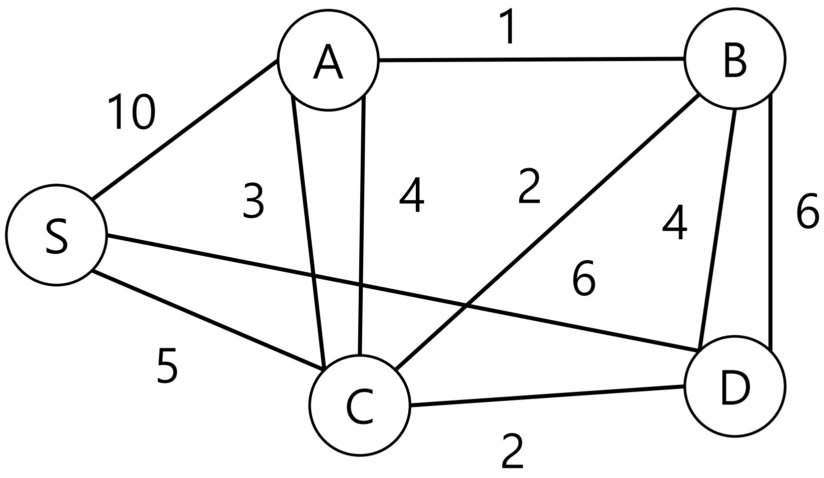
Greedy Algorithms
Minimum Spanning Tree

Greedy Algorithm

- 답을 찾기 위해 선택을 반복하는 알고리즘들 중
- 비교적(?) 간단한 방법으로 선택하고
- 선택한 후 바꾸지 않는 알고리즘

Shortest Path의 예

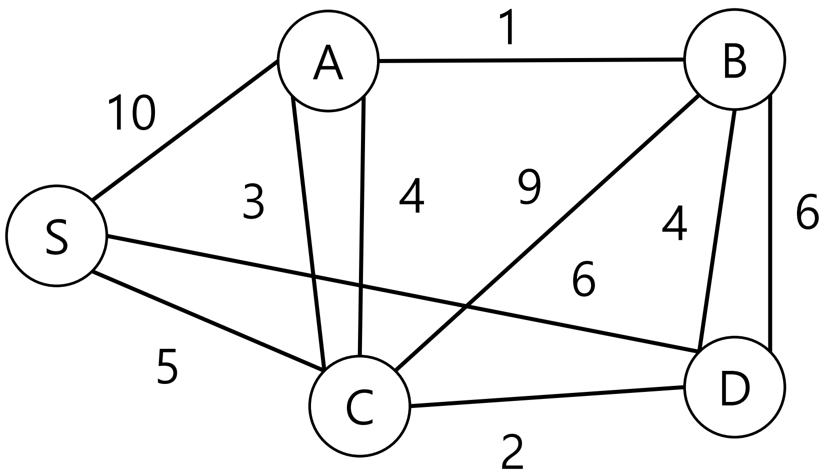


Selection Sort

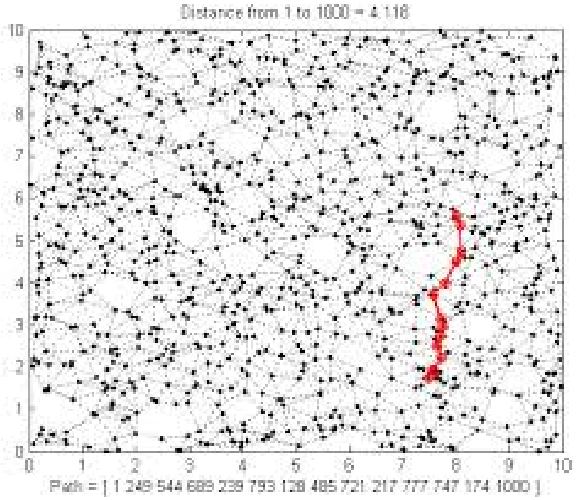
```
int sort(int a[], int n)
{
    int i, j, m, t;
    for (i = 0; i < n; i++) { *******
        // Find Minimum
        m = i;
        for (j = i; j < n; j++)
            if (a[m] > a[j]) m = j;
        t = a[i]; a[i] = a[m]; a[m] = t;
    }
    return;
}
```

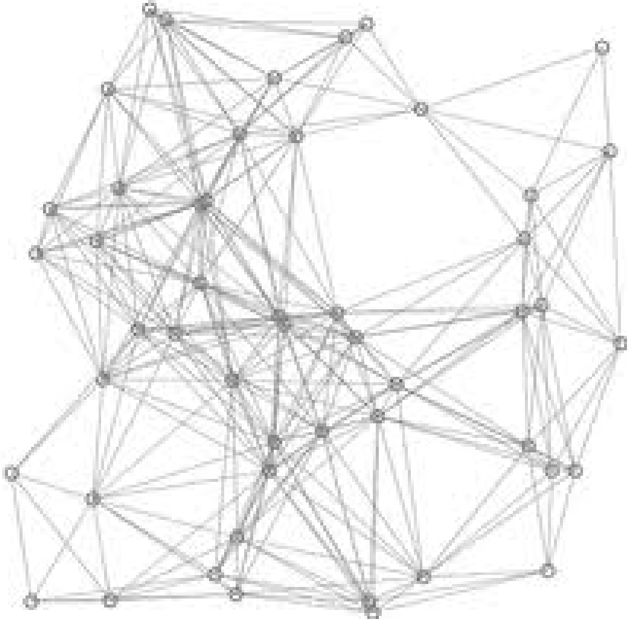
Minimum Spanning Tree

- Given a Graph, find Subset of Edges so that a Connected Graph results with Minimum Sum of Edge Costs



What about these?

