§The Greedy Method

Suppose that a problem can be solved by a sequence of decisions. The greedy method has that each decision is locally optimal. These locally optimal solutions will finally add up to a globally optimal solution.

Only a few optimization problems can be solved by the greedy method.

e.g. pick k numbers out of n numbers such that the sum of these k numbers is the largest.

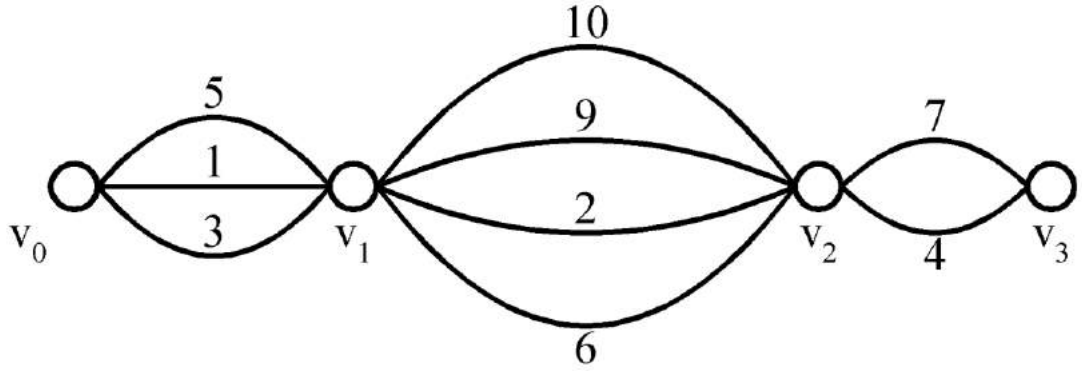
algorithm:

FOR i = 1 to k

pick out the largest number and delete this number from the input.

ENDFOR

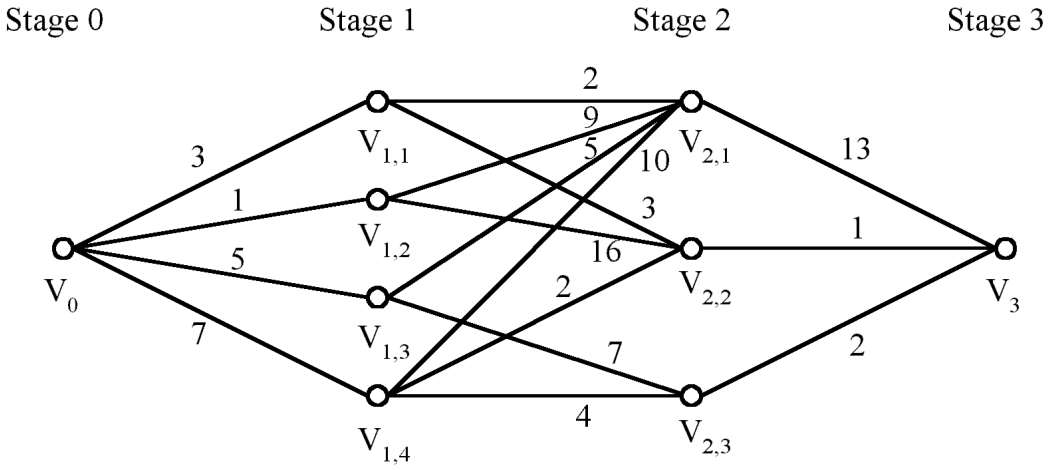
e.g. find a shortest path from v_0 to v_3.



The greedy method can solve this problem.

The shortest path: 1 + 2 + 4 = 7.

e.g. find a shortest path from v_0 to v_3 in the multi-stage graph.



greedy method: $v_0 \xrightarrow{1} v_{1,2} \xrightarrow{9} v_{2,1} \xrightarrow{13} v_3 = 23$

optimal: $v_0 \xrightarrow{3} v_{1,1} \xrightarrow{3} v_{2,2} \xrightarrow{1} v_3 = 7$

The greedy method does not work.

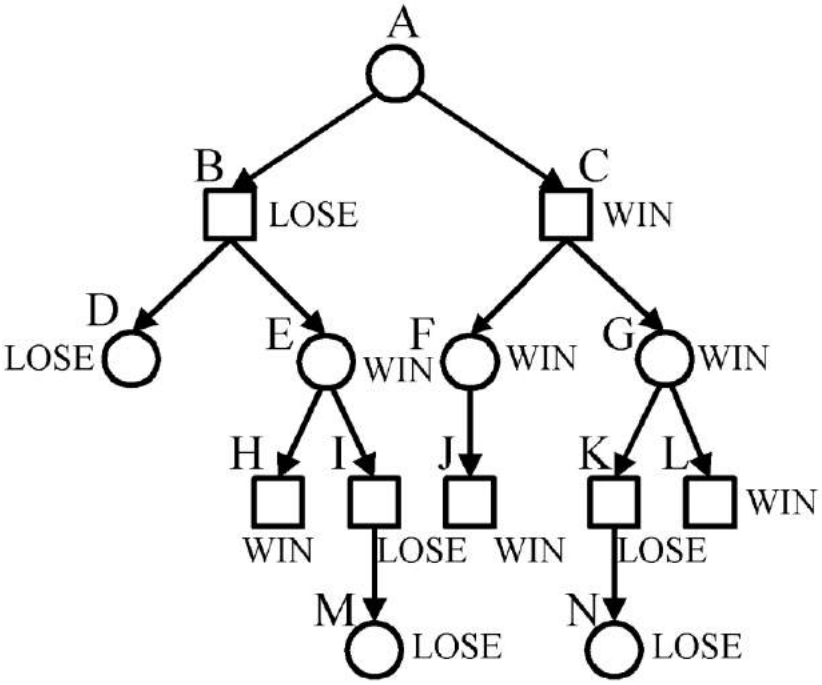
How to solve?

dmin(i,j): minimum distance between i and j.

$$dmin(v_0,v_3)=\min\begin{cases} 3+dmin(v_{1,1},v_3) \\ 1+dmin(v_{1,2},v_3) \\ 5+dmin(v_{1,3},v_3) \\ 7+dmin(v_{1,4},v_3) \end{cases}$$

This problem can be solved by the dynamic programming.

e.g. How to win in a game?



