Edmonds-Karp Algorithm

Shusen Wang

Edmonds–Karp Algorithm

- Edmonds-Karp algorithm [2] is almost the same as as Ford-Fulkerson algorithm [1].
- Edmonds-Karp algorithm uses the shortest path from source to sink. (Apply weight 1 to all the edges of the residual graph.)
- Edmonds-Karp algorithm is a special case of Ford-Fulkerson algorithm.

Reference

1. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8: 399–404, 1956.
2. J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM.* 19 (2): 248–264, 1972.

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to the capacities.

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to the capacities.
2. While augmenting path can be found:
    a. Find an augmenting path (on the residual graph.)
    b. Find the bottleneck capacity x on the augmenting path.
    c. Update the residuals. (residual $\leftarrow$ residual $- x$.)
    d. Add a backward path. (Along the path, all edges have weights of x.)

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to the capacities.
2. While augmenting path can be found:
   a. Find an augmenting path (on the residual graph.)
   b. Find the bottleneck capacity x on the augmenting path.
   c. Update the residuals. ($\text{residual} \leftarrow \text{residual} - x$.)
   d. Add a backward path. (Along the path, all edges have weights of x.)

Time complexity: $O(f \cdot m)$. (f is the max flow; m is #edges.)

Edmonds–Karp Algorithm

1. Build a residual graph; initialize the residuals to the capacities.
2. While augmenting path can be found:
   a. Find the shortest augmenting path (on the residual graph.)
   b. Find the bottleneck capacity x on the augmenting path.
   c. Update the residuals. (residual $\leftarrow$ residual $- x$.)
   d. Add a backward path. (Along the path, all edges have weights of x.)

When finding path, regard the residual graph as unweighted.

Time complexity: $O(m^2 \cdot n)$. (m is #edges; n is #vertices.)

Time Complexity Analysis

- m: number of edges.
- n: number of vertices.
- Each iteration has $O(m)$ time complexity.
  - The residual graph has at most $2m$ edges.
  - Finding the shortest path has $O(m)$ time complexity.

Time Complexity Analysis

- m: number of edges.
- n: number of vertices.
- Each iteration has $O(m)$ time complexity.
- The number of iterations is at most $m \cdot n$.
- The worst-case time complexity is $O(m^2 \cdot n)$.

Thank You!