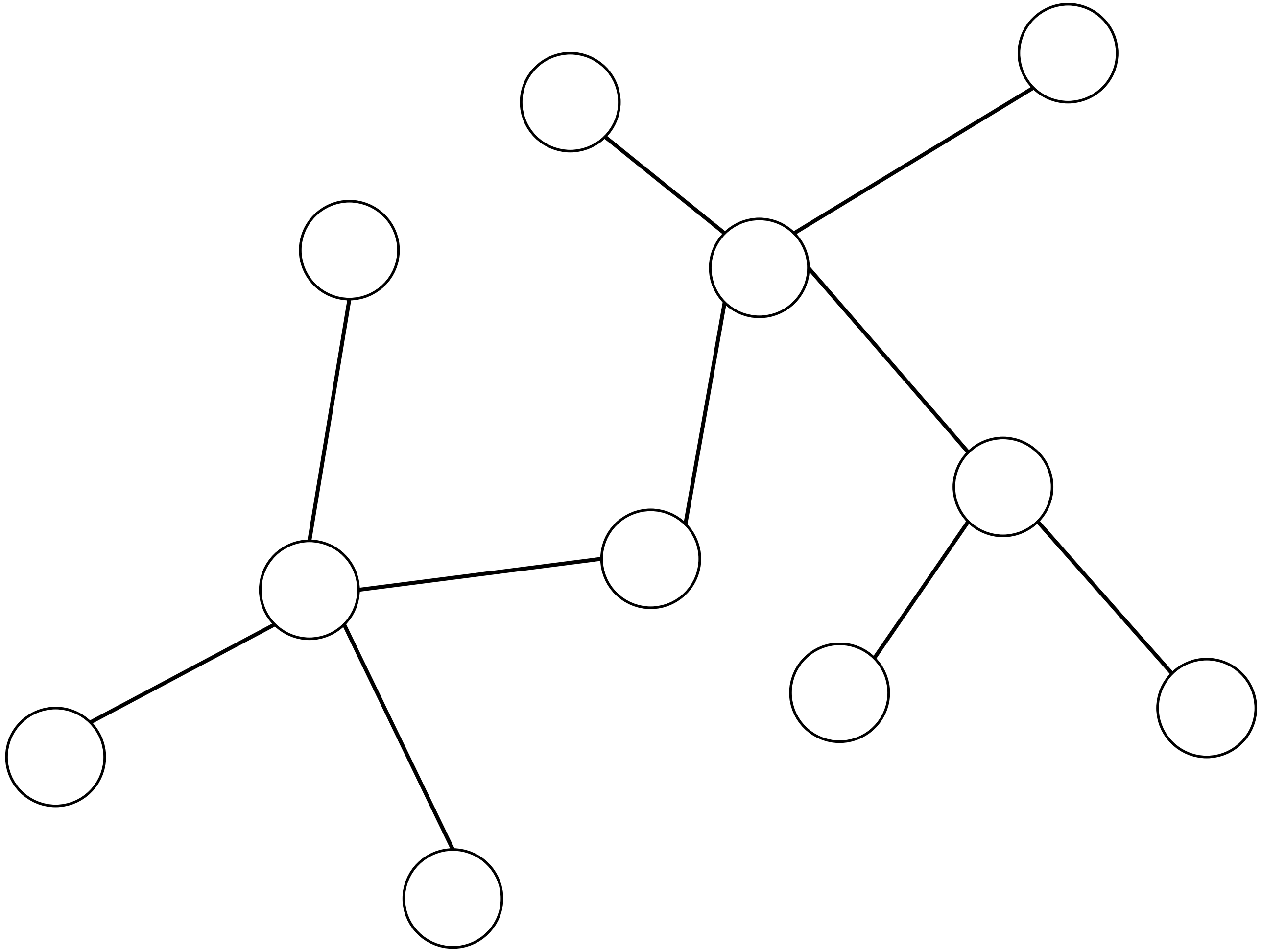




Prim Algorithm

- 아이디어
  - 하나의 노드를 가진 트리에서 시작 (아무 노드나)
  - 트리에 인접한 에지들 중 가장 작은 웨이트를 가진 에지를 추가
    - 단, 사이클을 만들지 않아야 함
  - Spanning Tree가 될때 까지 반복