The greedy method does not work because it never looks ahead.

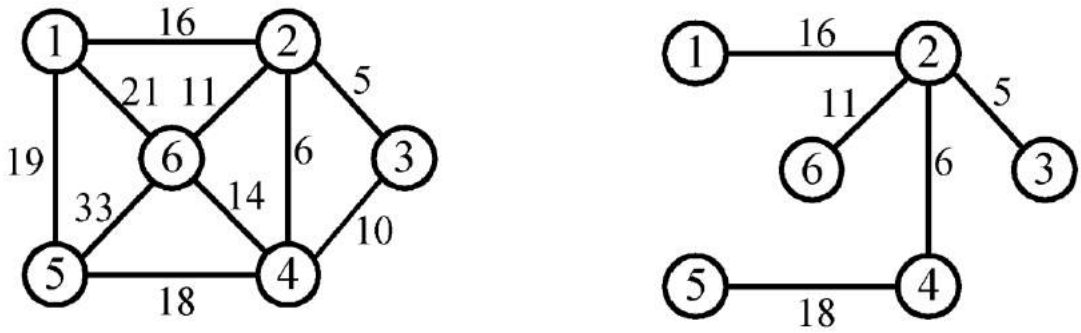
● Minimal Spanning Trees

It may be defined on Euclidean space points or on a graph.

G = (V, E): weighted connected undirected graph

spanning tree: S = (V, T), T $\subseteq$ E, undirected tree

minimal spanning tree (MST): a spanning tree with the smallest total weight.



A graph and one of its minimum costs spanning tree

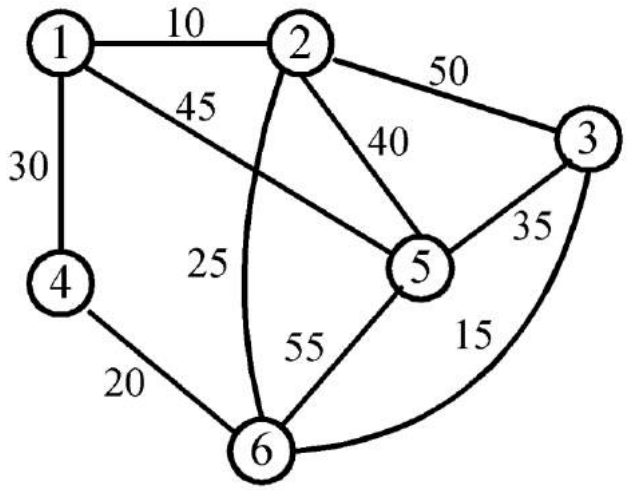
- **Kruskal's algorithm for finding MST**

  Step 1: Sort all edges

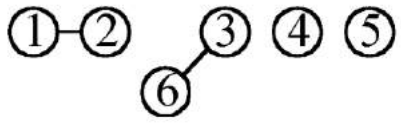
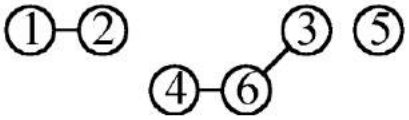
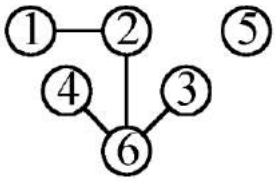
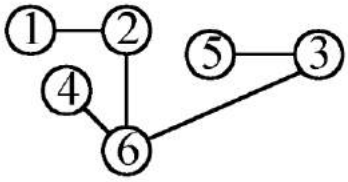
  Step 2: Add the next smallest weight edge to the forest if it will not cause a cycle.

  Step 3: Stop if n-1 edges. Otherwise, go to Step2.

e.g.



construct MST:

Edge	Cost	Spanning Forest
		
(1,2)	10	
(3,6)	15	
(4,6)	20	
(2,6)	25	
(1,4)	30	(reject)
(3,5)	35	

How do we check if a cycle is formed when a new edge is added?

By the **SET** and **UNION** method.

A tree in the forest is represented by a SET.

If (u, v) $\in$ E and u, v are in the same set, then the addition of (u, v) will form a cycle.

If (u, v) $\in$ E and $u \in S_1$, $v \in S_2$, then perform UNION of S_1 and S_2.

time complexity: $O(|E| \log|E|)$

Step 1 : $O(|E| \log|E|)$

Step 2, Step 3 : $O(|E| \alpha(|E|, |V|)$

- Ackermann's function

$$A(p, q) = \begin{cases} 2q, & p = 0 \\ 0, & q = 0, p \geq 1 \\ 2, & p \geq 1, q = 1 \\ A(p-1, A(p, q-1)), & p \geq 1, q \geq 2 \end{cases}$$

$\Rightarrow A(p, q+1) > A(p, q), \quad A(p+1, q) > A(p, q)$

$A(3,4) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ } 65536 two's

inverse of Ackermann's function:

$\alpha(m, n) = \min\{Z \geq 1 \mid A(Z, 4\lceil m/n \rceil) > \log_2 n\}$

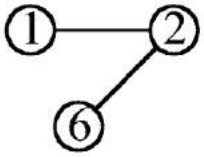
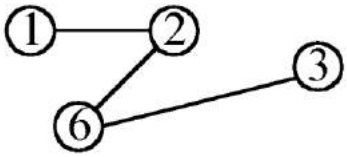
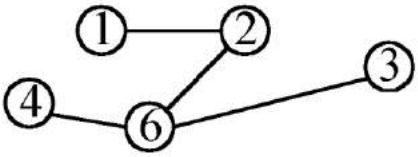
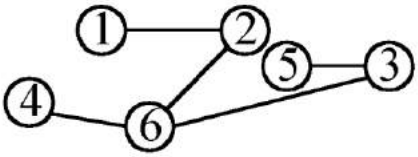
practically, $A(3,4) > \log_2 n$

$\Rightarrow \alpha(m, n) \leq 3$

$\Rightarrow \alpha(m, n)$ is almost a constant.

● **Prim's algorithm for finding MST**

e.g.

