Breadth First Search

\[ G = (V, E) \]

\[ w(x, y) = \text{weight on the edge } (x, y) \]

positive

"length" = weight = cost of the entire path

Task: finding shortest paths

\[ \rightarrow \text{Dijkstra's method (weighted, +ve)} \]

\[ \rightarrow \text{BFS (0/1, e,g,o)} \]

\[ G, \text{ source } x, \text{ find shortest path to all other vertices} \]

BFS:
Dijkstra's Algorithm

Replace # of hops with weighted hops.

Always pick the current smallest cost vertex and expand from.

\[ \text{PQ: Array} \quad \text{delete min: } O(1) \]
\[ \text{insert/decrease: } O(1) \]

\[ \text{Dijkstra's: } O(|V|) \quad \text{delete mins} \]

\[ \text{Total cost: } O((|V| + |E|) \log |V|) \]
Dijkstra's: $O(|V|) \text{ delete mins}$
$O(|V| + |E|) \text{ decrease keys}$

**Binary PQ**

$|V| \log |V| + (|E| \log |V|)$ vs $O(|V|^2)$

Good for sparse graphs

$|E| = O(|V|)$

$\approx O(|V| \log |V|)$

**Dense:**

$|E| = O(|V|^2)$

$|V|^2 \log |V| \approx O(|V|^2)$

Array PQ

Negative weights

---

Counting example to Dijkstra's
Bellman-Ford method

Update the weights of all vertices in each iteration:

For all vertices \( u \in V \) for all neighbors \( y \) of \( u \), update weight:

\[
\text{cost}(u) + w(u, y) \leq \text{cost}(y)
\]

\[
\text{cost}(y) = \text{cost}(u) + w(u, y)
\]

Max # of edges in any path from \( x \): \( |V| - 1 \)

Exact \( |V| - 1 \) iterations
Exactly \(|V|-1\) iterations

In each iteration we update all edges

Total cost: \((|V|-1) \times (|V| + |E|)\)

\[= \frac{|V|^2 + |V| \cdot |E|}{|V|} \]

Sparse: \(O(|V|)\)

Dense: \(O(|V| \cdot |E|)\)

\(\implies O(|V|^2)\)
If we restrict to "paks" Bellman Ford will not work

1) Find all possible paths between any pair $(X, Y)$
   choose the min cost path

Exponential time method

Greedy Algorithms

"Local" information

-> pick the best available option

O(1V^2 1E)
**Minimum Spanning Tree**

Output: \( G = (V, E) \), with weights

Output:

1) Tree

2) Span all the vertices

\[ T = (V', E') \]

Spanning tree: set of vertices in \( T \) include all vertices in \( G \)

\[ E' \subseteq E \]

Cost of \( T = (V', E') \)

\[ \text{Cost}(T) = \sum_{e \in E'} w(e) \]

Find a tree \( T \) with smallest possible \( \text{Cost}(T) \)

\[ T_1 \text{ is a spanning tree} \]

\[ \text{Cost}(T_1) = 23 \]
Prim's Algorithm (similar to Dijkstra's)

- \( O(|E_1| \log |E_1|) \)
  1) sort the edge in increasing order

  2) \( T = \{ \text{1st edge in sorted order} \} \)

  for each edge in sorted order \( e \in E_1 \)

  add 'if and only if one end point in T'

  add edge to \( T \)

  if one end point in \( T \) & we do not create a cycle

- \( O(|E_1| \log |E_1| + |E_1|) \)

(\( G \) is undirected)

1) \( ac, cd, ad, ab, be, bd, ef, ef, df \)

2) \( T \)

\( T = \{ \{a, c\} \} \)

picture

1) sort the edges

\( |T_2| = 16 \)
picture

\[
\begin{align*}
\{a, c, d\} \\
\{a, c, d, b\} \\
\{a, c, d, b, f\} \\
\{c, c, d, b, f\}
\end{align*}
\]

1) sort the edges
2) \( T = \{ \text{smallest edge} \} \)

Iteratively add the least weight edge with one end point not in \( T \)

greedy: add the least cost edge