Lecture 15

"Shortest paths" in graphs with negative edge weights allowed

- Bellman-Ford (single source)
- Floyd-Warshall (all pairs)

Suppose we have a (directed or undirected) graph

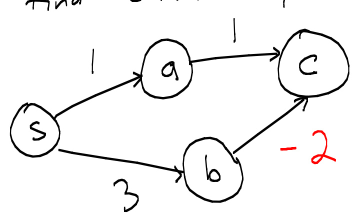
$G = (V, E)$

$V = \{v_1, v_2, \ldots v_n\}$

Let $s \in V$ be "starting vertex"

Recall: If a graph has positive and negative weights, then Dijkstra is not guarenteed to find shortest path.

Example:



Dijkstra finds shortest path to c is (s, a, c) whereas true shortest path is (s, b, c).

Dynamic Programming approach

- how to reduce problem size?
  - reduce number of vertices and edges in the graph?
  - something else?

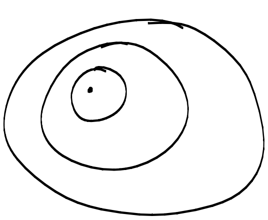
Bellman-Ford approach

Suppose we choose the minimum cost path from s to v

(s, ______ , v)

that has at most i edges

$i = 0, 1, 2, \ldots$



This gives us a set of increasing subproblems.

Let $Opt(i, v)$ be the minimum cost of all paths from s to v that have at most i edges.

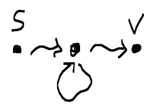
$$\text{Cost}(P) \equiv \sum_{\substack{\text{for all edges} \\ (u,v) \text{ in path } P}} c(u,v)$$

What is the range of i ?

$s \bullet \rightsquigarrow \bullet\ v$

A path with i edges has $i+1$ vertices
(with repeats, in case of "non simple" path)

A path with $|V| = n$ edges
thus must include a cycle.



We are not interested in paths with cycles
(for reasons described later).

Thus, we will only consider $i = 0, 1, .., n-1$.

"Base case"

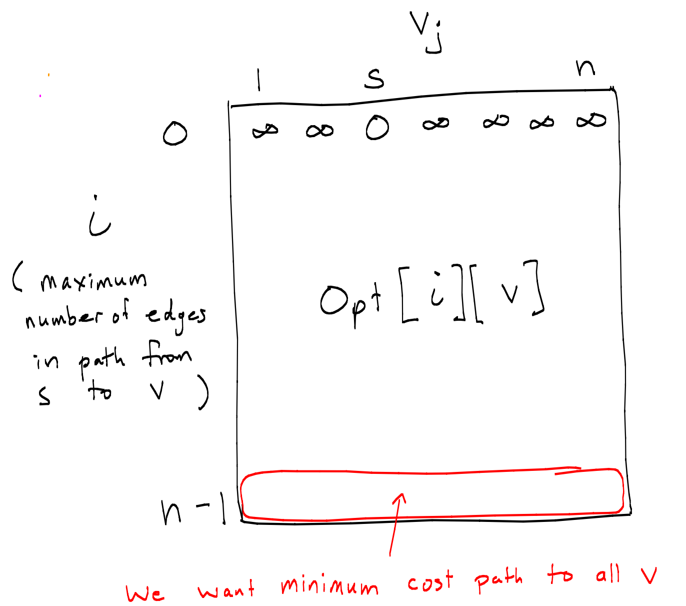
$s \bullet \rightsquigarrow \bullet\ v$

$Opt(0, s) = 0$

$Opt(0, v) = +\infty$ for $v \neq s$

"Induction step"

$$Opt(i, v) = \min \left\{ Opt(i-1, v), \min_{(u,v) \in E} \{ Opt(i-1, u) + c(u,v) \} \right\}$$



Same idea as previous DP problems.

