13.1 Minimum Spanning Tree (MST)

13.1.1 Problem Overview

Input: Undirected weighted graph $G(V, E)$, where V is the set of vertices in the graph G and E is the set of edges in G. It is assumed that G is a connected graph and that there are no edges of the same weights for simpler analysis. However, the algorithm works when there are edges of the same weights.

Output: A spanning tree ($T \subset E$) of minimum weight. In other words, output T is connected, acyclic and of minimum weight.

First, we are going to refresh our memory on "cut theorem", which underlies the algorithms for minimum spanning tree.

Theorem 13.1.1 *For any $U \subset V$, let e_u = minedge($\{(u, v) \in E | u \in U, v \in V - U\}$), then $e_u \in MST(V)$.*

Proof: Let $U \subset V$ and $e_u = (x, y)$, where $x \in U$ and $y \in V$ (refer to Figure 14.1.1). Since MST(V) is connected, there is a path from x to y, and there is a at least one edge connecting U and V. Suppose $e_u \notin$ MST(V), then there is an edge $h = (w, z)$ where $w \in V$ and $z \in U$, such that

1) $h \in$ MST(V);

2) the weight of h is greater than the weight of e_u;

3) there is a path P from x to w in MST(V) and there is a path Q from y to z in MST(V).

Then, e_u, P, h, Q form a cycle. By adding e_u to MST(V) and deleting h from MST(V), the resulting MST'(V) is of lower weight and still remains acyclic and connected, which is contradictory to the statement that MST(V) is a minimum spanning tree of V. Thus, it is true that $e_u \in MST(V)$.

■

There are several sequential algorithms to solve the minimal spanning tree problem, as listed below.

1) Prims: repeatedly add edge (u, v) of minimum weight such that $u \in V_{new}$ and $v \in V - V_{new}$, and add v to V_{new}, until $V = V_{new}$.

2) Kruskal: sort edges and then add edge of minimal weight first such that no cycles are created.

3) Boruvka: look at extra credit in assignment 4.

Karger-Klem-Tarjan: O(m) time with high probability. It uses the idea of contraction, which is commonly used in parallel algorithms.

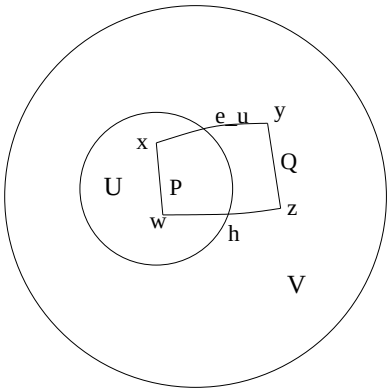

Figure 13.1.1: Choice of edge in MST.

13.1.2 Parallel Minimum Spanning Tree Algorithm

We are going to describe a parallel minimum spanning tree algorithm, random mate MST. In this algorithm, when there are concurrent writes, leftmost position wins. For example, E=[(4, 2), (5, 2), (5, 1), (3, 2), (2, 1)]. Code "forall $(u, v) \in E$ A[v]=u" results in A = [5, 4].

Algorithm 1 High level description of random mate MST.

while $|E| \neq 0$ **do**
1) Identify edge of minimum weight out of each vertex
2) Add these edges to MST
3) Use these to hook in random mate forming stars
4) Contract starts and remove self edges
end while

A high level pseudocode for random mate MST is shown in Algorithm 1. This algorithm is correct since step 1 and 2 select edges according to cut theorem and step 3 and 4 result in a tree at the end of the algorithm. More specifically, in step 3 and 4, each node represents a subtree. When there is a selected edge connecting two nodes, the two nodes are contracted to be one node, which is also a subtree. This is because two trees connected by an edge is a tree. So, by induction, algorithm results in one spanning tree since there is only one vertex (no non-self edges) at the end of the algorithm. Since the edges in the spanning tree is selected according to the cut theorem, the output of the algorithm is a minimum spanning tree. An example illustrating this algorithm is shown in Figure 14.1.2.

Before moving on to the more detailed version of the random mate MST algorithm, we will first define our edge array E to be of the form $[(u_1, v_1, i_1), ..., (u_m, v_m, i_m)]$, where u_j and v_j are two ends of the edge and i_j is the weight of the corresponding edge. For example, the edge array corresponding to the graph in Figure 14.1.3 is [(a, b, A), (a, c, B), (b, c,C), (b, a, A), (c, a, B), (c, b, C)]. Let E'=E in sorted order by edge weight using quicksort or mergesort.

