CS 161: Design and Analysis of Algorithms

Announcements

- Final Exam:
  - Friday August 17^{th}
  - 12:15 – 3:15pm
  - Skilling Auditorium

Dynamic Programming III: Shortest Paths/Traveling Salesman

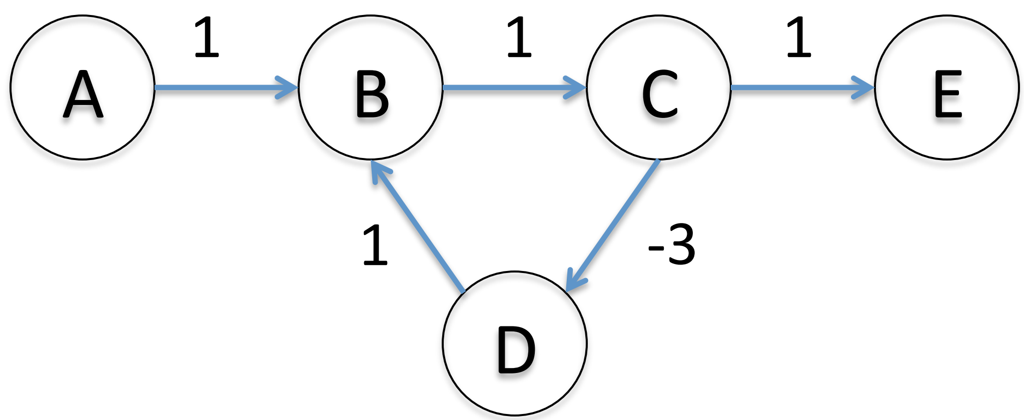
- Shortest paths problem, revisited
- All pairs shortest paths
- Traveling Salesman Problem

Shortest Paths, Revisited

- Single-Source Shortest Paths Problem:
  - Given start node v, computes distances from v to all other nodes u
- Dijkstra's algorithm
  - Solves single-source problem if no negative edges
- What about negative edge weights?