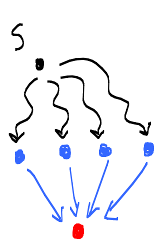
- iterative solution

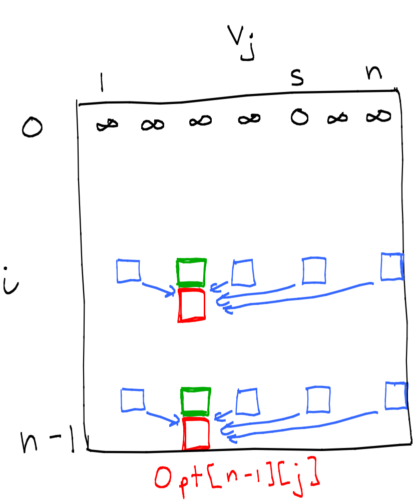
  for i = 1 to n-1
    for j = 1 to n
      Opt[i][j] = . . .

- recursive solution possible also

- once we have finished, we can find minimum cost path from s to any v by back tracking.

$$Opt(i, j) = \min \left\{ Opt(i-1, v), \min_{(u,v) \in E} \{ Opt(i-1, u) + c(u,v) \} \right\}$$

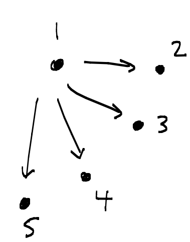




You might not need to go through the i loop $n-1$ times.
If Opt[i][j] == Opt[i-1][j] for all j
then we can stop.

Example

All paths are length ≤ 1 but $|V| = 5$.



How does Bellman-Ford differ from Dijkstra?

- Bellman-Ford: For loops do breadth first search. (Dijkstra does not do BFS)
- Dijkstra's correctness depends on all edges having non-negative weights.

  Bellman-Ford works even if edges have positive and negative weights. (it only disallows negative cycles – see below)

How long does Bellman-Ford take?
[Recall Dijkstra is $O(m \log n)$]

- $O(n^2)$? Two "for loops". But there's work to do at each cell Opt[i][j]
- $O(n^3)$? Two "for loops" and $O(n)$ operations at each cell.
- $O(n\, m)$? Two "for loops" but only need to check each incoming edge for second "for loop."

  $m = |E|$, $\; n \sum_v \text{in-degree}(v)$

Do we need $O(n^2)$ space?

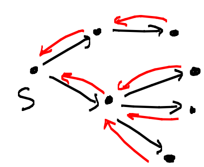
No, the iterative solution only uses the values from the previous row.

Yes, back tracking requires you have access to the whole Opt[ ][ ].

No, you can avoid backtracking as follows (similar to how Dijkstra keeps track of paths)

```
for all j = 1 to n {
    Opt [0][j] = +∞
    prev[j] = -1
}
Opt [0][s] = 0

for i = 1 to n-1
   for j = 1 to n
      for each (k,j) ∈ E {
         tmp = C(k,j) + Opt[i-1][k]
         if tmp < Opt[i][j] {
            Opt[i][j] = tmp.
            prev[j] = k
         }
      }
```



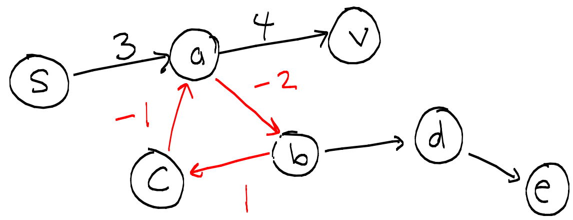
Instead of backtracking, to find the minimum cost path from s to v, we just run:

```
repeat {
   print v
   v = prev[v] }
until v == s
print s
```

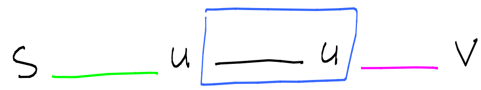
What if the graph has a cycle?

- negative cycle (" " < 0)
- positive cycle (total weights > 0)

If there were a path from s to v that contained a negative cycle then we could make a path with arbitrarily small cost by repeating the cycle many times (letting $i \to \infty$). So Bellman-Ford does not produce the correct result in that case.



Suppose a path from s to v has a positive cycle.



Then removing the cycle would give a path that has lower cost:

Thus, the Bellman Ford solution would be (simple) paths with no positive cycles.

Other dynamic programming approaches to shortest paths from s (in a graph with negative edges)?

