CS340 - Analysis of Algorithms

DP on Graphs

Announcements:

HW8 Due Wednesday (November 26)

Lecture Wednesday?

Final Exam Options:

1. Dec 12 1-4pm Park 230
2. Dec 15 9:30-12:30 Park 159

Agenda

- DP on graphs
  - Djikstra’s review
- Shortest Path algorithms for graphs with negative edges
  - Bellman-Ford
  - Floyd-Warshall

Dijkstra’s Review

- Greedy Algorithm
- Computes a shortest path from the origin s to every other node v in the graph
- Maintains a set S with the property that the shortest path from s to each node in S in known
- Start with S = {s} and greedily add elements to this set
- Consider the node u with minimum-distance in the PQ
    - Relax edges outgoing from u
- Dijkstra's algorithm is correct for graphs with non-negative edges

Dijkstra's Review - Runtime Complexity

```
dijkstra(G, s){
  for each (u in V) { d[u] = infinity  visited[u] = false  }
  d[s] = 0 pred[s] = null
  Q = priority queue of all vertices u keyed by d[u]
  while (Q is not empty) {
    u = extractMin from Q
    for each (v in Adj[u]) {
      if (d[u] + w(u, v) < d[v] && !visited[v])
        d[v] = d[u] + w(u, v)
        decrease v's key value in Q to d[v]
        pred[v] = u //keeps track of the tree
      }
    }
  }  visited[u] = true
}
```

Init:

$O(n) + O(n\log n)$

For each node extract min:

$n * O(\log n)$

For each edge, decrease key:

$m * O(\log n)$

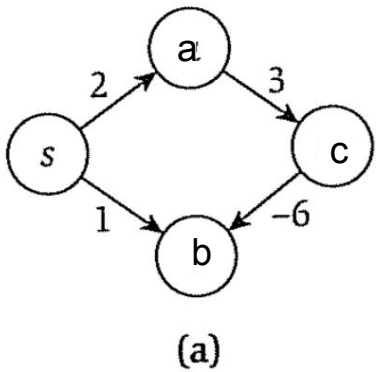
$O(n) + O(n\log n) + O(n\log n) + O(m\log n)$

$n < m < n^2$

$O(m\log n)$

Dijkstra's Review

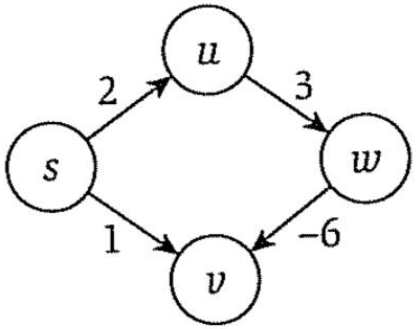
```
dijkstra(G, s){
  for each (u in V) { d[u] = infinity  visited[u] = false  }
  d[s] = 0 pred[s] = null
  Q = priority queue of all vertices u keyed by d[u]
  while (Q is not empty) {
    u = extractMin from Q
    for each (v in Adj[u]) {
      if (d[u] + w(u, v) < d[v] && !visited[v])
        d[v] = d[u] + w(u, v)
        decrease v's key value in Q to d[v]
        pred[v] = u //keeps track of the tree
      }
    }
  } visited[u] = true
}
```



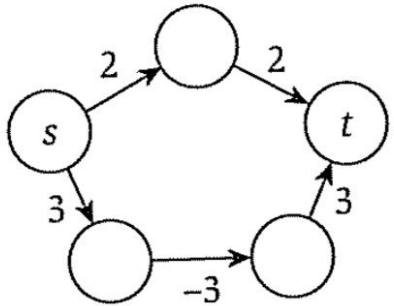
(a)

Shortest Paths with Negative Weights

Idea 1: add a constant to each edge



(a)



(b)

