Minimum Spanning Trees

(CLRS 23)

- Problem: Given connected, undirected graph $G = (V, E)$ where each edge (u, v) has weight $w(u, v)$. Find acyclic set $T \subseteq E$ connecting all vertices in V with minimal weight $w(T) = \sum_{(u,v) \in T} w(u, v)$.
- An acyclic set connecting all vertices is called a *spanning tree*. We want to find a spanning tree of *minimal weight*. We use *minimum spanning tree* as short for *minimum weight spanning tree*).
- MST problem has many applications
  - For example, think about connecting cities with minimal amount of wire or roads (cities are vertices, weight of edges are distances between city pairs).
- Example:

mst_example-eps-converted-to.pdf

  - Weight of MST is $4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = 37$
  - MST is not unique: e.g. (b, c) can be exchanged with (a, h)

1 PRIM's algorithm

- *Greedy* algorithm for computing MST:
  - Start with spanning tree containing arbitrary vertex r and no edges
  - Grow spanning tree by repeatedly adding minimal weight edge connecting vertex in current spanning tree with a vertex not in the tree
- Implementation:
  - To find minimal edge connected to current tree we maintain a priority queue on vertices not in the tree. The key/priority of a vertex is the weight of minimal weight edge connecting it to the tree. (We maintain pointer from adjacency list entry of v to v in the priority queue).

- For each node u maintain $visit(u)$ ($(u, visit(u))$ is the cuurently best edge connecting it to the tree.)

PRIM(r)

For each $v \in V$ DO

    INSERT(PQ, v, ∞)

DECREASE-KEY($PQ, r, 0$)
WHILE PQ not empty DO

    u = DELETEMIN(PQ)

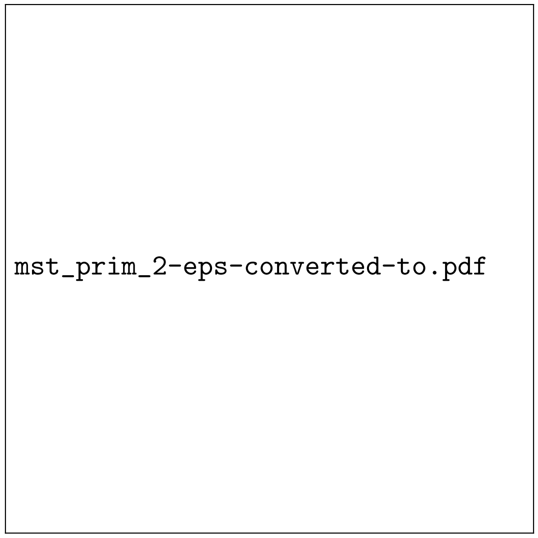
    (output edge $(u, visit(u))$ as part of MST)

    For each $(u, v) \in E$ DO

        IF $v \in PQ$ and $w(u, v) <$ key(v) THEN
            visit[v] = u
            DECREASE-KEY($PQ, v, w(u, v)$)

- On the example graph, the greedy algorithm would work as follows (starting at vertex a):

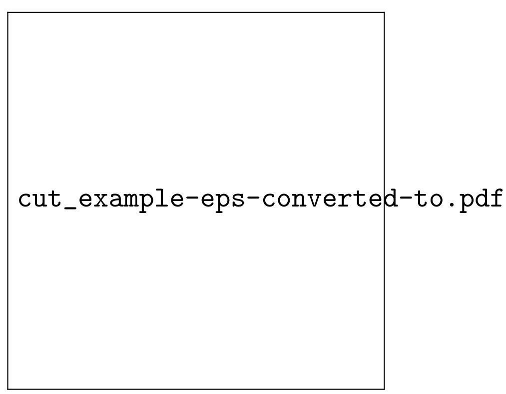
mst_prim_1-eps-converted-to.pdf



- Analysis:
  - While loop runs $|V|$ times $\Rightarrow$ we perform $|V|$ DELETEMIN's
  - We perform at most one DECREASE-KEY for each of the $|E|$ edges
    $\Downarrow$
    $O((|V| + |E|) \log |V|) = O(|E| \log |V|)$ running time.
- Correctness:
  - When designing a greedy algorithm the hard part is to prove that it works correctly.
  - We will prove a Theorem that allows us to prove the correctness of a general class of greedy MST algorithms:

    Some definitions
    * A *cut* $(S, V \setminus S)$ is a partition of V into sets S and $V \setminus S$
    * A *edge* (u, v) *crosses a cut* S if $u \in S$ and $v \in V \setminus S$ *or* $v \in S$ and $u \in V \setminus S$
    * A *cut* S *respects a set* $T \subseteq E$ if no edge in T crosses the cut

    Example: Cut S respects T



- *Theorem*: If $G = (V, E)$ is a graph such that $T \subseteq E$ is subset of some MST of G, and S is a cut respecting T **then** there is a MST for G containing T and the minimum weight edge $e = (u, v)$ crossing S.

