Algorithm, Design & Analysis

Lecture 18: Dijkstra's Algorithm

Muhammad Bilal & Muhammad Asharib

Information Technology University

March 27, 2025

About Your Fellows

- Hi there! We are **Asharib** and **Bilal**.
- We are Associate Students at ITU.

Introduction to Dijkstra's Algorithm

- Used to find the shortest path from a single source to all vertices or a specified vertex in a weighted graph.
- Using a priority queue (min-heap) for efficient vertex selection.
- Decrease-Key heap operation is used to maintain an optimized priority queue.

> Dijkstra works like a miser traveling the world—always taking the cheapest step forward to minimize cost.

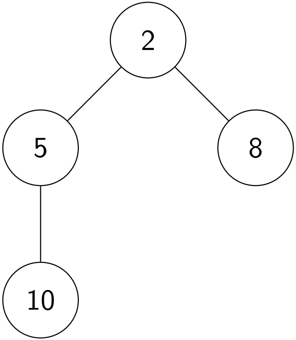
What is Decrease Key

Heap Operation – Decrease-Key

Definition:

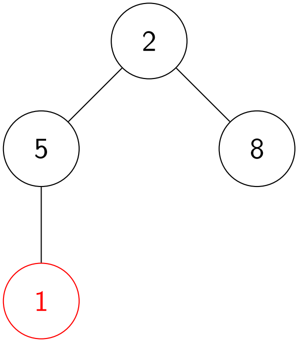
- The decrease-key operation is used when an edge relaxation lowers the shortest known distance of a vertex.
- Helps update distances and maintains the heap property by sifting up updated elements.

Step 1: Initial Min-Heap



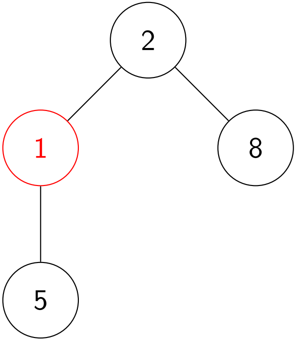
Before decrease-key: The key at node 10 is reduced to 1.

Step 2: Update Key (10 $\rightarrow$ 1)



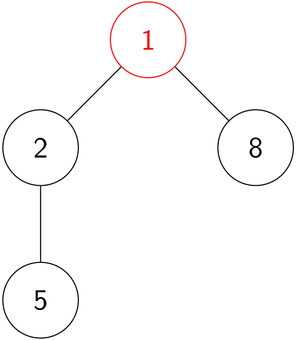
Key changed: Node 10 is now 1. Heap property is violated.

Step 3: Sift-Up (Swap 1 , 5)



Step: Node 1 swaps with its parent (5).

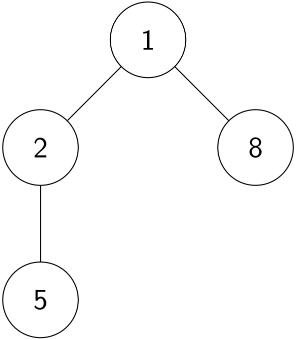
Step 4: Sift-Up (Swap 1 , 2)



Step: Node 1 swaps with its parent (2).

Min-Heap property restored.

Final Heap After Decrease-Key



Final Structure: Min-Heap is now valid!

Time Complexity Analysis

Time Complexity for Binary Heap

- **Insert:** $O(\log N)$
- **Delete Min:** $O(\log N)$
- **Decrease Key:** $O(\log N)$

Back to Dijkstra's Algorithm

Algorithm Explanation

Initialization:

- Set all distances except the source to infinity (∞).
- Set the distance from the source to 0.
- Store all vertices in a min-heap based on their current shortest distances.

