Lecture 6
Properties of MST

1. Properties of MST
2. Second-best MST
3. Bottleneck Spanning Tree

Properties of MST

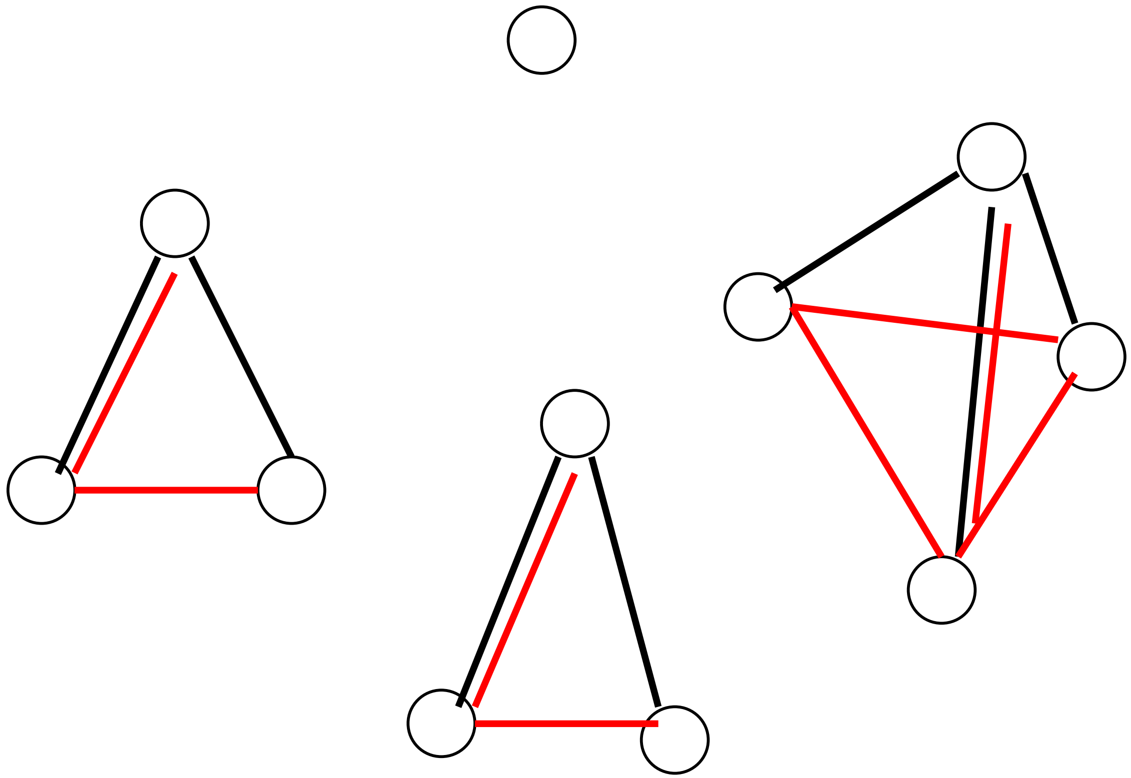
- Property 1:
  - Let (u, v) be a minimum-weight edge in a graph G = (V, E), then (u, v) belongs to some minimum spanning tree of G.
  - Proof: Let A be the empty set of edges. Then A is a subset of some minimum spanning tree, and A does not include (u, v). Then cut $(u, V\text{-}u)$ respects A. Edge (u, v) is a light edge crossing cut $(u, V\text{-}u)$. According to theorem 23.1, (u, v) is a safe edge for A. Hence, (u, v) belongs to some minimum spanning tree.
  - Another method: Suppose T is an arbitrary MST, and (u, v) is not in T. Then T+(u, v) contains a cycle. Let f be any edge other than (u, v) on the cycle, then T +(u, v) - f is another MST and it contains (u, v).

Properties of MST

- Property 2
  - Let *T* be a minimum spanning tree of a graph *G*, and let *L* be the sorted list of the edge weights of *T*, then for any other minimum spanning tree *T'* of *G*, the list *L* is also the sorted list of edge weights of *T'*.
    - A corollary of Property 3.
    - If all the edge weights are distinct, then MST is unique.
- Property 3
  - Let *T* be a minimum spanning tree of a graph *G*, and let *T'* be an arbitrary spanning tree of *G*, suppose the edges of each tree are sorted in non-decreasing weight order, that is, $w(e_1) \le w(e_2) \le \ldots \le w(e_{n-1})$ and $w(e_1') \le w(e_2') \le \ldots \le w(e_{n-1}')$, then for $1 \le i \le n-1$, $w(e_i) \le w(e_i')$.