정확성



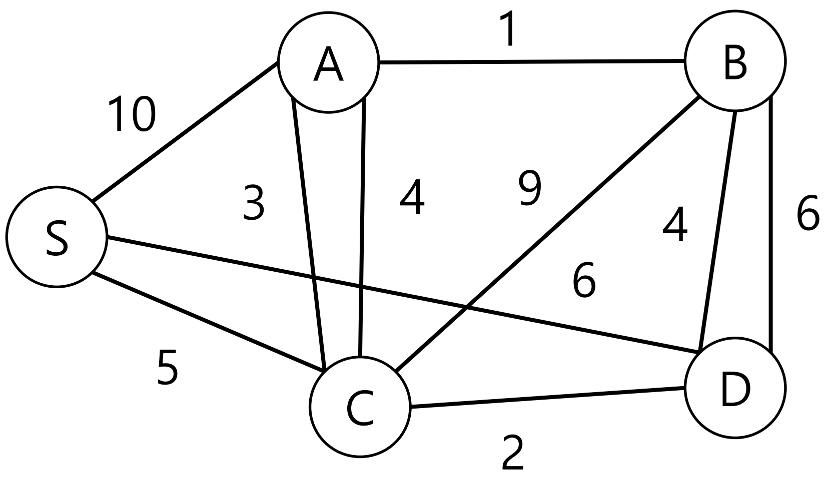
구현 및 성능

- Algorithm Prim(G, T)
  - T <- Empty Set
  - U <- {1}
  - While U != V do
    - U의 한 노드와 V-U의 한 노드를 잇는 에지들 중 웨이트가 제일 작은 것 uv
    - uv 를 T에 추가
    - v 를 U에 추가

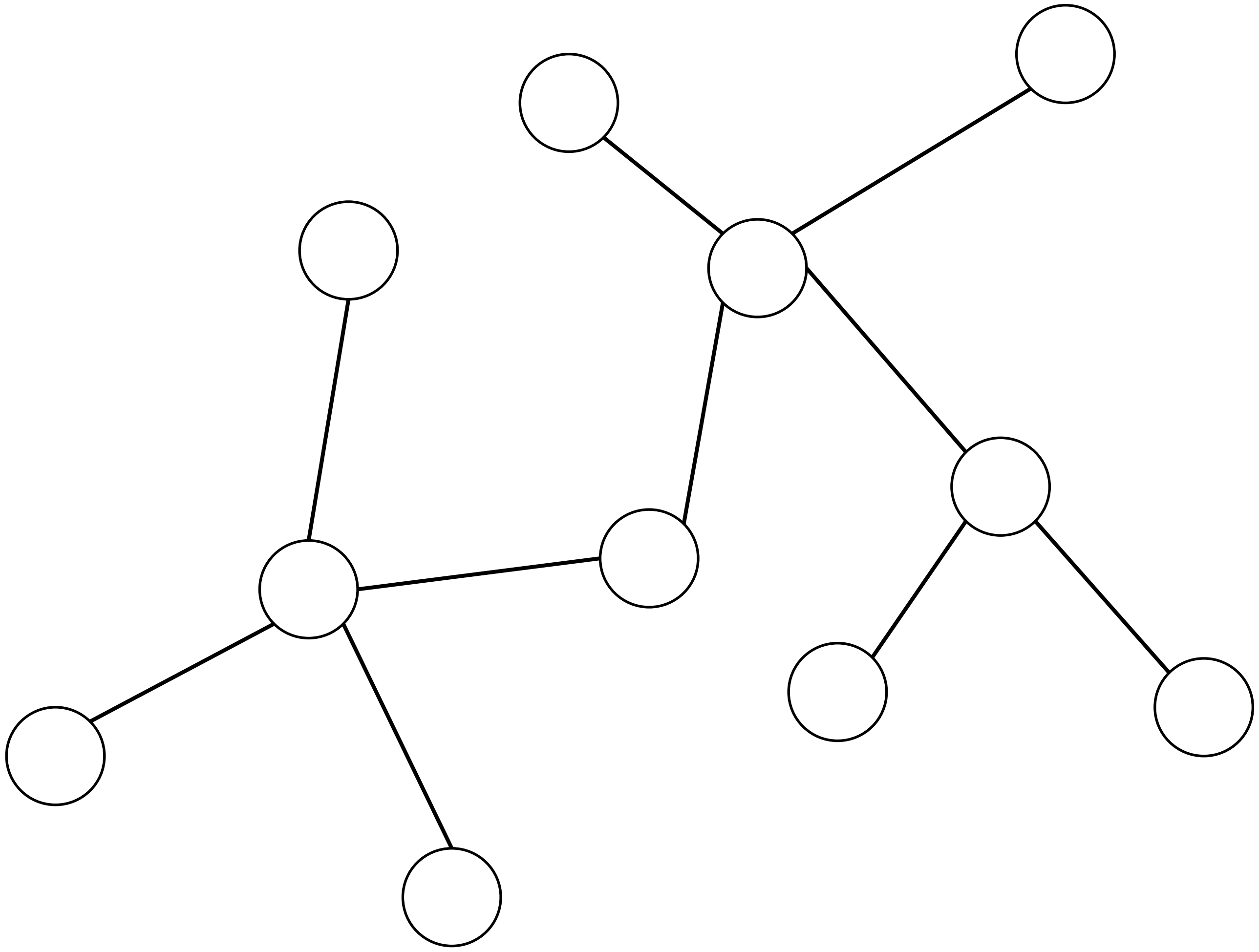
구현 및 성능

Kruskal Algorithm

- Keep adding Edges
  - From smaller weights
  - As long as no Cycle results
  - Until N-1 Edges are added



