



离散数学

Discrete Mathematics

for Computer Science

计算机学院计科系

薛思清 xuesiqing@cug.edu.cn



第16讲 树 Tree (2)

I think that I shall never see
A graph more lovely than a tree.

by Radia Perlman

3 最小生成树 Minimum Spanning Tree

Otakar Boruvka (1926).

- Electrical Power Company of Western Moravia in Brno.
- Most economical construction of electrical power network.
- Concrete engineering problem is now a cornerstone problem in combinatorial optimization.



3 最小生成树 Minimum Spanning Tree

MST is fundamental problem with diverse applications.

- Network design.
 - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
 - traveling salesperson problem, Steiner tree
- Indirect applications.
 - max bottleneck paths
 - LDPC codes for error correction
 - image registration with Renyi entropy
 - learning salient features for real-time face verification
 - reducing data storage in sequencing amino acids in a protein
 - model locality of particle interactions in turbulent fluid flows
 - autoconfig protocol for Ethernet bridging to avoid cycles in a network
- Cluster analysis.

3 最小生成树 Minimum Spanning Tree

最小生成树算法(MST)

Kruskal算法, Prim算法

Kruskal's Algorithm:

Sort the edges so that: $c(e_1) \leq c(e_2) \leq \dots \leq c(e_m)$

$T \leftarrow \emptyset$

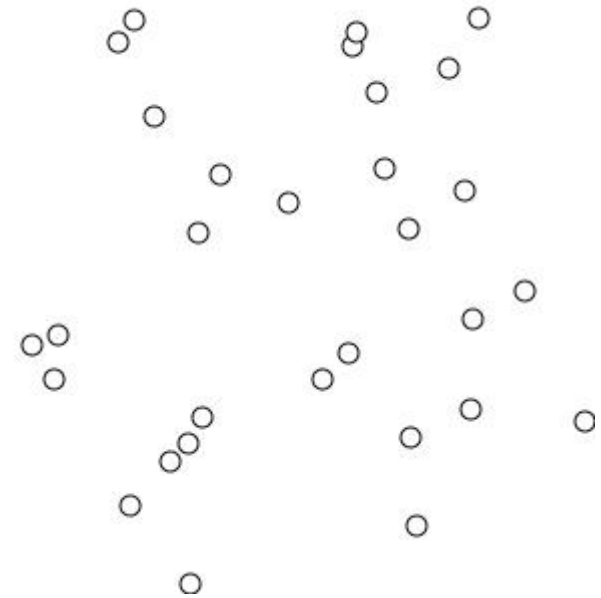
for $i : 1..m$

if $T \cup \{e_i\}$ has no cycle then

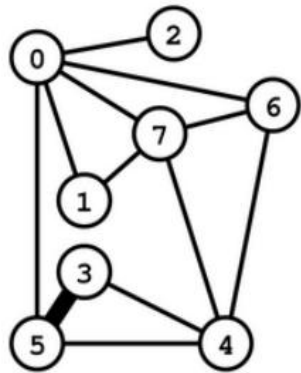
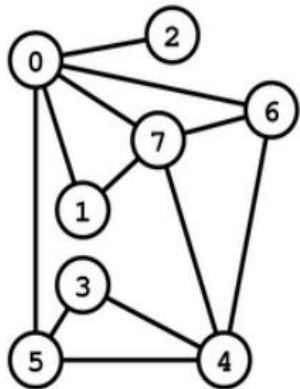
$T \leftarrow T \cup \{e_i\}$

end if

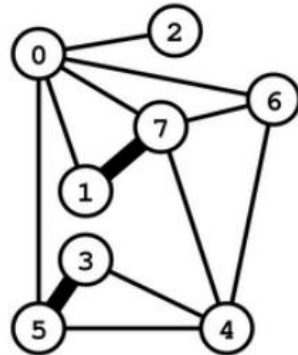
end for



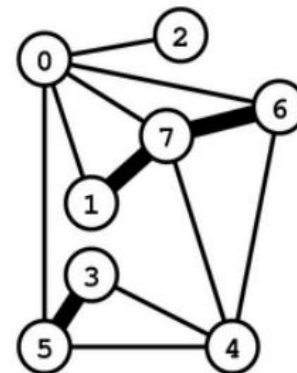
Kruskal算法证明?



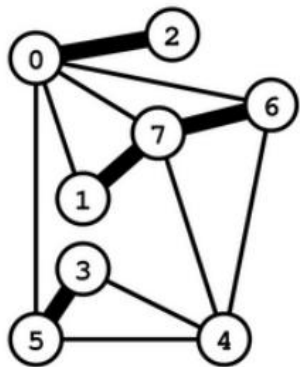
3-5



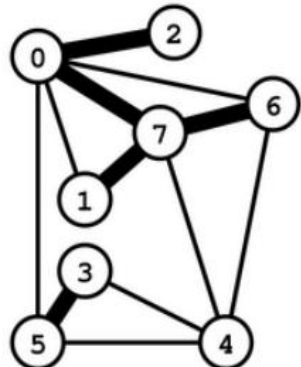
1-7



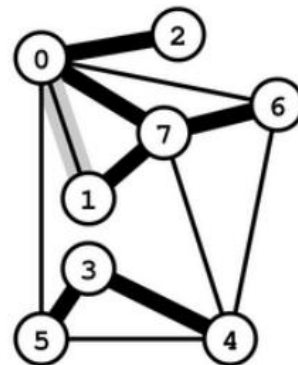
6-7



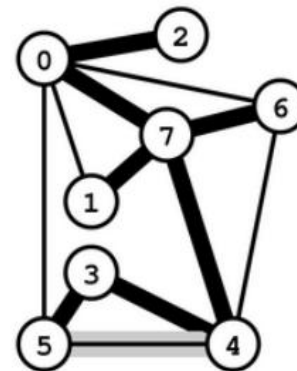
0-2



0-7



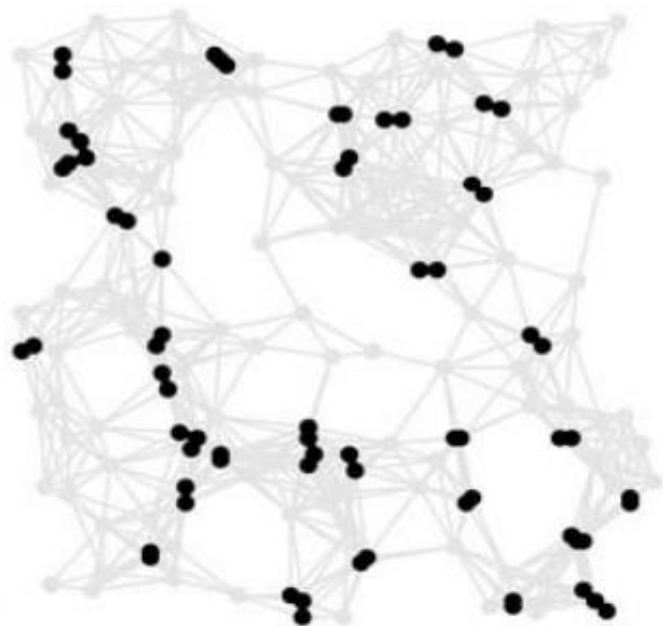
0-1 3-4



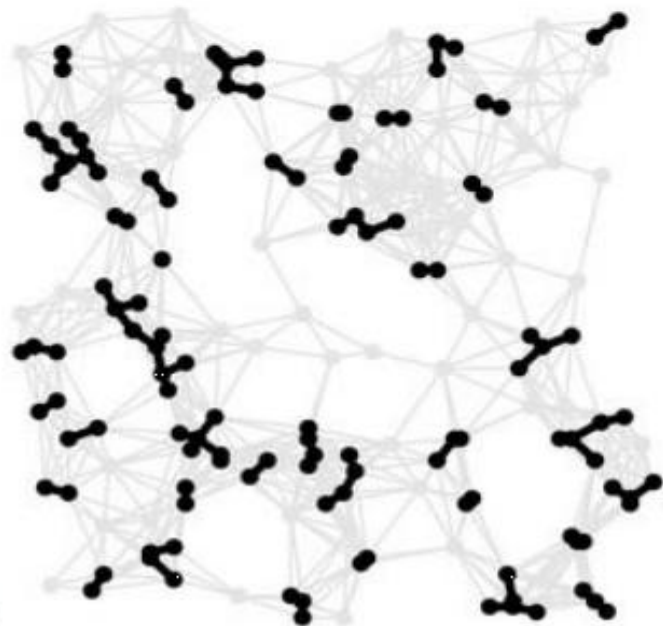
4-5 4-7

3-5	0.18
1-7	0.21
6-7	0.25
0-2	0.29
0-7	0.31
0-1	0.32
3-4	0.34
4-5	0.40
4-7	0.46
0-6	0.51
4-6	0.51
0-5	0.60