Processing Nodes (Main Loop):

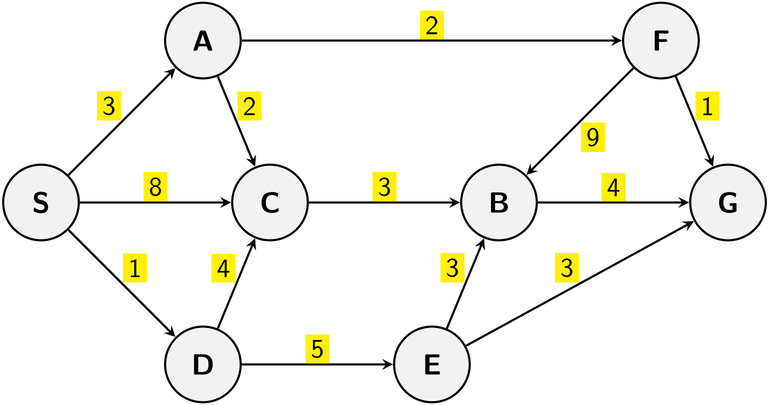
- Extract the vertex u with the smallest distance from the heap.
- Relax all its neighbors v:
  - If a shorter path to v is found through u, update $d[v]$.
  - Update the predecessor of v to track the shortest path.
  - The decrease-key operation is used to update the heap efficiently.

Pseudocode

Dijkstra's Algorithm:

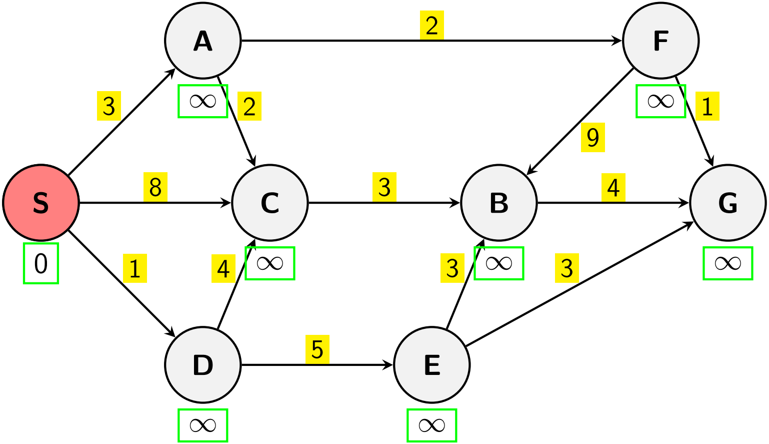
- $d[v] = \infty$ for all V.
- $\pi[v] =$ NIL for all V.
- $d[s] = \phi$
- H = Insert all vertices into a min-heap with their distances.
- For i = 1 to n:
  - u = $deleteMin(H)$
  - For all edges (u, v):
    - If $d[u] + w(u, v) < d[v]$:
    - $d[v] = d[u] + w(u, v)$.
    - $\pi[v] = u$.
    - $decreaseKey(H, v, d[v])$

Dijkstra's Algorithm - Example



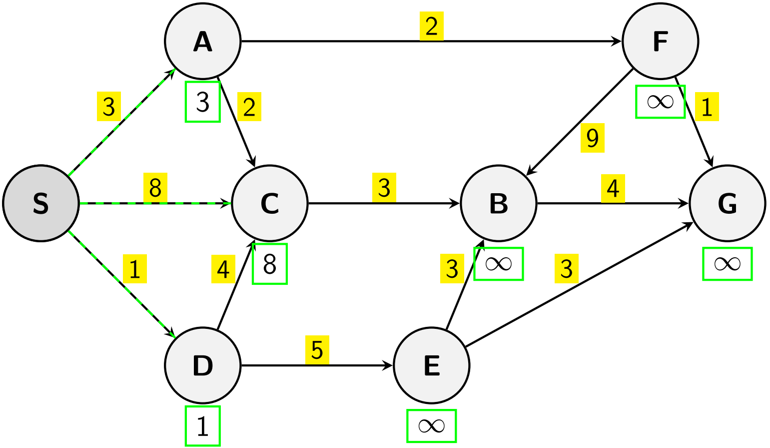
Example - Step by Step

Step 1: Starting with S ;



