Looking Into Minimum Spanning Trees

11811810 Jiyuan Chen

Topic One : What is a Tree and what is a Graph (brief introduction) ?

Graph: A graph G = (V , E) consists of V , a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints.[i]

- **Undirected Graph**

  A graph is undirected means that there is no limitation of directions on any edges in the graph connecting two vertices. If G is an undirected graph and E is an edge connecting vertex A and B, then by definition, you can either travel from A to B via E or from B to A via E.

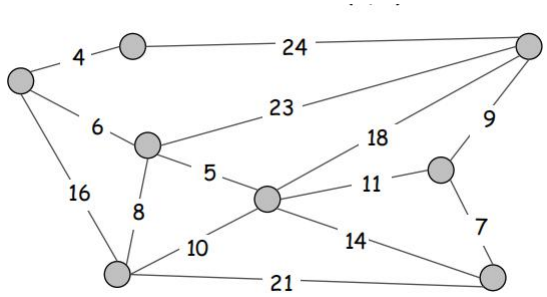
- **Weighted Graph**

  Graphs that have a number assigned to each edge are called weighted graphs.

- **Connected Graph**

  A graph is connected if any of its two vertices is connected by an edge.

An example of weighted undirected graph:

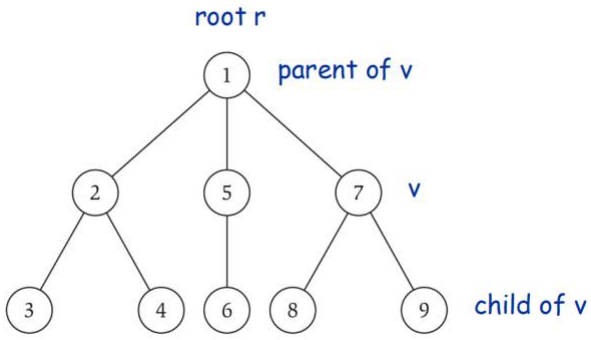


Tree: A tree is a special connected graph that has no circuits.

- **Rooted tree**

  Given a tree T, choose a root node r and orient each edge away from r.

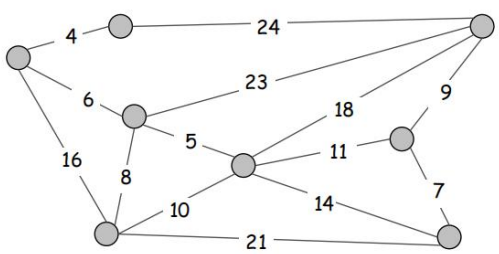
An example of rooted tree:



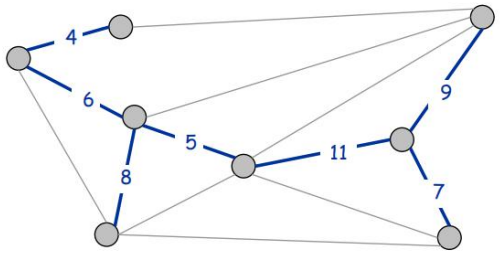
Topic Two : Minimum Spanning Trees and its Applications

Minimum spanning tree: Given a connected graph G = (V, E) with real valued edge weights c_e , an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized. Note that MST is always with respect to undirected weighted graph.

Example:



G = (V, E)



T, $\Sigma_{e \in T} c_e = 50$

Applications:

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

Example：

> Yuki is a wise girl and she rules a country named Blossom.
>
> Blossom is a big country with n cities, and there are m roads determined to be constructed. Due to many reasons during construction, different roads might have different costs to be built. The roads are **bidirectional**, that is a road from u to v **can** be passed from v to u.
>
> Since Yuki wants to make the construction more economical, she decides to choose some of the roads to construct and wants to spend the least money. However, to make the traffic in Blossom convenient, all the cities in Blossom are needed to be connected after construction. Your task is to determine the **minimum** cost of construction.

ii

It can clearly be viewed from the question that it is a classic MST problem . We can just construct a graph consisting of all cities and roads and then find out its MST to solve the problem.

