The Design of Facilities

In this chapter, we will use the word facility to mean a resource where productive activity is taking place in a company. A single enterprise may have to solve Facilities planning problems of different scale and scope:

(1) Global planning, or the ***Site location problem***: For example, the company must decide where to locate an assembly plant. Many HK companies must decide whether to place their factory in HK, China, Indonesia, Vietnam, etc. Many factors play a role in such decisions, e.g. labor availability, wage levels, transportation costs for materials, customs, tax and governmental subsidies, legal issues, etc.

(2) Site planning: How many buildings are required at the site, where are they located with respect to each other, how to connect them (for material flow, data connections, etc.)

(3) Building layout planning: Each building has one or more departments. Decisions made at this level include deciding the size and shape of each department, the layout of the departments in the building, and how materials can be moved between them.

(4) Department layout planning: A department may have many machines -- how to locate these machines with respect to each other, considering the flow of in-process materials between machines. Another example is the design of assembly-lines.

We first look at one example of each type of facilities planning problem. Next, we will consider one of these four problems, and study in detail how to solve it.

Example 1. Site location Problem
The New China Oil Company found oil in 7 different locations in the Gobi desert, as shown in the Map below. All oil must be sent in pipes to a central refinery to convert into different petroleum products. What is the best location for the refinery?

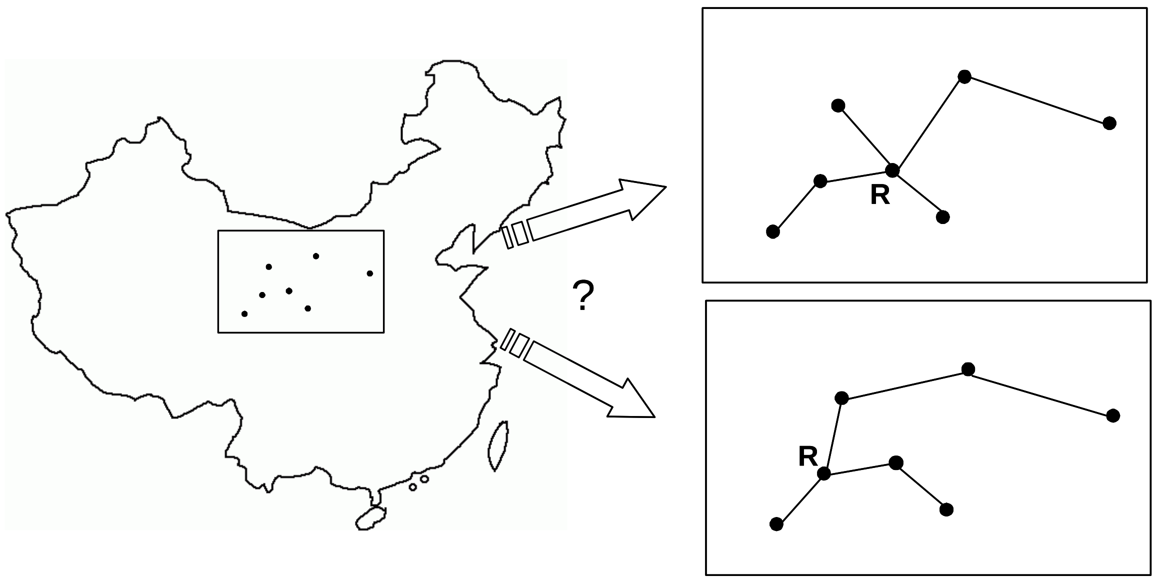


Figure 1. Facility location example

Example 2. Site Planning Problem

A footwear manufacturing company has a large factory, with several buildings, as shown in the figure below. The company wants to set up a network of fiber-optic cables for high-speed data communication between the buildings. The cables must be laid along the roads shown in the figure (this helps to perform repairs, and also ensures less disruption if more buildings are constructed in the empty spaces in the future). What is the shortest length of cables that can connect all buildings?

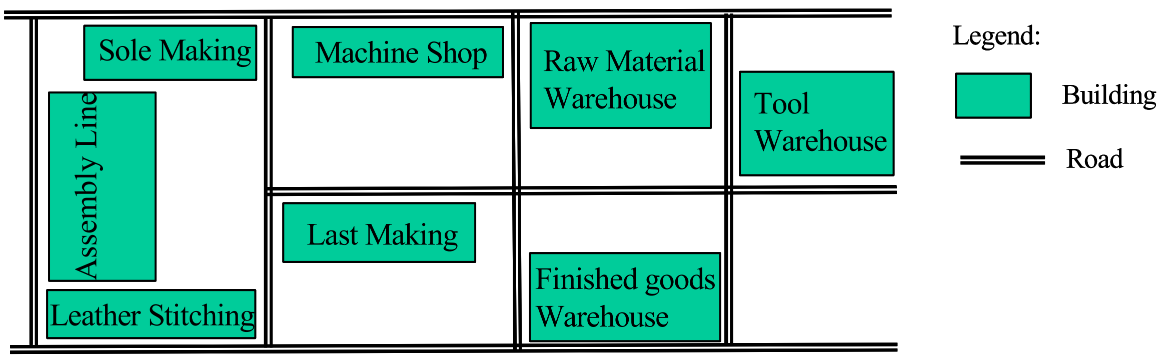


Figure 2. Map of factory and its buildings

Example 3. Building layout problem

The following figure shows images of different departments of a company that manufactures plastic components for consumer electronics (e.g. the plastic casing of mobile phones, or MP3 players). Each component is produced by a process called injection molding. The company has just purchased a new factory building (figure in center), and must now determine the size and location of each department.

[Note: for this class of problems, it is usually quite difficult to make a good mathematical model, and even more difficult to solve the problem mathematically for the best solution. In most cases, a 'good' solution is accepted, instead of hunting for the optimum one].