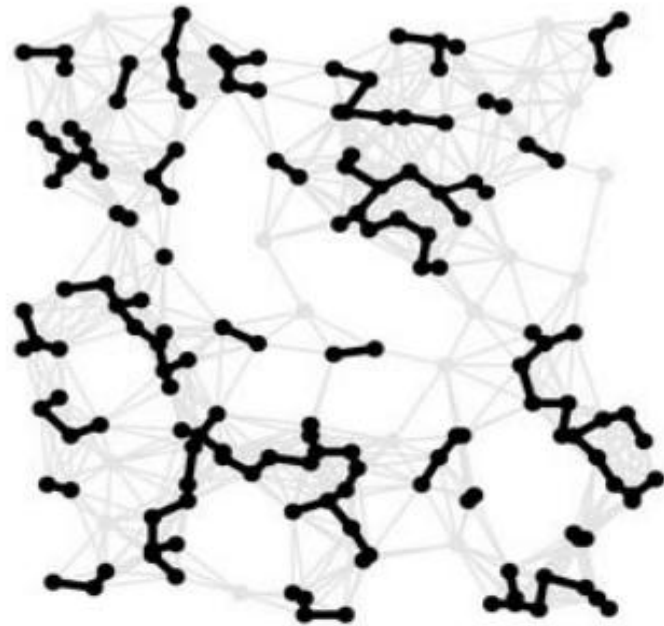
25%



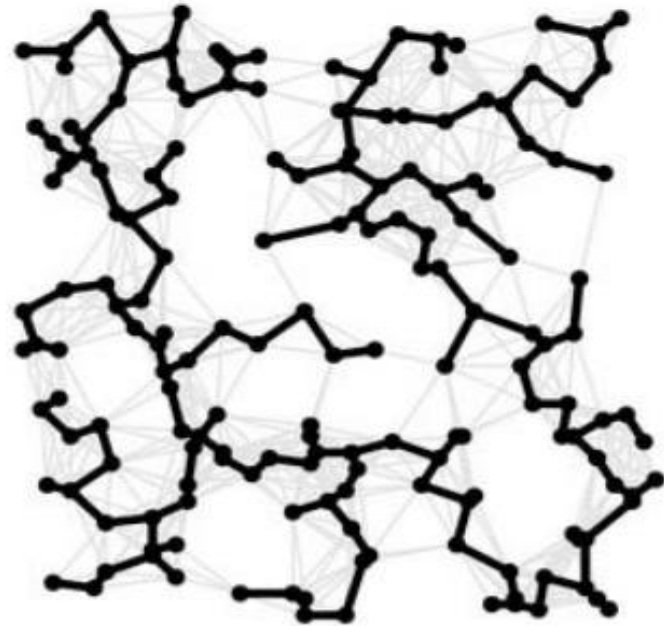
50%



75%



100%



3 最小生成树 Minimum Spanning Tree

算法正确性证明

3 最小生成树 Minimum Spanning Tree

算法正确性证明

设 T 不是最小生成树，则存在另一棵树 T^* ，为最小生成树。

下面证明 $T^*=T$ 。首先设 T 的所有边升序排列，假设 T 中有 m 条边不在 T^* 中，kruskal算法选择边的顺序不妨假设为 $e_1e_2e_3\dots e_m$ ：

下面证明可以将 e_1 加入 T^* 中得到生成树 T_1 ，且满足条件 $w(T_1) \leq w(T^*)$ ：

将 e_1 加入 T^* 中将形成环，此环中必然存在边 e_1' 在 T^* 中而不在 T 中，于是，删除 e_1' ，则得到生成树 T_1 ，按kruskal算法选择边的顺序知

$$w(e_1) \leq w(e_1'), \text{ 从而 } w(T_1) \leq w(T^*).$$

依此进行，可以将 e_k 加入到 T_{k-1} 中，将形成环，此环中必然存在边 e_k' 在 T^* 中而不在 T 中，于是，删除 e_k' ，则得到生成树 T_k 。而显然，两边序列 $e_1e_2e_3\dots e_k$ 与 $e_1e_2e_3\dots e_k'$ 均不构成环，而按kruskal算法，必然有 $w(e_k) \leq w(e_k')$ ，

从而 $w(T_k) \leq w(T_{k-1}) \leq w(T^*) \dots$ ，

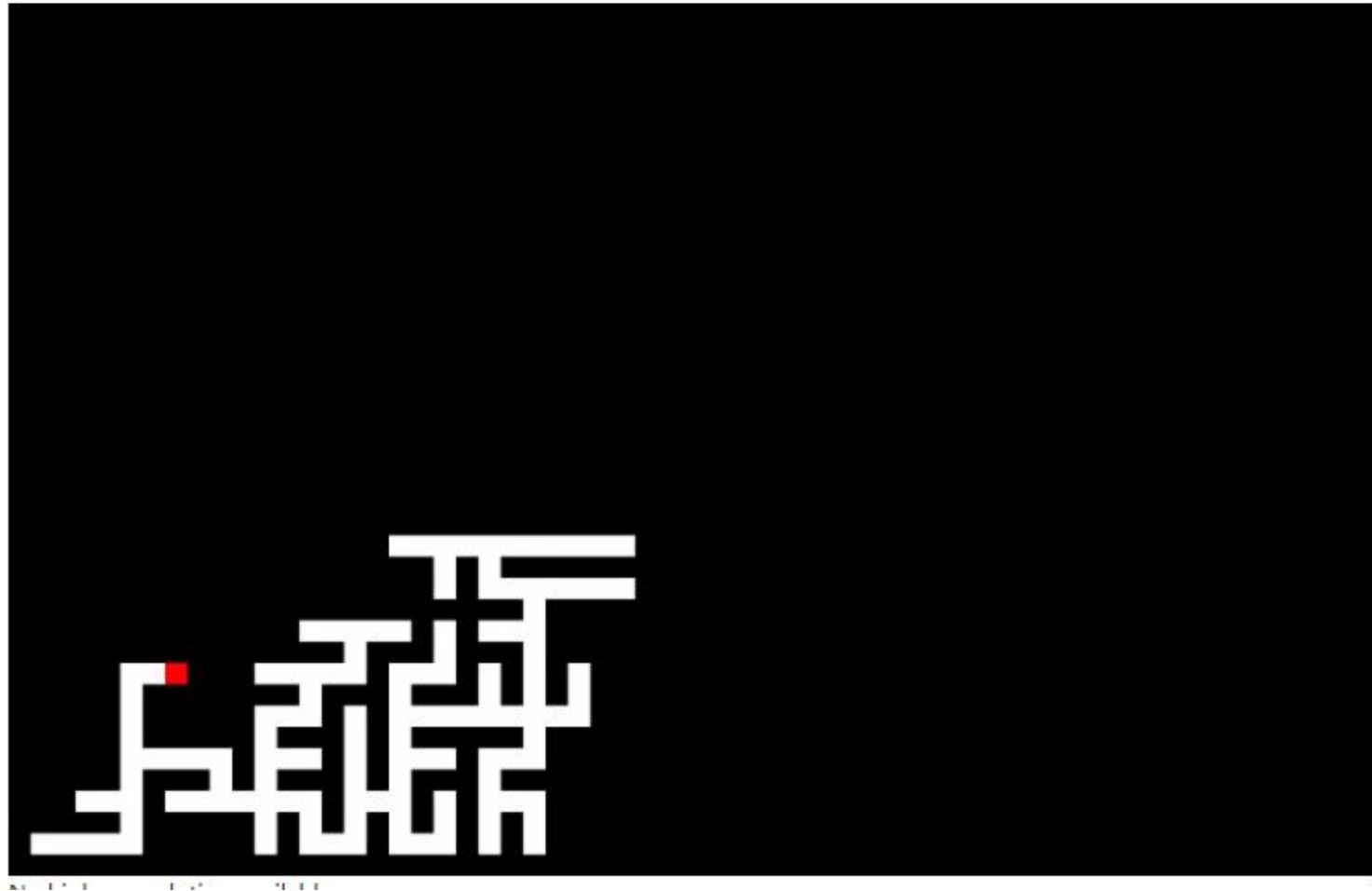
最后可以将 e_m 加入到 T_{m-1} 中，得到生成树 T_m ，且 $w(T_m) \leq \dots \leq w(T_k) \leq w(T_{k-1}) \leq \dots \leq w(T_1) \leq w(T^*)$ 。