Recall:

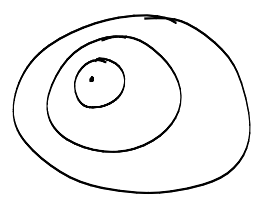
- weighted interval scheduling.
  - ↳ chose from subset of intervals $\{1, \dots i\}$
- segmented least squares
  - ↳ chose segmentation of $\{1, \dots i\}$
- sum of subsets / knapsack
  - ↳ chose from subset of weights $\{w_1 .. w_i\}$

How about this?

Suppose we choose the minimum cost path from s to v

$$(s, \_\_\_\_\_\_, v)$$

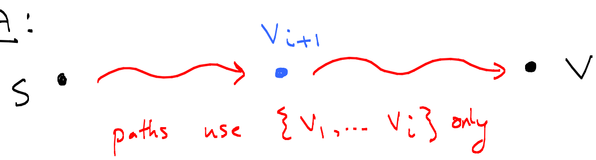
that only uses vertices $\{v_1, \dots v_i\}$ as intermediate vertices.



This would gives us a set of increasing size subproblems.

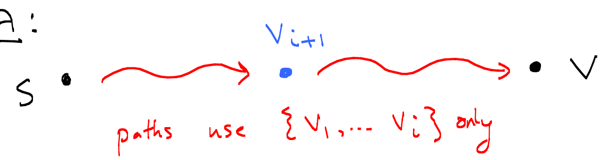
Suppose we have solution (shortest paths from s to all v) for some i, and we want to extend solution to i+1.

IDEA:



Suppose we have solution (shortest paths from s to all v) for some i, and we want to extend solution to i+1.

IDEA:



However, this doesn't work because we don't know the minimum cost path from v_{i+1} to v which uses only $\{v_1\ v_2 \dots v_i\}$ as intermediates.

... which brings me to ...

lecture 15

"Shortest paths" in graphs with negative edge weights allowed

- Bellman-Ford (single source)
- Floyd-Warshall (all pairs)

Suppose we want to find the minimum cost path between all pairs of vertices in a graph

- directed or undirected
- edges can have negative weight
- no negative cycles

Southern Ontario Distance Chart in KM (1 KM = 0.6 Miles)	Brantford	Collingwood	Elora	Fort Erie	Goderich	Grand Bend	Hamilton	Kitchener-Waterloo	London	Niagara Falls	Niagara-on-the-Lake	Oakville	Orangeville	Owen Sound	Point Pelee	Port Dover	Port Stanley	Sarnia	Port Elgin	St. Catharines	St. Thomas	Tobermory	Toronto	Windsor
Brantford	0	234	70	149	155	144	39	39	91	112	130	70	116	186	262	53	119	161	200	108	112	302	95	277
Colllingwood	234	0	110	282	176	220	200	140	310	256	262	169	75	58	477	273	336	409	107	240	325	170	145	486
Elora	70	110	0	208	128	149	76	30	136	177	153	94	42	121	303	114	164	239	125	131	151	224	114	317
Fort Erie	149	282	208	0	282	316	95	166	237	32	55	118	193	305	402	117	264	336	300	43	252	408	153	413
Goderich	115	175	128	282	0	49	172	115	91	235	258	201	140	123	279	207	130	116	89	239	126	239	214	301
Grand Bend	144	220	149	316	49	0	186	105	60	371	296	236	182	196	248	169	99	67	136	274	96	267	203	218
Hamilton	39	200	76	95	172	186	0	64	126	63	75	36	98	179	298	60	165	196	200	53	146	294	63	319
Kitchener -Waterloo	39	140	30	166	115	105	64	0	91	123	145	88	77	144	263	84	130	161	153	122	125	263	98	284
London	91	310	136	237	91	60	126	91	0	193	216	157	168	214	182	109	39	91	186	145	29	330	175	193
Niagara Falls	112	256	177	32	235	371	63	123	193	0	28	92	161	242	377	109	224	259	274	17	224	357	133	382
Niagara-on -the-Lake	130	262	153	55	258	296	75	145	216	28	0	97	172	284	382	128	243	316	280	23	231	387	132	393
Oakville	70	169	94	118	201	236	36	88	157	92	97	0	79	191	323	109	183	255	208	76	171	294	40	333
Orangeville	116	75	42	193	140	182	98	77	168	161	172	79	0	101	341	161	207	256	151	150	189	295	74	361
Owen Sound	186	58	121	305	123	196	179	144	214	242	284	191	101	0	361	240	274	239	53	263	272	116	203	434
Point Pelee	262	477	303	402	279	248	298	263	182	377	382	323	341	361	0	301	172	178	342	360	177	535	361	74
Port Dover	53	273	114	117	207	169	60	84	109	109	128	109	161	240	301	0	98	234	242	106	93	341	153	311
Port Stanley	119	336	164	264	130	99	165	130	39	224	243	183	207	274	172	98	0	127	244	222	15	387	220	182
Sarnia	161	409	239	336	116	67	196	161	91	259	316	255	256	239	187	234	127	0	214	294	115	336	293	108
Port Elgin	200	107	125	300	89	136	200	153	186	274	280	208	151	53	342	242	244	214	0	258	232	122	239	318
St. Catharines	108	240	131	43	293	274	53	122	145	17	23	76	150	263	360	106	222	294	258	0	210	366	111	371
St. Thomas	112	325	151	252	126	96	146	125	29	224	231	171	189	272	177	93	15	115	232	210	0	376	208	187
Tobermory	302	170	224	408	239	267	294	263	330	357	387	294	295	116	535	341	387	336	122	366	376	0	295	439
Toronto	95	145	114	153	214	203	63	98	175	133	132	40	74	203	361	153	220	293	239	111	208	295	0	371
Windsor	277	486	317	413	301	218	319	284	193	382	393	333	361	434	74	311	182	108	318	371	187	439	371	0