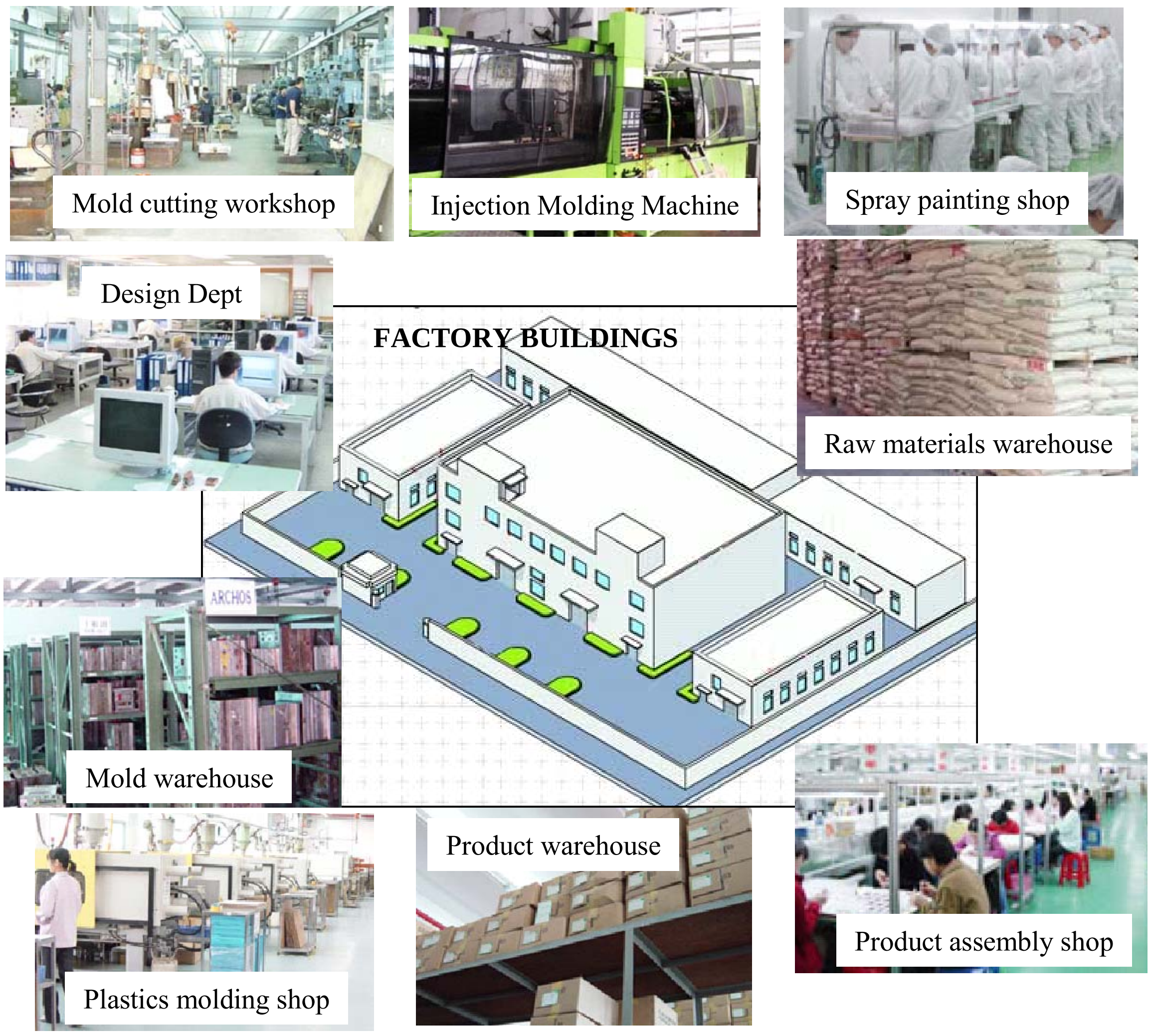


Figure 3. Example of a building layout problem -- several departments of different sizes must be located in the available space in a building.

Example 4. Department layout problem

The Royal Bicycle Company assembles and markets street bikes. A typical bicycle has over 100 components, as can be seen from the figure below. The factory must have the capacity to produce 1000 bicycles per day. Design the layout of an assembly line that is most suited for this task.

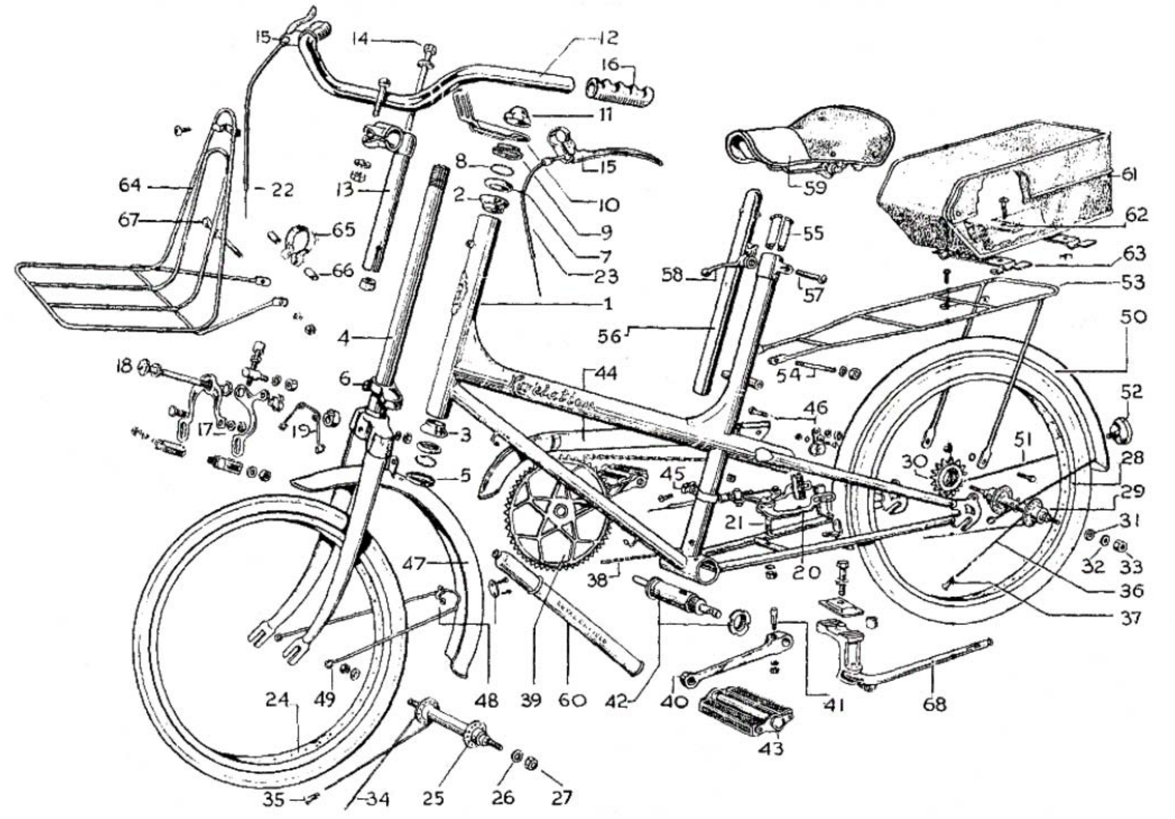


Figure 4. An exploded view of a bicycle

The above examples are presented to give you some idea of the nature of layout planning problems. The exact methods used to solve such design problems really depend on the precise details of the particular problem, and such details are mostly outside the scope of this course. Nevertheless, we shall look at a simpler type of layout design problem in detail below.