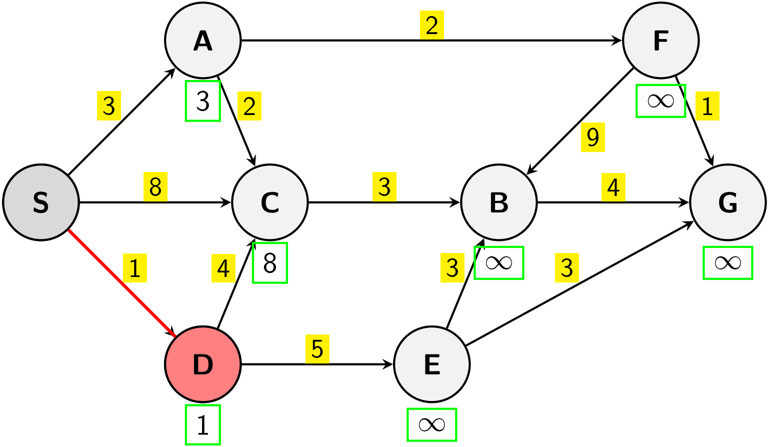
Example - Step by Step

Step 2: Updating Neighbors of S ;



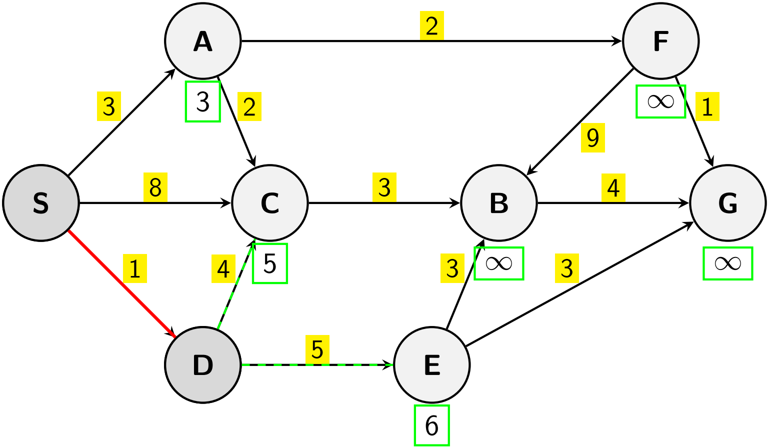
Example - Step by Step

Step 3: Selecting D(Smallest Distance) ;



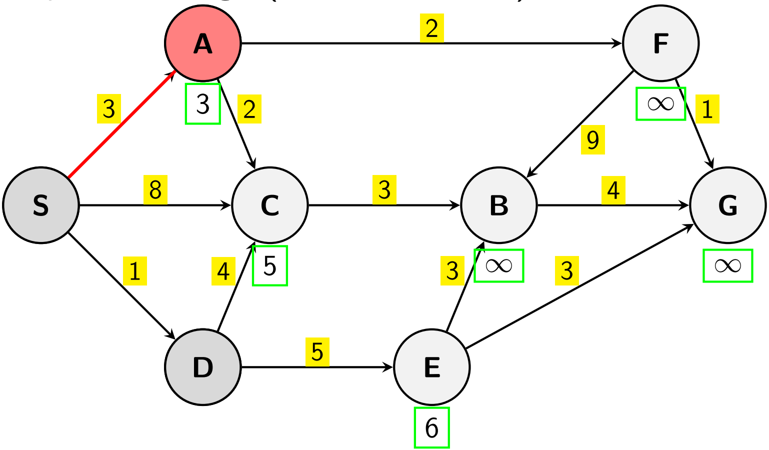
Example - Step by Step

Step 4: Updating Neighbors of D ;