Edge	Cost	Spanning tree
(1,2)	10	
(2,6)	25	
(3,6)	15	
(6,4)	20	
(3,5)	35	

Prim's algorithm

Step 1: $x \in V$, Let $A = \{x\}$, $B = V - \{x\}$

Step 2: Select $(u, v) \in E$, $u \in A$, $v \in B$ ∋ (u, v) has the smallest weight between A and B

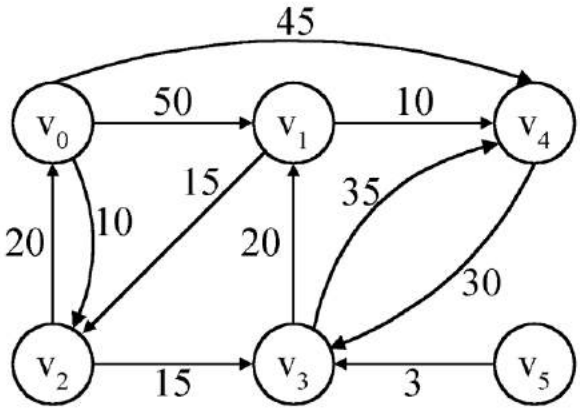
Step 3: (u, v) is in the tree. $A = A \cup \{v\}$, $B = B - \{v\}$

Step 4: If $B = \varnothing$, stop; otherwise, go to Step 2.

time complexity:

$O(n^2)$, $n = |V|$.

● The single-source shortest path problem



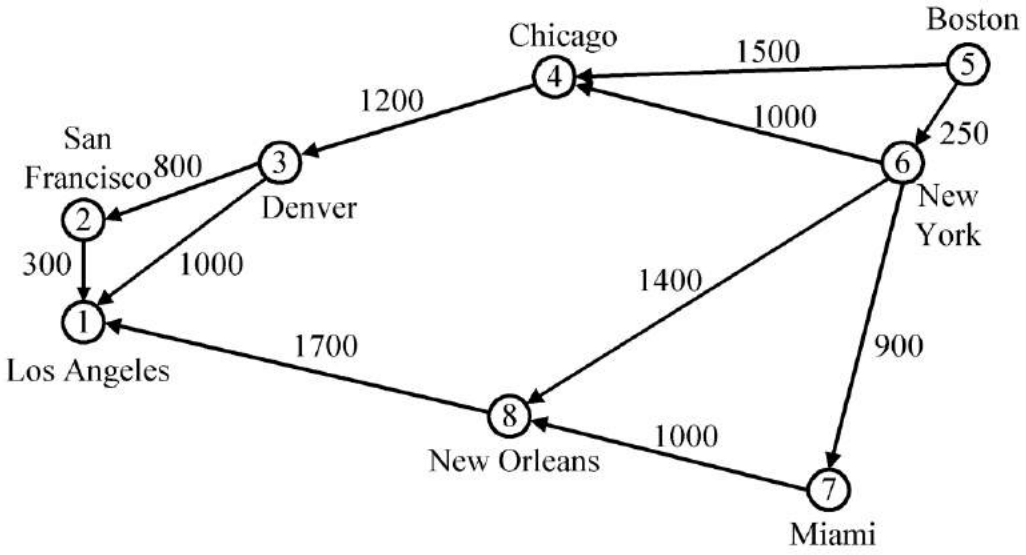
(a)

	Path	Length
1)	v_0v_2	10
2)	$v_0v_2v_3$	25
3)	$v_0v_2v_3v_1$	45
4)	v_0v_4	45

(b)

Graph and shortest paths from v_0 to all destinations

e.g.



Dijkstra's algorithm:

	1	2	3	4	5	6	7	8
1	0							
2	300	0						
3	1000	800	0					
4			1200	0				
5				1500	0	250		
6				1000		0	900	1400
7							0	1000
8	1700							0

Cost adjacency matrix. All entries not shown are $+\infty$.

Iteration	S	Vertex Selected	DIST	LA (1)	SF (2)	D (3)	C (4)	B (5)	NY (6)	M (7)	NO (8)
Initial		----									
1	5	6		+∞	+∞	+∞	1500	0	250	+∞	+∞
2	5,6	7		+∞	+∞	+∞	1250	0	250	1150	1650
3	5,6,7	4		+∞	+∞	+∞	1250	0	250	1150	1650
4	5,6,7,4	8		+∞	+∞	2450	1250	0	250	1150	1650
5	5,6,7,4,8	3		3350	+∞	2450	1250	0	250	1150	1650
6	5,6,7,4,8,3	2		3350	3250	2450	1250	0	250	1150	1650
	5,6,7,4,8,3,2			3350	3250	2450	1250	0	250	1150	1650

Action of SHORTEST_PATHS

time complexity: $O(n^2)$

Can we use Dijkstra's algorithm to find the longest path from a starting vertex to an ending vertex in an acyclic directed graph?

There are 3 possible ways to apply Dijkstra's algorithm:

(1)Convert all positive weights to be negative. Then find the shortest path.

(2)Give a very large positive number M. If the wright of an edge is w, now M-w is used to replace w. Then find the shortest path.

(3)Directly use "max" operations in stead of "min" operations.

All these 3 possible ways would not work!

The longest path(critical path) problem can be solved by the critical path method(CPM).

Step 1. Find a topological ordering.

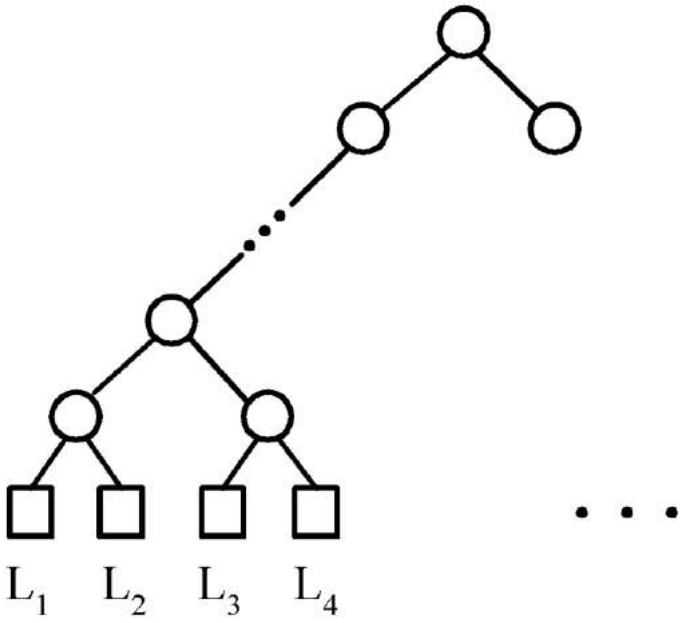
Step 2. Find the critical path.