Example. A city is planning to construct a metro railway system (like our MTR). The city has 75 population centers that need to be connected by the train system. The construction costs are proportional to the total length of the metro system. Given a map of the city, how should the railway system be constructed so as to minimize the construction costs?

Let's first look at a smaller version of this problem, using a real city, Delhi, as an example. Given below is a map of Delhi, with nine population centers marked as circles -- their abbreviated names are written inside the circle All feasible railway links between these nodes are marked using lines.

Of course, we need not make all the feasible connections in order to get full connectivity. For example, if we omit the line between KB and CK, one could anyway travel from KB to CK by going through KB → CP → CK. So the question is, what is the minimum length of lines we should construct such that there is at least one connection between every pair of circles?

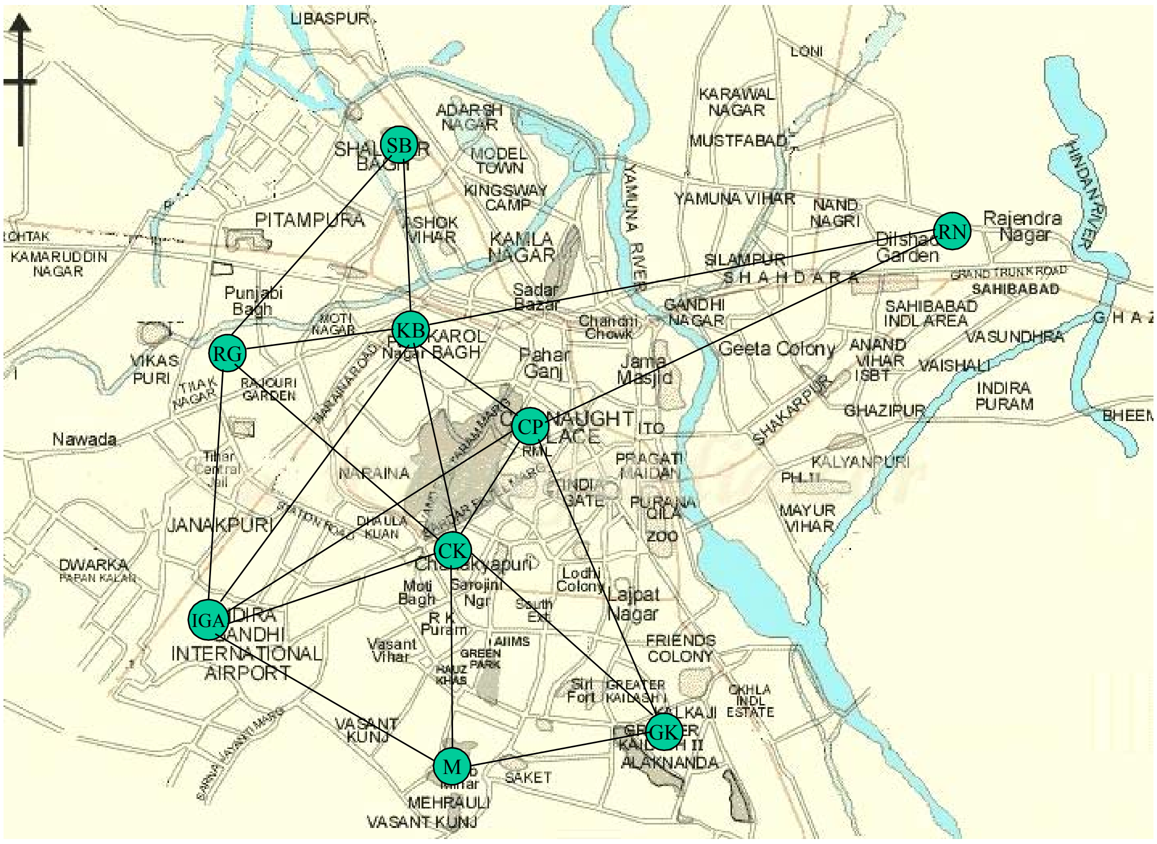


Figure 5. Map of Delhi, and graph of feasible railway links

Now it turns out that this, and thousands of other types of problems, can be solved using a very elegant tool in mathematics, called Graphs. We will shortly return to our problem, but first I will introduce some standard graph theory terms. It is useful to always use the same terminology, even when you model other problems.

Basic graph terminology

Graph: A graph is described by two sets of objects: a set of ***nodes*** and a set of ***edges***. Each *node* is usually depicted by its name. Each *edge* links exactly two nodes, and therefore it is depicted by a pair, (node1, node2). An edge is ***incident on*** each node on its ends.

It is conventional to chart out a graph as a figure, as shown in figure 1. Also, we will often denote the set of vertices as *V*, and the set of edges as *E*. In this convention, the graph will be denoted as *(V, E).*

Path: A path is a sequence of nodes, $n_0, n_1, \ldots, n_{k+1}$ such that each n_i is a node in *V*, and pair (n_i, n_{i+1}) for $i = 0, .., k$ is an edge in E. We can only move from one node to another if the two are connected by a path, and such "moving" is called ***traversing*** the graph. The ***length of a path*** = number of edges in the path.

Cycle: A path that has the same node as its start and end points is a cycle. Thus a cycle is a path of the form $n_0, n_1, \ldots, n_k, n_0$.

Weighted graph: A graph in which each edge is associated with a real number, called its weight, is a weighted graph. Weighted graphs are very useful in modeling, for example, street maps. Each connection of two roads is denoted by a node, each stretch of road is an edge, and the distance between the two nodes connected by the edge is its weight.

A Directed graph, or ***Digraph*** is a graph whose edges also have a direction; here we denote each edge by a pair (***tail***, ***head***), and one can only traverse a directed edge from tail to head. A directed edge is ***incident from*** the tail, and ***incident to*** the head. In such graphs, we will refer to the tail as the ***parent***, and the head as the ***child***. For example, in figure 1 (iv) below, node *c* is the parent of node *e*. Further, if there is a path <n_1, …, n_k> in a graph, then n_1 is an ***ancestor*** of n_k, and n_k is a ***successor*** of n_1. For instance, in figure 1 (iv), node *a* is an *ancestor* of node *e*; node *d* is a *successor* of node *f*, etc.