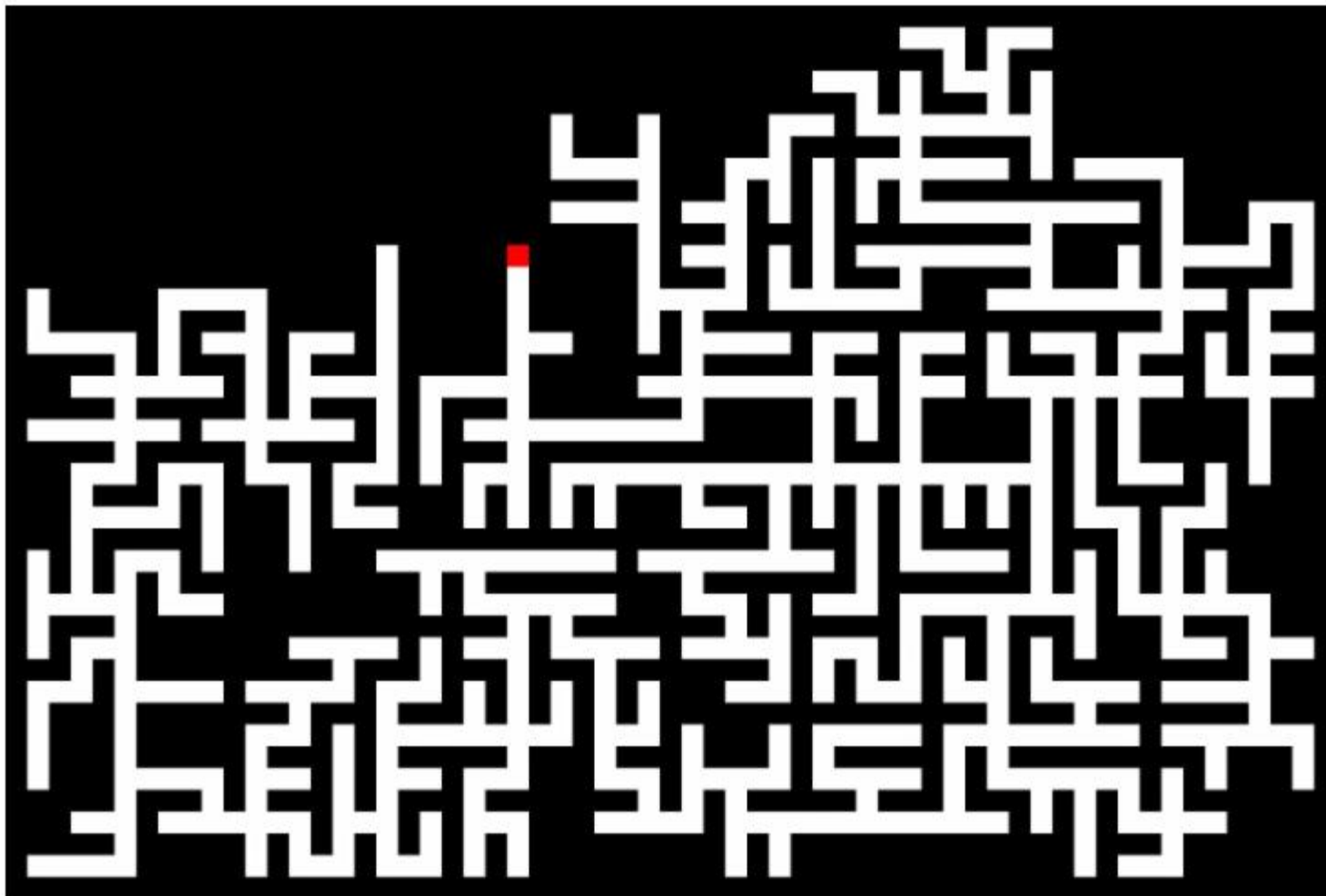
而此时，所 T 有边（包括与 T^* 中相同的边）都在到 T_m 中，即 $T_m=T$ 。故 $w(T) \leq w(T^*)$

因此， T 为最小生成树。

Prim's Algorithm

Prim's algorithm has many applications, such as in the generation of this **maze**, which applies Prim's algorithm to a randomly weighted grid graph.

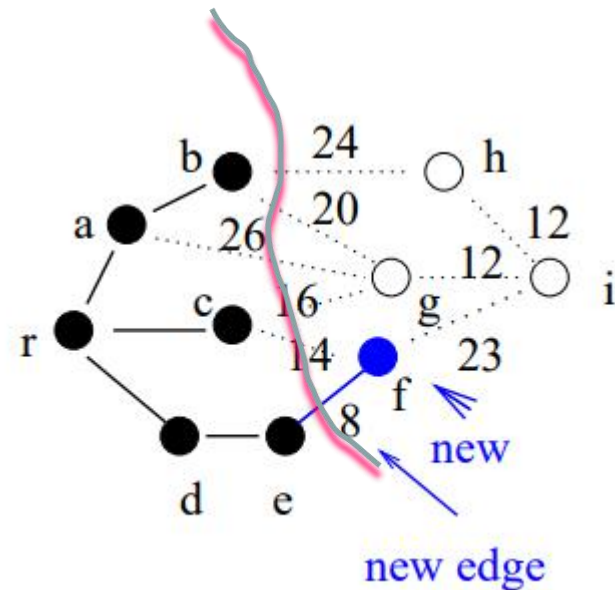
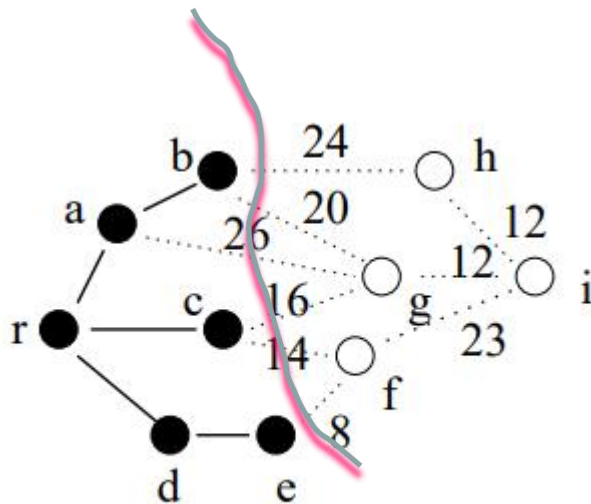




3 最小生成树 Minimum Spanning Tree

Prim's Algorithm

The idea: expand the current tree by adding the **lightest (shortest) edge** leaving it and its endpoint.