(see [Horiwitz and Sahni 1976])

- **The 2-way merging problem**

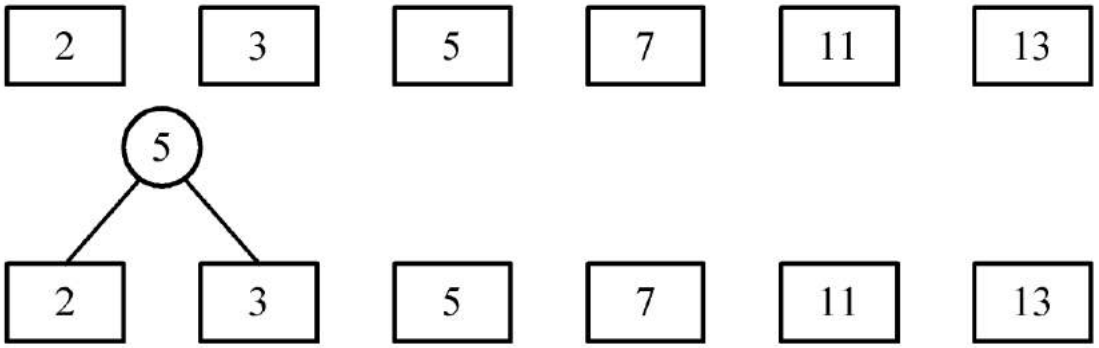
The number of comparisons required for the linear 2-way merge algorithm is m+n-1 where m and n are the lengths of the two sorted lists respectively.

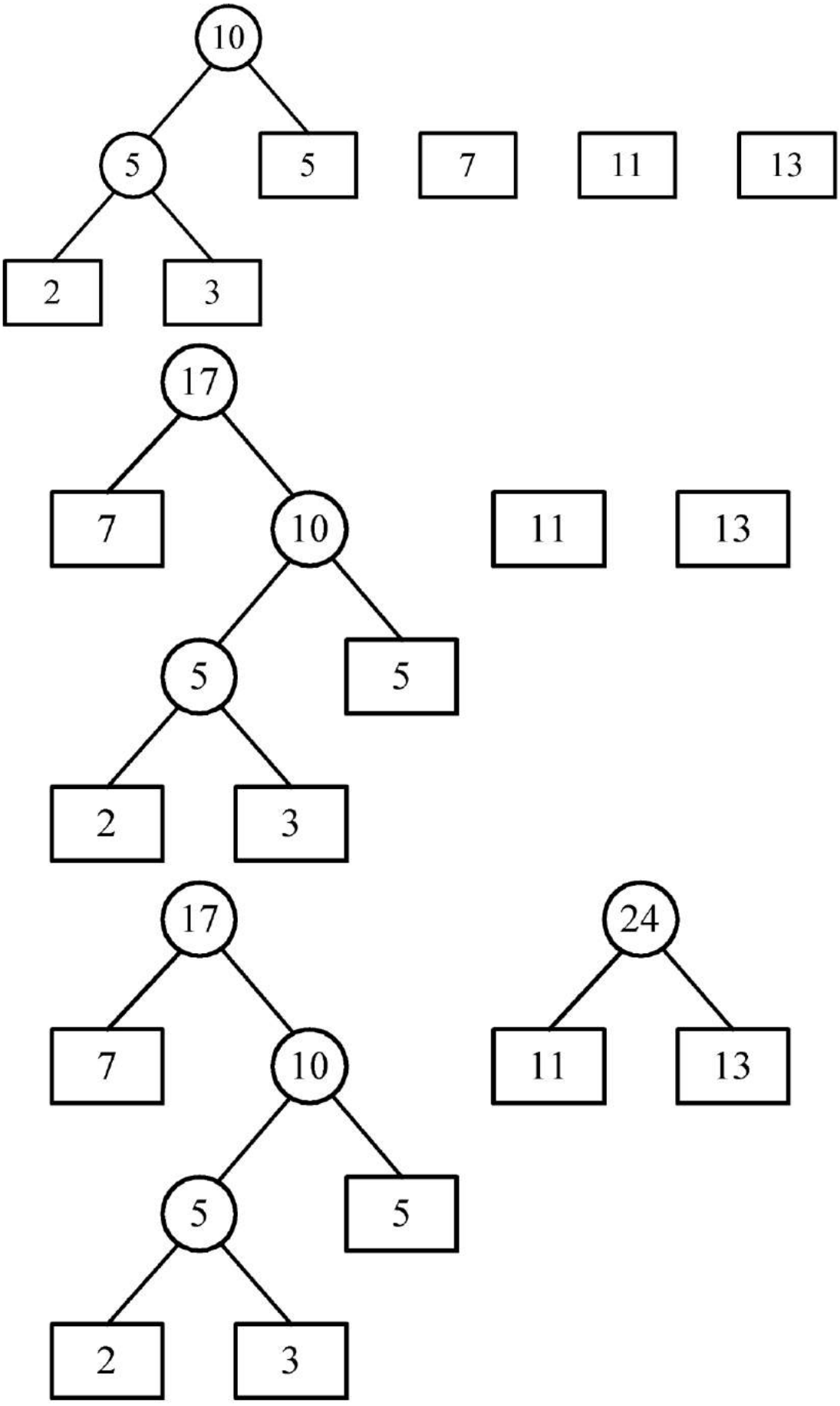
The problem: There are m sorted list, each of length n_i. What is the optimal sequence of merging process to merge these m lists into one sorted lists?