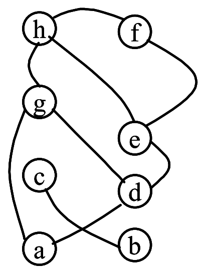
Degree of node is the number of edges that are incident on the node. In a digraph, we denote the number of incoming and outgoing edges by ***indegree*** and ***outdegree***, respectively.

A **connected graph** is one in which every pair of vertices has a path connecting them.

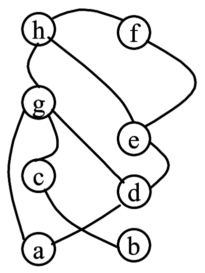
A **strongly connected digraph** is one in which every vertex is reachable from every other vertex.

A graph with no cycles is called **acyclic**. A **directed acyclic graph** is denoted as a **DAG**.

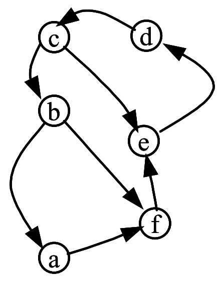
A **tree** is an undirected, acyclic, connected graph.



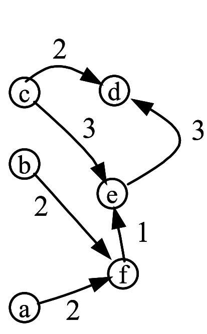
(i) Unconnected graph



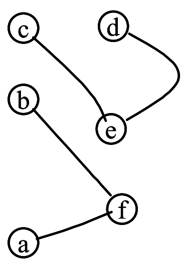
(ii) Connected graph



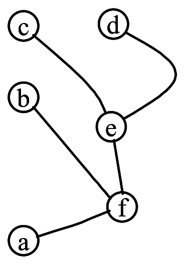
(iii) Strongly Connected Digraph



(iv) Weighted DAG



(v) Forest



(vi) Tree

Figure 6. Different types of graphs

Solving the Delhi railway line layout problem: The minimum spanning tree problem

It is easy to see that the problem can be naturally represented by a graph -- each station is a node, and the connection between two neighboring stations is an edge. The length of the railway line representing an edge is the weight of that edge. Now let us examine some properties of the solution:

Let G(V, E) be a graph representing the problem. V = the set of nodes (each node represents one point that needs to be connected), and E = set of edges (each edge shows a connection between two nodes; the weight of each edge is the distance between the two nodes that it connects).
Notice that not each pair of nodes may be connected with an edge; in our example, this may represent a condition that two population centers cannot be directly connected by rail (e.g. there is no direct connection between Tsim Sha Tsui and Causeway Bay due to the presence of the Victoria harbor in HK).

Property 1. The optimum set of connections is a sub-graph M(V', E') of G, such that V' = V, and $E' \subset E$.
Proof:
If $V' \neq V$, then either there is a node in V' that is not in V – which is impossible, or there is a node in V that is not in V', in which case our optimum solution cannot find a way to connect to this node. This violates our requirement that all nodes should be reachable.

Since all nodes of V are also part of the solution, we say that the solution ***spanning subgraph*** of the problem (i.e. it spans across all nodes).

Property 2. The optimum solution is a tree.
Proof.
If the optimum was not a tree, then it must contain a path that is a cycle, of the form $<n_a, n_b, \ldots, n_k, n_a>$.
We can now select one of the edges, and remove it from the solution. Observe that all nodes will still remain connected. At the same time, we have reduced the total cost by an amount equal to the weight of the edge that we cut. This should not be possible if we started at the optimum solution; therefore the optimum must not contain any cycle, and so it must be a tree.

Obviously the optimum solution is a spanning tree such that the sum of the weights of its edges is minimized. We call it the ***minimum spanning tree***, or ***MST***.

We look at a method to solve the MST problem first. Later, we shall get some insight into why it works. The method is systematic, and can be written in the form of a block diagram or a computer algorithm. However, let us work with a more intuitive description. Just imagine that each edge is made of an elastic band, and each node is a ball.

Prim's method to find the minimum spanning tree of a graph.

Step 1. Put the entire graph (all nodes and edges) in a bag.

Step 2. Pull any one node out of the bag; the edges that are incident on this node are now crossing the boundary of the bag.

Step 3. Among all edges that cross the boundary of the bag, pick the one with the minimum weight. Add this edge to the MST.

Step 4. Follow this edge to the node inside the bag, and pull that node out of the bag. In doing so, if some edge has both its end nodes outside the bag, then this edge must no longer cross the boundary of the bag.

Step 5. Repeat steps 3 and 4 until the bag is empty.

Let us look at an example to understand the method. In figure 7 below, the metro planning set for Delhi from our above example is repeated. For clarity, I am omitting the map itself. The number written on each edge is the length of the rail line corresponding to that link. Notice that this representation of our problem is a weighted, non-directed graph.

Let's apply Prim's method to our graph.