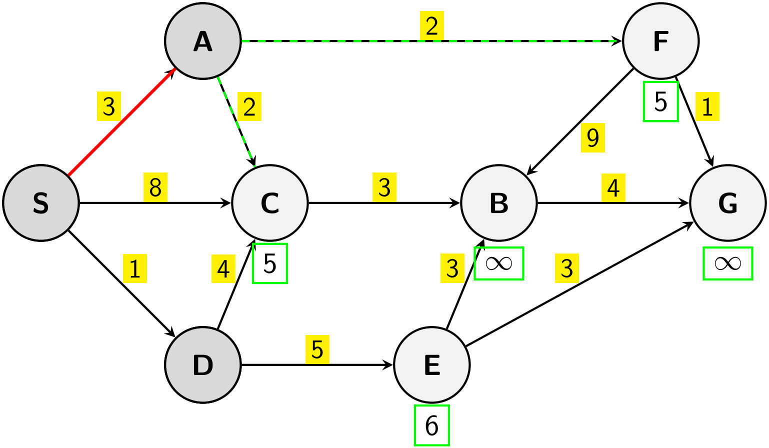
Example - Step by Step

Step 5: Selecting A(Smallest Distance) ;



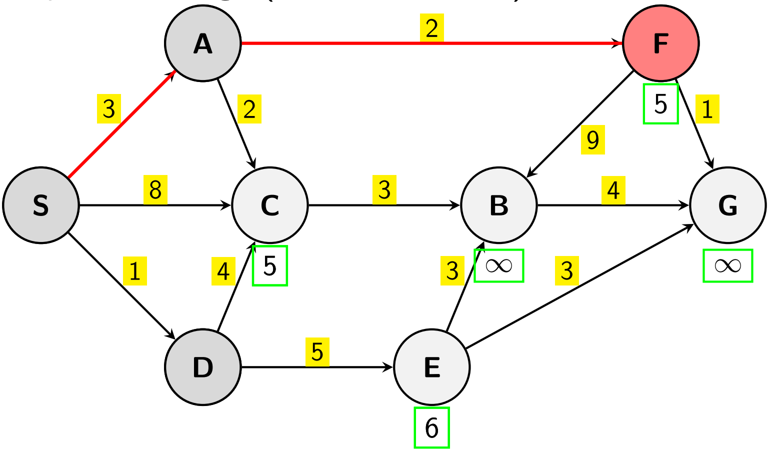
Example - Step by Step

Step 6: Updating Neighbors of A ;



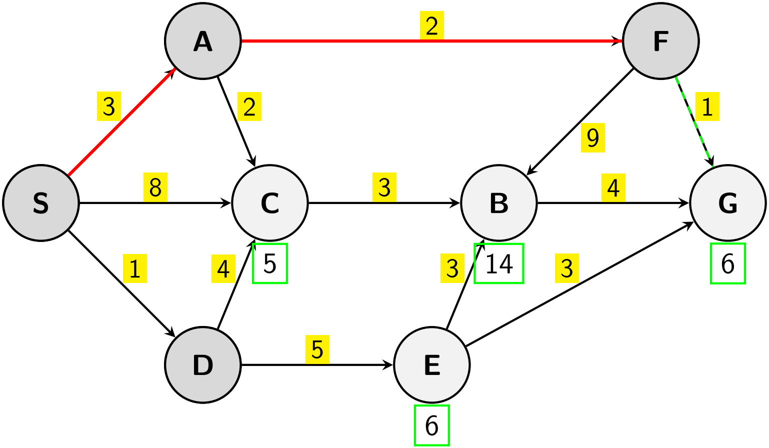
Example - Step by Step

Step 7: Selecting F(Smallest Distance) ;



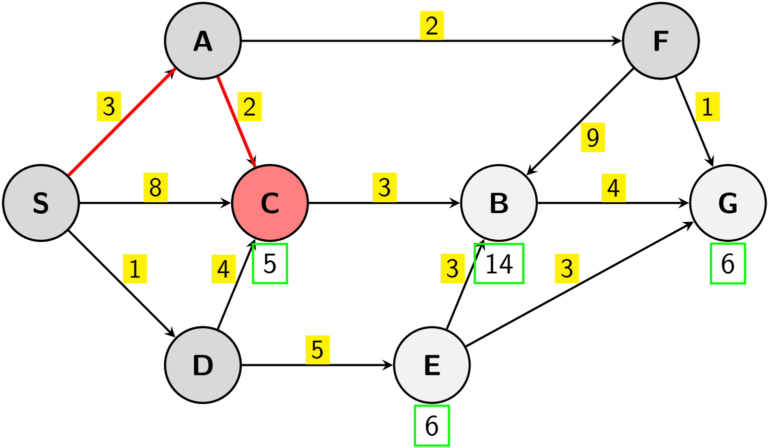
Example - Step by Step

Step 8: Updating Neighbors of F ;