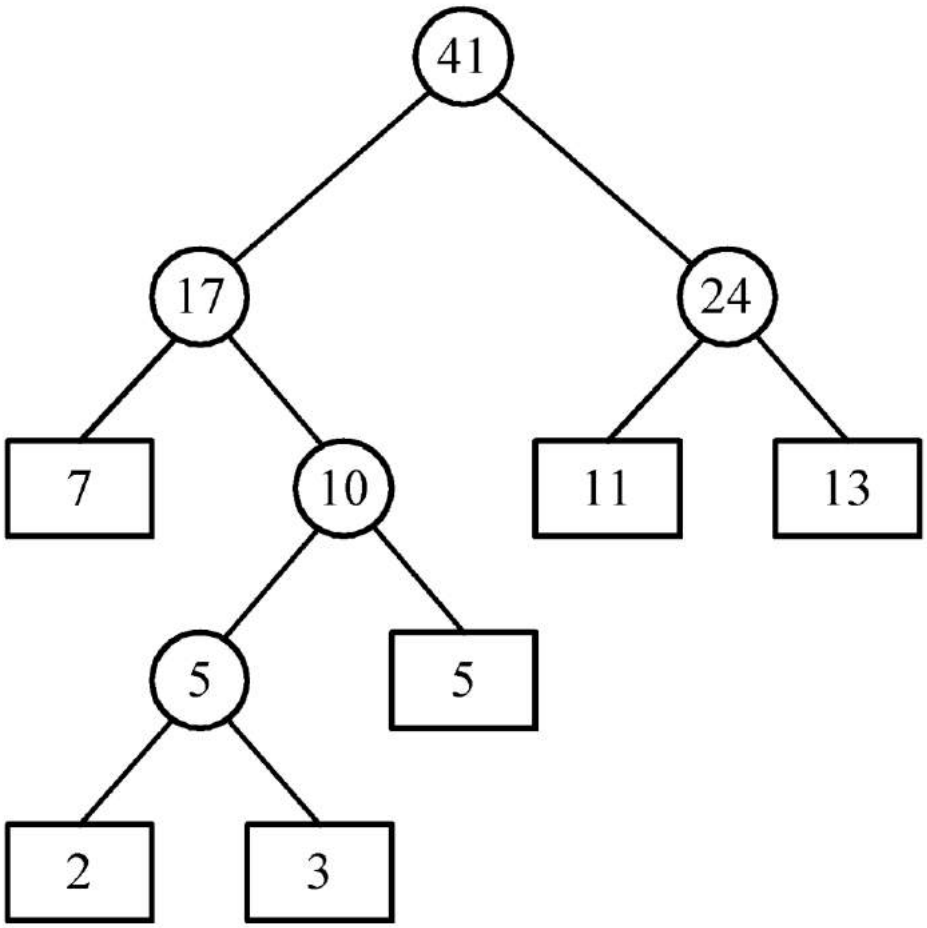


Extended Binary Tree Representing a 2-way Merge

e.g. There are 6 sorted lists with lengths 2, 3, 5, 7, 11 and 13.




10
5
5
7
11
13
2
3
17
7
10
11
13
5
5
2
3
17
24
7
10
11
13
5
5
2
3



Time complexity of the greedy algorithm to generate an optimal extended binary tree:

O(n log n)

(using a min-heap: (log n)·(n-1))

- **Huffman codes**

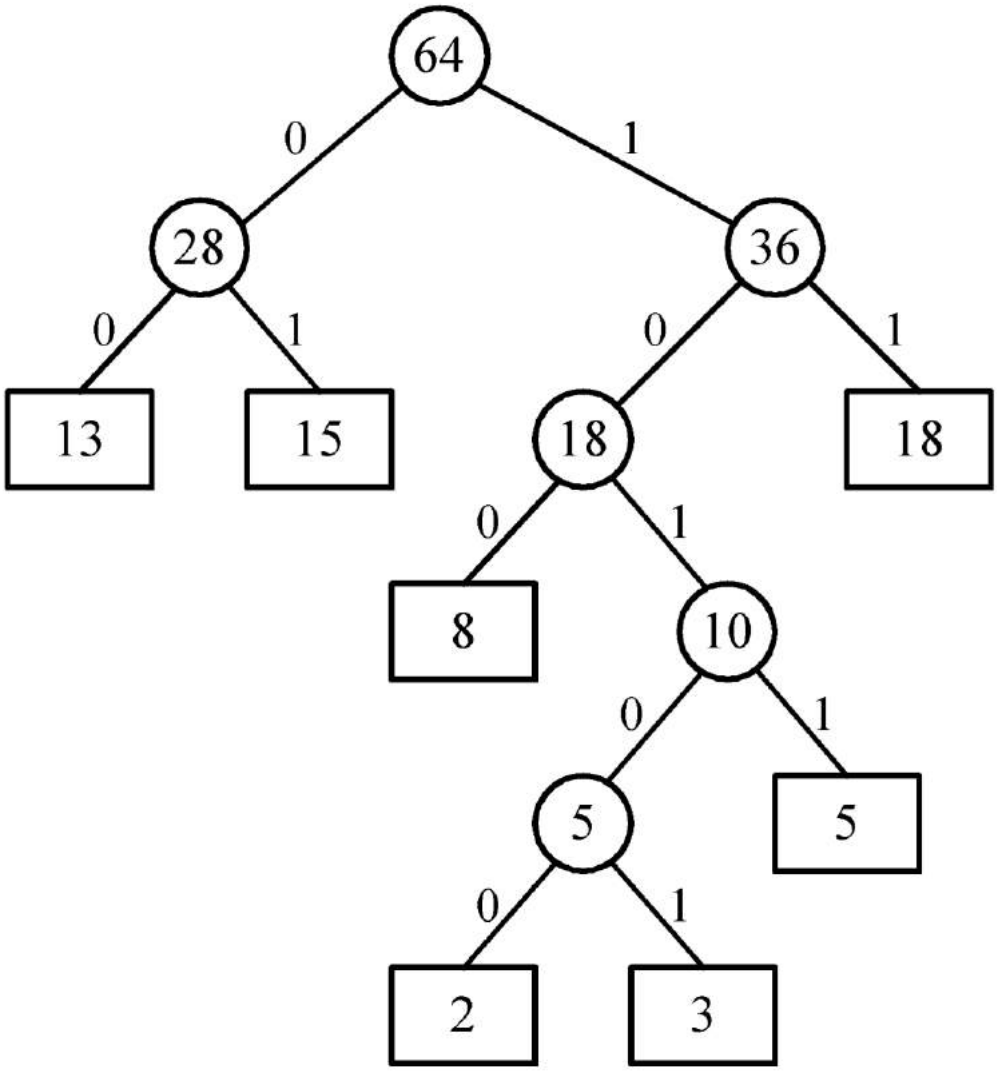
In telecommunication, how do we represent a set of messages, each with an access frequency, by a sequence of 0's and 1's?