정확성



구현 및 성능

- Algorithm Kruskal(G, T)
  - T <- Empty Set
  - While |T| < n-1 do
    - E에서 가장 작은 weigh인 에지 e 선택
    - E = E - {e}
    - If (V, T ∪ {e})에 사이클이 없으면, T <- T ∪ {e}

구현 및 성능

Prim and Kruskal can find ANY Solution

- Stable Sort Example
  - Some algorithms cannot find some of the possible solutions

- Prim and Kruskal can find any solution

- Why?

Proof

- Fix any solution T_{mst}
- Show Prim can find THAT solution

Is that Property Important?

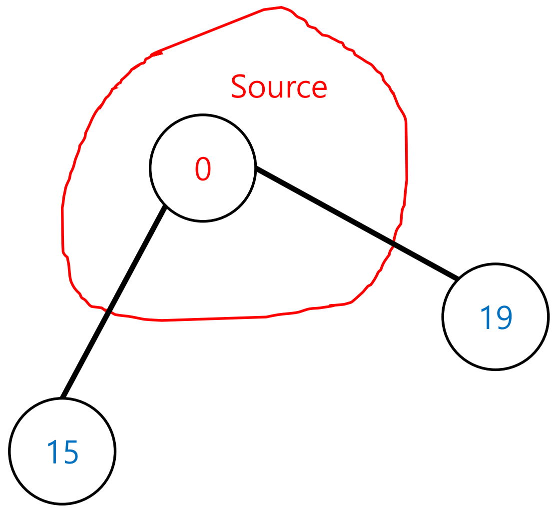
- Can be used to show that if all weights are different there is exactly one solution to the MST problem

Shortest Path

- Dijkstra Algorithm
- Versions of Dijkstra
  - Only Shortest Path Length for each Node
  - Actual Path for each Node also
  - All Possible Shortest Paths for each Node
- Alternate Explanations
  - As a Variation of BFS
  - Select from Nearest to Furthest

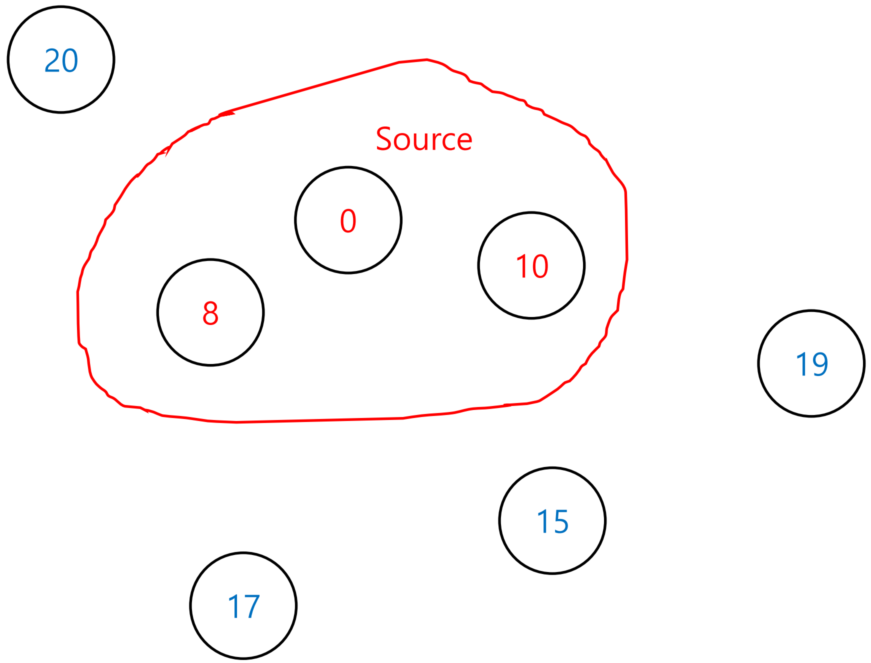
Dijkstra (Only Length)

- Algorithm Dijkstra(G, n, v_0, d_{min})
  - For u ∈ V do d_{min}(v) <- g(v_0, u)
  - R <- {v_0} // d_{min}(v) is Red for Red Nodes
  - While |R| < n do
    - Among nodes not in R, find u with smallest d_{min}(u)
    - R <- R ∪ {u}
    - For w not∈ R do d_{min}(w) <- min[d_{min}(w), d_{min}(u)+g(u, w)]
    - // d_{min}(w) is Blue for Blue Nodes



