4. Greedy Algorithms II

- red-rule blue-rule demo
- Prim’s algorithm demo
- Kruskal’s algorithm demo
- reverse-delete algorithm demo
- Boruvka’s algorithm demo

Section 4.5
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.

Repeat $n - 1$ times:

- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
  - Add to $T$ a min-weight edge with one endpoint in $S$.
  - Add new node to $S$. 
Prim’s algorithm demo

Initialize $S$ = any node, $T = \emptyset$.

Repeat $n - 1$ times:
  - Add to $T$ a min-weight edge with one endpoint in $S$.
  - Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}, \ T = \emptyset$.
Repeat $n - 1$ times:
• Add to $T$ a min-weight edge with one endpoint in $S$.
• Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
  • Add to $T$ a min-weight edge with one endpoint in $S$.
  • Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.

Repeat $n - 1$ times:

- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 

![Diagram showing Prim's algorithm progression]

Node connections with weights 7, 8, 11, 10, 2, 13.
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:

- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S =$ any node, $T = \emptyset$.
Repeat $n-1$ times:
- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 

![Graph diagram showing Prim's algorithm in action]

Note: The diagram illustrates the process of Prim's algorithm on a given graph, starting from an arbitrary node and iteratively adding the minimum weight edge to the growing tree $T$. The blue lines represent the edges added to $T$ as the algorithm progresses.
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
  • Add to $T$ a min-weight edge with one endpoint in $S$.
  • Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
• Add to $T$ a min-weight edge with one endpoint in $S$.
• Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.

Repeat $n - 1$ times:
- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S = \text{any node}$, $T = \emptyset$.
Repeat $n - 1$ times:
- Add to $T$ a min-weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim’s algorithm demo

Initialize $S$ = any node, $T = \emptyset$.
Repeat $n - 1$ times:
  • Add to $T$ a min-weight edge with one endpoint in $S$.
  • Add new node to $S$. 
4. Greedy Algorithms II

- red-rule blue-rule demo
- Prim’s algorithm demo
- Kruskal’s algorithm demo
- reverse-delete algorithm demo
- Boruvka’s algorithm demo

Section 4.5
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
- Add to \( T \) unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
  • Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
• Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:

- Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
- Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
  • Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
- Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
  • Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
- Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
- Add to $T$ unless it would create a cycle.
Consider edges in ascending order of weight:
- Add to $T$ unless it would create a cycle.
Kruskal’s algorithm demo

Consider edges in ascending order of weight:
- Add to $T$ unless it would create a cycle.
4. **Greedy Algorithms II**

- red-rule blue-rule demo
- Prim’s algorithm demo
- Kruskal’s algorithm demo
- reverse-delete algorithm demo
- Boruvka’s algorithm demo
Reverse-delete algorithm demo

Start with all edges in $T$ and consider them in descending order of weight:
- Delete edge from $T$ unless it would disconnect $T$. 
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:
- Delete edge from $T$ unless it would disconnect $T$. 

![Diagram showing a graph with weights and edges]
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:
- Delete edge from $T$ unless it would disconnect $T$. 

![Graph Diagram](image-url)
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:
- Delete edge from $T$ unless it would disconnect $T$. 

![Graph Diagram](image-url)
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:

- Delete edge from $T$ unless it would disconnect $T$. 
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:

- Delete edge from $T$ unless it would disconnect $T$. 
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:
• Delete edge from $T$ unless it would disconnect $T$. 
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:

- Delete edge from $T$ unless it would disconnect $T$. 
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:

- Delete edge from $T$ unless it would disconnect $T$. 
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:

- Delete edge from $T$ unless it would disconnect $T$. 

![Diagram](image)
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:
• Delete edge from $T$ unless it would disconnect $T$. 
Reverse-delete algorithm

Start with all edges in $T$ and consider them in descending order of weight:
- Delete edge from $T$ unless it would disconnect $T$. 

![Graph diagram](image-url)