To minimize the transmission and decoding costs, we may use short strings to represent more frequently used messages.

This problem can by solved by using an extended binary tree which is used in the 2-way merging problem.

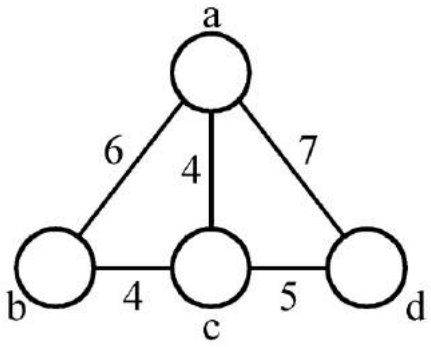
e.g. 7 messages: A, B, C, D, E, F, G
frequencies: 2, 3, 5, 8, 13, 15, 18
Huffman codes:

A: 10100	B: 10101	C: 1011
D: 100	E: 00	F: 01
G: 11		



A Huffman code Tree

- **The minimal cycle basis problem**



3 cycles:
A_1 = {ab, bc, ca}
A_2 = {ac, cd, da}
A_3 = {ab, bc, cd, da}

where $A_3 = A_1 \oplus A_2$

$(A \oplus B = (A \cup B)-(A \cap B))$

$A_2 = A_1 \oplus A_3$

$A_1 = A_2 \oplus A_3$

cycle basis: $\{A_1, A_2\}$ or $\{A_1, A_3\}$ or $\{A_2, A_3\}$

Def: A cycle basis of a graph is a set of cycles such that every cycle in the graph can be generated by applying $\oplus$ on some cycles of this basis.

Minimal cycle basis: smallest total weight of all edges in this cycle.

e.g. $\{A_1, A_2\}$

Algorithm for finding a minimal cycle basis:

Step 1: Determine the size of the minimal cycle basis, demoted as k.

Step 2: Find all of the cycles. Sort all cycles(by weight).

Step 3: Add cycles to the cycle basis one by one. Check if the added cycle is already combination of some cycles already existing in the basis. If it is, delete this cycle.

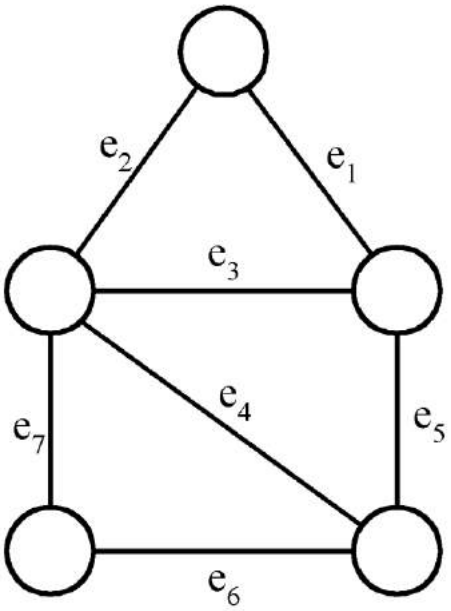
Step 4: Stop if the cycle basis has k cycles.

More detailed:

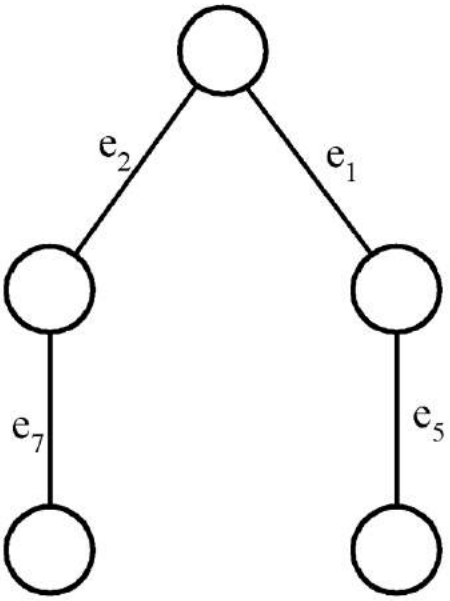
Step 1:

A cycle basis corresponds to the fundamental set of cycles with respect to a spanning tree.

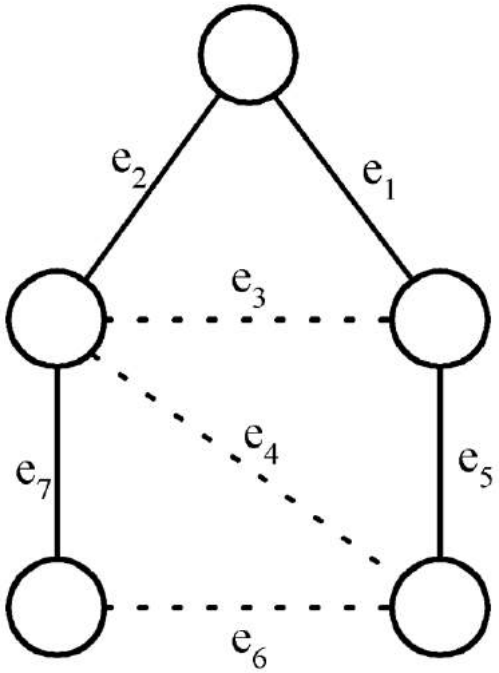
a graph:



a spanning tree:



a fundamental set of cycles:



# of cycles in a cycle basis
= k
= |E| - (|V|- 1)
= |E| - |V| + 1

Step 2:

How to find all cycles in a graph?