Here all edge weights are positive.

Running Bellman-Ford n times (once for each starting vertex) is too costly, namely

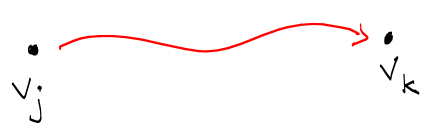
$O(nm \cdot n) = O(n^2 m)$ where $m = |E|$.

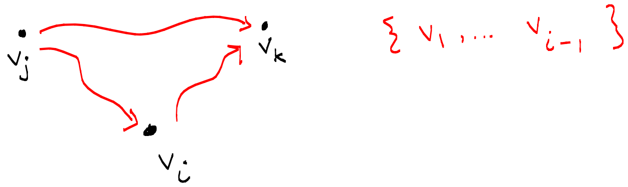
If graph is "dense", m is $O(n^2)$
"sparse", m is $O(n)$.

Define a 3D array

$Opt[i][j][k]$

= total cost of minimum cost path from v_j to v_k that only uses $\{v_1, \ldots v_i\}$ as intermediates.





initialization of Opt[ ][ ][ ] // base case

for $i = 1$ to $n-1$
  for $j = 1$ to n
    for $k = 1$ to n
      $Opt[i][j][k] = \min\{Opt[i-1][j][k],$
        $Opt[i-1][j][i] + Opt[i-1][i][k]\}$

initialization // base case

$$Opt[0][j][k] = \begin{cases} 0 & \text{, if } j = k \\ C(j,k) & \text{, if } (v_j\, v_k) \text{ is an edge.} \\ \infty & \text{, otherwise.} \end{cases}$$

Floyd - Warshall (all pairs) is $O(n^3)$ both space and time.

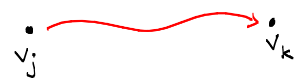
Compare to running Bellman-Ford n times (once for each starting vertex)

- If graph is "dense", time is $O(n^4)$
- " " "sparse", time is $O(n^3)$

Thus, Floyd-Warshall benefits us only if graphs are dense.

<u>Exercise</u>:

How to use Opt[ ][ ][ ] to reconstruct a min cost path from v_j to v_k ?



<u>Note</u>:

- use backtracking (but how?)
- for any j,k, $Opt[i][j][k]$ is "non-increasing" with i.
- minimum cost could be a negative

<u>Midterm 2 exam (Tues after Study Break)</u>

- similar format to midterm 1
- if you missed or messed up on midterm 1 you should still consider writing midterm 2.

Why?

- $\{49, 54, \dots 79, 84\} \Rightarrow$ I need a reason to bump you up
- . . . . .