Proof of Property 3

- (Prove by contradiction) Suppose without loss of generality that there exists an index $i \geq 1$ such that $w(e_1) \leq w(e_1')$, $w(e_2) \leq w(e_2')$, …, $w(e_{i-1}) \leq w(e_{i-1}')$, but $w(e_i) > w(e_i')$. This indicates that $w(e_{n-1}) \geq w(e_{n-2}) \geq \ldots \geq w(e_i) > w(e_i') \geq w(e_{i-1}') \geq \ldots \geq w(e_1')$. So, if we add any $e_x' \in \{ e_1', e_2', \ldots, e_i'\}$ onto T, then either e_x' is an edge of $\{e_1, e_2, \ldots, e_{i-1}\}$, or a cycle is formed which includes **only edges in $\{e_x', e_1, e_2, \ldots, e_{i-1}\}$.** Thus each $e_x' \in \{ e_1', e_2', \ldots, e_i'\}$ is included in a connected component of $\{e_1, e_2, .., e_{i-1}\}$. Thus if we add all the edges $\{ e_1', e_2', \ldots, e_i'\}$ onto T and delete $\{e_i, \ldots, e_{n-1}\}$, $\{ e_1', e_2', \ldots, e_i'\}$ are still included in the connected component of $\{e_1, e_2, .., e_{i-1}\}$. Thus $\{e_1', e_2', \ldots, e_i', e_1, e_2, .., e_{i-1}\}$ have the same number of connected components as $\{e_1, e_2, .., e_{i-1}\}$ have, i.e., n-i+1. A contradiction, since $\{e_1', e_2', \ldots, e_i'\}$ are tree edges, they can have at most n-i connected components.



$\{e_1', \ldots, e_i'\}$ are all included in the components of G_{i-1},
So, red edges cannot have less components.

Second-best MST

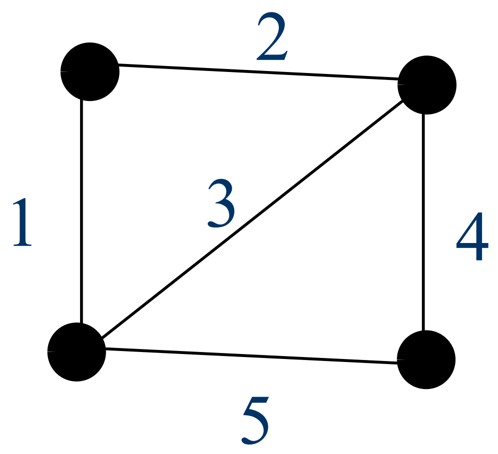
- Let $G = (V, E)$ be an undirected, connected graph with weight function w: $E \rightarrow \mathbf{R}$, and suppose that $|E| \geq |V|$ and all edge weights are distinct, a second-best minimum spanning tree is a spanning tree T such that $w(T) = \min_{T'' \in \Gamma - \{T'\}} \{w(T'')\}$, where Γ is the set of all spanning trees of G, and T' is a minimum spanning tree of G.

