do allocate a new node z
           left[z] ← x ← EXTRACT-MIN(Q)
           right[z] ← y ← EXTRACT-MIN(Q)
           f[z] ← f[x] + f[y]
           INSERT(Q, z)
return EXTRACT-MIN(Q)                ▷ Return the root of the tree.
```

Complexity O(nlogn)

PREFIX CODE

If the string has C different characters then the tree will have C leaves

Since it is a full binary tree, therefore it will have C-1 leaves

The length of the code representing a character is equal to the depth of the tree d(T)

Let the frequency of the character be f(C)

Number of bits to store the string (cost of the tree)

- Goal: to minimize this cost

$$B(T) = \sum_{c \in C} f(c) d_T(c) ,$$

PROOF OF CORRECTNESS-I

Let x and y be two characters of the lowest frequencies.

Then there exists an optimal prefix code for C in which the code for x and y have the same length and differ only in the last bit

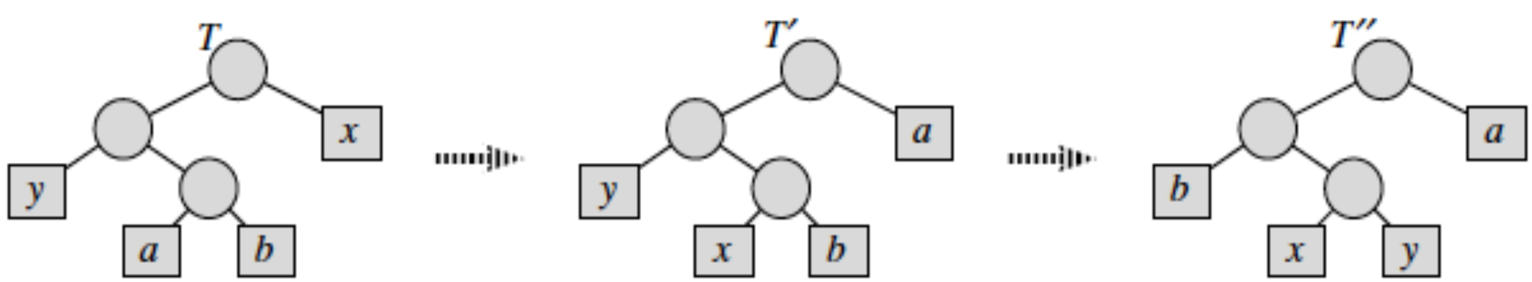
Thus x and y will be leaves with same parent in tree (T'')

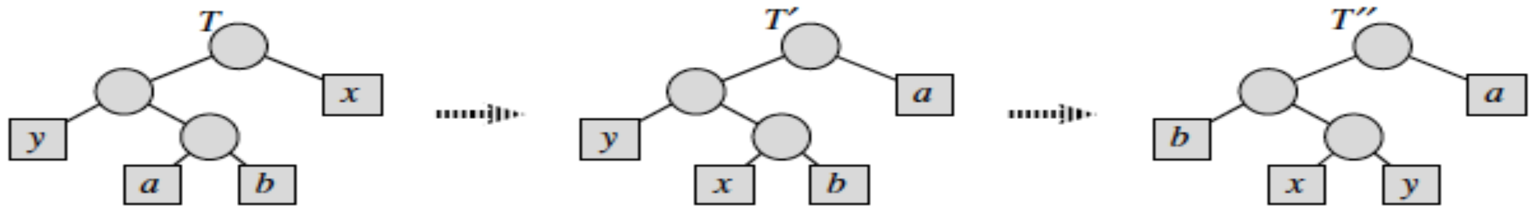
Let there be an optimal Huffman code where x and y are not siblings and not on the lowest branch.

Instead a and b are siblings in the lowest branch(T)

We will transform from T to T''

We can then remove the parent node of x and y, and recursively transform the rest of the trees





CORRECTNESS-II

In the String

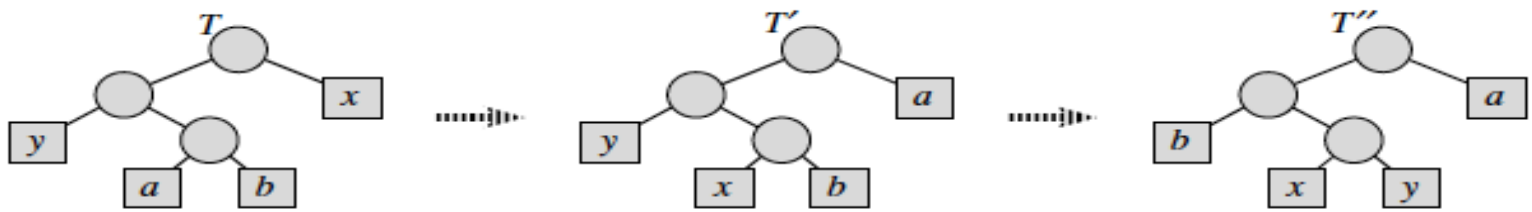
- Frequency of a=fa
- Frequency of b=fb
- Frequency of x=fx
- Frequency of y=fy

In T

- Depth of a=da
- Depth of b=db
- Depth of x=dx
- Depth of y=dy

In T" (exchange position of a and x ; y and b

- Depth of a =Depth of x in T=dx
- Depth of b =Depth of y in T=dy
- Depth of x =Depth of a in T=da
- Depth of y =Depth of b in T=db



CORRECTNESS-III

Difference in Storage in T and T'', assuming all other letters are not moved

In T

- fa*da+fb*db+fx*dx+fy*dy

In T''

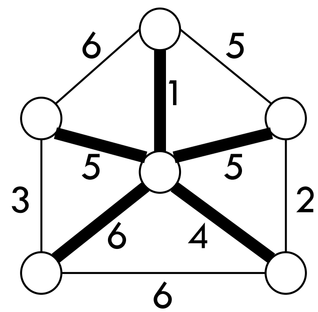
- fa*dx+fb*dy+fx*da+fy*fb