Dijkstra 진행

빨간 숫자는 최종 정답

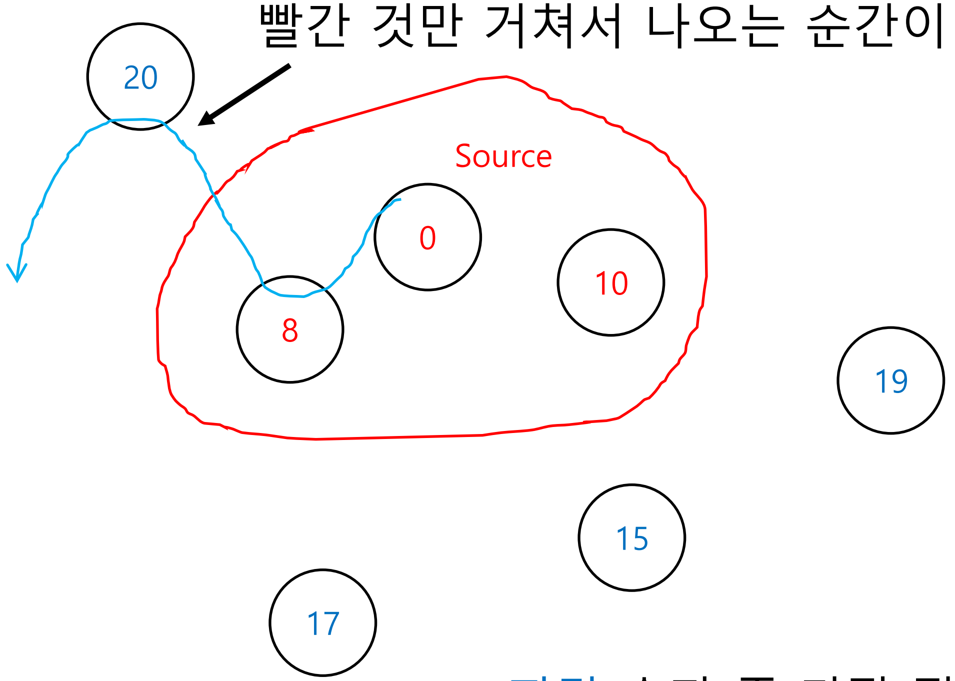


파란 숫자는:
빨간 노드들만 거쳐가는 가장 짧은 길의 길이

Dijkstra 진행

빨간 숫자는 최종 정답

빨간 것만 거쳐서 나오는 순간이 있어야 함

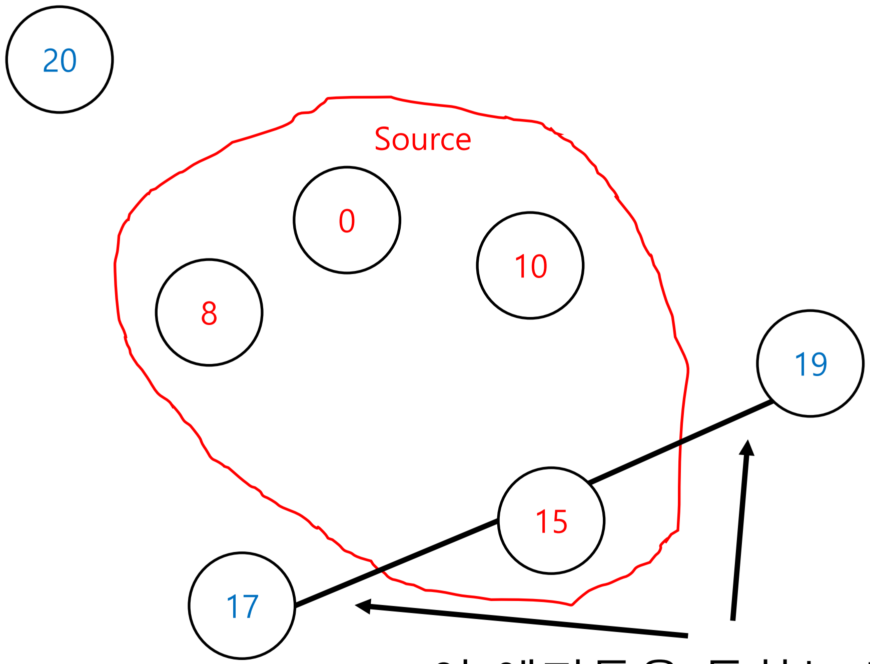


파란 숫자 중 가장 작은 값은 정답이다!

