Lec 2)

Lec 1 - Page 28 Hamilton path problem :

↳ 모든 Vertex 들을 1번씩 꼭 지나야 함
2번은 안된다.

- If there is a negative cycle
- Just identify a negative cycle & don't do it anything

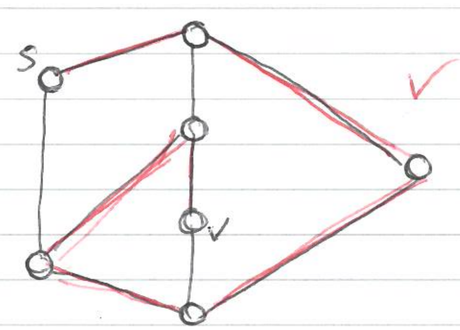
- If there is a no negative cycle
- then compute shortest path.

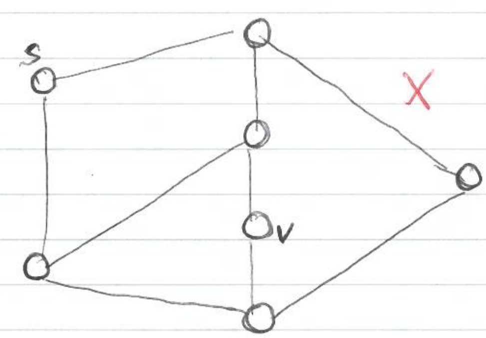
The hamilton path problem : for a given undirected graph and two nodes s and v, find a path from s to v which passes through each node in the graph exactly once, or decide that there is no such path.

⇒ This problem is computationally hard (NP-hard)

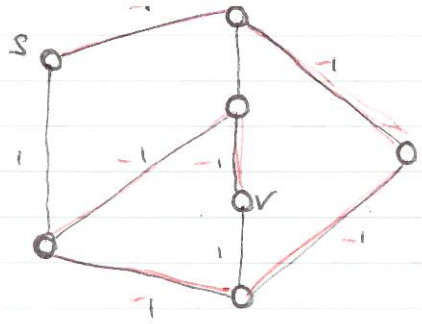
Show that the hamiltonian-path problem can be solved by an algorithm which finds a shortest simple path s to v.

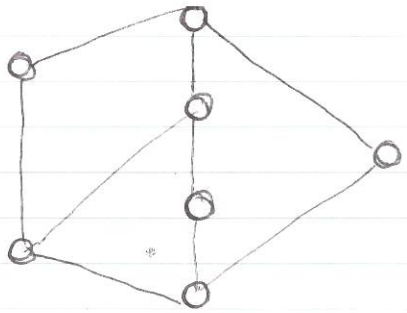
Hint : How should we define the weights of the edges so that a Hamiltonian-path is a shortest path?





1) brute-force - 노가다, 모든 경로 일일이 체크





1) each edge doesn't have a weight
= all edges have a same weight

2) Let's assume ever edge has same weight which is 1

3) Start 부터 v 까지의 최대 weight를 구한다.

4) 우리는 최단거리를 구해야 할 것 이므로 최소 → 최대로 변환
즉 모든 edge의 −를 붙인다.
= 즉 6 (최대 값) → −6 (최소 값) 으로 변환된다.

*) 만약 output이 output > −(n−1) 이면
Hamiltonian path는 존재하지 않는다.

*) Hamilton path = 모든 꼭지점을 포함하는 경로.

Hamilton cycle = 해밀턴 순환

Hamilton Graph = 해밀턴 순환을 갖는 그래프

Traceable Graph = 해밀턴 경로를 갖는 그래프.

결정문제 { 해밀턴 경로 문제 = Traceable Graph 인가?
해밀턴 순환 문제 = 이 그래프가 해밀턴 그래프인가?

Lec 2)
Lec 1 - page 11)

Relaxation technique

구성 = ① $d[v]$ array + ② PARENT $[v]$
총 2개의 Array 필요.

technique 순서 = ① Initialization (G, S) + ② Relaxation (u, v, w) {relax edge(u,v)

여기 Relaxation technique 은 알고리즘이 아니고 technique 이기 때문에 언제 Stop 해야 하는지가 Main Issue이다.

Main Issue

No negative cycle | negative cycle

- finite effective relax operation.
- PARENT pointers always form a tree rooted at S
- 마지막으로 No effective relax operation
① each node v, $d[v] = \delta(S,v)$
② PARENT pointers form a shortest path tree.

- Always $d[v] > d[u] + w(u,v)$
= Always effective operation is possible
↳ That's why we can't believe condition of effective relax operation.
- PARENT array form a cycle.
- 위 조건으로 인해서 PARENT Node들을 검사해서 negative cycle 이 있는지 확인 가능하다.

Page 16) The Bellman-Ford algorithm.

- The most general Algorithm when we have a negative ~~cycle~~ weight

Bellman-Ford (G, w, s) { G = (V, E) }
  Initialization (G, s) // Relaxation technique을 사용한 Initialization 부터 시작
  ①: for i ← 1 to n-1 do { n = |V| }
      for each edge (u,v) ∈ E do
        RELAX (u,v,w) // Initialization 하고 RELAX operation 하기로.
  ②: for each edge (u,v) ∈ E do
      if d[v] > d[u] + w(u,v) then
        Return FALSE (a negative cycle is reachable from s)
  RETURN TRUE (no negative cycle; for each v∈V, d[v] = δ(s,v))

Bellman-Ford의 main part 부분

Checking of effective Relaxation

Running time of Bellman-Ford algorithm

= (PART 1 부분) Θ(mn) + Θ(n) PART 2 부분

∴ Θ(mn)