Difference T-T''

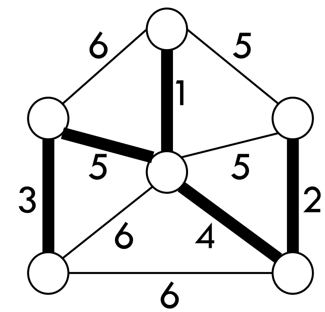
- (fa-fx)(da-dx)+(fb-fy)(db-dy)
- Since da>=dx; db>=dy; fa>=fx; fb>=fy ; difference is >=0
- Which means T>=T''
- But T has the minimum storage according to assumption
- Thus T=T'' and Huffman coding also produces minimum storage.

Spanning Tree

- A *spanning tree* of a graph G is a tree that contains every vertex of G.
- The *weight* of a tree is the sum of its edges' weights.



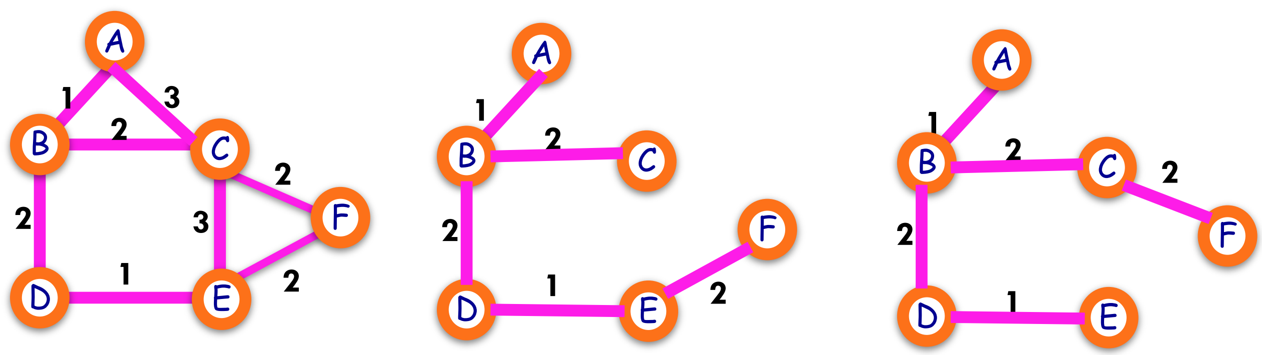
weight = 21



weight = 15

- A *minimal spanning tree* is a spanning tree with lowest weight.

MINIMUM WEIGHTED SPANNING TREE (MST)



Prim's Method (1957):

Start with a vertex

Pick the smallest weighted edge

Keep picking smallest edge going out of the current tree

Do not form cycles

Done when all vertex are connected

Kruskal's Method (1956):

Start with set of disconnected nodes

Pick the smallest weighted edge

Keep picking smallest edge from the set of all edges

Do not form cycles

Done when all vertex are connected

DISJOINT SET UNION FIND

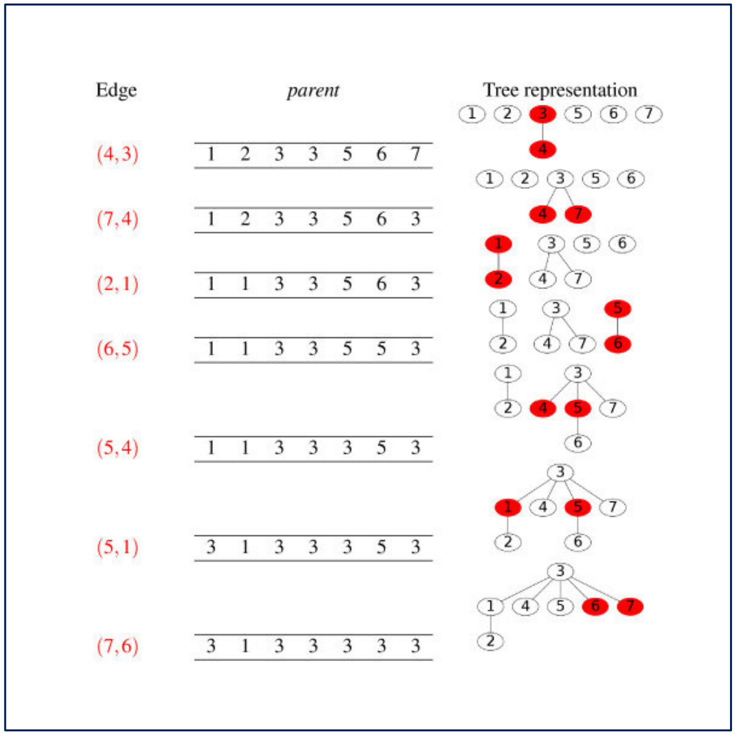


Image from Ultra-fast sequence clustering from similarity networks with SiLiX

COMPLEXITY

Prim's Algorithm

Time to sort edges by weight in heap = O(ElogE)

Find smallest edge =O(1)

Check if new vertex added =O(V)

Together it comes to O(ElogE)=O(ElogV), since E<=V*V

Can go down to O(VlogV) when sorted using vertices and using Fibonacci heap.

Kruskals's Algorithm

Initialization of vertices for Union Join =O(V)

Time to sort edges by weight in heap = O(ElogE)

Time to extract minimum weighted edge =O(1)

Time to check for cycle using union find=O(VlogV)

Together it comes to O(ElogE)=O(ElogV), since E<=V*V

DIJKSTRA'S ALGORITHM

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S ← Ø
Q ← V[G]
while Q ≠ Ø
      do u ← EXTRACT-MIN(Q)
         S ← S ∪ {u}
         for each vertex v ∈ Adj[u]
             do RELAX(u, v, w)
```