Topic Three : Prim Algorithm and its Correctness

Assumption: All edges in the graph have different costs

The pseudocode of Prim Algorithm:

ALGORITHM 1 Prim's Algorithm.

procedure *Prim*(G: weighted connected undirected graph with n vertices)
T := a minimum-weight edge
for $i := 1$ **to** $n - 2$
  e := an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T
  $T := T$ with e added
return T {T is a minimum spanning tree of G}

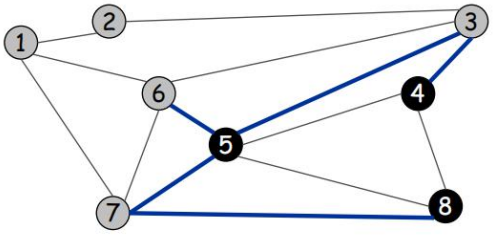
iii

Let us say that the set of all vertices of the Graph G is V , and the set of vertices in the Minimum Spanning Tree is MST . Then we can analyze as follows: Prim Algorithm is derived from Greedy Algorithm to find the optimal solution by each time adding the closest vertex from V-MST to MST. The Algorithm will stop after adding |V| vertices into MST.

Some Basic Knowledge to know:

- **Cut Property:**Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.
- **Cycle property.** Let C be any cycle, and let f be the max cost edge belonging to C. Then the MST does not contain f.
- **Cutset.** A cut S is a subset of nodes. The corresponding cutset D is the subset of edges with exactly one endpoint in S

  Example：



Cut S = { 4, 5, 8 }
Cutset D = 5-6, 5-7, 3-4, 3-5, 7-8

- **A cycle and a cutset always intersect in an even number of edges**.

- **Proof of Cut Property (Using contradiction):**

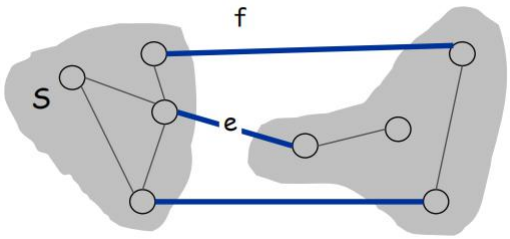
(1). Suppose that e does not belongs to MST and f belongs to MST (we are sure about this because there must be an edge belongs to both the Cutset of S and MST).

(2). Then we can find that we will have another spanning tree MST* .

MST* = MST ∪ e - f

(3) . Then because cost(e)<cost(f), we find thatcost(MST*)<cost(MST).

So MST is not the Minimum Spanning Tree of G. Contradiction!



- **Proof of Cycle Property (Using contradiction):**

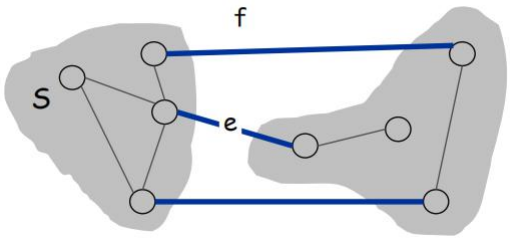
(1).Suppose there is a cycle C in graph G and f is the largest cost edge in the cycle. Also f is in MST.

(2). Then we delete f from MST, this will certainly create a Cut S in MST

(3). We can see that in Graph G , f is both on the Cutset D corresponding to S and the cycle C. This leads to the assertion that there must be another edge e both on the Cutset D corresponding to S and the cycle C (See what we have claimed in Basic Knowledge)

(4). Then there is a spanning tree MST* = MST ∪ e - f , and

cost(MST)>cost(MST*).Contradiction!