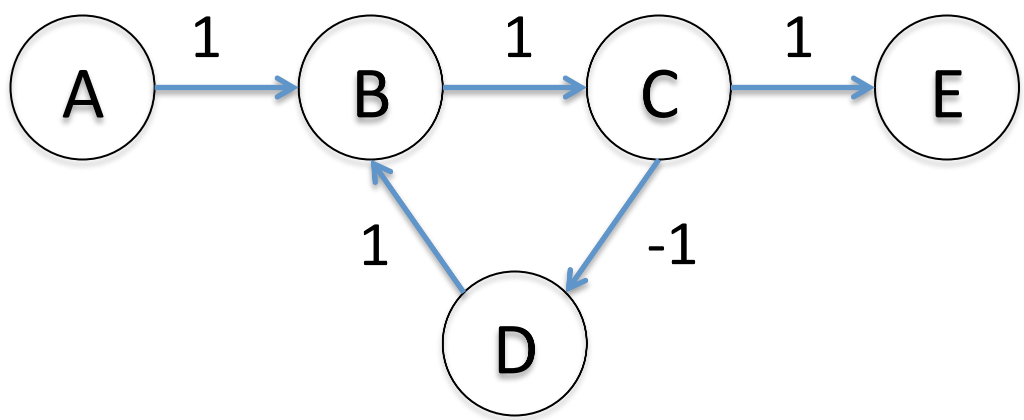
Negative Cycles

- If a graph has a negative cycle, shortest path does not exist



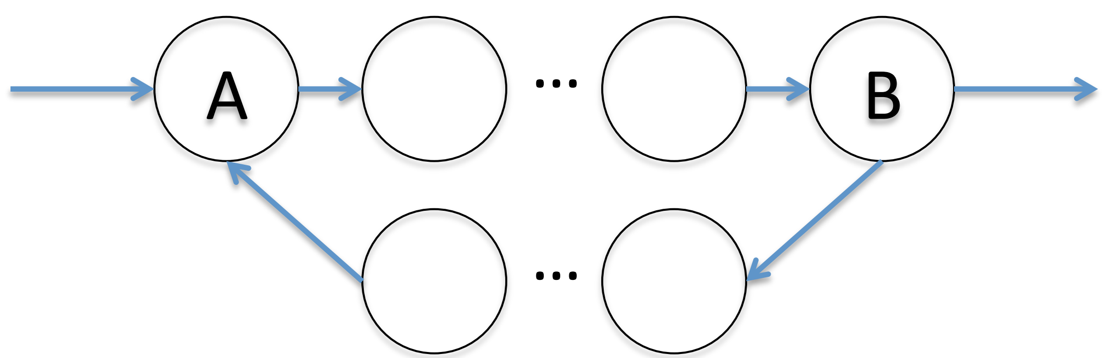
Negative Cycles

- If no negative cycles, may still have shortest path



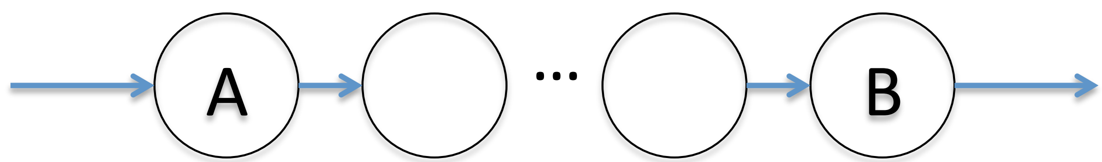
Negative Cycles

- **Theorem**: If a graph has no negative cycles, then given a path P from v to u, we can construct a simple path P' from u to v such that length(P') ≤ length(P).



Negative Cycles

- **Theorem**: If a graph has no negative cycles, then given a path P from v to u, we can construct a simple path P’ from u to v such that length(P’) ≤ length(P).



Weight decreases by length of cycle. If all cycles are non-negative, weight can only decrease

Negative Cycles

- **Corollary**: If a graph has no negative cycles, then the shortest path between two nodes exists, and can be taken to be simple
  - To find shortest path, we only need to look at simple paths
  - Only a finite number of simple paths
  - Minimum must exist

Negative Cycles

- Therefore, the shortest path problem is well-defined if and only if there are no negative cycles
- Two problems:
  - Dijsktra only works with non-negative edge weights.
  - How do we detect negative cycles?

Dynamic Programming Solution

- We want length of shortest path from v to all other nodes u
- What is a good subproblem?
- Dijsktra: length of shortest path from v to some subset of nodes

Dynamic Programming Solution

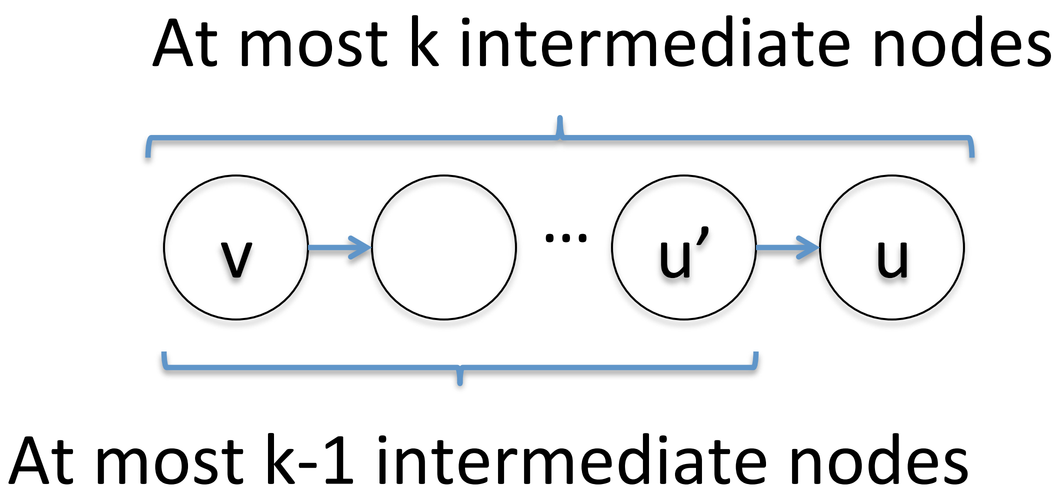
- What is a good subproblem?
  - In the presence of a negative cycle, obtaining arbitrarily short paths requires paths with an arbitrary number of nodes/edges
  - If we cap the number of nodes/edges in a path, shortest path problem is well-defined.
  - Good subproblem: shortest path from v to all other nodes u, limited to at most k intermediate nodes

Dynamic Programming Solution

- Let S(u,k) be the length of the shortest path from v to u, using at most k intermediate nodes
  - S(u,k) = ∞ if no such path exists
- S(u,0) = w(v,u) if the edge (v,u) exists, ∞ otherwise

Dynamic Programming Solution

- How do we compute S(u,k) values in terms of S(u',k-1) values?