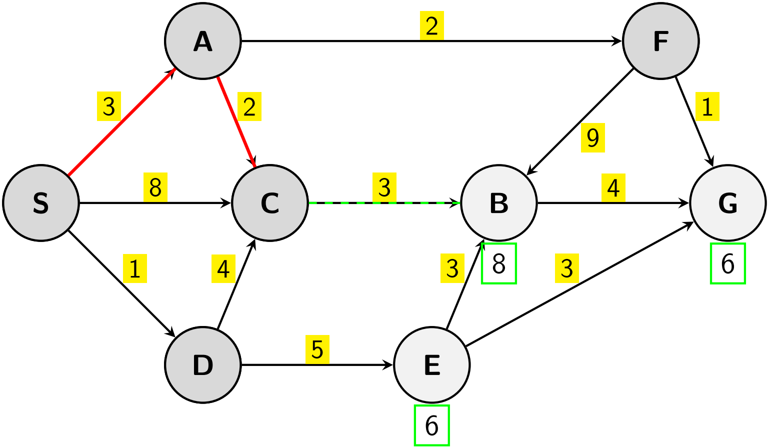


Example - Step by Step

Step 9: Selecting C(Smallest Distance) ;

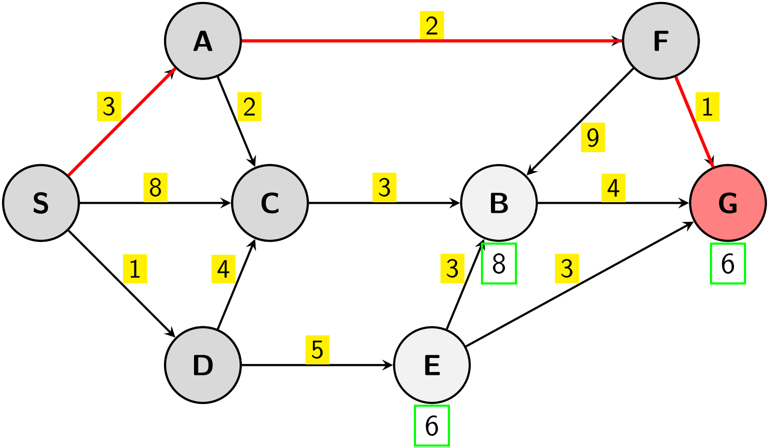


Step 10: Updating Neighbors of C ;



Example - Step by Step

Step 11: Selecting G :Smallest Distance(Goal Reached!) ;



Time Complexity Analysis

Time Complexity of Dijkstra's Algorithm

- **Using Binary Heap:** $O((V + E \log V)$
- **Using Fibonacci Heap:** $O(V + E)$
- **Why Fibonacci Heap?** - Improves performance by making decrease-key $O(1)$ amortized time.

Understanding of Fabonacci Heap & Amortized Cost

Fibonacci Heaps & Efficiency

- Fibonacci heaps allow faster updates by reducing decrease-key cost.

Time Complexity of Decrease-Key

- **Binary Heap:** $O(\log V)$
- **Fibonacci Heap:** $O(1)$ amortized time

- Analogy with binary counting:
  - The cost of incrementing a binary number depends on bit flips.
  - Similar logic applies to heap restructuring.

Amortized Cost

Amortized Cost

The amortized cost is the average cost of each operation in an algorithm when spread over a sequence of operations, even if some are more expensive.It gives a clearer picture of overall efficiency.