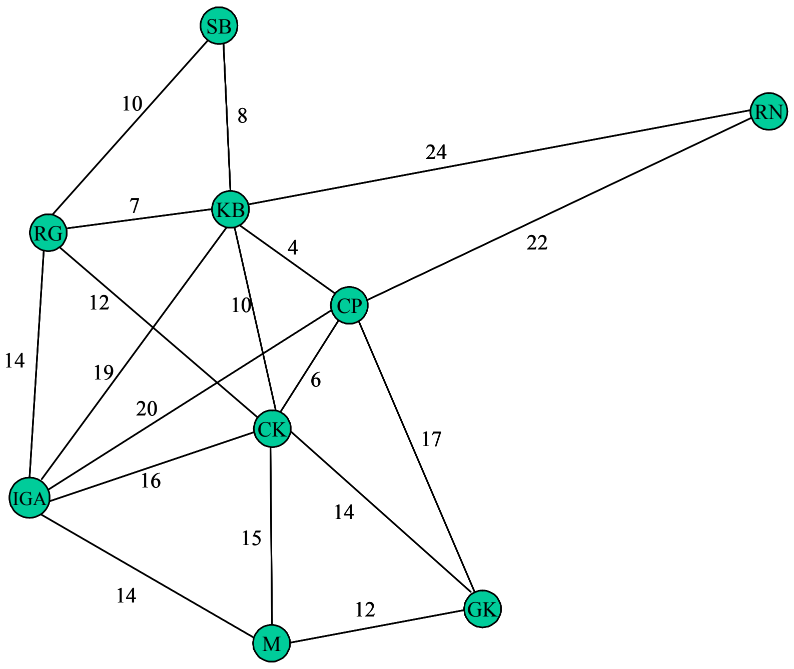


Figure 7. The network of train stations and link distances

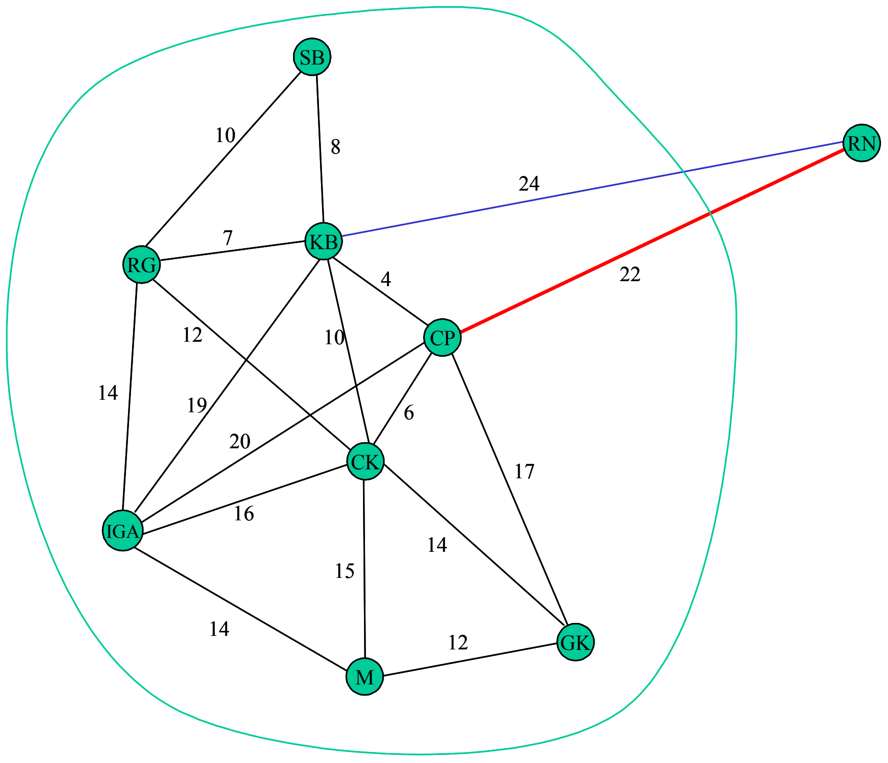


Figure 8. Prim's algorithm, 1st iteration. Green line represents the bag; pull any node out of the bag -- here, we arbitrarily select node RN. RN has two edges crossing the bag, RN-CP and RN-KB. RN-CP has the lower weight, and is selected (marked in red).

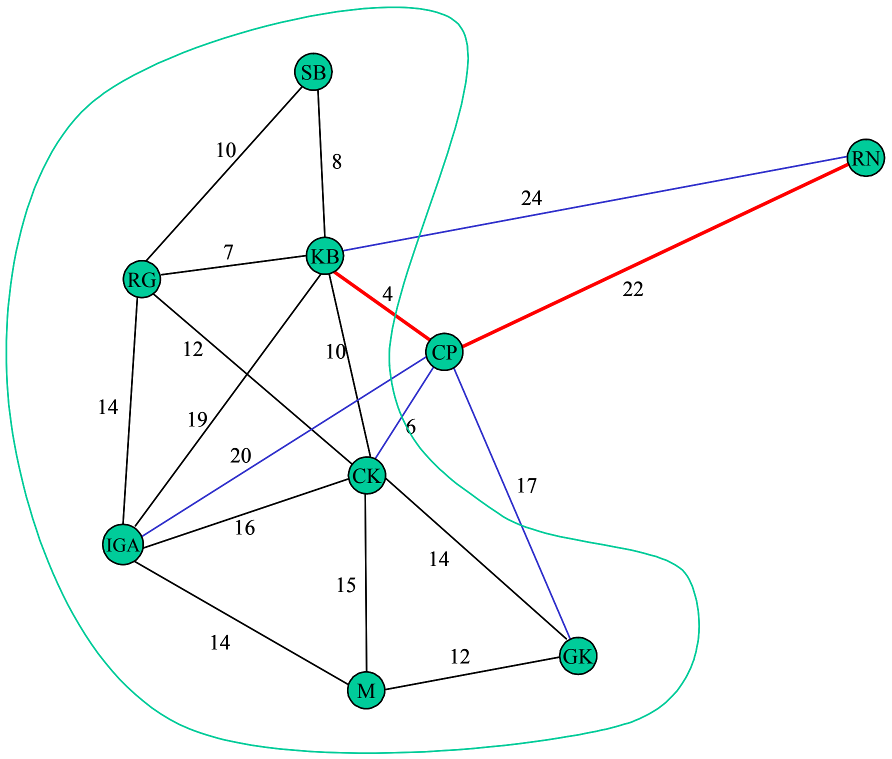


Figure 9. Prim's algorithm, 2nd iteration. Since RN-CP was selected, node CP is pulled out of the bag. Out of the five edges crossing the bag now, CP-KB is cheapest.