Dynamic Programming Solution

- S(u,k) = S(u',k-1) + w(u',u) for some u' with (u',u) in E
- To find S(u,k), just do minimum over all u' with (u',u) in E:

$$S(u,k) = \min_{(u',u)\in E} \left(S(u',k-1) + w(u',u)\right)$$

Dynamic Programming Solution

- How large of k do we need?
- If no negative cycles, then shortest path is simple
- Simple paths have at most |V| nodes, so at most |V|-2 intermediate nodes
- Only need to go up to k = |V| - 2

Dynamic Programing Solution

- For all u in V: set S(u,0) = w(v,u)
- For k = 1, …, |V|-2:
  - For all u in V: set

$$S(u,k) = \min_{(u',u)\in E} \left(S(u',k-1) + w(u',u)\right)$$

- For all u in V, report distance(v,u) = S(u,|V|-2)

Running Time?

- For each k,
  - |V| updates
  - Each update looks at indegree(u) other nodes
  - Running time for all |V| updates: O(|V| + |E|)
- Total running time: O(|V||E|)

Optimization

- Keep track of S(u)
- Initially, let S(u) = w(v,u)
- Whenever we set S(u,k), also update S(u):

$$S(u) = \min_{(u',u) \in E} \left(S(u') + w(u',u) \right)$$

$$S(u,k) = \min_{(u',u) \in E} \left(S(u',k-1) + w(u',u) \right)$$

Optimization

- Claim: $S(u,k) \leq S(u,k')$ for all $k' < k$

Optimization

- Claim: Wherever we set S(u,k) and update S(u), S(u) ≤ S(u,k)
  - S(u) = S(u,k) at the beginning
  - Suppose true before particular update

$$S(u,k) = \min_{(u',u)\in E} \left(S(u',k-1) + w(u',u)\right)$$

$$S(u) = \min_{(u',u)\in E} \left(S(u') + w(u',u)\right)$$

Optimization

- Claim: S(u) is always at least the length of some path from v to u
  - True at the beginning
  - Suppose true before a particular update

$$S(u) = \min_{(u',u)\in E} \left(S(u') + w(u',u)\right)$$

  - All of the S(u') + w(u',u) must be at leas the length of the shortest path from v to u

Optimization

- Therefore, if $S(u,|V|-2)$ is the correct length of the shortest path from v to u, so is $S(u)$

Optimization

- For all u in V: set S(u) = w(v,u)
- For k = 1, …, |V|-2:
  - For all u in V:

$$S(u) = \min_{(u',u)\in E} (S(u') + w(u',u))$$

- For all u in V, report distance(v,u) = S(u)

- Called Bellman-Ford algorithm

Alternatively

- For all u in V: set S(u) = w(v,u)
- For k = 1, …, |V|-2:
  - For all (u’,u) in E:

$$S(u) = \min\left(S(u), S(u') + w(u', u)\right)$$

- For all u in V, report distance(v,u) = S(u)

- Called Bellman-Ford algorithm

Similarity to Dijkstra

- Dijkstra is also just sequence of updates

$$S(u) = \min\left(S(u), S(u') + w(u', u)\right)$$

  – Update (u’,u) where S(u’) is minimum

Detecting Negative Cycles

- What if we update one more time?
- S(u) is always at least as long as some path from v to u
- If no negative cycles, after k = |V| - 2, S(u) is at most the length of the shortest path, so it is equal
  - Future updates will not change S(u)

Detecting Negative Cycles

- In general, if S(u) values don't decrease in one iteration, will never decrease again

$$S(u) = \min\left(S(u), S(u') + w(u', u)\right)$$

- S(u) ≤ S(u,k)
- If negative cycles, S(u,k) will get arbitrarily small
- S(u) must keep decreasing

Detecting Negative Cycles

- To detect a negative cycle, run one extra time
- If distances change, there is a negative cycle
- If distances don't change, there is no negative cycle, and distances are correct

All Pairs Shortest Paths

- What if we want to compute the length of the shortest path between all pairs of nodes v and u
- Can run all-pairs shortest paths from all nodes v
  - $|V|$ * (running time of single-source algorithm)
  - Bellman-Ford: $O(|V|^2 |E|)$
  - Dijkstra: $O(|V|^2 \log |V| + |V||E|)$

All Pairs Shortest Paths

- If we want to handle negative edges, cannot use Dijsktra. Bellman-Ford gives $O(|V|^2|E|)$
- Can we do better?

