Review:

Goal: Determine distances from source $s$ in graphs

Single Source Shortest Path Algorithms:

- Breath First Search $\text{BFS}$ ✓
- Dijkstra's Algorithm ✓
- Bellman-Ford Algorithm today

**BFS**

**Idea:**
- Start from source $s$
- Find its neighbors
- Given set $S$ of vertices of distance $d$ from $s$,
  find all not yet seen neighbors
  of the vertices in $S$
  $\rightarrow$ vertices at distance $d+1$

**Example:**

$$
\begin{array}{c}
\begin{array}{ccc}
A & B & C \\
\hline
1 & 0 & 1 \\
\end{array} \\
\begin{array}{c}
\begin{array}{ccc}
D & E & F \\
\hline
1 & 1 & 1 \\
\end{array} \\
\end{array}
\end{array}

\Rightarrow

\begin{array}{c}
\begin{array}{ccc}
A & B & C \\
\hline
\emptyset & 0 & 1 \\
\end{array} \\
\begin{array}{c}
\begin{array}{ccc}
D & E & F \\
\hline
\emptyset & 1 & 1 \\
\end{array} \\
\end{array}
\end{array}

\text{BFS-tree}

**Running Time:** $O(|V| + |E|)$
Dijkstra’s Algorithm

Goal: Find distances from source \( s \) when edges have lengths \( \ell(u,v) \) length of \( (u,v) \)

\[
\ell(s,v) = \text{length of shortest path from } s \text{ to } v
\]

\[\text{dijkstra}(\ell, s)\]

- \( \text{dist}[s] = 0 \quad \text{prev}[s] = s \)
- \( \forall v \neq s \quad \text{dist}[v] = \infty \)
- Build priority queue \((V, \text{dist}[\cdot])\)
  - \( \forall u \in V \quad \text{insert}(u, \text{dist}[u]) \)
  
while \( U \neq \emptyset \)
  - \( u = \text{Delete Min} \)
  - \( \forall \text{ edges } (u,v) \in E \)
    - If \( \text{dist}[u] + \ell(u,v) < \text{dist}[v] \)
      - \( \text{dist}[v] = \text{dist}[u] + \ell(u,v) \)
      - \( \text{prev}[v] = u \)
      - \( \text{Decrease Key}(v, \text{dist}[v]) \)

Idea: Dijkstra discovers vertices in the order of their distance from the source, updating an estimate for \( d(s,v) \) that is equal to

\[
\text{dist}(v) = \begin{cases} 
\ell(s,v) & \text{if } v \notin V \setminus U \\
\text{the length of shortest path } \omega : s \rightarrow v \text{ with all edges in } V \setminus U \text{ except for the last one} & \text{if } v \in U
\end{cases}
\]
This work because

- Dijkstra finds correct next vertex \( v \)
- Dijkstra updates \( \text{dist} \[\cdot \] \) correctly

new shortest path to \( w \)
could go through \( v \)
instead of \( u' \)

\[
V \setminus U \quad U
\]

Remark: For applications, we often want to keep track of the shortest paths from the source, not just the distance \( d(s,v) \). The above pseudo code does this by updating \( \text{pre}(v) \), the predecessor of \( v \) in the shortest path found.

**Running Time**

\( n = |V| \) inserts \& delete\(\min\)

\( m = |E| \) decrease\(\text{key}\)

| Implementation   | delete\(\min\) | insert/ decrease\(\text{key}\) | \( V \times \text{delete\(\min\)} + (|V|+|E|) \times \text{insert} \) |
|------------------|---------------|--------------------------------|------------------------------------------------|
| Array            | \( O(n) \)    | \( O(1) \)                      | \( O(n^2) \)                                  |
| Binary heap      | \( O(\log n) \) | \( O(\log n) \)               | \( O(n+m \log n) \)                           |
| Fibonacci heap   | \( O(\log n) \) | \( O\left(\frac{\log n}{\log d}\right) \) | \( O(nd + m \frac{\log n}{\log d}) \)         |
Bellman-Ford (Graphs with neg. weights)

Dijkstra does not work
Why?
What is $d(5, D)$?