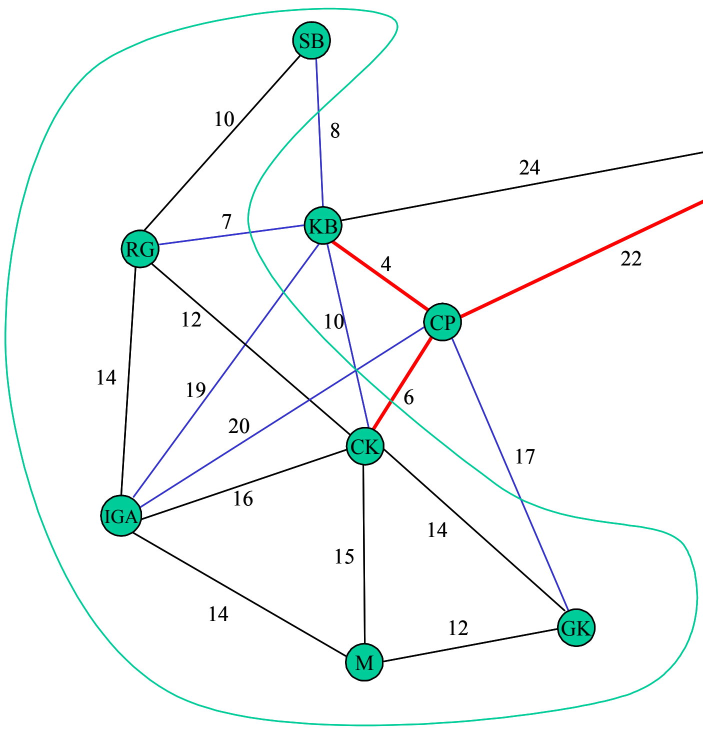


Figure 10. Prim’s algorithm, 3rd iteration

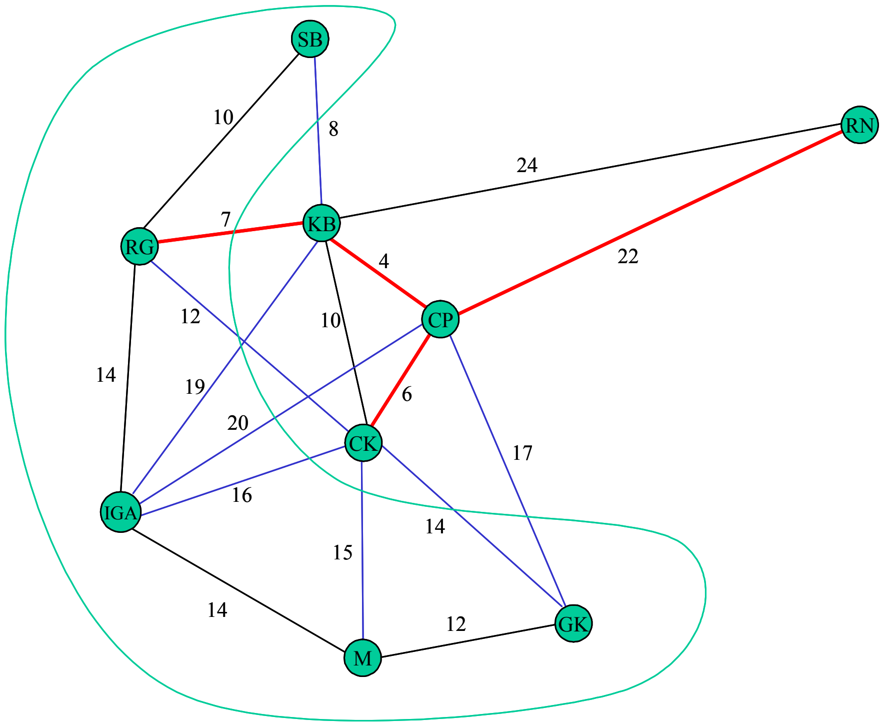


Figure 11. Prim’s algorithm, 4th iteration

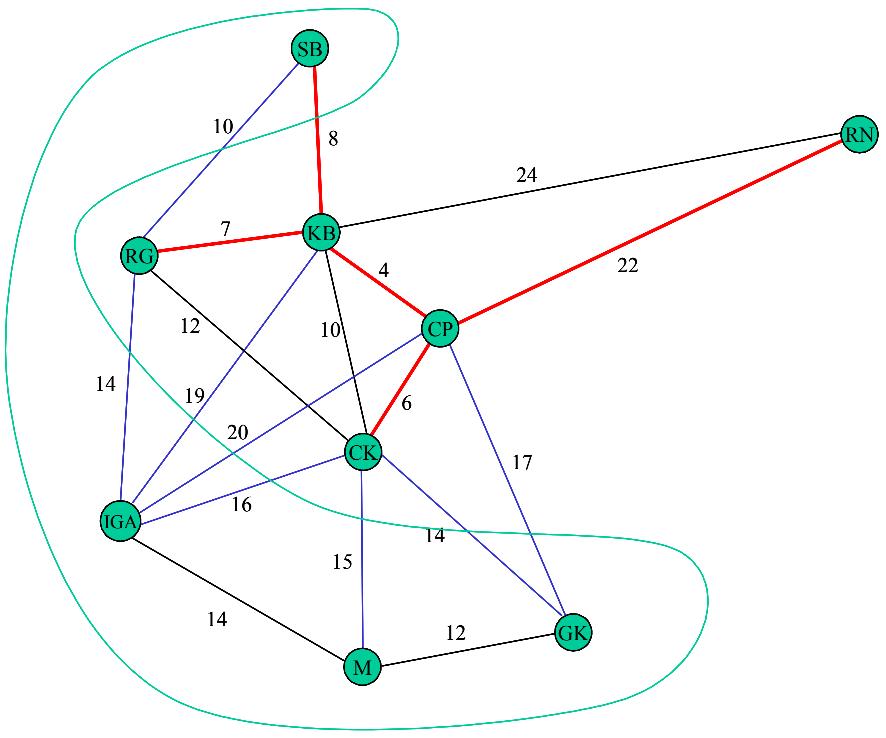


Figure 12. Prim's algorithm, 5th iteration

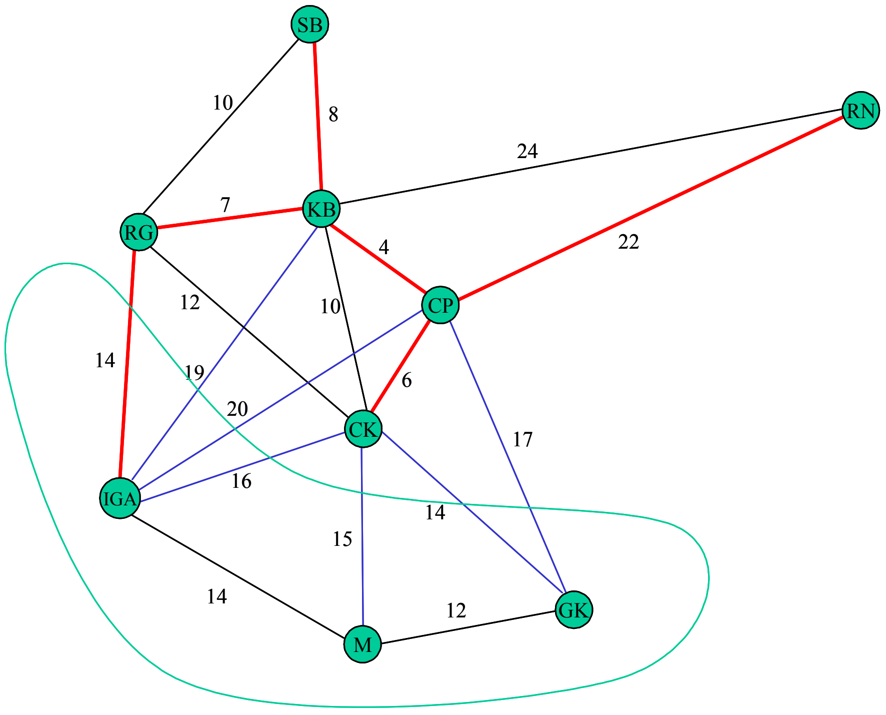


Figure 13. Prim's algorithm, 6th iteration. Notice that there are two edges that have the same minimum weight in this step, RG-IGA and GK-CK. We may select any one.