All Pairs Shortest Paths

- Good subproblem:
  - Arbitrarily number the nodes {1,…,|V|}
  - dist(i,j,k} = length of shortest path from i to j, where intermediate nodes come from {1,…,k}
- How many subproblems: $|V|^3$

All Pairs Shortest Paths

- How to compute dist(i,j,k):
  - Either k is in shortest path, or it isn't
  - If k is in shortest path,

    dist(i,j,k) = dist(i,k,k-1)+dist(k,j,k-1)
  - If not, dist(i,j,k) = dist(i,j,k-1)
  - Therefore, just set dist(i,j,k) to be the minimum

All Pairs Shortest Paths

```
for i = 1, …, |V|:
        for j = 1, …, n:
                dist(i,j,0) = ∞
for all (i,j) in E:
        dist(i,j,0) = w(i,j)
for k = 1, …, |V|:
        for i = 1, …, |V|:
                for j = 1, …, |V|:
                        dist(i,j,k) = min( dist(i,j,k-1),
                                           dist(i,k,k-1)+dist(k,j,k-1) )
Return dist(i,j,|V|) for all
```

All Pairs Shortest Paths

- Running time?
  - $|V|^3$ subproblems
  - Each takes time O(1) to solve
  - $O(|V|^3)$

Travelling Salesman Problem

- Given a complete, weighted, undirected graph $G = (V,E)$, a **tour** is a simple cycle covering all nodes
- **Travelling Salesman Problem**: find tour of least total weight

Alternate Formulation

- Given array of values $d_{i,j}$ for i,j in {1,…,n}
- For any permutation {p(1),p(2), …, p(n)} of the set {1,…,n}, let the cost be

$$d_{p(1),p(2)} + d_{p(2),p(3)} + \ldots + d_{p(n-1),p(n)} + d_{p(n),p(1)}$$

- Find the permutation p with minimum cost
- Invariant to cyclic shifts, so we can take p(1) = 1

Brute-Force Algorithm

- Check all permutations, keeping the minimum
- Number of permutations?
  - We can take p(1) to be 1
  - p(2) has n-1 choices
  - p(3) has n-2 choices
  - …
  - Total choices: (n-1)!
- Running time: O(n (n-1)!) = O(n!)

Nitpicking

- Is $n! = \theta((n-1)!)$?
  - $(n-1)! < n!$, so $(n-1)! = O(n!)$
  - $n!/(n-1)! = n \rightarrow \infty$, so $(n-1)! = o(n!)$
- Therefore, $(n-1)!$ is technically asymptotically smaller than $n!$

A Dynamic Programming Solution

- Assign nodes labels 1,…,|V|
- For a node j and a subset S including 1 and j, let C(S,j) be length of shortest path visiting each node in S exactly once, starting at 1, ending at j
- Define C(S,1) = ∞ for |S| > 1 since we cannot both start and end at 1

A Dynamic Programming Solution

- What is the second-to-last node?
  - Must be some i in S
  - Path to i must contain all nodes in S other than j
  - $C(S,j) = C(S-\{j\},i) + w(i,j)$
  - Simply pick the best i over all i in S other than j

A Dynamic Programming Solution

$C(\{1\},1) = 0$

for $s = 2, \ldots, |V|$:

  for all subsets S of $\{1,\ldots,|V|\}$ of size s containing 1:

    $C(S,1) = \infty$

    for all j in S, $j \neq 1$:

      $C(S,j) = \min(C(S-\{j\},i)+w(i,j): i \text{ in } S, i \neq j)$

Return $\min(C(\{1,\ldots,|V|\},j) + w(j,1))$

Running Time

- $2^{|V|}$ different subsets S
- Up to $|V|$ different j
- Minimize over up to $|V|$ different i
- Running time: $O(|V|^2 \, 2^{|V|})$
- Much better than $O(|V|!)$