Only well defined if cycles have positive length

Define

update($u, v$)

$$\text{dist}[v] = \min \{ \text{dist}[v], \text{dist}[u] + l(u, v) \}$$

We consider an arbitrary algorithm which starts with

$$\text{dist}(s) = 0, \quad \text{dist}(v) = \infty \quad \forall v \neq s$$

and then calls update($u, v$) successively for different edges ($u, v$), possibly several times for a given edge

Properties

1) This maintains upper bounds on $d(s, v)$ (it is safe)

2) If $u$ is the second to last node on a shortest path to $v$ and
\[ \text{dist} [v] = \text{dist}(v) \Rightarrow \text{dist}(v) = d(n,v) \text{ after update}(u,v) \]

**Proof 1:** Let \( \text{dist}'[v] \) be the value after calling \( \text{update}(u,v) \).

By induction on the number of times \( \text{update} \) has been called, we may assume \( \text{dist}(v) \geq d(n,v) \), \( \text{dist}(u) \geq d(n,v) \)

\[ \Rightarrow \text{dist}'[v] \geq \min \{ d(n,v), d(n,u) + l(u,v) \} \]

But \( d(n,u) + l(u,v) \) is the length of some path from \( n \) via \( u \) to \( v \), and thus at most the length, \( d(n,v) \) of the shortest path from \( n \) to \( v \) \( \Rightarrow \) claim

**Proof 2:** If \( \pi \) is a shortest path \( n \rightarrow v \)

\[ \Rightarrow \text{path length}(\pi) \text{ must be shortest as well, and hence equal to } d(n,v) \]

\[ d(n,v) = \text{length of } \pi = \text{length of } \pi + l(u,v) \]

\[ = d(n,u) + l(u,v) \]

\[ \downarrow \text{ by assumption in (2) } \]

\[ = \text{dist}[u] + l(u,v) \]

\[ \geq \min \{ \text{dist}[v], \text{dist}[u] + l(u,v) \} \]

\[ = \text{dist}[v] \text{ after update}(u,v) \]

By (1), we also have the bound \( \text{dist}[v] \geq d(n,v) \)

\( \Rightarrow \) claim
Properties 1 and 2 imply

**Property 3**: Let \( s, u_1, u_2, \ldots, u_n, t \) be a shortest path from \( s \) to \( t \). If we make the updates

\((s, u_1), (u_1, u_2), \ldots, (u_n, t)\)

in that order (possibly with other steps inbetween),

then \( \text{dist}(t) = d(s, t) \).

**Claim**: If we run the updates through all edges \((n-1)\) times, \( \text{dist}[v] = d(s, v) \) \( \forall v \).

**Proof**: Any shortest path has at most \( n-1 \) edges (otherwise vertices are repeated, leading to a cycle, which can't be part of a shortest path).

\( \implies \) For each \( v \), the edges on the shortest path from \( s \) to \( v \) are updated as required by property 3 \( \implies \) claim

**Bellman-Ford (6, 2, 0)**

Output:

\[ \text{dist}(v) = \text{dist}(n, v) \quad \forall \ v \in V \]

\[ \forall \ u \in V \text{ set } \text{dist}(u) = \infty \]

Repeat \( n-1 \) times

\[ \forall (u, v) \in E \]

update \((u, v)\)
A: Run one more time, i.e. IVI times
    if \( \exists \) neg. cycle \( \text{dist}(u) \) goes down for at least one \( v \)

**Shortest Paths in DAGs**

Look at DAG with vertices reverse ordered by post(\( u \)) times of DFS

\[ \Rightarrow \text{all edge go forward.} \]

\( \Rightarrow \) all path in DAG run forward. Thus,

IF we run update(\( u, v \)) respecting order of \( u \),

* on each shortest path, edges are update in order

\[ \Rightarrow \text{dist}(v) = \text{dist}(\alpha, v) \text{ for all vertices } v \]

**DAG-SSSP(\( G, \pi, \gamma \))**

\[
\forall u \in V \text{ dist}(u) = \infty \\
\text{dist}(\alpha) = 0 \\
\text{Run DFS and rev. order vertices by post(\( u \))} \\
\forall u \text{ in this order} \\
\forall (u, v) \in E : \text{Update}(u, v)
\]

**Running Time:** \( O(IVI + 1E1) \)

**Correctness:** Let \( \nu_1, \nu_2, \ldots, \nu_n \) be reverse ordered by post(\( u \))

\( \Rightarrow \text{all edges point forward} \)

Let \( \omega = \nu_1, \nu_2, \ldots, \nu_k \) be a shortest path from \( \alpha \) to \( v \),