파란 숫자는:
빨간 노드들만 거쳐가는 가장 짧은 길의 길이

Dijkstra 진행

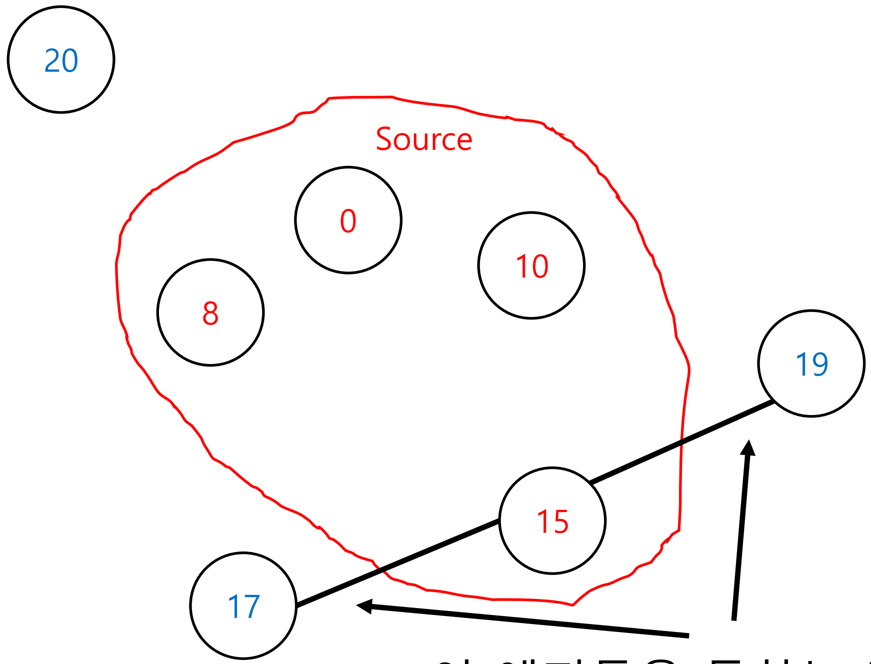
빨간 숫자는 최종 정답



이 에지들을 통하는 것이 더 좋다면 update!!!

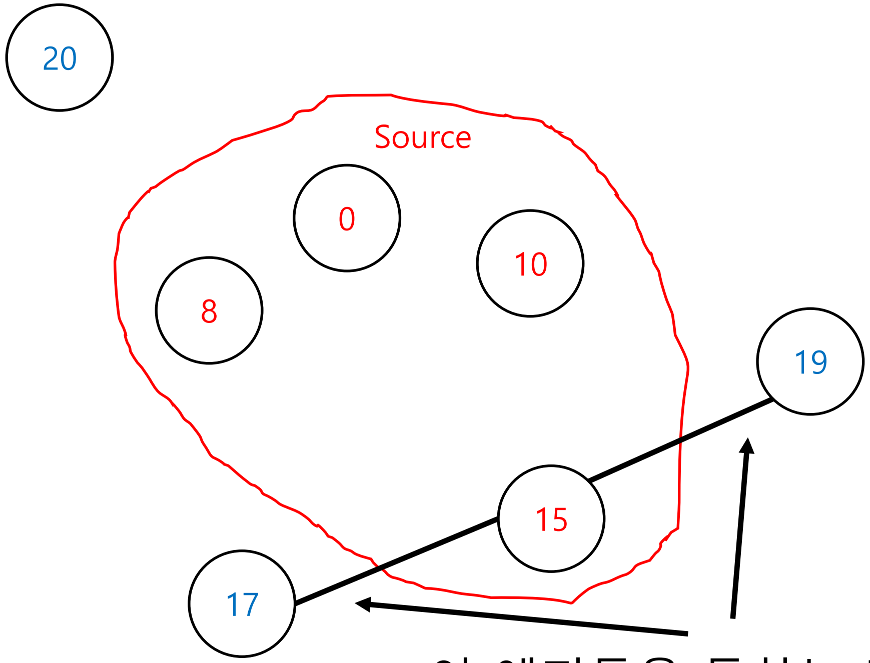
파란 숫자는:
빨간 노드들만 거쳐가는 가장 짧은 길의 길이

Find Actual Path



이 에지들을 통하는 것이 더 좋다면
update!!!

Find All Paths



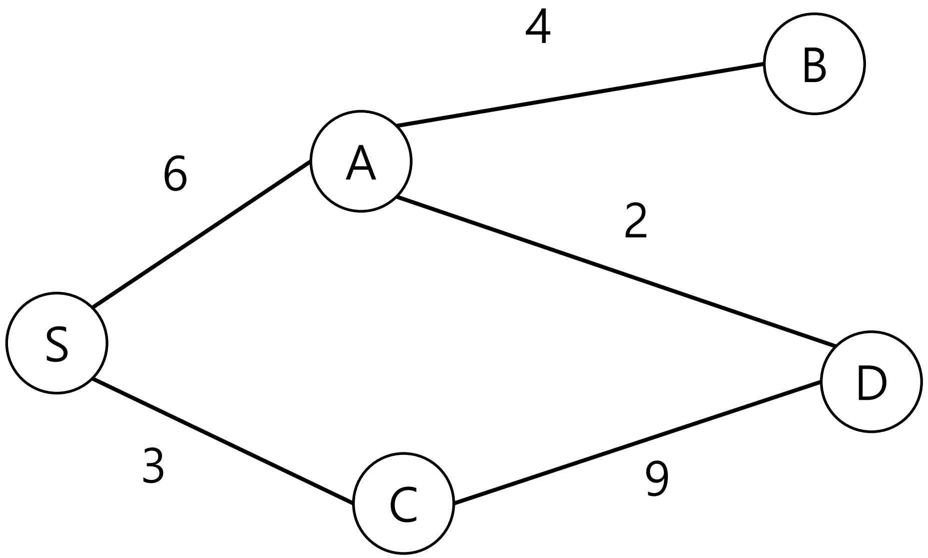
이 에지들을 통하는 것이 더 좋다면
update!!!