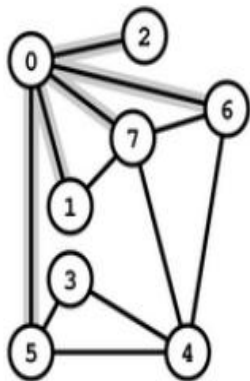


Prim's Algorithm

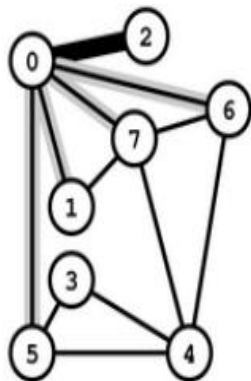
Step 0: Choose any element r ; set $S = \{r\}$ and $A = \emptyset$. (Take r as the root of our spanning tree.)

Step 1: Find a lightest edge such that one endpoint is in S and the other is in $V \setminus S$. Add this edge to A and its (other) endpoint to S .

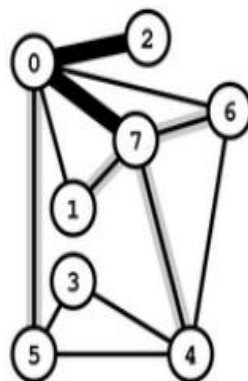
Step 2: If $V \setminus S = \emptyset$, then stop & output (minimum) spanning tree (S, A) . Otherwise go to Step 1.



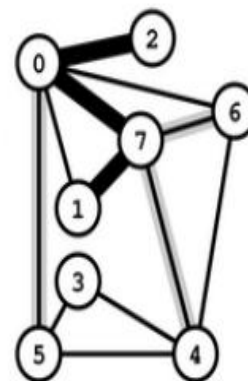
0-2 0-7 0-1 0-6 0-5



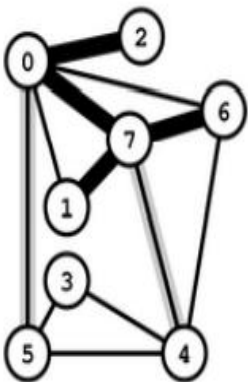
0-7 0-1 0-6 0-5



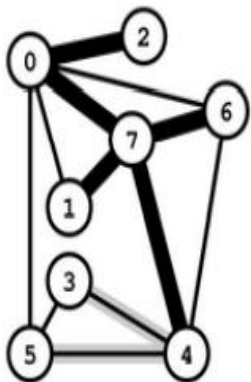
7-1 7-6 7-4 0-5



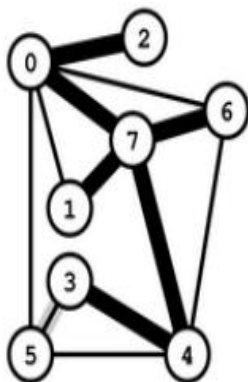
7-6 7-4 0-5



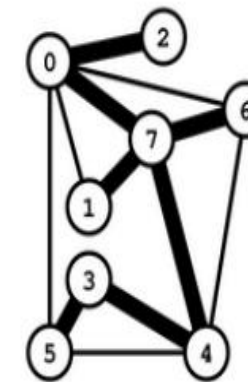
7-4 0-5



4-3 4-5

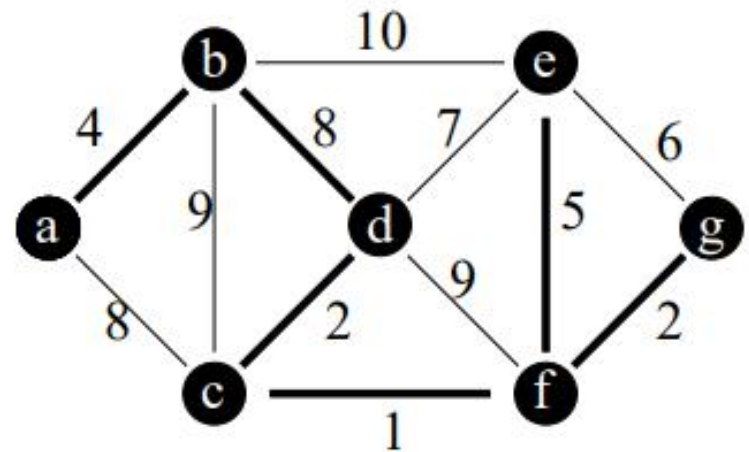
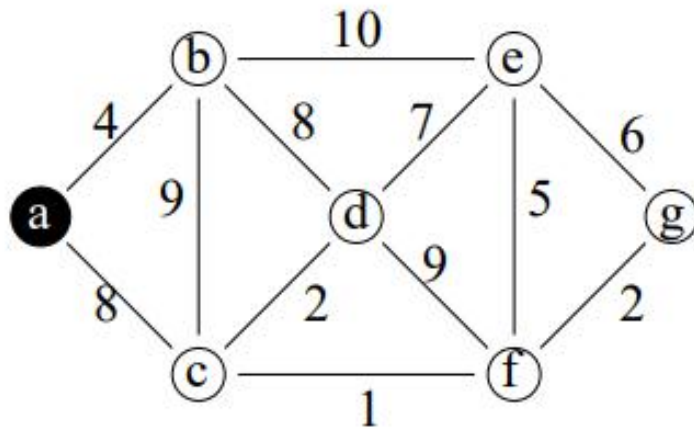


3-5

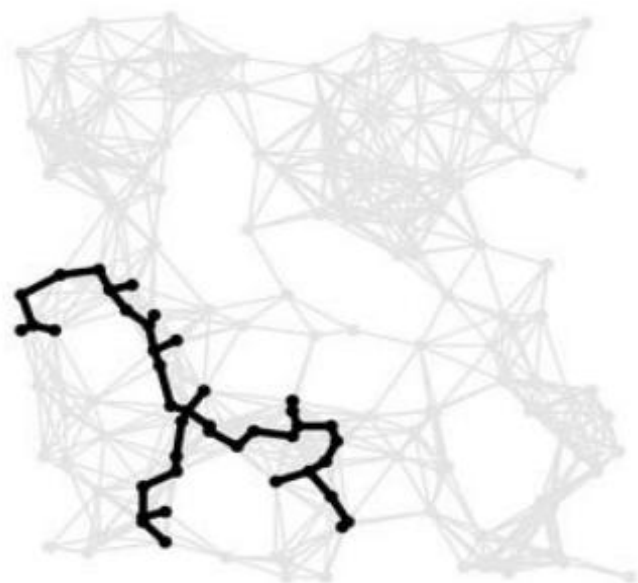


0-1	0.32
0-2	0.29
0-5	0.60
0-6	0.51
0-7	0.31
1-7	0.21
3-4	0.34
3-5	0.18
4-5	0.40
4-6	0.51
4-7	0.46
6-7	0.25