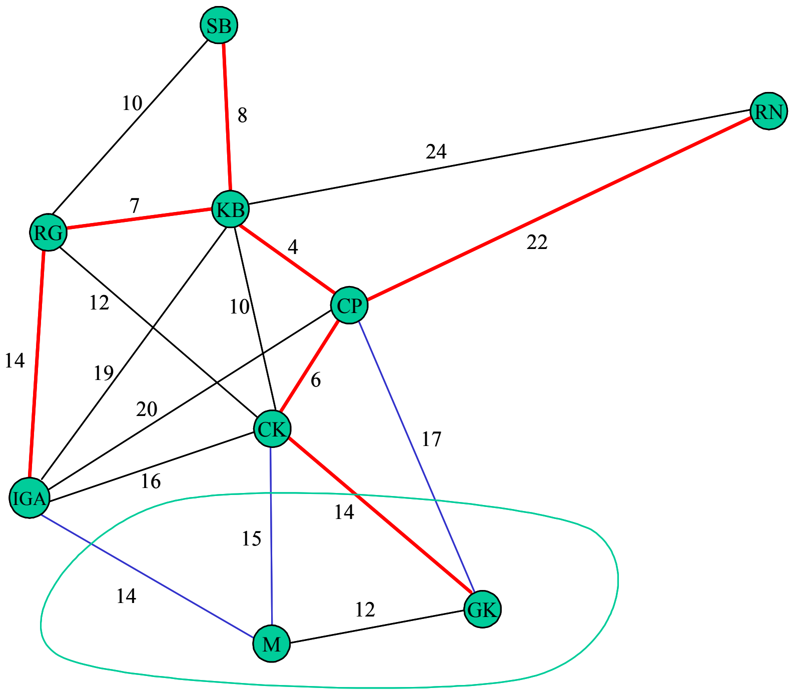


Figure 14. Prim’s algorithm, 7th iteration

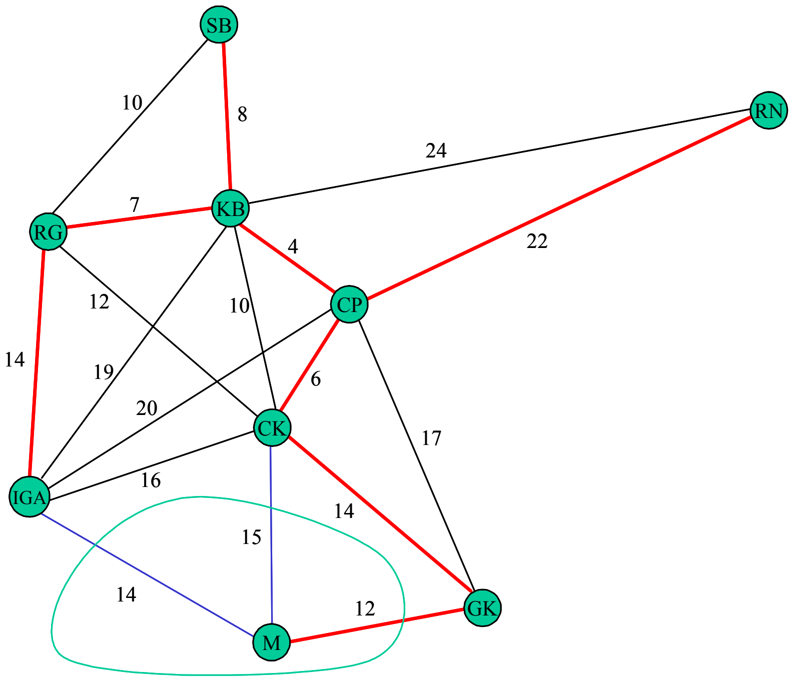


Figure 15. Prim’s algorithm, 8th iteration

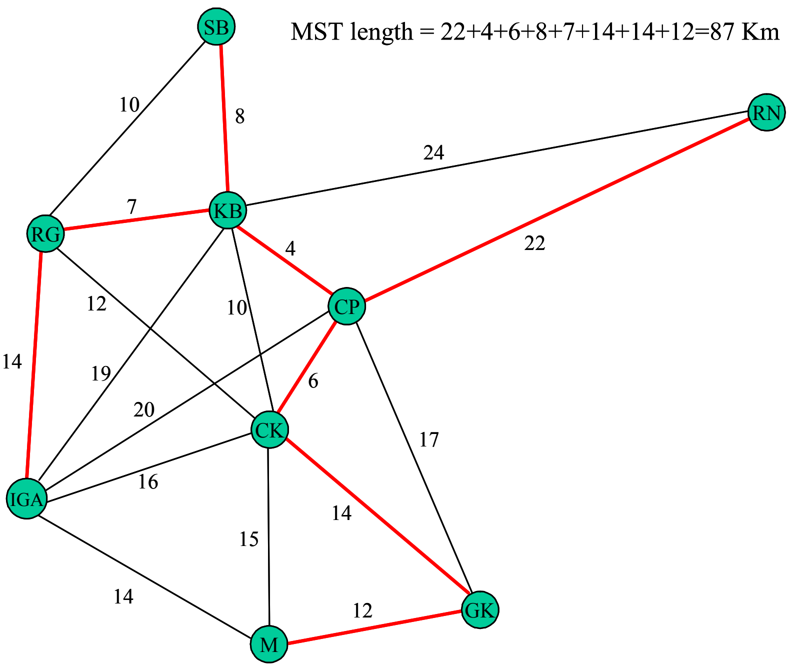


Figure 16. Prim’s algorithm: an MST

Some important pints to note:

1. For a given graph, there may be more than one different Minimum Spanning Trees (each with the same weight, of course.) For example, in our case, the figure below shows an alternate MST.