Amortized Cost (Binary Counter)

Understanding the Cost of Incrementing a Binary Counter

- Each increment operation flips bits from 0 to 1 or 1 to 0.
- Worst case: All n bits flip (e.g., $1111 \to 0000$).
- m operations can have a complexity of $O(m \cdot n)$.

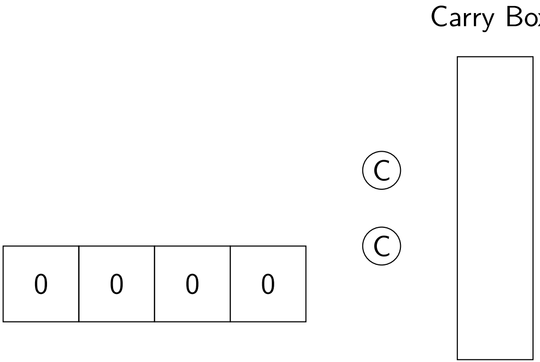
Amortized Analysis Using the Coin Method

Basic Idea:

- Each increment earns 2 coins.
- Flipping 0 to 1 costs 1 coin.
- Flipping 1 to 0 costs 1 coin.
- At the end of increment, the remaining coins move to savings.

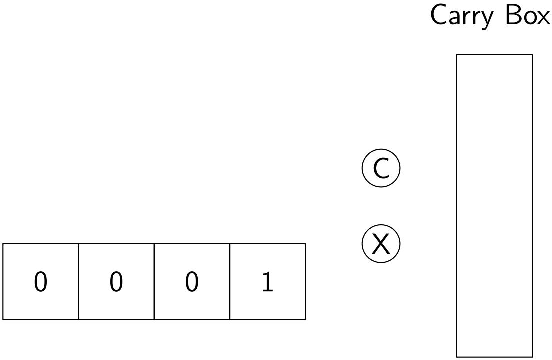
Step 1: Incrementing from 0000 to 0001

- We get 2 coins at start of the incrementing step



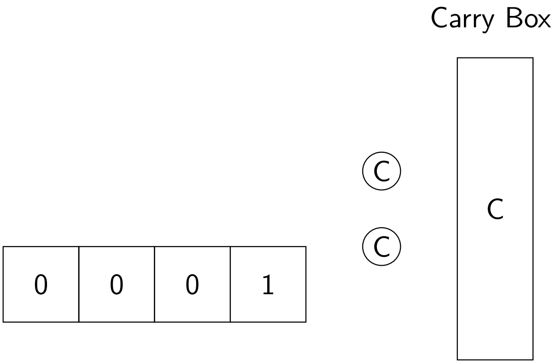
Step 1: Incrementing from 0000 to 0001

- 1 coin is spent on flipping 0 to 1
- The remaining 1 coin will be moved to carry box



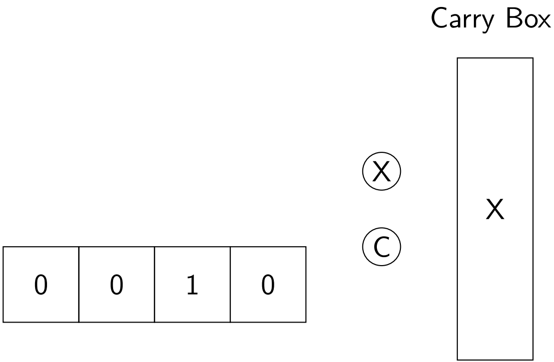
Step 2: Incrementing from 0001 to 0010

- We get 2 coins again



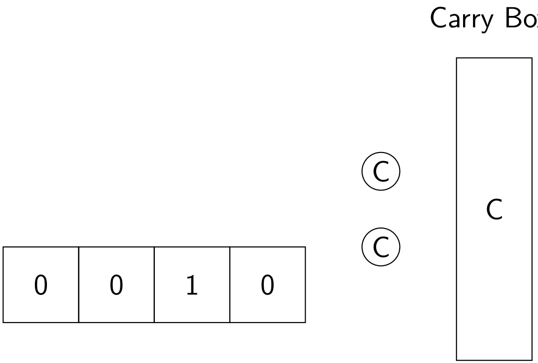
Step 2: Incrementing from 0001 to 0010

- 1 carry coin utilized for flipping 1 to 0
- 1 coin is spent on flipping 0 to 1
- The remaining 1 coin will be moved to carry box