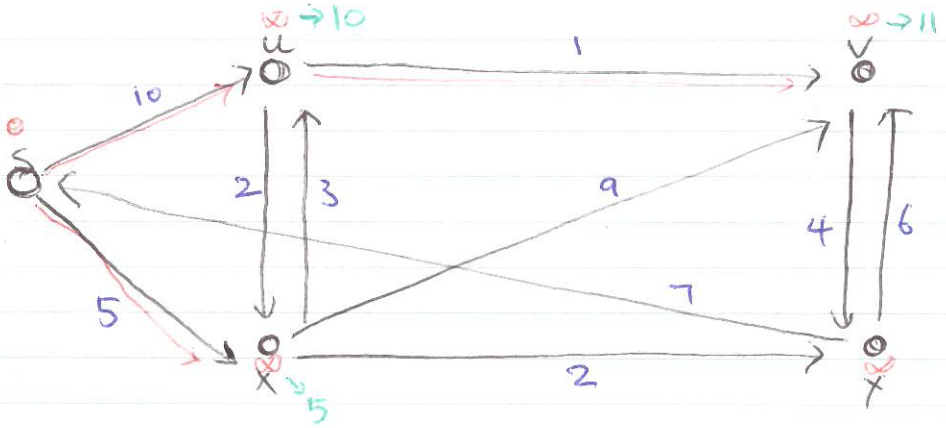
↳ 결국

* Worst running time of any algorithm for the single source shortest path

Ω(mn) because of negative weight

Example the computation by the Bellman-Ford algorithm.

Page 16)



Assume that the edges are given in this order:

(s,u) , (s,x) , (y,s) , (v,y) , (u,v) , (x,v) , (y,v) , (x,u)

(x,y) , (u,x)

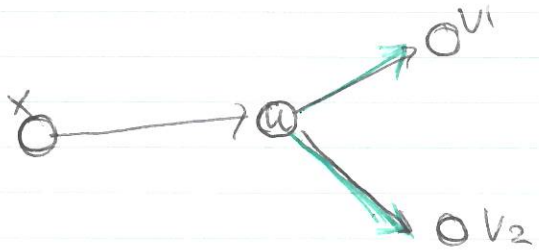
1) Initialization → $S = \emptyset$ $v \in V - \{s\} = \infty$

2) RELAX operation. Iteration $(n-1)$

3) Bellman-Ford-Algorithm.

Page 21)

active vertex : Improved shortest path estimate but not applied relaxed operation to out going edges



- Perform RELAX only on edge outgoing from active vertices
- Store active vertices in the Queue data structure.

The Bellman-Ford algorithm with a FIFO queue.

```
BF-with-FIFO (G, w, s)                    {G=(V,E)}
  Initialization (G,s)
  Q ← ∅ ; EnQueue (Q,s)                   {Queue Q - FIFO queue of active ver
  while Q ≠ ∅ do
  u ← head(Q) ; DeQueue(Q)
  for each v ∈ Adj[u]                     {for each node v adjacent to u}
     do RELAX(u, v, w)
return TRUE                               {for each v∈V, d[v] = δ(s,v)}
```

```
RELAX (u, v, w) :
  if d[v] > d[u] + w(u,v) then
     d[v] ← d[u] + w(u,v) ; PARENT[v] ← u
     If v is not in Q then EnQUE (Q, v)
```

- Adj[u] : the adjacent list of node u (the list of all neighbors of

* If there is negative cycle ⇒ never stop (Queue is never empty)
So this wrong algorithm.

* A mechanism for detecting negative cycles must be added : (3)
Periodically check PARENT pointers for a cycle.
O(nm) = running time.

Page 24)

Bellman-Ford : the Basic Bellman-Ford

Bellman-Ford 2 : the basic Bellman-Ford with termination,
If no effective relax operation in the current iteration.

Bellman-Ford-FIFO : Bellman-Ford with active nodes in FIFO

Lec 2)
Page 2

- Lec 2 is about restricted version of single source path problem.

    1) only non-negative edge weights allowed:
    Dijkstra's shortest-paths problems (교통거리 등 Negative weight 없을때)

    2) the input graph is acyclic (a DAG - a directed acyclic graph)
    Single source shortest paths algorithms for DAG'S.

- Single-source single-destination shortest-path problem.

    1) If we have additional informations (coordinates of nodes) then we can improve average running time. (Not the worst case)
    - Adoption of Dijkstra's shortest-paths problem.

- All pairs shortest-paths problem : find shortest paths for all source-destinations pairs of nodes.
- Johnson's algorithm.

Page 3) Dijkstra's shortest-path algorithm.

- Crucial assumption : all edge weights are nonnegative.
- For convinience, assume that all nodes are reachable from s.

```
Dijkstra(G, w, s)                    { G = (V, E) }
~~~~~~
Initialization(G, s)                 { relaxation technique initialization }
S ← ∅                                { nodes v for which we know that d[v] = δ[s,v] }
Q ← V                                { the other nodes, as Priority Queue }

While Q ≠ ∅ do         ← Priority Queue
u ← Extract-Min(Q)                   { u has the min d[] value in Q }
S ← S ∪ {u}
for each node v ∈ Adj[u] do   RELAX(u, v)
end_while.
```

This is for analysis (← arrow pointing to the line under "S ← ∅")

Running time of Dijkstra's Algorithm.

Initialization = n

while

- Extract_min(Q) = $O(n)$ because setting decrease. $\approx n$
- for ... do RELAX(u,v) = $O(n)$ ^{2}

End_while

$O(n^2)$

page(11) 참조, 12도 참고해서 정리하기.