- Note: Correctness of Prim's algorithm follows from the Theorem by induction—cut consist of current spanning tree.

- Proof:

  - Let T^* be MST containing T
  - If $e \in T^*$ we are done
  - If $e \notin T^*$:
    - There must be (at least) one other edge $(x, y) \in T^*$ crossing the cut S such that there is a unique path from u to v in T^* (T^* is spanning tree)

      cut_proof-eps-converted-to.pdf

    - This path together with e forms a cycle
    - If we remove edge (x, y) from T^* and add e instead, we still have spanning tree
    - New spanning tree must have same weight as T^* since $w(u, v) \leq w(x, y)$

      $\Downarrow$

      There is a MST containing T and e.

- The Theorem allows us to describe a very abstract greedy algorithm for MST:

  $T = \emptyset$
  While $|T| \leq |V| - 1$ DO
      Find cut S respecting T
      Find minimal edge e crossing S
      $T = T \cup \{e\}$

  - Prim's algorithm follows this abstract algorithm.
  - Kruskal's algorithm is another implementation of the abstract algorithm.

2 Kruskal's Algorithm

- Kruskal's algorithm is another implementation of the abstract algorithm.
- Idea in Kruskal's algorithm:
  - Start with $|V|$ trees (one for each vertex)
  - Consider edges E in increasing order; add edge if it connects two trees
- Example:

mst_kruskal_1-eps-converted-to.pdf

mst_kruskal_2-eps-converted-to.pdf

- Implementation:

  We need (Union-Find) data structure that supports:

– Make-set(v): Create set consisting of v

– Union-set(u, v): Unite set containing u and set containing v

– Find-set(u): Return unique representative for set containing u

```
KRUSKAL

T = ∅
FOR each vertex v ∈ V MAKE-SET(v)
Sort edges of E in increasing order by weight
FOR each edge e = (u, v) ∈ E in order DO
    IF FIND-SET(u) ≠ FIND-SET(v) THEN
        T = T ∪ {e}
        UNION-SET(u, v)
```

- Analysis:
  - We use $O(|E| \log |E|)$ time to sort edges and we perform $|V|$ Make-Set, $|V|-1$ Union-set, and $2|E|$ Find-Set operations.
  - We will discuss a simple solution to the *Union-Find problem* such that Make-Set and Find-Set take $O(1)$ time and Union-Set takes $O(\log V)$ time amortized.

    $\Downarrow$

    Kruskal's algorithm runs in time $O(|E| \log |E| + |V| \log |V|) = O((|E| + |V|) \log |E|) = O(|E| \log |V|)$ like Prim's algorithm.
- Correctness
  - follows from Theorem above: If minimal edge connects two trees then there exists a cut respecting the current set of edges (cut consisting of vertices in one of the trees)

3 Union-Find

- The *Union-Find problem*: Maintain a set system under:
  - Make-set(v): Create set consisting of v
  - Union-set(u, v): Unite set containing u and set containing v
  - Find-set(u): Return unique representative for set containing u
- Simple solution:
  - Maintain elements in same set as a linked list with each element having a pointer to the first element in the list (unique representative)

    Example:

union-find_linked-eps-converted-to.pdf

  - MAKE-SET(v): Make a list with one element $\Rightarrow O(1)$ time
  - FIND-SET(u): Follow pointer and return unique representative $\Rightarrow O(1)$ time
  - UNION-SET(u, v): Link first element in list with unique representative FIND-SET(u) after last element in list with unique representative FIND-SET(v) $\Rightarrow O(|V|)$ time (as we have to update all unique representative pointers in list containing u)

- With this simple solution the $|V|-1$ UNION-SET operations in Kruskal's algorithm may take $O(|V|^2)$ time.

- We can improve the performance of UNION-SET with a very simple modification: Always link the smaller list after the longer list ($\Rightarrow$ update the pointers of the smaller list)

  - One UNION-SET operation can still take $O(|V|)$ time, but the $|V|-1$ UNION-SET operations takes $O(|V|\log|V|)$ time altogether (one UNION-SET takes $O(\log|V|)$ time *amortized*):
    * Total time is proportional to number of unique representative pointer changes
    * Consider element u:
      After pointer for u is updated, u belongs to a list of size at least double the size of the list it was in before
      $\Downarrow$
      After k pointer changes, u is in list of size at least 2^k
      $\Downarrow$
      Pointer can be changed at most $\log|V|$ times.

- With improvement, Kruskal's algorithm runs in time $O(|E|\log|E| + |V|\log|V|) = O((|E| + |V|)\log|E|) = O(|E|\log|V|)$ like Prim's algorithm.