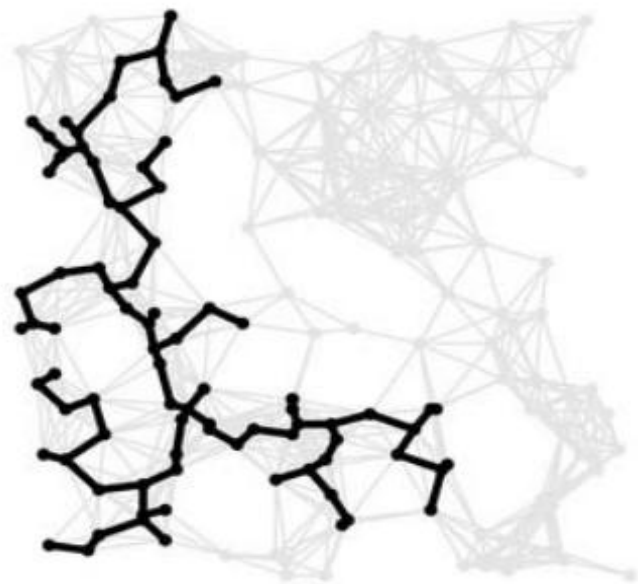
Another Example



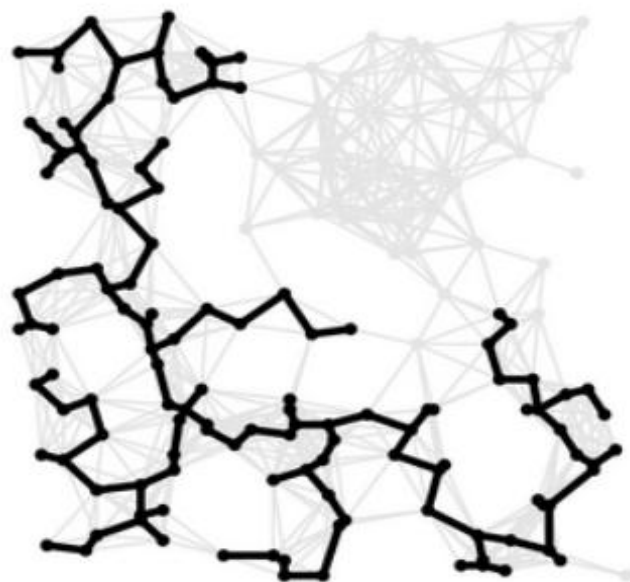
25%



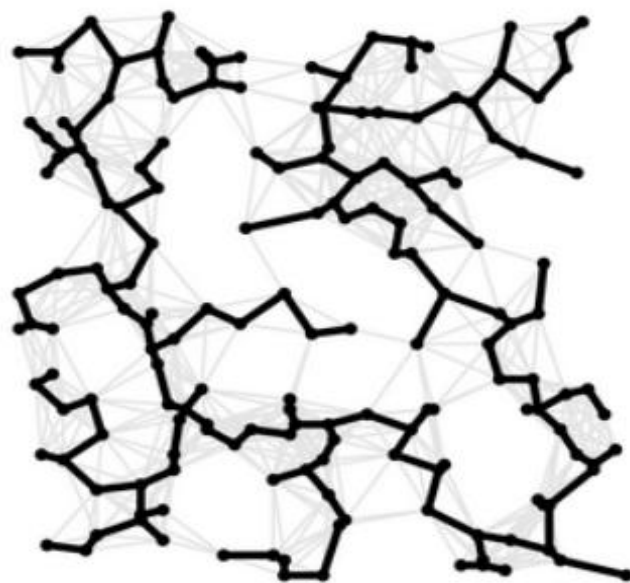
50%



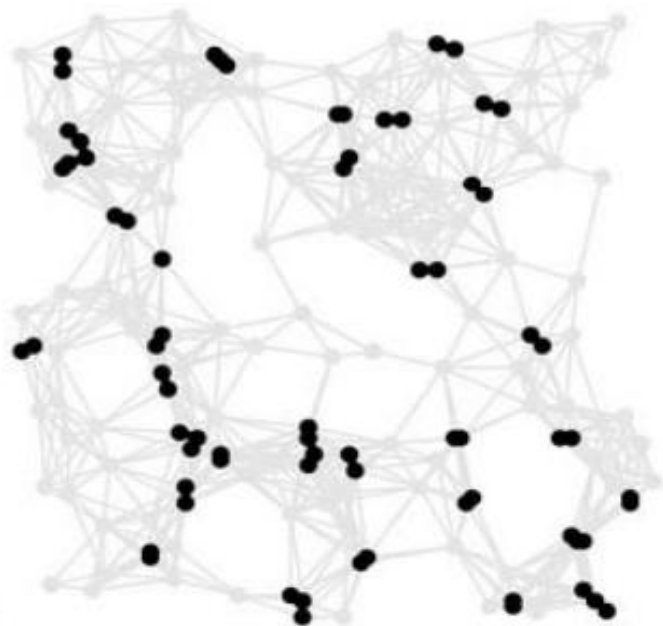
75%



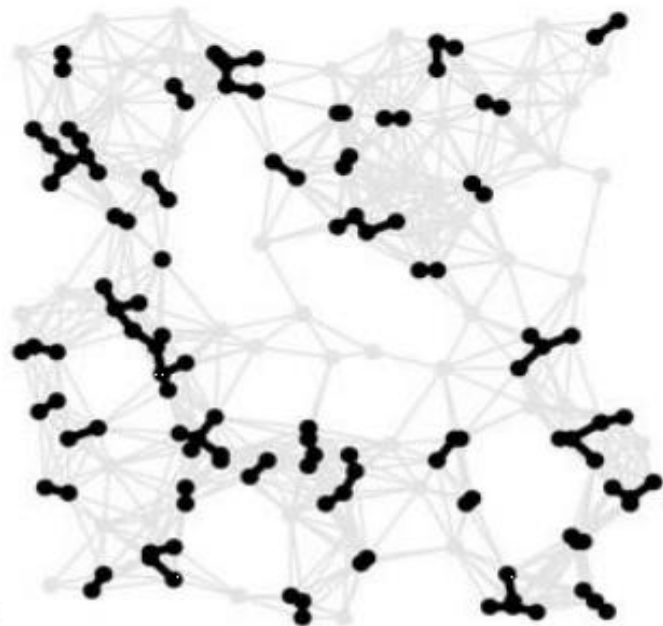
100%



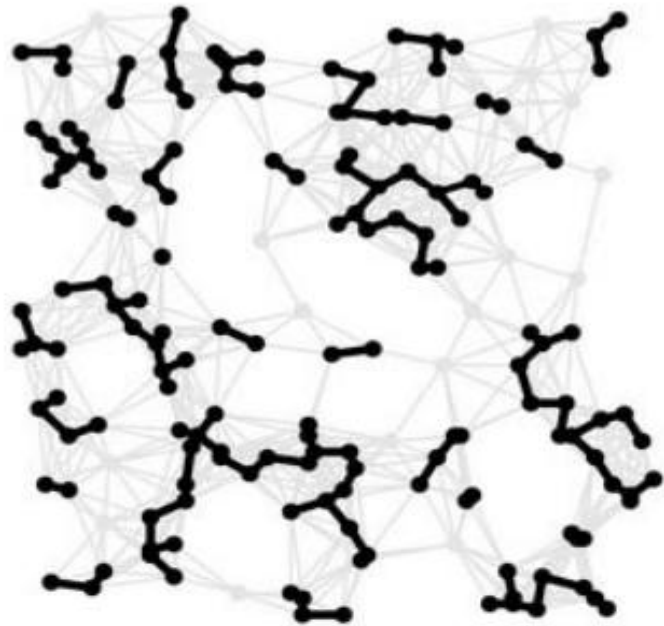
25%



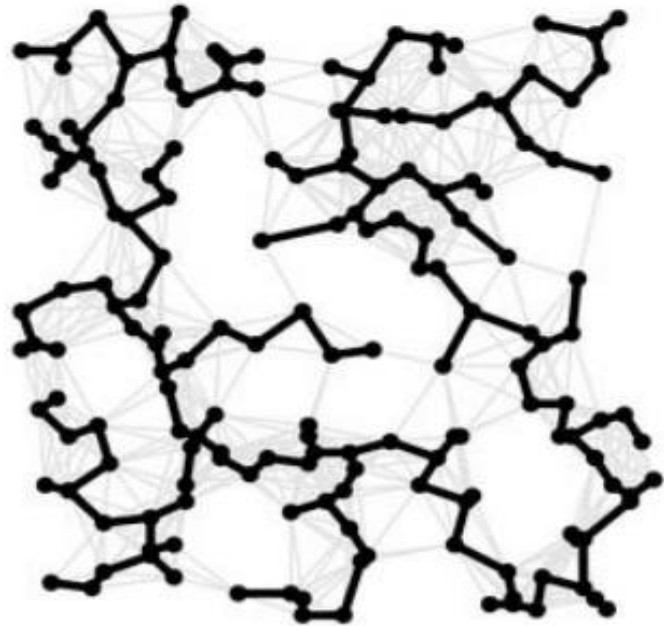
50%



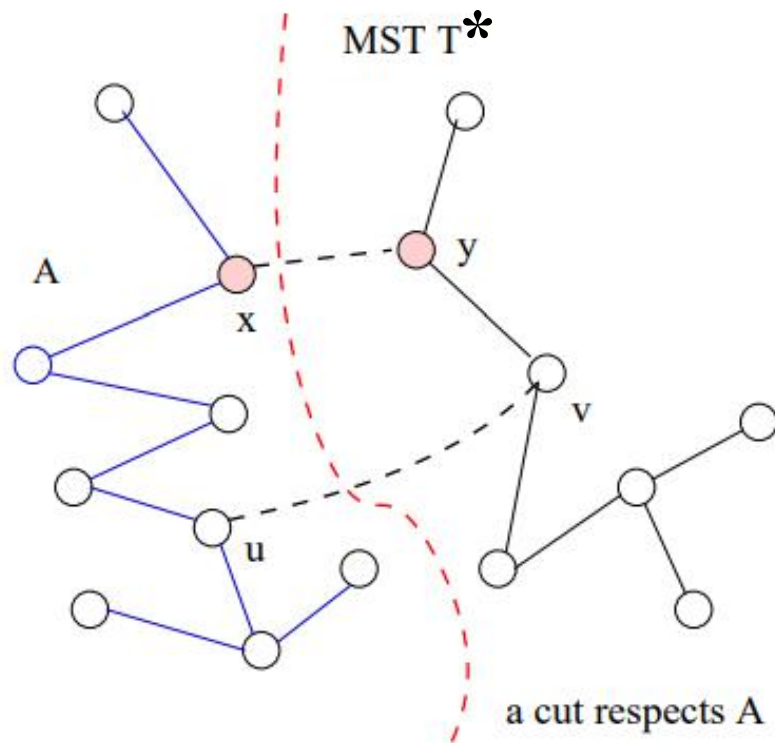
75%



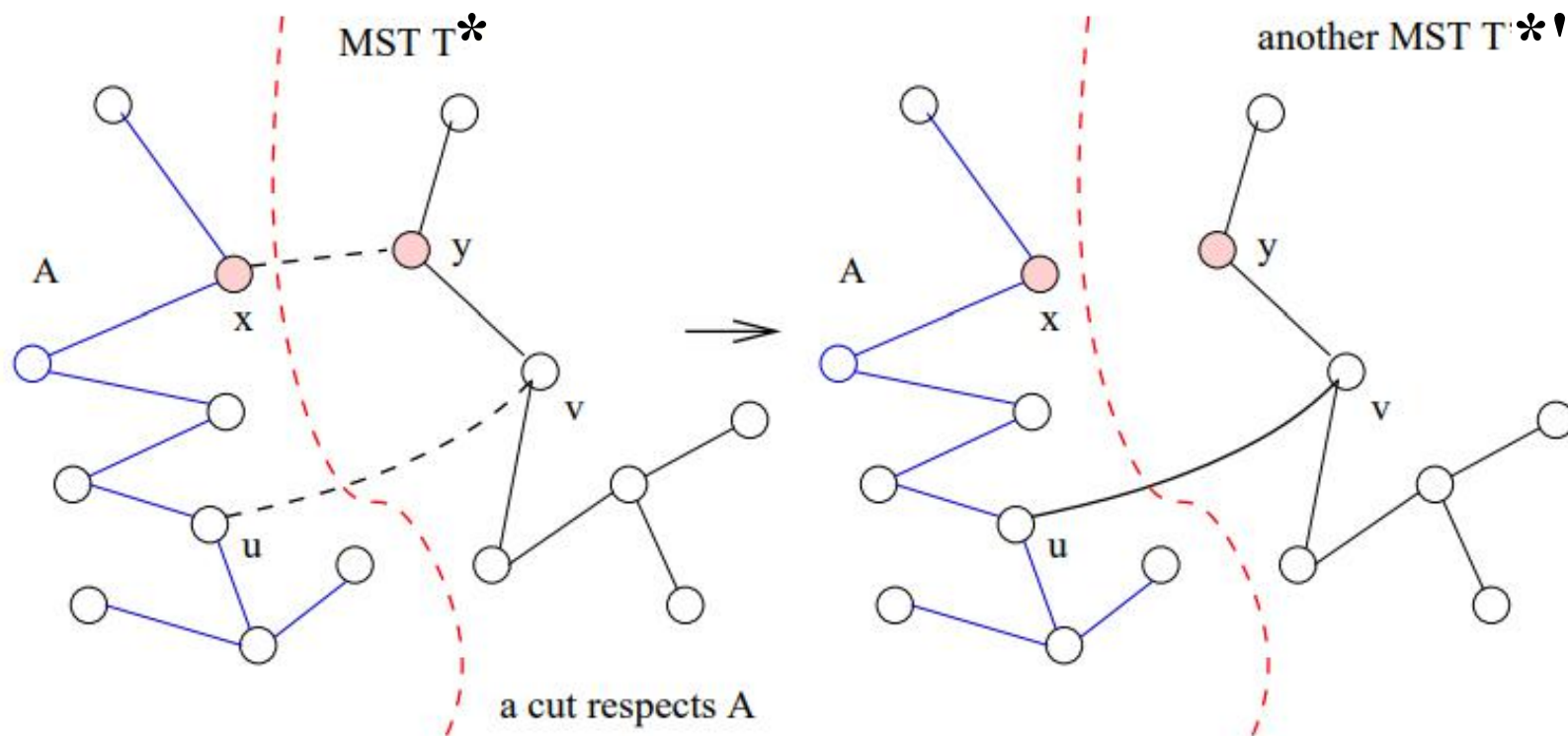
100%



Proof



Proof



Theorem: If G is a connected, weighted graph, Prim's algorithm correctly finds an MST in G .

Proof: Let T be the spanning tree found by Prim's algorithm and T^* be any MST of G . We will prove $c(T) = c(T^*)$. If $T = T^*$, then $c(T) = c(T^*)$ and we are done.

Otherwise, $T \neq T^*$, so we have $T - T^* \neq \emptyset$. Let (u, v) be any edge in $T - T^*$. When (u, v) was added to T , it was a least-cost edge crossing some cut $(S, V - S)$. 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 $x \in S$ and $y \in V - S$. Since (u, v) is a least-cost edge crossing $(S, V - S)$, we have $c(u, v) \leq c(x, y)$.

Let $T^{*'} = T^* \cup \{(u, v)\} - \{(x, y)\}$. Since (x, y) is on the cycle formed by adding (u, v) , this means $T^{*'}$ is a spanning tree. Notice $c(T^{*'}) = c(T^*) + c(u, v) - c(x, y) \leq c(T^*)$. Since T^* is an MST, this means $c(T^{*'}) \geq c(T^*)$, so $c(T^*) = c(T^{*'})$.