Since all edges point forward
\[ u_1 < u_2 < \ldots < u_n \text{ in the above order} \]

\[ \Rightarrow \text{update first updates (} u_1, u_2 \text{), then (} u_1, u_3 \text{),} \ldots \]

(by property 3) \[ d[v] = \text{dist}(v, v) \]

---

**Greedy Algorithms**

**Goal:** Optimize some function in some multi-step process (Chess, Scrabble, ...) 

**Greedy:** Don't think ahead, just do what looks best at the time

**Example: Scheduling**

**Input:** n jobs with start and end times

\[
(s_1, e_1) \ldots (s_n, e_n)
\]

**Task:** Schedule as many as possible without overlap

\[
\begin{array}{c}
J_1 \quad J_2 \quad J_3 \quad J_4 \quad J_5 \\
\hline
J_1 \quad J_3 \quad J_5
\end{array}
\]

Optimal: \[ J_1, J_3, J_5 \]

**Strategies:**

- Shortest First
- First Start Time
- First Finish Time

**Counterexample**

- None
Claim: First finish time is optimal

Proof Strategy “Exchange proof”

Consider optimal strategy \( \pi \)
transform it to greedy, step by step

--- greedy

--- optimal

---

Lemma: Greedy is optimal

By: Let

\[ \text{Greedy} = [\pi_1, t_1], \ldots, [\pi_k, t_k] \]

\[ \text{Optimal} = [\pi'_1, t'_1], \ldots, [\pi'_n, t'_n] \]

Claim 0: \( k \leq n \)

Claim 1: For all \( k \leq l \),

\[ 0_k = \{ [\pi_1, t_1], \ldots, [\pi_k, t_k], [\pi_{k+1}, t_{k+1}], \ldots, [\pi_n, t_n] \} \]

is optimal

By: \( l = 0 \) ν

\[ l \rightarrow l+1: \quad 0_l = \{ [\pi_1, t_1], \ldots, [\pi_k, t_k], [\pi_{k+1}, t_{k+1}], \ldots \} \]
greedy = \{ (n_1, t_1), \ldots, (n_e, t_e), (n_{e+1}, t_{e+1}) \ldots \}

Both \(O_e\) and \(\text{greedy}\) have no overlaps

- Definition of \(\text{greedy}\) \(\Rightarrow t_{e+1} \leq t_{e+1}'\)
- \(\text{greedy has no overlaps}\) \(\Rightarrow t_{e+1} > t_e\)
- \(O_e\) has no overlaps \(\Rightarrow t_{e+1}' < s_{e+2}'\)

\(\Rightarrow t_e < s_{e+1}'\) and \(t_{e+1} < s_{e+2}'\)

\(\Rightarrow O_{e+1} = \{(n_1, t_1), \ldots, (n_e, t_e), (n_{e+1}, t_{e+1}), (n_{e+2}', t_{e+2}')\}\)

has no overlap.

Same # of jobs \(\Rightarrow O_{e+1}\) is still optimal.

**Claim 2:** \(n > k\) is not possible

\(O_k = \{(n_1, t_1), \ldots, (n_k, t_k), (n_{k+1}, t_{k+1}) \ldots \}\)

\(-\text{greedy}\)

\(\Rightarrow\text{greedy could have added}\ [n_{k+1}', t_{k+1}'] \Rightarrow \&\)

---

**Compression**

**Goal:** Encode text with \(T\) letters from a finite alphabet \(\Gamma\) with frequency \(f_i\) for \(i \in \Gamma\)

**Ex.:** \(\Gamma = \{A, B, C, D\} \quad T = 100\)

**Naive:** \(A = 00\), \(B = 01\), \(C = 10\), \(D = 11\) \(\Rightarrow 200\) Bits
What if $A$ appears much more often

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
<th>Code</th>
<th>Code 2</th>
<th>Code 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>80</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>01</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>11</td>
<td></td>
<td>111</td>
</tr>
<tr>
<td><strong>Cost:</strong></td>
<td>200</td>
<td>110</td>
<td>130</td>
<td></td>
</tr>
</tbody>
</table>

**Prefix-Problem:** In Code 2, how to decode

$10 = BA$ or $C$?

- $B$ and $C$ have same prefix

**Prefix-Free Property:**

No codeword can be prefix of another, e.g. Code 3

**Tree Representation**

Binary tree:

- 0 in $i^{th}$ position $\iff$ go left in level $i$

Codewords on leaves $\iff$ prefix-free

---

Full binary tree:

- Every node has 0 or 2 children