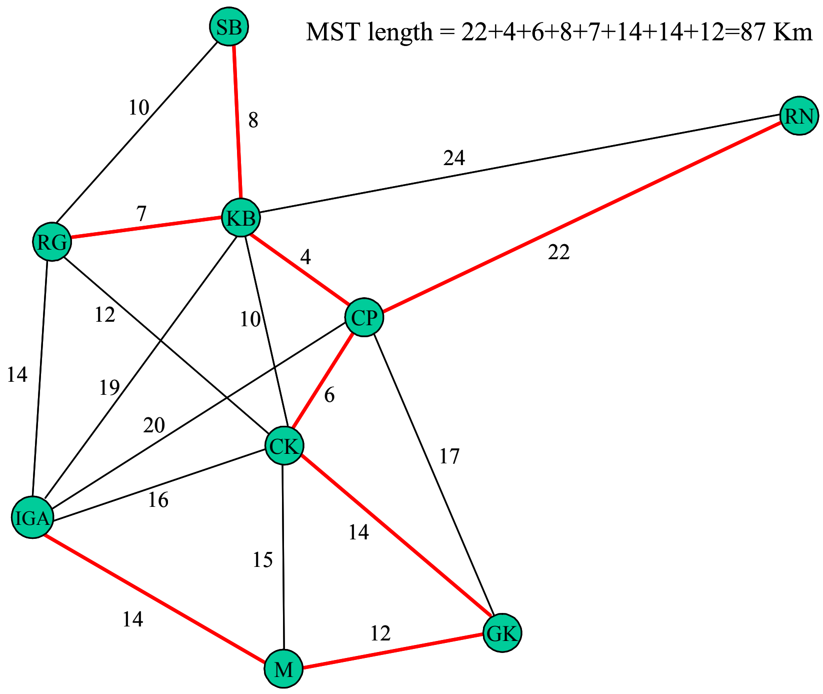


Figure 17. An alternate MST on the same graph

2. The size of the MST will be the same regardless of which node we begin our procedure.

3. Prim's method is very efficient – we could get the MST without really having to examine many of the combinations (a large graph may have millions of sub-graphs).

Proof of correctness:

Notice that at each step in our algorithm, the partially constructed MST is outside the bag, and at each step, we pick one of the edges crossing the bag as the next edge to join the MST.

We will give an inductive proof. Assume that at some intermediate step of Prim's algorithm, we have connected a set of nodes, ***partialMST***, which are (i) all outside the bag, and (ii) which form a subset of an MST.

At this stage, each edge that crosses the bag (i.e. has one node outside, and the other node inside) is called a ***candidate***; the least weight candidate is called a ***light-edge***, connecting a node outside the bag, $\boldsymbol{e_{out}}$, with some node inside the bag, $\boldsymbol{e_{in}}$.

Assume that the ***light-edge*** is not a part of the MST. Then we should be able to find some path, ***p***, that (i) connects $\boldsymbol{e_{out}}$ and, $\boldsymbol{e_{in}}$, (ii) does not contain ***light-edge***, and (iii) is a subset of the MST.

This path, ***p***, must contain some edge, ***heavy-edge***, that is a candidate (because the path must go from inside the bag to outside). Now, ***heavy-edge*** has a greater weight than ***light-edge***, so if we replaced ***heavy-edge*** by ***light-edge*** in the MST, we will get a lower total weight. But this contradicts our assumption that ***light-edge*** is not a part of the MST.

Therefore ***light-edge*** must be a part of the MST. The figure below illustrates this idea.

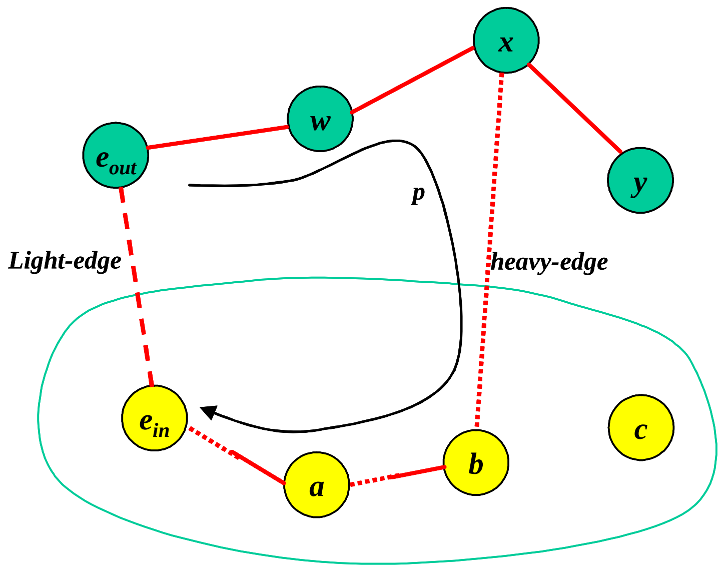


Figure 18. Proof of correctness of Prim's MST algorithm

Notes:

1. I have skipped some technical details about the case when ***heavy-edge*** and ***light-edge*** have the same weight, but the main idea of the proof remains the same.
2. The base case of the induction, i.e. the analysis for the first node to be pulled out of the bag, is similar to the proof for the intermediate stage.

We conclude by noting two things:
(a) The MST has many practical applications. It is an excellent tool to know if you will be involved in design of networks, communication systems in your factory, pipelines, etc. At the same time, you must also recall that the "optimal" design is just a starting point for the best design. For example, one weakness of an MST-type solution is that the failure of any one link will result in a situation where some nodes cannot be connected in any way to some others (this is because the MST is a tree, and cutting any edge will break a tree into two ***disconnected*** sub-trees). In real life, this may be unacceptable, so some redundant links are often built to allow for alternate routings.
Another issue is that while the MST solution optimizes total cost of the network, but it may result in unacceptably long travel distance between two physically close nodes (e.g. consider traveling from IGA to M in the first solution to our MST example).

(b) There are many spanning sub-graphs of a given graph. It is extremely time consuming to list them all in attempting to find the minimum weight one. However, some clever analysis allowed us to set up a really easy way that is guaranteed to give us the best possible solution. The difference between a "good feasible solution" and an "optimum" solution may be sufficient to convert a failed project to a successful one.

Summary of important points from this chapter
- Facility layout problems are concerned with the optimum design of a system.
- There are many different types of facility layout problems, each of which may require a different mathematical method if we wish to find the best possible solution.
- We learnt the basic terminology of graphs
- We learnt Prim's method for finding MST on graphs, and saw how it can be used to solve one type of facility layout problem.