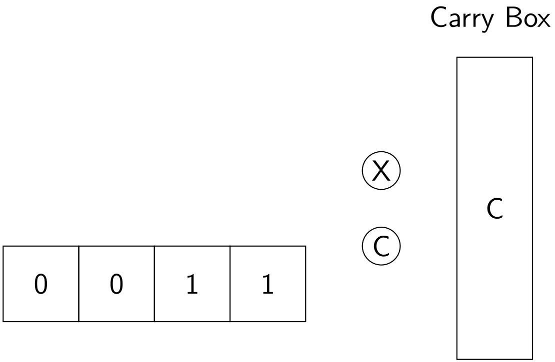
Step 3: Incrementing from 0010 to 0011

- We get 2 coins again



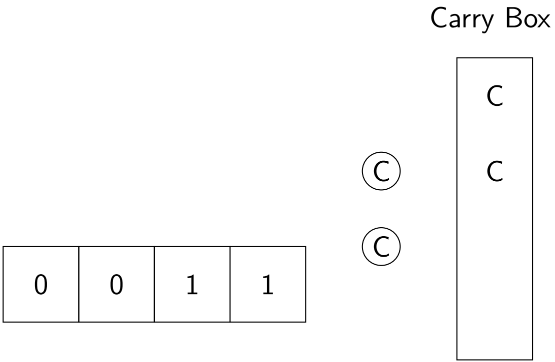
Step 3: Incrementing from 0010 to 0011

- 1 coin is spent on flipping 0 to 1
- The remaining 1 coin will be moved to carry box



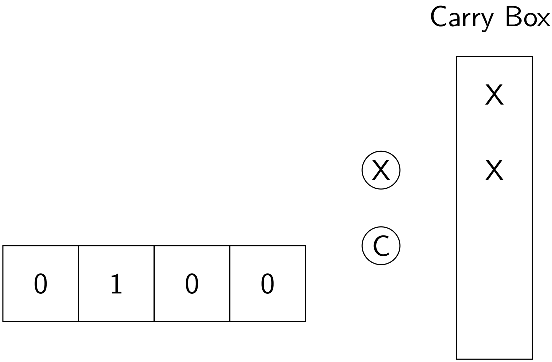
Step 4: Incrementing from 0011 to 0100

- We get 2 coins again



Step 4: Incrementing from 0011 to 0100

- Both carry coins are utilized for flipping 1's to 0's
- 1 coin is spent on flipping 0 to 1
- The remaining 1 coin will be moved to carry box

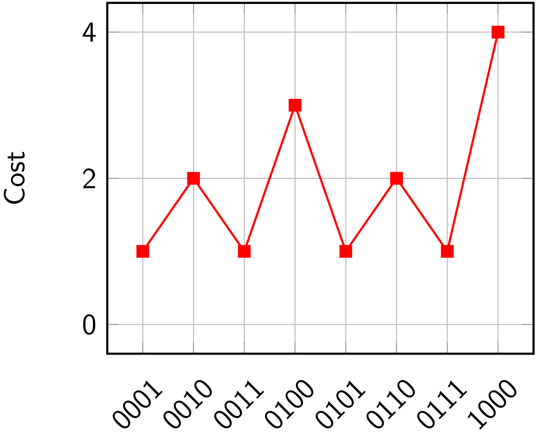


Why This Improves the Cost

- Each bit flip collects coins for future flips.
- When a bit flips, it spends coins collected earlier.
- Each increment operation only requires 1 new coin.
- Even if multiple bits flip, the cost is already covered by previous savings.

> - Total flips in m operations: $O(m)$ instead of $O(m \cdot n)$.
> - Average cost per operation: $O(1)$.