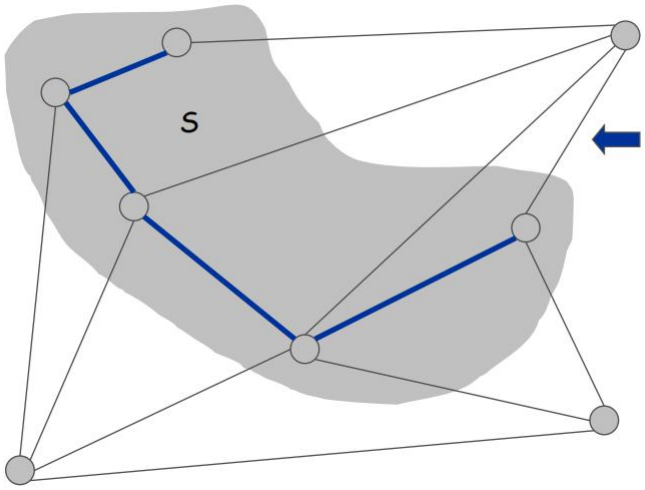
Proof of Correctness of Prim Algorithm:

(1). Initialize S = any vertex.

(2). Apply cut property to S.

(3). Add min cost edge in Cutset corresponding to S to MST, and add one newly explored vertex v to S

(4). It is really clear that all the edges we add into the MST belongs to MST



Topic Four : Kruskal Algorithm and its Correctness

Assumption: All cost of edges in the graph are distinct

The pseudocode of *Kruskal* Algorithm:

ALGORITHM 2 Kruskal's Algorithm.

procedure *Kruskal*(G: weighted connected undirected graph with n vertices)
T := empty graph
for $i := 1$ **to** $n - 1$
  e := any edge in G with smallest weight that does not form a simple circuit when added to T
  $T := T$ with e added
return T {T is a minimum spanning tree of G}

iv

We can see that Kruskal is a really really greedy algorithm to find out the Minimum Spanning Tree.We can first sort all the edges in Graph G by their cost, from low to high.Then we initial MST as an empty set.We next continually add the smallest edge that doesn't belong to MST into MST. Be aware that the edge adding into MST can't form a cycle within MST.(We can later use a data structure called **Union-Find** to solve this problem)

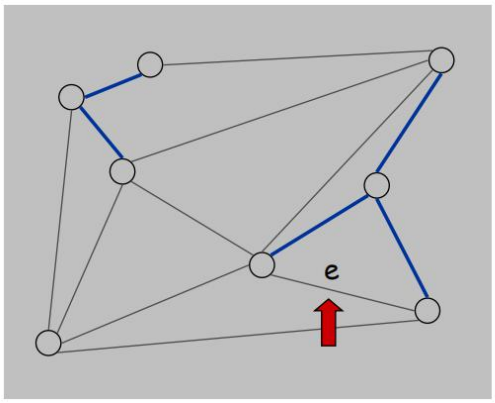
Proof of Correctness of Kruskal Algorithm (Direct proof):

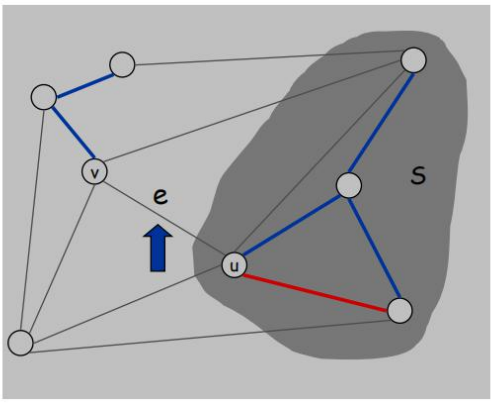
(1). We let the currently smallest edge not belonging to MST as e

(2). If adding e to MST will create a cycle in MST,then according to previous cycle property,we know that e doesn't belong to any Minimum Spanning Tree of the current graph G.

(3). If not,we can see that e is in the Cutset of MST and it's the smallest

among all .So we can see that e must belong to MST and thus we add it into MST.