Note that $|T - T^{*'}| = |T - T^*| - 1$. Therefore, if we repeat this process once for each edge in $T - T^*$, we will have converted T^* into T while preserving $c(T^*)$. Thus $c(T) = c(T^*)$. ■

Summary: proof idea

Proposition. Both greedy algorithms compute an MST.

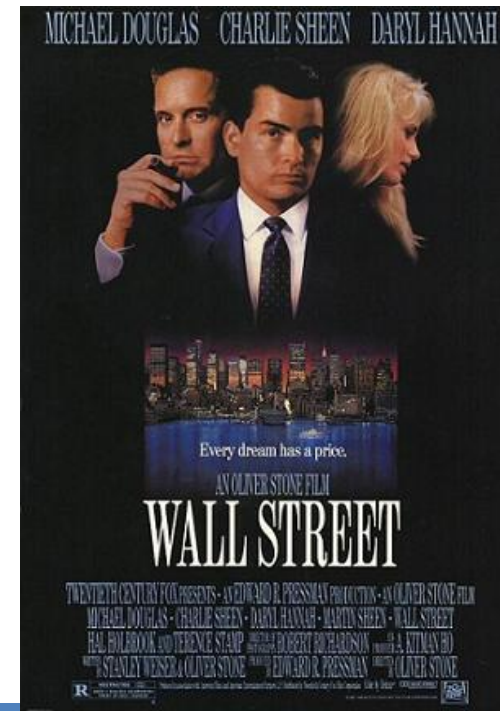
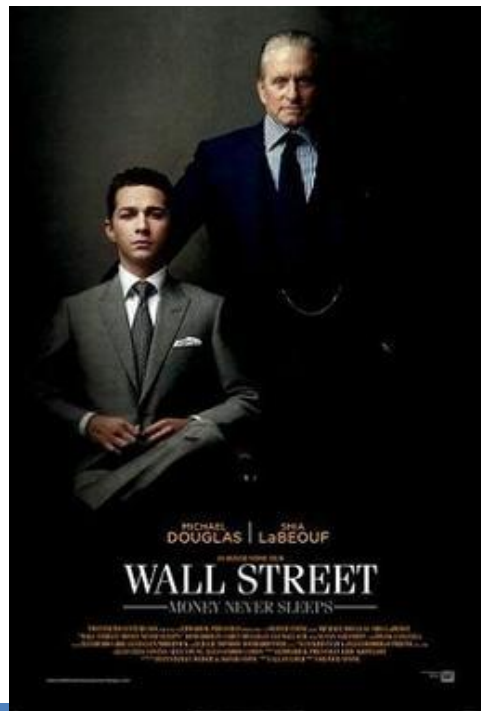
Summary: proof idea

Proposition. Both greedy algorithms compute an MST.

- Let X be the object produced by a greedy algorithm and X^* be *any* optimal solution.
- If $X = X^*$, the algorithm is optimal.
- Otherwise, show that you can *exchange* some piece of X^* for some piece of X without deteriorating the quality of X^* .
- Argue that this process can be iterated repeatedly to turn X^* into X without changing its cost.
- Conclude that X is optimal.

More——

Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit." - Gordon Gecko



Three Apparently Unrelated Optimization Problems

A “SEMIMATCHING” PROBLEM

Let $W = (w_{ij})$ be an $m \times n$ nonnegative matrix. Suppose we wish to choose a maximum weight subset of elements, subject to the constraint that no two elements are from the same row of the matrix. Or, in other words, the problem is to

maximize

$$\sum_{i,j} w_{ij} x_{ij}$$

subject to

$$\sum_j x_{ij} \leq 1 \quad (i = 1, 2, \dots, m)$$

$$x_{ij} \in \{0, 1\}.$$

$$W = \begin{bmatrix} 4 & \textcircled{6} & 4 & 5 \\ & & & 6 \\ 2 & 9 & 2 & \textcircled{10} \\ 1 & 2 & 3 & \textcircled{18} \\ 3 & \textcircled{8} & 1 & \end{bmatrix}$$

Choose the elements one at a time in order of size, largest first, rejecting an element only if an element in the same row has already been chosen.

Three Apparently Unrelated Optimization Problems

A SEQUENCING PROBLEM

A number of jobs are to be processed by a single machine. All jobs require the same processing time, e.g., one hour. Each job j has assigned to it a deadline d_j , and a penalty w_j , which must be paid if the job is not completed by its deadline. What ordering of the jobs minimizes the total penalty costs?

Job j	Deadline d_j	Penalty w_j
1	1	10
2	1	9
3	3	7
4	2	6
5	3	4
6	6	2

Choose the jobs one at a time in order of penalties, largest first, rejecting a job only if its choice would mean that it, or one of the jobs already chosen, cannot be completed on time.

Three Apparently Unrelated Optimization Problems

THE MAXIMAL SPANNING TREE PROBLEM

A television network wishes to lease video links so that its stations in various cities can be formed into a connected network. Each link (i, j) has a different rental cost a_{ij} . How can the network be constructed with minimum total cost?

Choose the arcs one at a time in order of their weights, largest, first, rejecting an arc only if it forms a cycle with arcs already chosen.

Greedy algorithm

matroid (拟阵)

More——

Greedy algorithms are by far one of the easiest and most well-understood algorithmic techniques. There is a wealth of variations, but at its core the greedy algorithm optimizes something using the natural rule, “pick what looks best” at any step. So a greedy routing algorithm would say to a routing problem: “You want to visit all these locations with minimum travel time? Let’s start by going to the closest one.”

Can we characterize when greedy algorithms give an optimal solution to a problem?

The answer is yes, and the framework that enables us to do this is called a **matroid (拟阵)**.

That is, if we can phrase the problem we’re trying to solve as a matroid, then the greedy algorithm is guaranteed to be optimal.

Matroids(拟阵)

Why study matroids?

- Matroids are common mathematical structures.
- In a matroid, we can always find the **minimum-weight maximal independent set** using the **greedy algorithm**.
- MST Algorithm: Apply the Red and Blue rules arbitrarily.

... ..

Theorem. The blue edges form a MST.

Matroids(拟阵)

A **matroid** is a pair (S, \mathcal{I}) where S is a finite set and \mathcal{I} is a family of subsets of S such that

- (1) \mathcal{I} is nonempty.
- (2) Elements of \mathcal{I} are called the independent sets. If I is in \mathcal{I} , then every subset of I is independent set.
- (3) If A, B are in \mathcal{I} with $|A| = |B| + 1$, then there is an element a in $A - B$ such that $B \cup \{a\}$ is in \mathcal{I} .

Matroids(拟阵)

- Graph $G = (V, E)$
 - \mathcal{I} is the set of forests in G (acyclic subgraphs).
- Vector space V
 - \mathcal{I} is the set of all linearly independent subsets of V .
- Columns/rows of a matrix A
 - \mathcal{I} is the set of all bases of A .

More——

Greedy Algorithm

- $B \leftarrow \emptyset$
- Sort nonnegative elements of \mathcal{X} in decreasing order of weight
 - $\{1, \dots, n\}$ with $w_1 \geq w_2 \geq \dots \geq w_n \geq 0$.
- For $i = 1$ to n :
 - if $B \cup \{i\} \in \mathcal{I}$, add i to B .

Theorem

The greedy algorithm returns the maximum weight set for every choice of weights if and only if the set system $(\mathcal{X}, \mathcal{I})$ is a matroid.

The Maximum/Minimum Spanning Tree Problem

The problem of testing for independence is equivalent to the problem of testing a subset of arcs for the existence of a cycle.