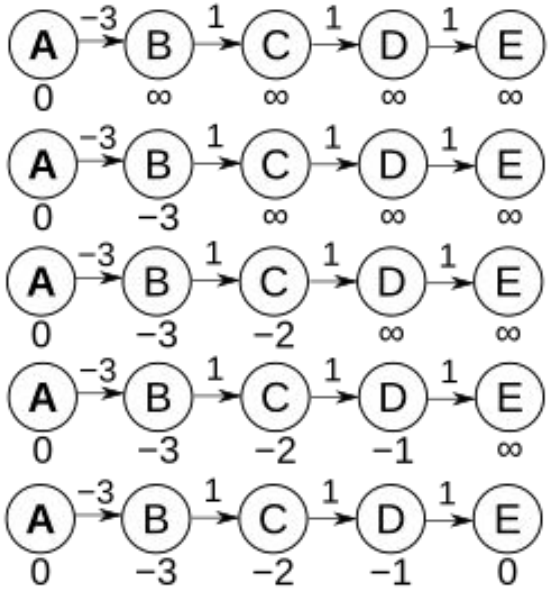
Changing the weights of all edges by an equal constant can change the identity of shortest paths

Shortest Paths with Negative Weights

We will try to use DP to solve the problem of finding a shortest path when there are negative edges but no negative cycles

Dijkstra's algorithm greedily selects the closest vertex that has not yet been processed, and relaxes the outgoing edges

We want to relax all the edges, regardless of if they have been processed or not, n-1 times



Bellman Ford (non DP)

```
BellmanFord(G=(V,E), s):
    // Step 1: Initialize distances
    for each vertex v in V:
        d[v] = ∞

    d[s] = 0

    // Step 2: Relax all edges |V| - 1 times
    for i = 1 to n - 1:
        for each edge (u, v) in E:
            if d[u] + w(u,v) < d[v]:
                d[v] = d[u] + w(u,v)
```

Shortest Paths with Negative Weights

Break down the problem by **expressing the optimal solution in terms of optimal solutions to sub-parts**

- DP algorithms we've seen so far:
  - Weighted interval scheduling
    - $OPT(j) = \max(w_j + OPT(p(j)), OPT(j-1))$
  - Knapsack
    - OPT(n, W) = max(V[n] + OPT(n-1, W-w_n), OPT(n-1, W))
  - CMM, Least Common Subsequence ...
- How can we represent the shortest paths problem in terms of sub-parts?
  - Shortest path using only the first i nodes

Bellman Ford Algorithm

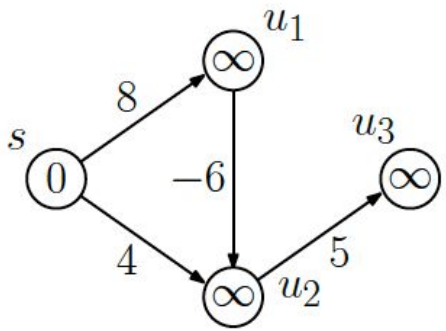
OPT(v, i) = length of shortest s-v path using at most i edges

- path uses at most i-1 edges:
  - OPT(i,v) = OPT(i-1,v)
- path uses i edges, and the last edge is (u,v):
  - OPT(i,v) = w(u,v) + OPT(i-1, u)

OPT(i,v) = min(OPT(i-1, v), forall (u,v) in E: min (OPT(i-1, u) + w(u,v)))

Bellman Ford Algorithm - Trace

```
BellmanFordDP(G=(V,E), s):
      // dp[i][v] = shortest distance to v using at most i edges
      n = |V|
      m = |E|

      // Initialize DP table
      for i = 0 to n-1:
        for each vertex v:
            dp[i][v] = ∞

      dp[0][s] = 0

      // Fill DP table: n-1 iterations
      for i = 1 to n-1:
        // Copy previous values first
        for each vertex v:
            dp[i][v] = dp[i-1][v]

        // Relax all m edges
        for each edge (u, v) in E:
          if dp[i-1][u] != ∞:
            dp[i][v] = min(dp[i][v], dp[i-1][u] + w(u,v))

      return dp[n-1]
```



