Minimum Cost Spanning Trees (MSTs)

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

Outline

Cost & Minimum-Cost Spanning Tree

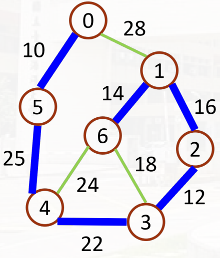
- The cost of a spanning tree of a weighted undirected graph is the sum of the weights of the edges in the spanning tree.

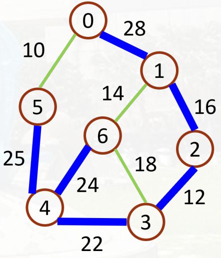
Cost & Minimum-Cost Spanning Tree

- The cost of a spanning tree of a weighted undirected graph is the sum of the weights of the edges in the spanning tree.
- A minimum-cost spanning tree is a spanning tree of least cost.

Cost & Minimum-Cost Spanning Tree

- The cost of a spanning tree of a weighted undirected graph is the sum of the weights of the edges in the spanning tree.
- A minimum-cost spanning tree is a spanning tree of least cost.





cost $= 10+25+22+12+16+14 = 99.$

cost $= 28+16+12+22+24+25 = 127.$

Greedy Methods

- We will introduce three different greedy algorithms can be used to obtain a minimum cost spanning tree of a connected undirected graph.

Greedy Methods

- We will introduce three different greedy algorithms can be used to obtain a minimum cost spanning tree of a connected undirected graph.
  - Kruskal's algorithm
  - Prim's algorithm
  - Sollin's algorithm

⋆ Further reading: Greedy Algorithms & Matroids [link]

Greedy Method (1/2)

- Construct an optimal solution in stages.
- A feasible solution is one which works within the constrains specified by the problem.
- At each stage, we make a decision that is the best decision at that time.
- Typically, The selection of an item at each stage is based on either a least cost or a highest profit criterion.

Greedy Method (2/2)

- For minimum spanning trees, we use a least cost criterion. Our solution must satisfy the following constrains:

Greedy Method (2/2)

- For minimum spanning trees, we use a least cost criterion. Our solution must satisfy the following constrains:
  - We must use only edges within the graph.

Greedy Method (2/2)

- For minimum spanning trees, we use a least cost criterion. Our solution must satisfy the following constrains:
  - We must use only edges within the graph.
  - Exactly $n - 1$ edges are used.
  - Never use edges that would produce a cycle.

TBD

```
void transpose(term a[], term b[]) { // b is set to the transpose of a
    int n, i, j, currentb;
    n = a[0].value; // total number of elements
    b[0].row = a[0].col; // rows in b = columns in a
    b[0].col = a[0].row; // columns in b = rows in a
    b[0].value = n;
    if (n > 0) { // dealing with a nonzero matrix
        currentb = 1;
        for (i=0; i<a[0].col; i++) // transpose by the columns in a
            for (j=1; j<=n; j++) // find elements from the current column
                if (a[j].col == i) { // element is in current column, add it to b
                    b[currentb].row = a[j].col;
                    b[currentb].col = a[j].row;
                    b[currentb].value = a[j].value;
                    currentb++;
                }
    }
}
```

Outline

Kruskal's Algorithm

- Build a minimum cost spanning tree T by adding edges to T one at a time.

Kruskal's Algorithm

- Build a minimum cost spanning tree T by adding edges to T one at a time.
- Select the edges for inclusion in T in nondecreasing order of their cost.

Kruskal's Algorithm

- Build a minimum cost spanning tree T by adding edges to T one at a time.
- Select the edges for inclusion in T in nondecreasing order of their cost.
- An edge is added to T if it does not form a cycle with the edges that are already in T.

Kruskal's Algorithm

- Build a minimum cost spanning tree T by adding edges to T one at a time.
- Select the edges for inclusion in T in nondecreasing order of their cost.
- An edge is added to T if it does not form a cycle with the edges that are already in T.
- Exactly $n-1$ edges will be selected for inclusion in T.

- Time complexity: $O(e \log e)$.

Kruskal's Algorithm

- Build a minimum cost spanning tree T by adding edges to T one at a time.
- Select the edges for inclusion in T in nondecreasing order of their cost.
- An edge is added to T if it does not form a cycle with the edges that are already in T.
- Exactly $n - 1$ edges will be selected for inclusion in T.
- Time complexity: $O(e \log e)$.

Kruskal's Algorithm

- Build a minimum cost spanning tree T by adding edges to T one at a time.
- Select the edges for inclusion in T in nondecreasing order of their cost.
- An edge is added to T if it does not form a cycle with the edges that are already in T.
    - **How to do this?**
- Exactly $n - 1$ edges will be selected for inclusion in T.

- Time complexity: $O(e \log e)$.

