Kruskals

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Freq. | Cost | Total |
| Build queue | 1 | E | E |
| Delete Min | E | logE | **E\*logE** |
| Union | V | logV | V\*logV |
| Connected | E | logV | E\*logV |

Prims

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PQ Impementation | Insert | Delete-min | Decrease-key | Total |
| Unordered Array | 1 | V | 1 | V^2 |
| Binary Heap | logV | logV | logV | E\*logV |
| D-way heap | Log4V | D\*logdV | logdV | E\*log(e\4)V |
| Fibonacci heap | 1 | logV | 1 | E+V\*logV |

- Array is best for dense graphs.

- Binary heap is best for sparse graphs.

- 4-way heap worth the trouble in performance-critical situations.

- Fibonacci best in theory but not worth implementing.

Kruskals is better for sparse, has simpler data structures.

Prims is better for dense, better data structures.

**Single Source Shortest Path:**

Dijkstra's (Updates previous paths):

- Works when weight(n,v) >= 0

- Initialize distance array.

- Pick source node (S), relax edges.

- Choose closest vertex to S, and relax all it's edges.

- Choose closest vertex to S, and relax all it's edges.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Queue | S | A | B | C | D | E |
| S | 0 | Inf | Inf | inf | inf | inf |
| S,B |  | 5,S | 1,S | 8,B | inf | inf |
| S,B,C |  | 5,S |  | 4,B | inf | 8,B |
| S,B,C,A |  | 5,S |  |  | inf | 8,B |
| S,B,C,A,D |  |  |  |  | 7,A | 8,B |
| S,B,C,A,D,E |  |  |  |  |  | 8,B |

- In Djikstra's each edge will be relaxed once.

- Big Oh(|V|+|E|\*log|V|).

**Bellman-Ford (Solving inequalities):**

- Works for general weights and it can detect negative cycle.

- In general Bellman-Ford you iterate n-1 times to find the shortest distance, and the nth iteration is to check for negative cycles.

- Every inequality is a directed line.

- The "negative" x is the source, the other is the target.

- Because there is 5 vertices excluding v0, we will do 6 iterations.

- Create a v0 node, connect it to all inequalities with weight of 0.

- Relax all nodes.

- Check the incoming edge for change, only ONE update per iteration, so if a node updates you must wait to reflect that in it's children

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Node | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| V0 | 0 | 0 | 0 | 0 | 0 | 0 | \ |
| V1 | Inf | 0,v0 | -3,v5 | -7,v4 | -9,v2 | -9,v2 | \ |
| V2 | inf | 0,v0 | 0 | 0 | 0 | -1,v1 | \ |
| V3 | inf | 0,v0 | 0 | 0 | 0 | 0 | \ |
| V4 | inf | 0,v0 | -2,v2 | -2,v2 | -2,v2 | -2,v2 | -3,v2 |
| V5 | inf | 0,v0 | -4,v4 | -6,v2 | -6,v2 | -6,v4 | \ |

- There was a change in the v+1th iteration, therefore the graph contains a negative cycle.

- In Bellman-Ford an edge may be relaxed many times.

- Big Oh (|V|\*|E|)

**Chain Matrix Multiplication:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 15,000 |  |  |
| 2 |  | 0 | 6,000 |  |
| 3 |  |  | 0 | 60,000 |
| 4 |  |  |  | 0 |

(A1 = 50,10), (A2 = 10,30), (A3 = 30,20), (A4 = 20,100)

(A1 x A2) 50 x 10 x 30 = 15,000

(A2 x A3) 10 x 30 x 20 = 6,000

(A3 x A4) 30 x 20 x 100 = 60,000

-

A1\*(A2 x A3) = m[1,1] + m[2,3] + (50\*10\*20) = **0 + 6,000 + 10,000**

(A1 x A2)\*A3 = m[1,2] + m[3,3] + (50\*30\*20) = 0 + 60,000 + 30,000

-

A2\*(A3 x A4) = m[2,2] + m[3,4] + (10\*30\*100) = 0 …

(A2 x A3)\*A4 = m[2,3] + m[4,4] + (10\*20\*100) = 0 …

-

m[1,1] + m[2,4]+ (50\*10\*100) =

m[1,2] + m[3,4] + (50\*30\*100) =

m[1,3] + m[4,4] + (50\*20\*100) =