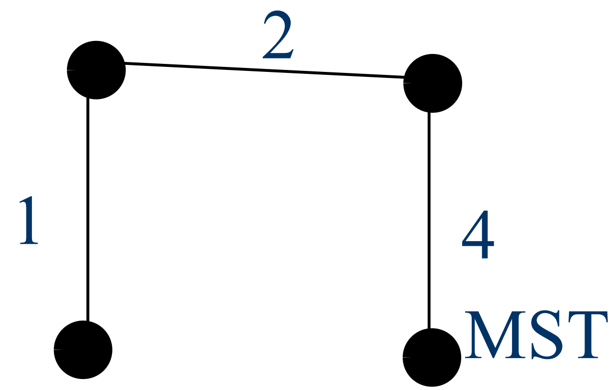
MST Property

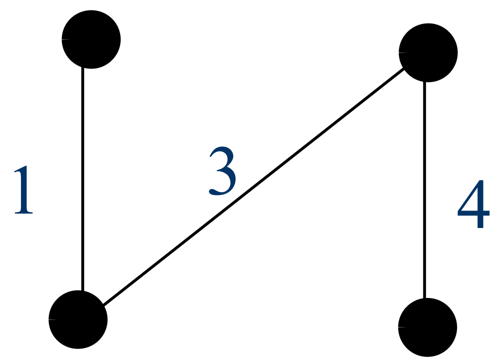
- Let $G = (V, E)$ be an undirected, connected graph with weight function w: $E \rightarrow \mathbf{R}$, and suppose that $| E | \geq | V |$ and all edge weights are distinct. Then the minimum spanning tree of G is unique.
  - According to Property 2.

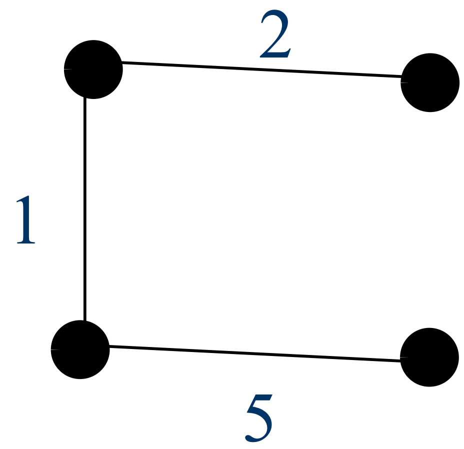
Second-best MST Properties

- The second-best minimum spanning tree can not be unique.









Second-best MST Properties

- A second-best minimum spanning tree can be obtained from the minimum spanning tree by replacing a single edge from the tree with another edge not in the tree.
  - By contradiction. Suppose T_1 is the MST and T_2 is a second-best MST, and they differ by at least two edges, we will get another spanning tree T_3 which is not equal to T_1 but better than T_2, thus contradicts that T_2 is second-best.

- Suppose e_1 is the minimum edge in $T_1 \setminus T_2$.
- Then $T_2 \cup \{e_1\}$ contains a cycle on which there must be some edge e which belongs to T_2 but not belongs to T_1, since otherwise T_1 has a cycle.

- Let $T_3 = T_2 \cup \{e_1\} \setminus e$. Then T_3 is a spanning tree.

We claim that

1. **T_3 is better than T_2.**
2. **T_3 is not equal to T_1.**

Thus, contradicts to that T_2 is a second-best MST.

Proof of 1: We claim that $w(e) > w(e_1)$. Proof by contradiction: assume that $w(e) < w(e_1)$. If we add e to T_1, we get a cycle C, which contains some edge e' in T_1 but e' is not in T_2. (otherwise, T_2 would contain a cycle). Therefore, the set of edges $T_4 = T_1 - \{e'\} \cup \{e\}$ form a spanning tree, and we must have $w(e') < w(e)$, since otherwise T_4 would be a spanning tree with weight less than $w(T_1)$. Thus, $w(e') < w(e) < w(e_1)$, which contradicts our choice of e_1 as the edge in $T_1 - T_2$ of minimum weight. Since $T_3 = T_2 \cup \{e_1\} \setminus e$, T_3 is better than T_2.

Proof of 2: T_3 differs from T_2 by only one edge, while T_1 differs from T_2 by at least two edges, so, T_3 is not equal to T_1.

Thus, we have formed a spanning tree T_3 whose weight is less than $w(T_2)$ but is not T_1. Hence, contradicts that T_2 is a second-best minimum spanning tree.

Compute the Second-best MST

- Idea: use the property in previous page:
  - (u, v): an edge of T, (x, y): an edge in G but not in T
  - $w(T') = w(T) + (w(x, y) - w(u, v))$
- Let T be a spanning tree of G and for any two vertices $u, v \in V$, let $max[u, v]$ be the maximum weight among all edges on the unique path between u and v in T.
- How to compute $max[u, v]$?

```
Compute_max(T)
    for each pair of vertices u, v ∈ V
        do max[u, v] ← 0
    for each vertex s ∈ V
        do BFS (T, s)        //E(T)=O(V)
```

```
BFS(T, s)
    for each vertex u ∈ V
        do color[u] ← WHITE
    color[s] ←GRAY
    Q ←Φ
    ENQUEUE(Q, s)
    while Q ≠ Φ
        do u ← DEQUEUE(Q)
            for each v ∈ Adj[u]
                do if color[v] = WHITE
                    then color[v] ←GRAY
                        max[s, v] ← max{w(u, v), max[s, u]}
                        ENQUEUE(Q, v)
        color[u] ← BLACK
```

Compute the Second-best MST

Second-Best-MST(G)

1. Compute a MST T for G
2. Compute $max[u, v]$ for each pair of vertices in T
3. For each edge (u, v) in G but not T, compute the difference $w(u, v) - max[u, v]$
4. Find the smallest positive difference and replace the $max\_edge[u, v]$ with corresponding edge (u, v).