Why This Improves the Cost: Line Graph

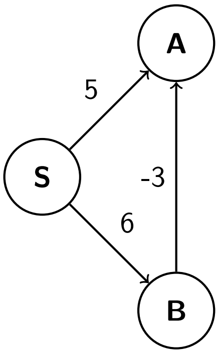


Observation:
The cost is going up and down at each step, Resulting in an amortized cost of $2m = O(m)$ instead of $O(mn)$.

Dijkstra's Limitations

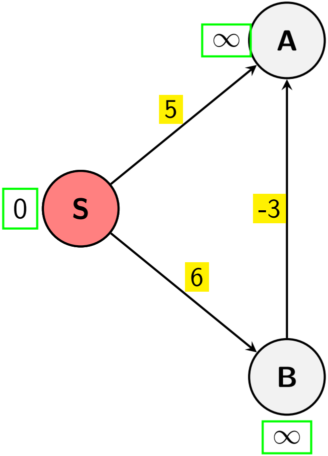
- Can not handle negative weights.

Dijkstra's Limitations



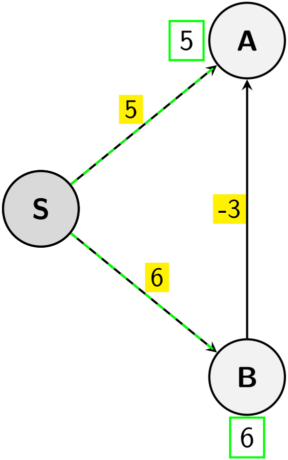
Dijkstra's Limitations

Step 1: Start with S (Goal is A) ;



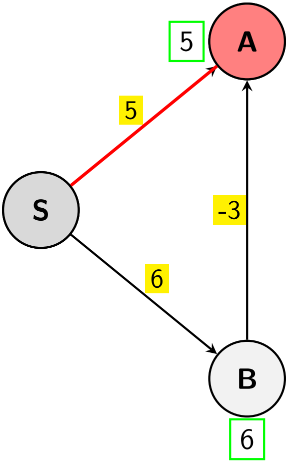
Dijkstra's Limitations

Step 2: Updating Neighbors of S ;

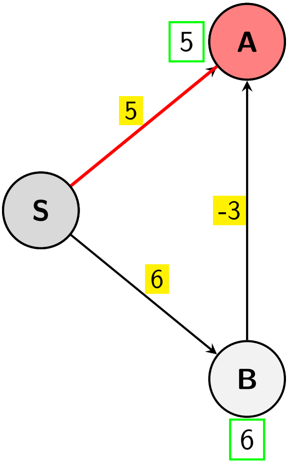


Dijkstra's Limitations

Step 3: Selecting A :Smallest Distance (Goal Reached!) ;

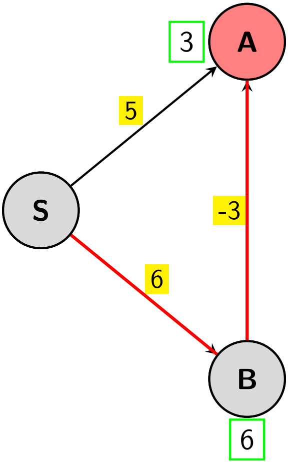


Incorrect Shortest Distance ;



Dijkstra's Limitations

The Real Shortest Distance ;



Why Dijkstra's Algorithm Failed?

- Dijkstra assumes all edge weights are **non-negative**.
- **The negative weight edge** (B $\rightarrow$ A, cost $=$ -3) breaks this assumption.
- Once a node is visited in Dijkstra's algorithm, it is **never reprocessed**, leading to incorrect results.
- In this case, the shortest path S $\rightarrow$ B $\rightarrow$ A (cost$=$3) is **never found**, as Dijkstra **incorrectly finalizes** d(A)$=$5 **too early**.

Applications of Dijkstra's Algorithm

- **Network Routing:** Finding shortest paths in computer networks.
- **GPS & Navigation:** Used in Google Maps to find the shortest routes.
- **Social Networks:** Finding the shortest connection path between users.
- **Airline Route Optimization:** Finding the cheapest travel routes.