Bellman Ford Algorithm - Runtime Analysis

```
BellmanFordDP(G=(V,E), s):
      // dp[i][v] = shortest distance to v using at most i edges
      n = |V|
      m = |E|

      // Initialize DP table
      for i = 0 to n-1:
        for each vertex v:
            dp[i][v] = ∞

      dp[0][s] = 0

      // Fill DP table: n-1 iterations
      for i = 1 to n-1:
        // Copy previous values first
        for each vertex v:
            dp[i][v] = dp[i-1][v]

        // Relax all m edges
        for each edge (u, v, weight) in E:
          if dp[i-1][u] != ∞:
            dp[i][v] = min(dp[i][v], dp[i-1][u] + weight)

      return dp[n-1]
```

1. Init:

   $O(n^2)$

2. Copy previous values:

   $O(n^2)$

3. Relax all edges:

   $O(mn)$

Total runtime: $O(mn)$

Bellman Ford Algorithm - Proof of Correctness

By induction

Floyd-Warshall Algorithm

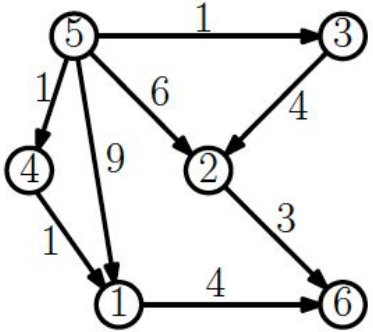
An algorithm for finding shortest paths *between all nodes* in a directed graph with any value edges (negative or positive or zero allowed)

Negative cycles are still not allowed

For each pair of vertices (i, j), we consider whether going through an intermediate vertex k gives a shorter path than the current best path from i to j

Floyd-Warshall Algorithm

- Define d_{ij}^{k} to be the length of the shortest path from i to j such that any intermediate vertices on the path are chosen from the set $\{1, 2, \ldots, k\}$



$d_{5,6}^{(0)} =$

$d_{5,6}^{(1)} =$

$d_{5,6}^{(2)} =$

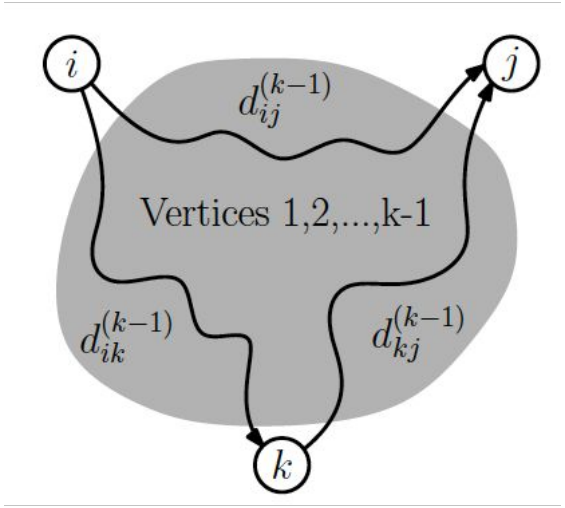
$d_{5,6}^{(3)} =$

$d_{5,6}^{(4)} =$

Floyd-Warshall Algorithm

How to compute d_{ij}^{k} assuming that we have computed all of d^{k-1}?