在组合数学中，拟阵是一个对向量空间中线性独立概念的概括与归纳的数学结构。拟阵理论广泛地借用了线性代数和图理论的术语，因为它是这些领域的重点概念的抽象。拟阵在图论、格论、几何、拓扑学、组合优化，网络理论和编码理论上都有很多应用。它抽象了很多图的性质，为组合优化问题和设计多项式算法提供了强有力的工具。

Matroids provide a link between graph theory, linear algebra, transcendence theory, and semimodular lattices, etc..

有关术语

有向树

根树 (Rooted Tree), 树高度

有序树: (完全)m分树、 (完全)二叉树

父结点/孩子结点;

分支结点/叶子结点, 内部结点/外部结点;

孩子结点/非孩子结点

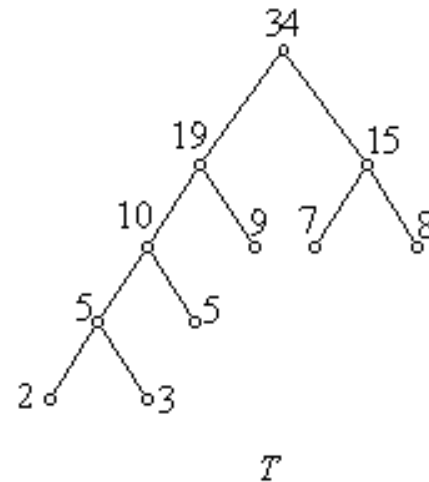
问题探讨

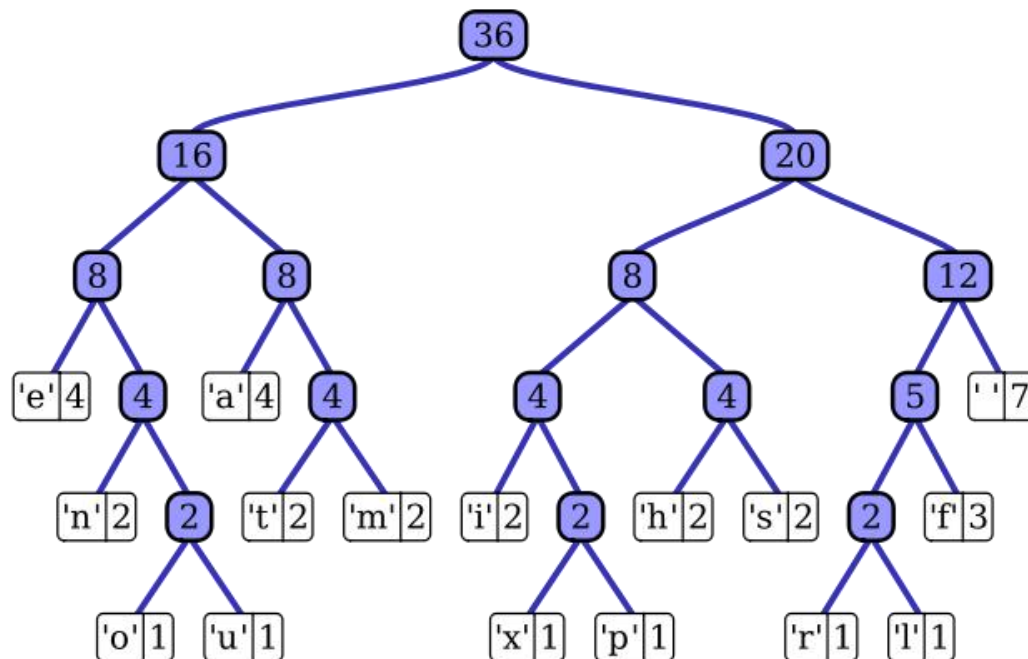
- (1) 存在一棵有 n 片叶的完全两分树;
- (2) 某完全 m 分树的叶子结点数为 t , 分支结点数为 i , 则 $(m-1)i=t-1$;
- (3) T 为有 t 片叶的完全两分树, 则 T 有 $2(t-1)$ 条边.

应用

- (1) 二叉树, 树的遍历, 逆波兰式
- (2) 前缀码, 最优二分树, Huffman算法

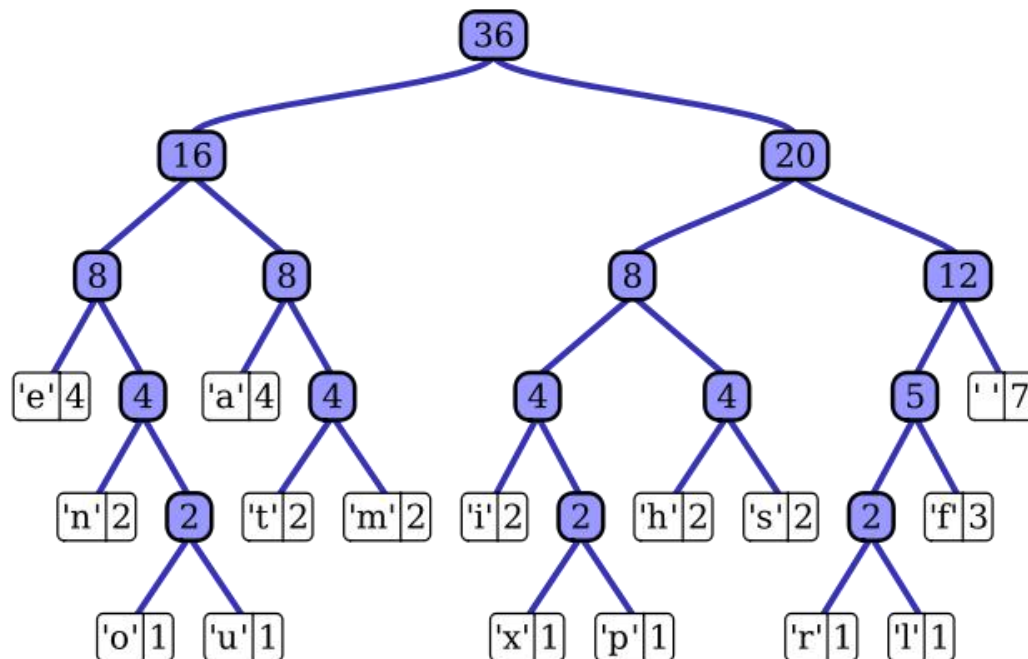
示例 求带权为 2,3,5,7,8,9 的最优二分树





Char ↕	Freq ↕	Code ↕
space	7	
a	4	
e	4	
f	3	
h	2	
i	2	
m	2	
n	2	
s	2	
t	2	
l	1	
o	1	
p	1	
r	1	
u	1	
x	1	

Huffman tree generated from the exact frequencies of the text "this is an example of a huffman tree". The frequencies and codes of each character are above. Encoding the sentence with this code requires 135 bits, as opposed to 288 (or 180) bits if 36 characters of 8 (or 5) bits were used. (This assumes that the code tree structure is known to the decoder and thus does not need to be counted as part of the transmitted information.)



Char ↕	Freq ↕	Code ↕
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010

Huffman tree generated from the exact frequencies of the text "this is an example of a huffman tree". The frequencies and codes of each character are below. Encoding the sentence with this code requires 135 bits, as opposed to 288 (or 180) bits if 36 characters of 8 (or 5) bits were used. (This assumes that the code tree structure is known to the decoder and thus does not need to be counted as part of the transmitted information.)

4 有序树 Order Tree

应用

二叉查找树

决策树

平衡树

1 高度为h的m元树：树叶数 $t \leq m^h$

2 若高度为h的m元树树叶数为t,

则 $h \geq \lceil \log_m t \rceil$,

若m元树为完全的平衡的, 则 $h = \lceil \log_m t \rceil$