The more detailed version of random mate MST is shown in Algorithm 2. This algorithm is slightly different

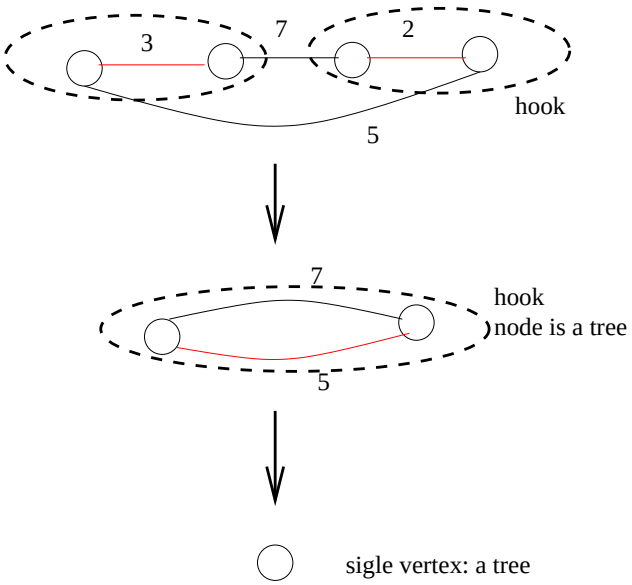

Figure 13.1.2: Example of computing MST.

from the one described in Algorithm 2 in that edges of minimum weights are not added unconditionally. This leads to adding an edge to MST exactly once. The resulting E in the packing step maintains the same order as before. Same as the graph connectivity algorithm, the work of the algorithm is O(m*log(n)), where m is the number of edges. The depth of the algorithm is O($log^2(n)$), since there are O(log(n)) rounds and in each round the depth is packing is O(log(n)).

Algorithm 2 Random Mate MST.

while E is not empty **do**
  forall $v \in V$, C[v]=heads or tails (each with probability 50%)
  forall $(u, v, i) \in E$, do R[u]=i
  forall $(u, v, i) \in E$, do
  if C[u]==head and C[v]==tail and R[u]==i **then**
    L[u]=v, MST[i]=true
  end if
  E=$\{(L(u), L(v)) : (u, v) \in E | L(u) \neq L(v)\}$
end while

13.2 Maximal-Independent Set (MIS)

Independent set (IS) is a set of vertices $U \subset V$, such that $\{\{(u, v) : u \in U, v \in U\} \cap E = \emptyset\}$. Maximal independent set is an IS such that no more vertices can be added. Vertices that are circled in Figure 14.2.4 is a maximal independent set. Maximal independent set is different from the maximum independent set where the latter one finds the global maximum and is an NP complete problem. MIS is useful in symmetry

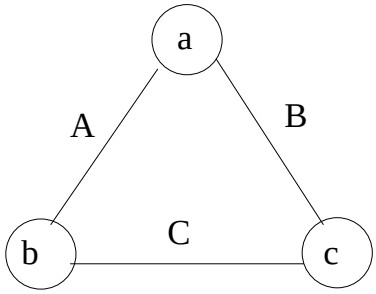

Figure 13.1.3: Example for edge array.

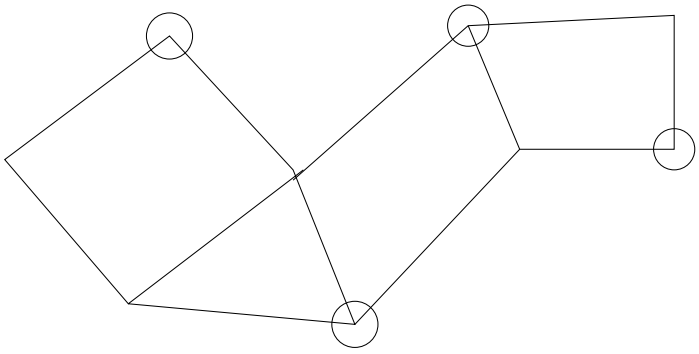

Figure 13.2.4: Maximal independent set.

Algorithm 3 Maximal independent set.

while E is not empty **do**
  Pick each vertex with probability $\frac{1}{2*d(v)}$, and add them to set S
  for each edge (u, v) **do**
    if $u \in S$ and $v \in S$ **then**
      Discard the one smaller degree from S (break ties arbitrarily)
    end if
  end for
  I=S, add I to MIS
  Remove I and N(I) from G.
end while

breaking in parallel algorithms. This is because performing operations on vertices sharing edges in parallel will result in conflicts. MIS algorithm is shown in Algorithm 3. $d(v)$ is the degree of vertex v, and $N(I)$ is the set of neighbors of I. This algorithm takes O(log(n)) rounds, O(n) work.