Note: $max\_edge[u, v]$ means the edge with maximum weight in the path connecting u and v in T. It can be easily computed.

Time Complexity Analysis

- Compute-max(T): $O(V^2)$
- Second-Best-MST(G): $O(E\log V + V^2)$

Bottleneck Spanning Tree

- A bottleneck spanning tree T of a connected, weighted and undirected graph G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G.
  - Let T_1, T_2, …, T_m are all the spanning trees G, and the largest edge of each tree is e_{t1}, e_{t2}, …, $e_{tm.}$ If $w(e_{ti}) \leq$ w(e_{tj}) for $1 \leq j \leq m$ and $j \neq i$, then T_i is a bottleneck spanning tree.
  - The value of a BST T is the weight of the maximum-weight edge in T.
  - The bottleneck spanning tree may not be unique.

BST vs MST

- Every minimum spanning tree is a bottleneck spanning tree.
  - Property 3 implies it.
  - Another proof: Let T be a MST and T' be a BST, let the maximum-weight edge in T and T' be e and e', respectively. Suppose for the contrary that the MST T is not a BST, then we have $w(e)>w(e')$, which also indicates that the weight of e is greater than that of any edges in T'. Removing e from T disconnects T into two subtrees T_1 and T_2, there must exist an edge f in T' connecting T_1 and T_2, otherwise, T' is not connected. $T_1 \cup T_2 \cup \{f\}$ forms a new tree T'' with $w(T'') = w(T) - w(e) + w(f) < w(T)$, A contradiction to the fact that T is MST, thus, a MST is also a BST.

The BST Problem

- The Problem:
  - Input: A weighted, connected and undirected graph G.
  - Output: A BST T of G.
- A solution:
  - Kruskal's and Prim's Algorithm works for the BST problem.
- More efficient algorithm exist for the problem?

A Verification Problem

- A verification problem:
  - Input: A graph $G = (V, E; W)$ and an integer b
  - Output: **TRUE** if the value of the bottleneck spanning tree of G is at most b and **FALSE** otherwise

How to solve this problem in linear time?

```
CHECKBOTTLENECK(G, b)
    for each vertex u ∈ V[G]
        do color[u] ← WHITE
    u ← randomly chosen vertex in G
    DFS(u, b)                                  //O(V+E)
    for each vertex u ∈ V[G]
        do if color[u] = WHITE
            then return FALSE
    return TRUE
```

```
DFS(u, b)
    color[u] ← GRAY
    for each v ∈ Adj[u]
        do if color[v] = WHITE and w(u, v) ≤ b
            then DFS(v, b)
    color[u] ← BLACK
```

Solve the BST Problem

BOTTLENECK(G)

sort the weights of edges of G in non-decreasing order: $e[1], e[2], \ldots, e[|E|]$

$start \leftarrow 1$

$end \leftarrow |E|$

while $start < end$

  do $middle \leftarrow \lfloor (start + end)/2 \rfloor$

    if CHECKBOTTLENECK(G, $e[middle]$) = FALSE

      then $start \leftarrow middle$

    else $end \leftarrow middle$ *//in the end there is a BST with value =e[end],*

$u \leftarrow$ randomly chosen a vertex of G

call DFS(u, $e[start]$) to build a spanning tree

Solve the Problem in Linear Time

- Time complexity:
- Sorting:O(ElogV)
- Finding the value of BST: O((V+E)logV)=O(ElogV)
- Total:O(ElogV)

- Can the problem be solved in linear time?
- Please think it carefully, anyone who finds the solution will be given an extra bonus.
- Will it be unsolved all our lives?

