Discussion 10: Shortest Path

(Dijkstra, Bellman-Ford, A*, Floyd-Warshall)

CS101 Fall 2024

CS101 Course Team

December 9, 2024

Bellman-Ford

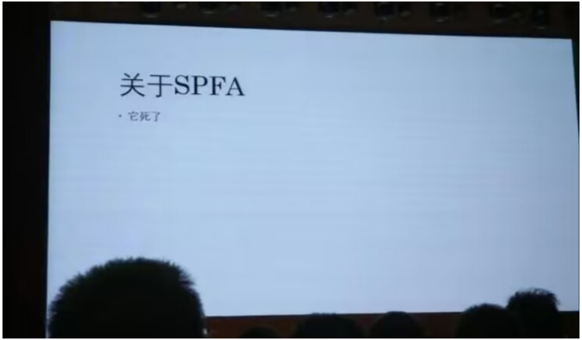
dist$[x][i]$: The shortest path length from the starting point s to x, passing through no more than i edges. And we can eliminate the second dimension with a scrolling array.

```
for(i = 1; i <= V - 1; ++i)
{
    for(const auto& e : Edge)
        if (dist[e.v] > dist[e.u] + e.w)
            dist[e.v] = dist[e.u] + e.w;
}
```

Bellman-Ford can deal with graphs with negative weights. It could also be used to detect if there is a negative cycle in the graph. (Hint: if there is a negative cycle, then it would be updated that there exists a path passing through $|V|$ edges)

Shortest Path Faster Algorithm(SPFA)*

Actually Bellman-Ford Algorithm with queue optimization. Put the node into a queue after a relaxation happens.
Sometimes really fast, but the worst case is still $O(nm)$.



Dijkstra

```
S = empty set
Q = G.V
while Q is not empty                    // O(V)
    u = ExtractMin(Q)                   // O(V)
    S = S + {u}
    for each vertex v in G.Adj[u]       // O(deg(u))
        Relax(u, v, w)                  // O(1)
```

Time complexity: $O(|V|^2 + \sum_{v \in V} deg(v)) = O(|V|^2 + |E|)$.

With binary heap optimization:

ExtractMin $\rightarrow O(\log |V|)$

Relax $\rightarrow O(\log |V|)$

Time complexity:

$O(|V| \log |V| + \sum_{v \in V} deg(v) \log |V|) = O(|V| \log |V| + |E| \log |V|) = O(|E| \log |V|)$.

Dijkstra

Time complexity

1. Adjancy matrix $O(|V|^2)$
2. Adjancy list $O(|V|^2 + |E|)$
3. Adjancy list + binary heap $O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$
4. ~~Adjancy list + Fibonacci heap $O(|V|\log|V| + |E|)$~~ (not in CS101)

Dijkstra

Dijkstra has now been proved to be Universal Optimality.

Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps

Bernhard Haeupler*
bernhard.haeupler@inf.ethz.ch
ETH Zurich & Carnegie Mellon University

Richard Hladík
rihl@uralyx.cz
ETH Zurich

Vaclav Rozhon†
vaclavrozhon@gmail.com
ETH Zurich

Robert E. Tarjan‡

Jakub Tětek

However, Dijkstra cannot deal with graphs with negative weights.

Dijkstra

Have a try!
https://acm.shanghaitech.edu.cn/d/CS101_2024Fall/p/11

Floyd-Warshall

$dist[k][i][j]$: The shortest path length from i to j that only passes through nodes in $\{1, 2, \ldots, k\}$. And we can eliminate the first dimension with a scrolling array.

```
for (int k = 0; k < V; ++k)
    for (int i = 0; i < V; ++i)
        for (int j = 0; j < V; ++j)
            if (dist[i][j] > dist[i][k] + dist[k][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
```

Floyd-Warshall can deal with graphs with negative weights. It could also be used to detect if there is a negative cycle in the graph. (Hint: if there is a negative cycle, then it has $dist[i][i] < 0$, otherwise $dist[i][i] = 0$)

Shortest Path Algorithms

Algorithm	Type	Time complexity	negative weights	judge negative circle
Bellman-Ford	Single Source	$O(\|V\|\|E\|)$	yes	yes
Dijkstra	Single Source	$O(\|E\| \log \|V\|)$ $O(\|V\|^2 + \|E\|)$	no	no
Floyd	All pair of nodes	$O(\|V\|^3)$	yes	yes

Special cases

1. DAG
   Topological sort. $O(|V| + |E|)$.
2. All weights are equal.
   BFS. $O(|V| + |E|)$.
3. **Dense** graph with no negative weights
   Dijkstra **without** heap optimization. $O(|V|^2 + |E|)$ ✓
   Dijkstra **with** heap optimization. $O(|E| \log |V|) = O(|V|^2 \log |V|)$ ✗

A*

- Tree search: regard yourself searching on a tree(never worry about repeatedly accessing nodes that have already passed)
- Graph search: regard yourself searching on a tree(record the nodes that have already been traversed and cannot repeat them)
- Admissible heuristic function $h(u) \leq d(u, t)$: **never overestimates**.
- Consistent heuristic function $h(u) \leq h(v) + w(u, v)$: **triangle inequality**.

If $h(n)$ is admissible, then the tree search is optimal.
If $h(n)$ is consistent, then the graph search is optimal.
If $h(n)$ is consistent, then it must be admissible. But if $h(n)$ is admissible, then it may not be consistent.

A* examples

1. Let h_1 be a consistent heuristic and h_2 be an inconsistent one. Then:
   - $\frac{h_1+h_2}{2}$ is necessarily consistent. $\times$
   - $\max(h_1, h_2)$ is necessarily consistent. $\times$
   - $\min(h_1, h_2)$ is necessarily consistent. $\times$
2. If h_1 and h_2 are admissible, then judge whether the following heuristics are admissible or not.
   - $h_1 + h_2$ $\times$
   - $\max(h_1, h_2)$. $\checkmark$
   - $\min(h_1, h_2)$. $\checkmark$
   - $\alpha h_1 + (1-\alpha)h_2, \alpha \in [0, 1]$ $\checkmark$