Kruskal's Algorithm

- Build a minimum cost spanning tree T by adding edges to T one at a time.
- Select the edges for inclusion in T in nondecreasing order of their cost.
- An edge is added to T if it does not form a cycle with the edges that are already in T.
  - **How to do this? $\Rightarrow$ Union-Find Operations!**
- Exactly $n - 1$ edges will be selected for inclusion in T.

- Time complexity: $O(e \log e)$.

Illustration of Kruskal's Algorithm

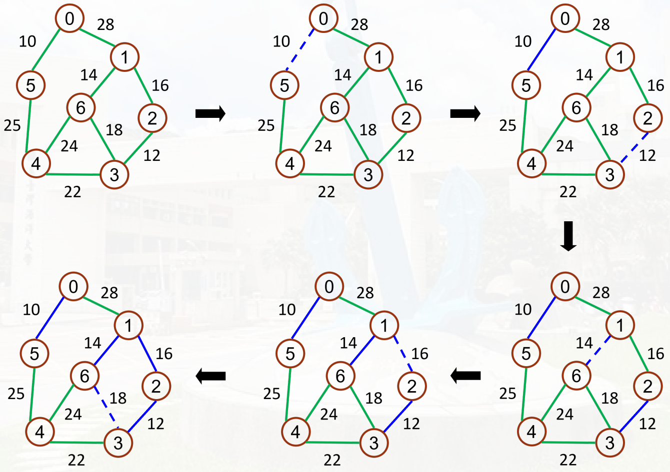
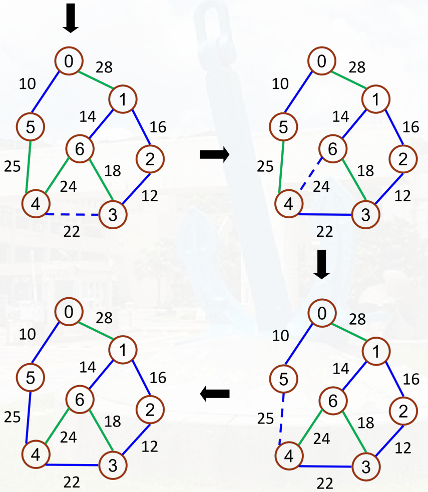


Illustration of Kruskal's Algorithm

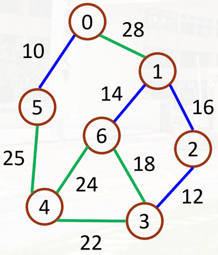


Union-Find Operations

- Union-Find Operations: to determine whether or not adding an edge would cause a cycle.

Union-Find Operations

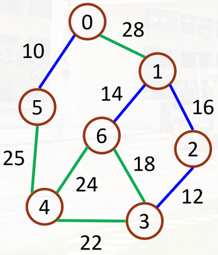
- Union-Find Operations: to determine whether or not adding an edge would cause a cycle.
- For example,



- $\{0, 5\}, \{1, 2, 3, 6\}, \{4\}$: the sets corresponding to existing subtrees.

Union-Find Operations

- Union-Find Operations: to determine whether or not adding an edge would cause a cycle.
- For example,



- $\{0, 5\}, \{1, 2, 3, 6\}, \{4\}$: the sets corresponding to existing subtrees.
- Vertex 3 and 6 are already in the same set $\Rightarrow$ edge $(3, 6)$ is rejected!

The Pseudo-code of Kruskal's Algorithm

```
T = { };
while (T contains fewer than n-1 edges && E is not empty) {
    choose a least cost edge (v,w) from E;
    delete (v,w) from E;
    if ((v,w) does not create a cycle in T)
        add (v,w) to T
    else
        discard (v,w);
}
if (T contains fewer than n-1 edges)
    printf("No spanning tree\n");
```

- How could "No spanning tree" happen?
- e union-find takes $O(e \log^* e)$ time (supplementary)

Outline

Prim's Algorithm (1/3)

- Another greedy MST algorithm.

Prim's Algorithm (1/3)

- Another greedy MST algorithm.
- The main difference:

Prim's Algorithm (1/3)

- Another greedy MST algorithm.
- The main difference:
  - The set of selected edges forms a tree at all times in Prim's algorithm.
  - The set of selected edges in Kruskal's algorithm forms a *forest at each stage*.

Prim's Algorithm (2/3)

- Prim's algorithm begins with a tree T that contains one arbitrary single vertex.

Prim's Algorithm (2/3)

- Prim's algorithm begins with a tree T that contains one arbitrary single vertex.
- Next, we add a **least cost edge** (u, v) to T such that $T \cup (u, v)$ is also a tree.