Difference between Prim and Kruskal:

(1). Prim Algorithm is more like extending its domain and finally get the answer. We can see that it always temps to add the edges that is connected with current MST.After all it is derived from Dijkstra.

(2). However, Kruskal is more arbitrary.The edges that added into MST each time does not necessarily connected with current MST.This is a very clever algorithm that use cycle property and cut property very well.

(3). For both algorithm, the plain implementation will cost

$O(num(edge) * \log_{num(edge)})$

Topic five: Some optimization while implementing

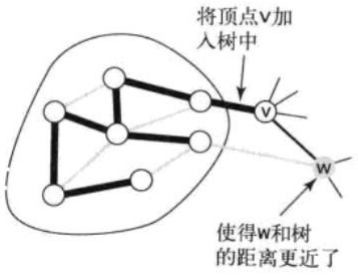
For Prim:

(1) . We can always use a **Priority Queue** to store the edges that are now the Cutset of MST. We sort the edges in Priority Queue with its cost

from low to high , and each time when we want to add an edge into MST,we just simply pop the top of Priority Queue.

(2) .**Lazy-Prim** is the lazy version of Prim Algorithm.It adds all the edges that in the Cutset of MST and sort them.The algorithm will stop after adding |V| vertices or |V-1| edges.

(3) But we quickly find out there is a problem in **Lazy-Prim.**For example:



v

Suppose the circle contains all the nodes that are currently in MST.And now in Priority Queue we have two edges,one is from MST to vertex V , **e1** , and one is from MST to vertex W, **e2**. Clearly that the edge to V has less cost, so in the next stage we add V into MST . Now we further find that V also have an edge to W. If follows the convention of **Lazy-Prim,**we certainly have to add edge(V,W) ,**e3**, into the Priority Queue . But after consideration ,we find the cost(e3) is less than cost(e2), which means , according to **Prim Algorithm** that always adds the closest vertex to MST, the edge e2 is of no use any more because even if we want to add W into MST , the edge connecting W and MST will be e3 rather than e2.

(4) The problem we find above give us the thought to invent another form of Prim Algorithm called **Active-Prim**. In this algorithm, when we come across the above problem, we simply change the edge connecting MST and W from e2 to e3 in the Priority Queue. These steps will certainly decrease the sorting time in Priority Queue and thus improves the performance.

For Kruskal:

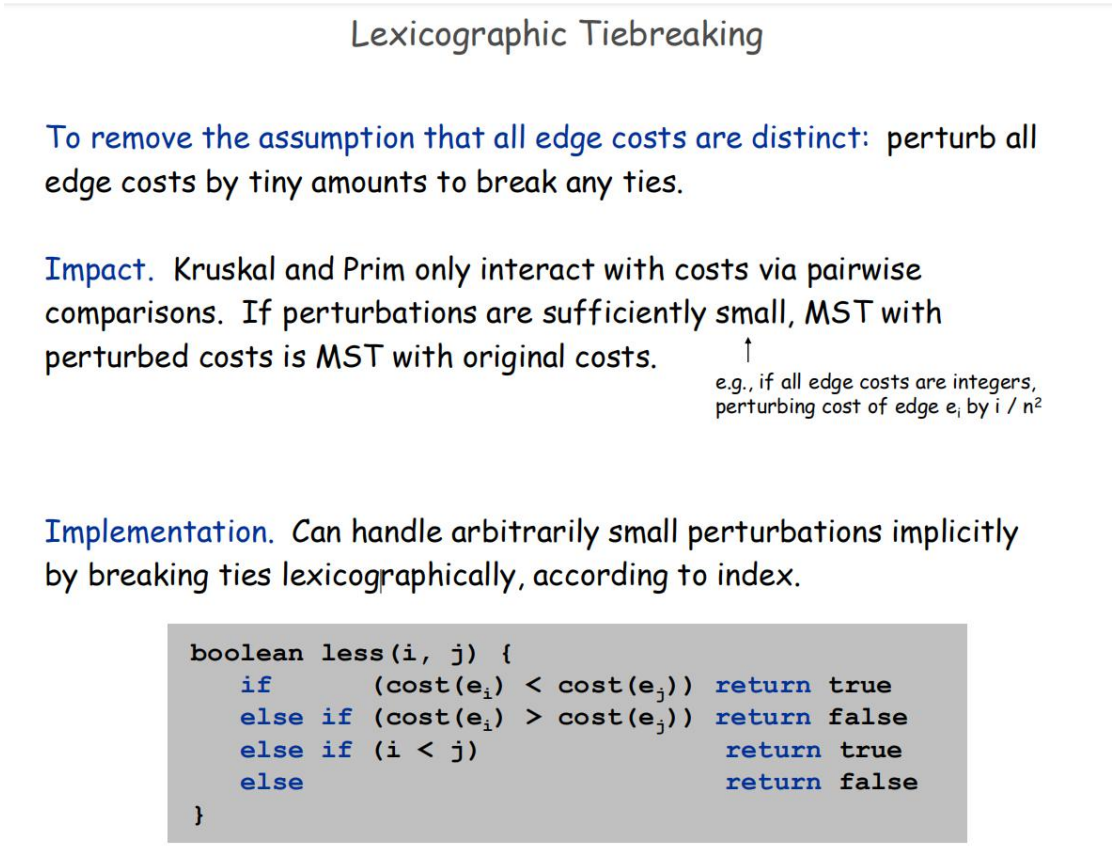
The difficulty in Kruskal is undoubtedly how to indicate if the newly added edge will form a cycle inside MST. We can implement a data structure called **Union-Find** to sovle this problem.But it is not the topic that we try to cover in this paper.We have provided with Java code that implement **Union-Find** for you to have a better understanding.

Java code:

```java
public class Union_Find {

    private int[] id;// 父链接数组
    private int[] sz;//size
    private int count;// 一共几个联通分量

    public Union_Find(int N) {
        count = N;
        id = new int[N];
        sz = new int[N];
        for (int i = 0; i < N; i++) {
            id[i] = i;// 一开始每个人自己就是一个联通分量
            sz[i] = 1;// 一开始大家的sz都是1
        }
    }

    public int count() {
        return count;
    }

    public int find(int p) {
        while (id[p] != p) {
//            将p节点的父节点设置为它的爷爷节点，下面一条语句是路径压缩
            id[p] = id[id[p]];//就这个是路径压缩
            p = id[p];
        }
        return p;
    }

    public boolean connected(int p, int q) {
        return find(p) == find(q);
    }

    public void union(int p, int q) {
        int k1 = find(p);// 拿到两个根节点
        int k2 = find(q);

        if (sz[k1] < sz[k2]) {
            id[k1] = k2;
        } else {
            id[k2] = k1;
        }
        count--;
    }
}
```

For Both:

We all knows that the theory of Prim and Kruskal is based on a fact that all the cost of edges are distinct. Although in real implement both two Algorithm runs well,how about the theoretical part ?



vi

The key is to slightly change the cost of the edge so that it fits into theory again.Also, the little change can have maybe no effect on the MST.

Topic six: Expansion

- First let's look at the following problem:

Yuki is an ambitious girl and she is addicted to a game called *Honor of Kings*.

In the game, Yuki is controlling the King to move in a grid of n rows and m columns, where rows are numbered from 1 to n and columns are numbered from 1 to m. The cell at the i-th row and the j-th column is denoted by (i, j). Each cell in the grid contains a *point coefficient*, denoted by C_{ij}.

At first, Yuki can place the King on the grid arbitrarily, that is any cells in the grid can be the initial position for the King. Every turn Yuki can move the King between the cells sharing a **common edge**. For example, when the King is at (i, j), it can be chosen to move to $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ or $(i, j+1)$, if the destination is not out of the boundary.