Implementation

- Code Fairly Similar to Prim's Algorithm
- Algorithm Dijkstra(G, n, v_0, d_{min})
  - For u $\in$ V do d_{min}(v) <- g(v_0, u)
  - R <- {v_0} // d_{min}(v) is Red for Red Nodes
  - While |R| < n do
    - Among nodes not in R, find u with smallest d_{min}(u)
    - R <- R $\cup$ {u}
    - For w not$\in$ R do d_{min}(w) <- min[d_{min}(w), d_{min}(u)+g(u, w)]
    -      // d_{min}(w) is Blue for Blue Nodes

Alternate Explanations

- From BFS

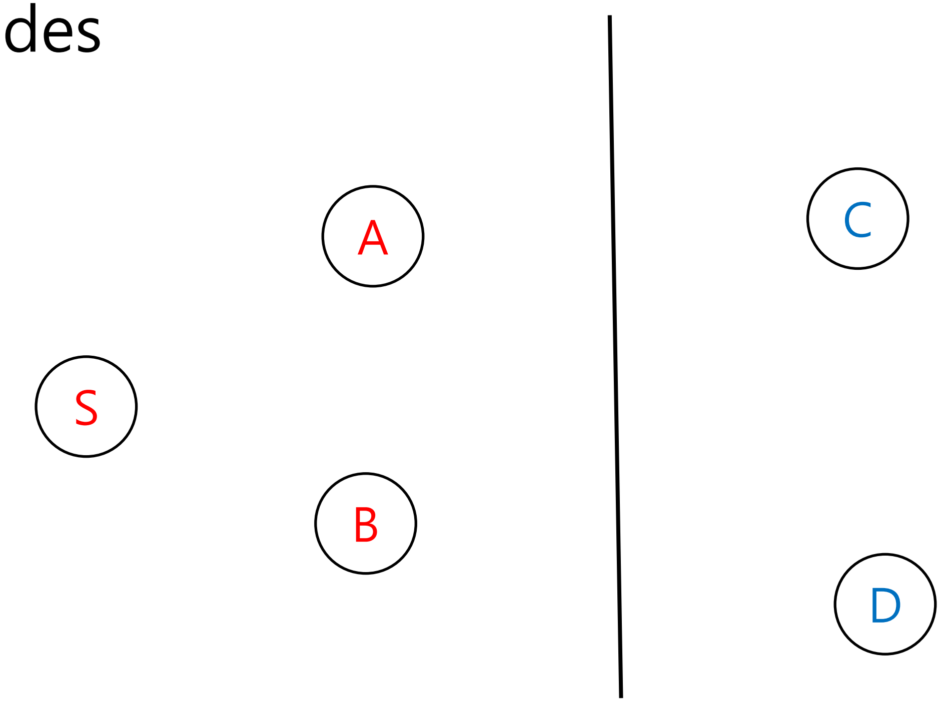


Incidentally

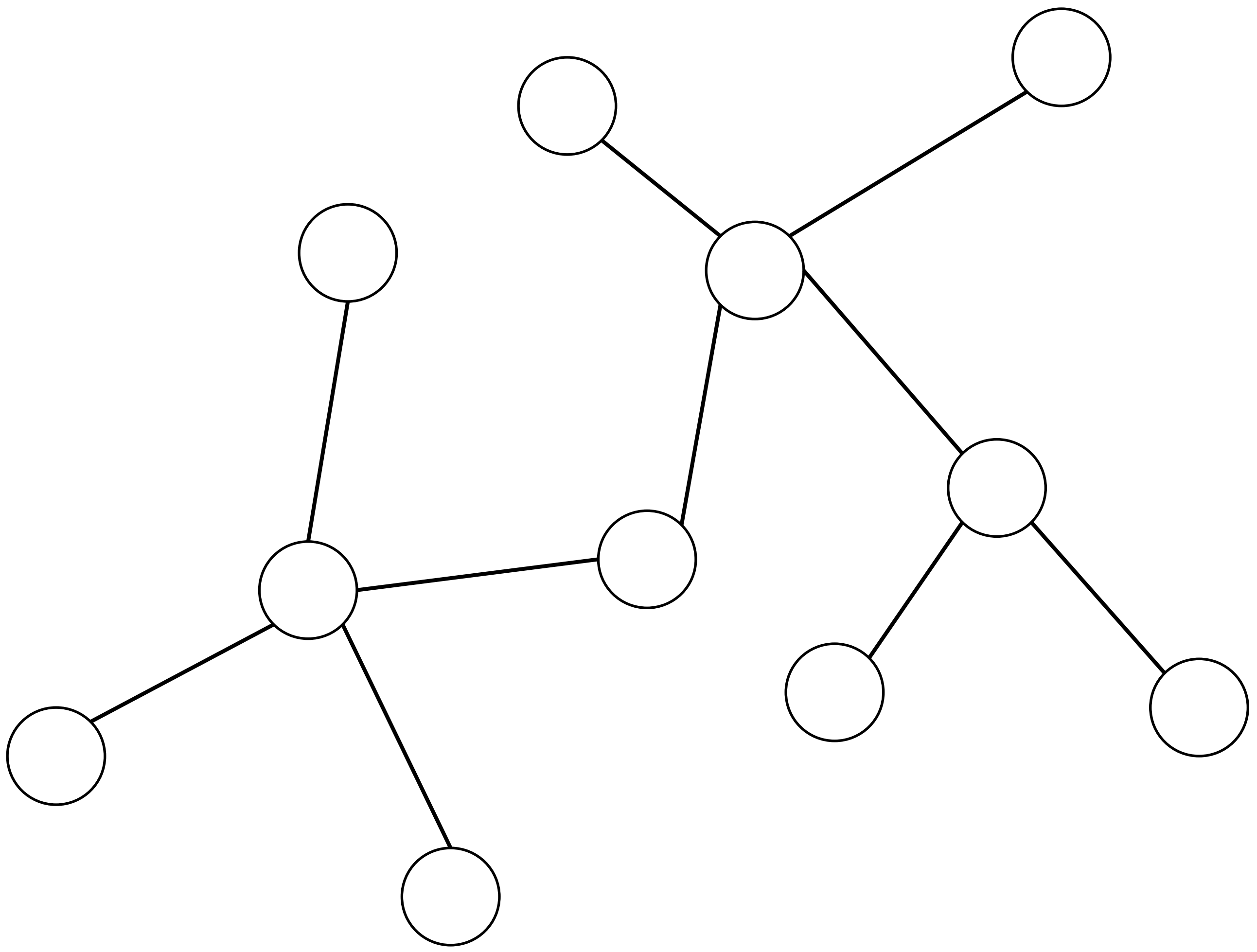
- Dijkstra Selects Node from Nearest (Smaller Shortest Path) to Furthest
- Why?