Start at a vertex

Add vertex to priority queue;

priority is lowest weight first

While priority queue not empty

Extract first element in queue; element with lowest distance

If not visited

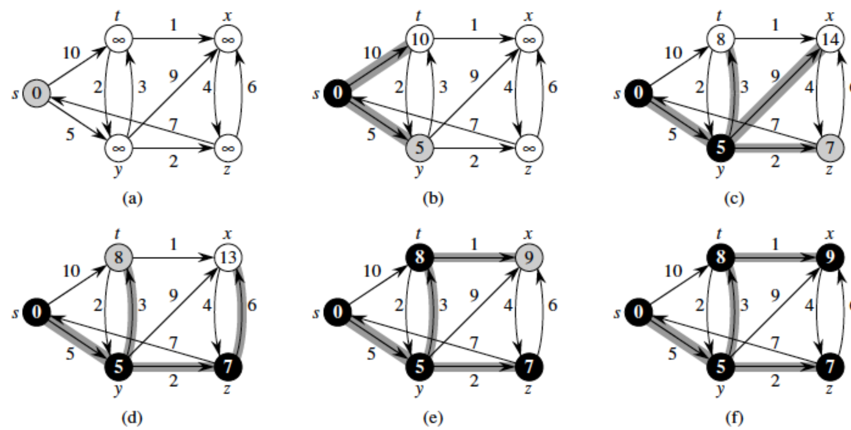
Mark as visited

Relax distance to neighbors

Add all unvisited neighbors to queue

End if

End while



EXAMPLES

For each problem, give the algorithm and the proof

Cost of merging two sorted arrays is equal to the sum of their length. If you have more than two sorted arrays in which order would you merge them to minimize the cost

- Example arrays of length 4,2,3,6,5

We want to obtain a given amount V, using the fewest number of dollar bills. We have dollar bills of the following denominations ={1,2,5,10,20,50,100}. We can use as many bills as necessary

- Example what is the fewest number of bills to make up $70