Now every time when the King is moved from one cell to an **unvisited** cell, Yuki will gain the points which is equal to the product of two *point coefficients*. It means that Yuki will get $C_{xy} \cdot C_{ij}$ points when the King moves from (i, j) to (x, y) and visits (x, y) at the **first** time.

Yuki can stop the game at any time, and she wonders the **maximum** score she can gain.

vii

We can also construct a graph according to the question. But how to find the maximum score ? We have already know the algorithm to find the minimum score of the question, then how to convert it into maximum?

These all leads to the **Maximum Spanning Tree** problem .Maximum Spanning Tree of a connected weighted undirected graph is a spanning tree with the largest possible weight.

In fact Maximum Spanning Tree can also be well solved by Prim or Kruskal. We just simply change the sign of each edge while constructing the graph , and then we imply Prim or Kruskal to find the MST .
Finally ,we change the sign of the total weight of MST again and we will get a Maximum Spanning Tree.

- We all knows that Prim or Kruskal will find out only one MST when the graph G is connected and the cost of edges are distinct. How about a graph G that is not connected ?

  The answer is clear: For each connected component of G,Prim or Kruskal will find a MST, together all ,we will have a **MST Forest**.

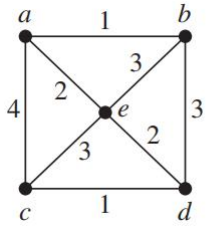
- Other Currently discovered MST Algorithm.

Fredman-Tarjan	V	$E+V\log V$
Chazelle	V	非常接近但还没有达到 E

viii

- For directed weighted connected graph, it's "Minimum Spanning Tree" is actually another problem called **Minimum Directed Spanning Tree** problem . And the same as MST , a famous greedy algorithm called **Edmonds(Chu-Liu) Algorithm** will easily solve the problem.

Topic seven: Demo Running

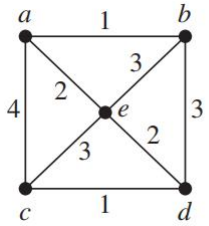


ix

Using Prim:

We start with vertex a, and we add edge(a,c),edge(a,b),edge(a,e) into PQ.

Next we choose the edge of min cost,which is edge(a,b).Now b is added into MST,and several edges, edge(b,e),edge(b,d) is also added into PQ. Next vertex to be added into MST is e ,which connected MST with edge(a,e) is currently the smallest cost edge.Then edge(e,d) ,edge(e,c) is added into PQ. Further d is added into MST with edge(e,d) of cost 2,and edge(d,c) is added into PQ.Finally c is added into MST with edge(d,c) of cost 1. Now we have add all the vertices into MST,and the min cost is 6.



Using Kruskal:

We use the same graph to run the algorithm.

Of all the edges we first sort them with cost ordered from low to high.

edge(a,b) = edge(c,d) < edge(a,e) = edge(e,d) < edge(b,e) = edge(b,d) = edge(c,e) < edge(a,c)

Then we add edge(a,b) ,edge(c,d) ,edge(a,e) ,edge(e,d) into MST,when we try to add edge(b,e) or edge(b,d) into MST, we find that it will form a circle inside MST, so we drop this edge(b,e) and edge(b,d) and continue. We further find it's the same case for edge(c,e) and edge(a,c).so we drop them again.Now in MST we have edge(a,b), edge(c,d),edge(a,e) and edge(e,d),the total cost is 6 the same as Prim.

i Definition from , Discrete Mathematics and Its Applications , Kenneth H. Rosen

ii Taken from Sustech OJ System (www.acm.sustech.edu.cn)

iii Taken from Discrete Mathematics and Its Applications , Kenneth H. Rosen

iv Taken from Discrete Mathematics and Its Applications , Kenneth H. Rosen

v Taken from Algorithm,4th Edition

vi Taken from Professor Yuhui Shi's PPT

vii Taken from Sustech OJ System (www.acm.sustech.edu.cn)

viii Taken from Algorithm,4th Edition

ix Taken from Discrete Mathematics and Its Applications , Kenneth H. Rosen