Prim's Algorithm (2/3)

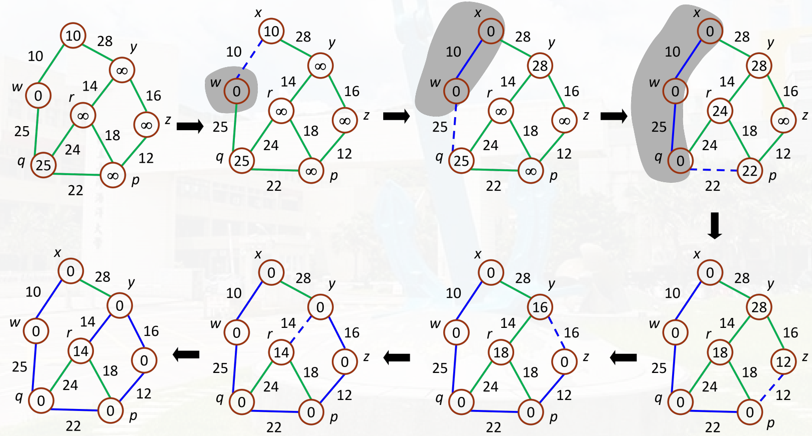
- Prim's algorithm begins with a tree T that contains one arbitrary single vertex.
- Next, we add a **least cost edge** (u, v) to T such that $T \cup (u, v)$ is also a tree.
- Repeat this edge addition until T contains $n-1$ edges.

Prim's Algorithm (2/3)

- Prim's algorithm begins with a tree T that contains one arbitrary single vertex.
- Next, we add a **least cost edge** (u, v) to T such that $T \cup (u, v)$ is also a tree.
- Repeat this edge addition until T contains $n - 1$ edges.

Time complexity: $O(n^2)$.

Illustration of Prim's Algorithm



Prim's Algorithm (3/3)

```
T = {}; TV = {0};
while (T contains fewer than n-1 edges) {
    let (u,v) be a least cost edge such that u is in TV and
    v is not in TV;
        if (there is no such edge )
            break;
        add v to TV;
        add (u,v) to T;
}
if (T contains fewer than n-1 edges)
    printf("No spanning tree\n");
```

- Each vertex $v \notin TV$ has a companion vertex "near(v)" such that near(v) $\in TV$ and cost(near(v), v) is minimum over all such choices for near(v).
- Therefore, it takes $O(n)$ time to choose an edge.
- We can implement Prim's algorithm in $O(n^2)$ time.

Outline

Sollin’s Algorithm (1/2)

- Sollin’s algorithm selects several edges for inclusion in T at each stage.
- At the start of a stage, the selected edges, together with all the n vertices, form a spanning forest.
- During each stage, we select one edge for each tree in the forest.
    - This edge is a minimum cost edge that has exactly one vertex in the tree.

Sollin's Algorithm (2/2)

- **Note:** two trees in the forest could select the same edge.

Sollin's Algorithm (2/2)

- **Note:** two trees in the forest could select the same edge.
    - We need to eliminate duplicate edges.

Sollin's Algorithm (2/2)

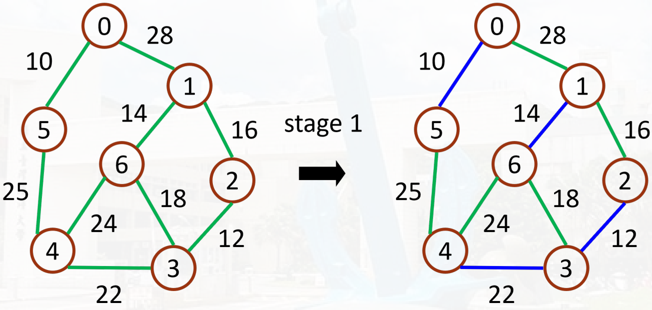
- **Note:** two trees in the forest could select the same edge.
  - We need to eliminate duplicate edges.
- At the start of the first stage the set of selected edges is empty.

Sollin's Algorithm (2/2)

- **Note:** two trees in the forest could select the same edge.
  - We need to eliminate duplicate edges.
- At the start of the first stage the set of selected edges is empty.
- The algorithm terminates when there is only one tree at the end of a stage or no edges remain for selection.

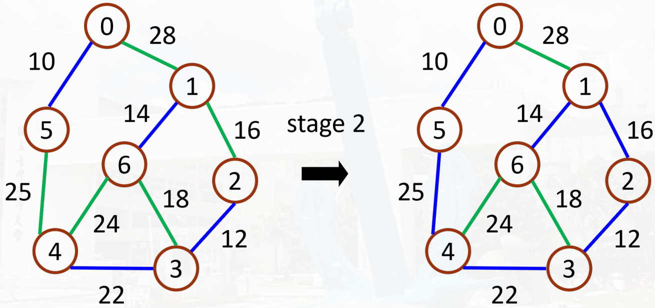
We can implement Sollin's algorithm in $O(e \log v)$ time.

Illustration of Sollin's Algorithm



- Stage 1: $(0,5), (1,6), (2,3), (3,2), (4,3), (5,0)$, and $(6,1)$ are selected, and then duplicates are removed.

Illustration of Sollin's Algorithm



- Stage 2: $(5, 4), (1, 2)$, and $(2, 1)$ are selected, and then duplicates are removed.

Discussions