23.1-5

- 证明:设$C=v_0, v_1,\ldots,v_k$是一个圈,其中边$e=(v_0, v_k)$是权值最重的边。
- 只需要构造一棵不包含$e=(v_0, v_k)$的MST即可。
- 设T是一棵包含$e=(v_0, v_k)$的MST，则删除e会使T变成两个连通分支V_1，V_2，$v_0\in V_1$，$v_k\in V_2$。依次检测顶点$v_1,\ldots,v_k$，找到第一个在V_2中的顶点v_i(这样的v_i一定能找到，因为$v_k\in V_2$)，从而$e'=(v_{i-1}, v_i)$是穿过割(V_1, V_2)的一条边，并且$w(v_0, v_k)\geq w(v_{i-1}, v_i)$。则$T'=T-e+e'$是一棵新的MST。

23-4(a)

- 证明: 设T中的边按照权值非递减顺序依次为$e_1, e_2, \ldots, e_{n-1}$,
- 即算法依次保留边$e_1, e_2, \ldots, e_{n-1}$。设边集$A_i = \{e_1, e_2, \ldots, e_i\}$, $1 \le i \le n-1$则只需要证明每个A_i都是某棵最小生成树的子集。
- 用归纳法证明。
- $i=1$时，设T’是一棵最小生成树，如果$(u, v)=e_1 \in T'$, 结论自然成立。如果$e_1 \notin T'$，则在T’中存在u到v的路径p。因为删除e_1会使图不连通，即删除e_1会使顶点集合V划分为两个子集V_1和V_2，其中$u \in V_1$，$v \in V_2$。则路径p中存在1条边(x, y)满足$x \in V_1$，$y \in V_2$，并且(x, y)已经被删除了，否则如果p中所有边都没被删除，删除e_1不会使图不连通。既然(x, y)已经被删除了，根据算法是按照权值由大到小的顺序删边的，所以$w(x, y) \ge w(u, v)$。则$T'' = T' - (x, y) + (u, v)$必然是一棵最小生成树。

Continued

- 设对边集A_i时结论成立，现在证明边集A_{i+1}也是某棵最小生成树的子集。
- 设$A_i=\{e_1, e_2,\ldots, e_i\}$是最小生成树T’的子集，如果$(u, v)=e_{i+1}\in T'$，结论自然成立。如果$e_{i+1}\notin T'$，则在T’中存在u到v的路径p。因为删除$e_{i+1}$会使图不连通，即删除$e_1$会使顶点集合V划分为两个子集$V_1$和$V_2$，其中$u\in V_1$，$v\in V_2$。则路径p中存在1条边$(x, y)$满足$x\in V_1$，$y\in V_2$，并且$(x, y)$已经被删除了，否则如果p中所有边都没被删除，删除$e_1$不会使图不连通。既然$(x, y)$已经被删除了，根据算法是按照权值由大到小的顺序删边的，所以$w(x, y)\geq w(u, v)$。则$T''=T' - (x, y)+(u, v)$必然是一棵最小生成树。现在只需要证明T’’包含$A_i=\{e_1, e_2,\ldots, e_{i+1}\}$中的所有边，因为T’’与T’只有1条边不同，所以只需要证明$(x,y)\notin A_i$，这显然是成立的，因为(x, y)已经被删除了，而$\{e_1, e_2,\ldots, e_i\}$是没被删除的。
- 证明完毕！

23-4(c)

- 证明：算法实际上是在图G中删除一些圈上权值最重的边，最后得到一棵MST。
- 设删除的边依次为$e_1, e_2, \ldots, e_{m-n+1}$，剩余的图依次是$G_0, G_1, \ldots, G_{m-n+1}$，其中$G=G_0$，$G_{m-n+1}=T$，$m=|E|$，$n=|V|$。
- 我们证明G_{i+1}的MST同时也是G_i的MST即可。
- 前面23.1-5已经证明了存在G_{i+1}的MST T’同时也是G_i的MST，而G_{i+1}的所有MST的大小与T’一样的，所有它们都与G_i的MST的大小一样，所以他们都是G_i的MST。
- 从而G_{m-n+1}必然是$G_{m-n}, \ldots, G_o$的MST。

Experiment-1

- 给定一个有向图G，从G中删除一些边，将G变为有向无环图。
- 要求：
  - 图用邻接链表存储
  - 能对具体小事例运行出正确结果
  - 能跟助教讲清楚你的算法思想
  - 不要求删除的边数最少