Alternate Explanations

- Finalize Node from Nearest to Furthest
  - The next nearest Node is directly connected to one of the known nodes

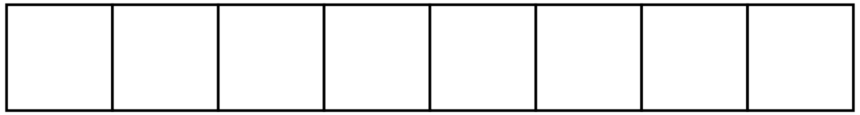


Prim vs. Dijkstra



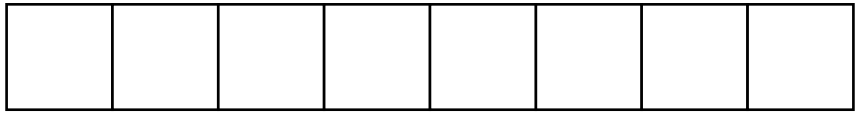
Deadline Scheduling

- Problem Definition
  - N Jobs, J_1, J_2, …, J_N
  - Each J_i = (D_i, P_i)
    - D_i: Deadline
    - P_i: Profit
  - There are 1-Hour Time Slots where you can schedule jobs
  - Each job takes 1 hour to finish
- Example: {(2, 2), (1, 3), (1, 1)}



Assumptions

- All deadlines are ≤ N
  - Why?
- Jobs are given in nonincreasing order of profits
  - Why?
- In an optimal Schedule, no job appears after its Deadline
  - Why?



Intuition/Algorithm

- Schedule Higher Profit First
- If there are Slots available for Current Job, Schedule as late as possible

Correctness

- Let's call the Algorithms (partial) Schedule A
- Invariant: After Deciding J_i, There is at least one optimal solution S such that the followings are true
  1. For $j \leq i$, J_j appears in A iff J_i appears in S
  2. Further, such J_j appears at the same slot in A and S

Proof of Invariant

- Base) $i = 0$, Vacuously True
- Step) Assume Invariant is True for i, Prove for i+1

Performance

- If you do the scheduling naively, it takes $O(N^2)$ time
- Use Balanced Tree
  - Insert every Slot initially
  - At each step with deadline D_i, query the Tree to find
    - "Maximum value in Tree less than or equal to Di"
  - Delete the Slot from Tree if scheduled
- This results in $O(N\log N)$ time

Another Job Scheduling Problem

- Problem Definition
  - N Jobs, J_1, J_2, …, J_N
  - Each J_i = (S_i, T_i)
    - S_i: Start Time
    - T_i: End Time
  - No Two Jobs Can Run Simultaneously

Solution

- Sort by End Time
- Going through Jobs, schedule if possible.
- That is, greedily schedule earliest-ending Job
- Proof?

Tape Storage

- Problem Definition
  - N Data Items, Parameters L_i, F_i
    - L_i: Size of data = Length on Tape
    - F_i: Frequency of Usage
- Read and Write Data on a Tape
  - Write once, everything
  - Read many times
  - EACH READ STARTS from the BEGINNING of Tape

Algorithm

- Just Store the data in the decreasing order of F_i/L_i

- Larger F_i -> Better to be in Front of Tape
- Smaller L_i -> Better to be in Front of Tape

- But Why F_i/L_i ? How about F_i^2/L_i^2, F_i-L_i, ...?

Correctness

- Assume $F_i/L_i < F_{i+1}/L_{i+1}$
- We can prove that this is not optimal

Performance

- Sort once so O(NlogN)