[Reingold, Nievergelt and Deo 1977]

How many cycles in a graph in the worst case?

In a complete digraph of n vertices and n·(n-1) edges:

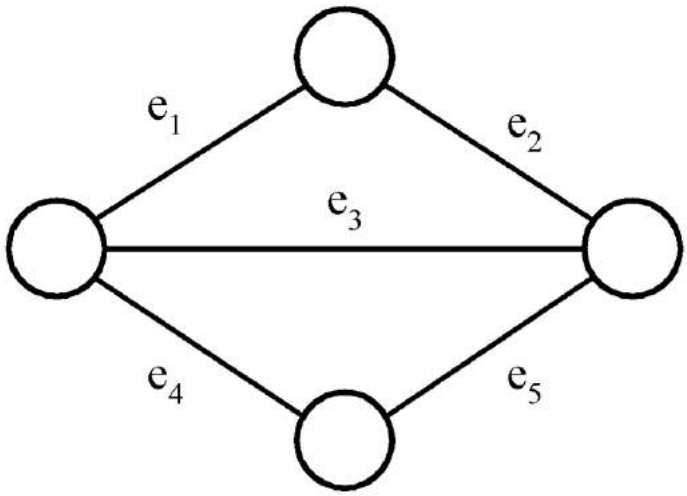
$$\sum_{i=2}^{n} C_i^n (i-1)! \; > (n-1)!$$

Step 3:

How to check if a cycle is a linear combination of some cycles?

Using Gaussian elimination.

e.g.



2 cycles C_1 and C_2 are represented by a 0/1 matrix:

	e_1	e_2	e_3	e_4	e_5
C_1	1	1	1		
C_2			1	1	1

add C_3:

	e_1	e_2	e_3	e_4	e_5
C_1	1	1	1		
C_2			1	1	1
C_3	1	1		1	1

$\oplus$ on row1 and row3:

	e_1	e_2	e_3	e_4	e_5
C_1	1	1	1		
C_2			1	1	1
C_3			1	1	1

$\oplus$ on row 2 and row 3: empty row

$\therefore C_3 = C_1 \oplus C_2$

A polynomial time algorithm to find the minimal cycle basis[Horton 1987]:

Step 1: Find all pairs shortest paths.

Step 2: For each vertex $v \in V$ and for each edge $(x,y) \in E$, the cycle consists of shortest path (v,x), edge (x,y) and shortest path (y,v) is a candidate. Compute the weight of each candidate.

Step 3: Sort all candidate cycles.
(by weight)

Step 4: Add the candidate cycles to the minimal cycle basis one by one.

Time complexity: $O(n^7)$

$|V| = n$, $|E| = m$ ($m=O(n^2)$ in the worst case)

Step 1: $O(n^3)$

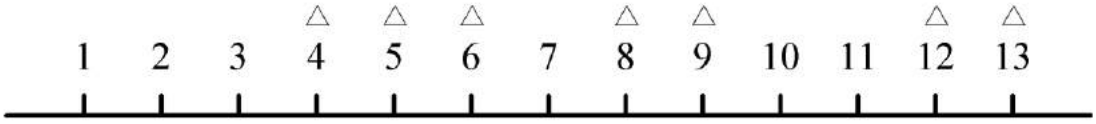
Step 2: $O(mn^2)$, find m n cycles

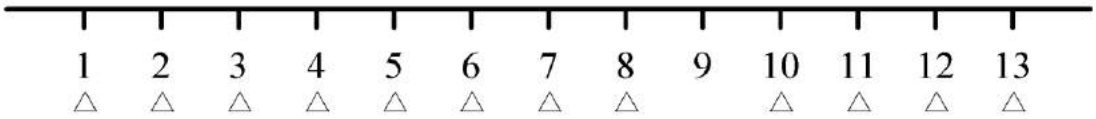
Step 3: $O(mn \log(mn))$

Step 4: $O(m \cdot s \cdot mn)$, $s = m - n + 1$

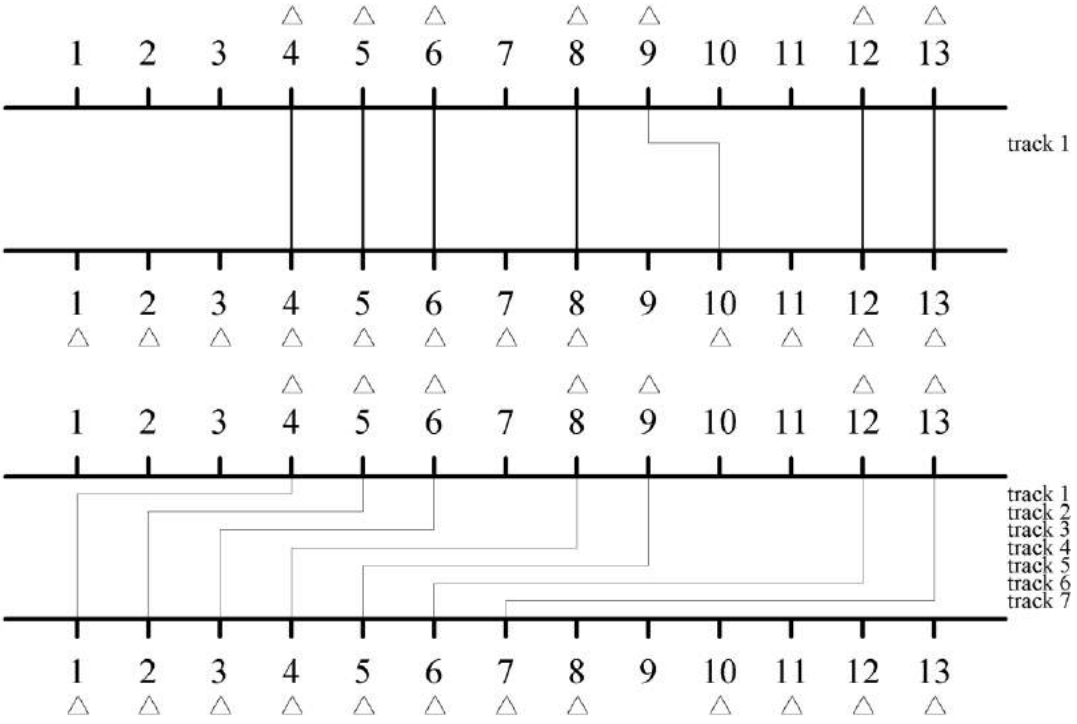
$= O(m^3 n)$

- The 2-terminal one to any problem special channel routing problem.