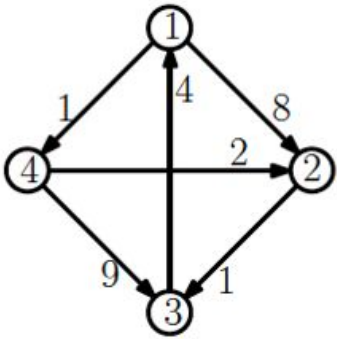
- p doesn't go through k: $d_{ij}^{k} = d_{ij}^{k-1}$
- p does go through k:
- $d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$



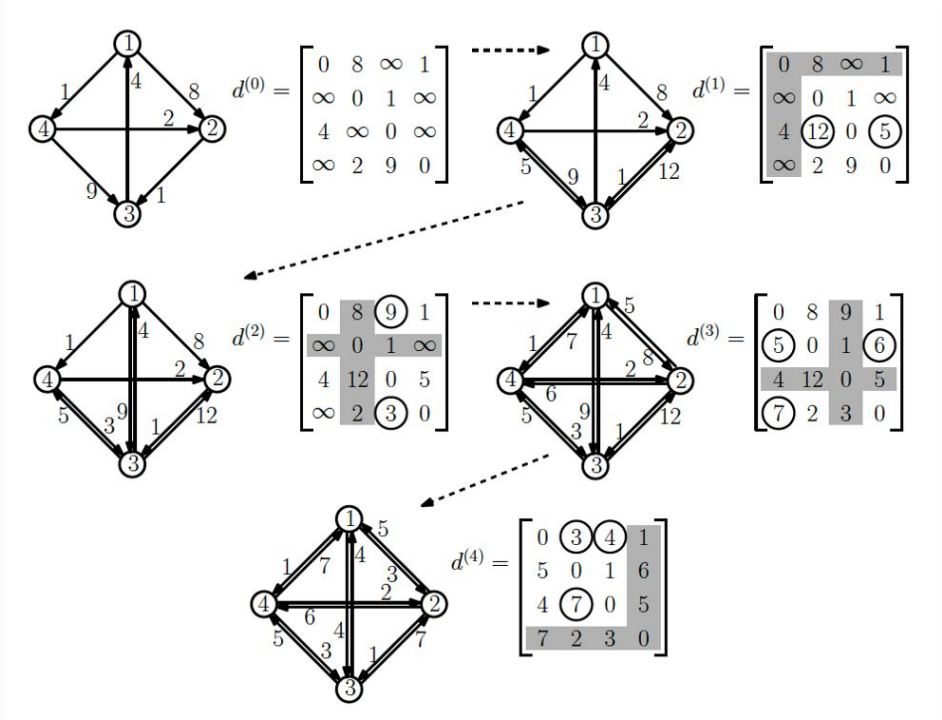
$$d_{ij}^{k} = \begin{cases} w_{ij}, & k = 0 \\ \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}), & k \geq 1 \end{cases}$$

Floyd-Warshall Algorithm Trace

```
Floyd-Warshall(n) {
  d = new array[1..n][1..n] mid = new array[1..n][1..n]
  set all d[i][j] = w[i][j]; set all mid[i][j] to 0
  for (k = 1 to n) {    // use intermediates 1..k
    for (i = 1 to n) {  // from i
      for (j= 1 to n) { // to j
        if (d[i][k]+d[k][j] < d[i, j]) {
           d[i][j] = d[i][k] + d[k][j]
           mid[i][j] = k
        }
}}}}
```



Floyd-Warshall Algorithm Trace



Floyd-Warshall Algorithm - Runtime Analysis

```
Floyd-Warshall(n) {
  d = new array[1..n][1..n] mid = new array[1..n][1..n]
  set all d[i][j] = w[i][j]; set all mid[i][j] to 0
  for (k = 1 to n) {    // use intermediates 1..k
    for (i = 1 to n) {  // from i
      for (j= 1 to n) { // to j
        if (d[i][k]+d[k][j] < d[i, j]) {
           d[i][j] = d[i][k] + d[k][j]
           mid[i][j] = k
        }
}}}}
```

$O(n^3)$

Summary

- DP on Graphs: solves optimization problems by breaking them into subproblems over vertices / edges and stores results in arrays (memoization)

- Bellman-Ford: DP for single source shortest paths. Supports negative weights. O(mn)

- Floyd-Warshall: DP for all pairs shortest paths. Supports negative weights. $O(n^3)$

- **HW8 due Wednesday**