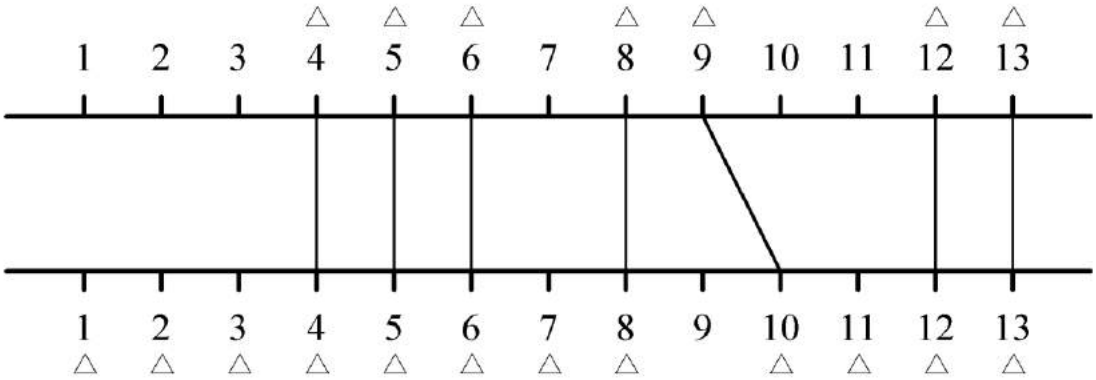




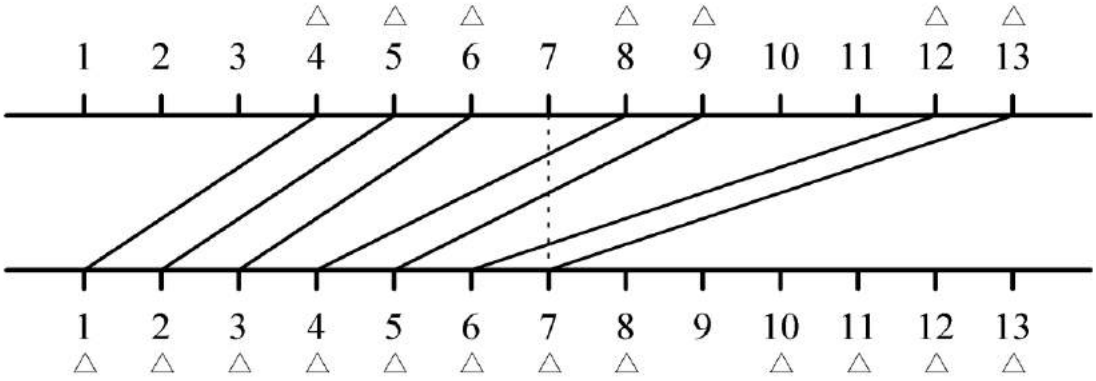
2 feasible solutions:



redrawing:



(a) optimal solution



(b)

At each point, the local density of the solution is # of lines the vertical line intersects.

The problem: to minimize the density. The density is a lower bound of # of tracks.

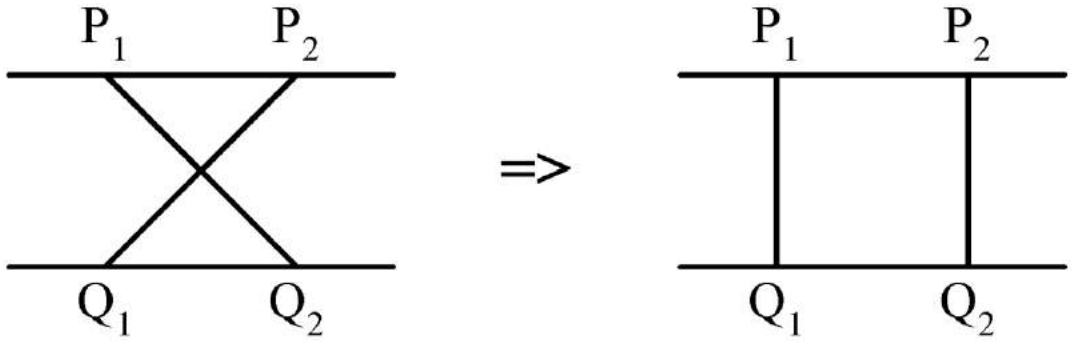
upper row terminals: $P_1, P_2, \cdots, P_n$ from left to right

lower row terminals: $Q_1, Q_2, \cdots, Q_m$ from left to right

$m > n$.

Suppose that we have a method to determine the minimum density, d, of a problem instance.
The greedy algorithm:
Step 1: P_1 is connected Q_1.
Step 2: After P_i is connected to Q_j, we check whether P_{i+1} can be connected to Q_{j+1}. If the density is increased to d+1, try to connect P_{i+1} to Q_{j+2}.
Step 3: Repeat Step2 until all P_i's are connected.

It would have a higher density:



- **The knapsack problem**

  n objects, each with a weight $w_i > 0$

  a profit $p_i > 0$

  capacity of knapsack: M

  Maximize $\sum_{1 \le i \le n} p_i x_i$

  subject to $\sum_{1 \le i \le n} w_i x_i \le M$

  $0 \le x_i \le 1, 1 \le i \le n$

The greedy algorithm:

Step 1: Sort p_i/w_i into non-increasing order.

Step 2: Put the objects into the knapsack according to the sorted sequence as possible as we can.

e.g. n = 3, M = 20, $(p_1, p_2, p_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

sol: $p_1/w_1 = 25/18 = 1.32$

$p_2/w_2 = 24/15 = 1.6$

$p_3/w_3 = 15/10 = 1.5$

optimal solution: $x_1 = 0, x_2 = 1, x_3 = 1/2$