Shortest-Path Algorithms
Anatomy of Programming Contest

Mix of problems testing different skills
Anatomy of Programming Contest

Mix of problems testing different skills
• one "easy" question
Anatomy of Programming Contest

Mix of problems testing different skills

• one “easy” question
• one shortest path problem “with a twist”
Anatomy of Programming Contest

Mix of problems testing different skills

- one “easy” question
- one shortest path problem “with a twist”
- one dynamic programming problem
Anatomy of Programming Contest

Mix of problems testing different skills

- one “easy” question
- one shortest path problem “with a twist”
- one dynamic programming problem
- one simulation
Anatomy of Programming Contest

Mix of problems testing different skills
• one “easy” question
• one shortest path problem “with a twist”
• one dynamic programming problem
• one simulation
• one number theory / geometry problem
Anatomy of Programming Contest

Mix of problems testing different skills

- one “easy” question
- one shortest path problem “with a twist”
- one dynamic programming problem
- one simulation
- one number theory / geometry problem
- some “hard” problems
Anatomy of Programming Contest

Mix of problems testing different skills
• one “easy” question
• one shortest path problem
• one dynamic programming problem
• one simulation
• one number theory / geometry problem
• some “hard” problems
Last Time

start state

goal states
What If Edges Have Costs?
What If Edges Have Costs?

Use a (minimum) priority queue to store the next nodes to visit

- cost = length of path from start to node
What If Edges Have Costs?

Use a (minimum) priority queue to store the next nodes to visit

• cost = length of path from start to node

This guarantees that nodes will be visited in order from closest to farthest
What If Edges Have Costs?

- goal states
- in queue
- visited
- current
What If Edges Have Costs?

![Diagram with nodes and edges labeled with numbers, indicating various states like goal states, in queue, visited, and current.]

- **goal states**
- **in queue**
- **visited**
- **current**
What If Edges Have Costs?

Invariant: visited paths are shorter than unvisited paths
What If Edges Have Costs?

Invariant: visited paths are shorter than unvisited paths
What If Edges Have Costs?

Invariant: visited paths are shorter than unvisited paths
General Graphs

- Green circle: start state
- Red circle: goal states
Dijkstra’s Algorithm
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

Invariant: red paths are shortest possible
Dijkstra’s Algorithm

dijkstra(A, B)
for each vertex v:
    v.visited = false;
p_queue Q = {((A,0))};

while(!Q.empty())
    v = Q.pop();
    if(v.node == B) return v.cost;
    if(v.node.visited) continue;
    v.node.visited = true;
    for each neighbor w:
        if(!w.visited)
            Q.push(((w, v.dist + d_{vw}));
    return infinity;
Dijkstra's Algorithm w/ Path

dijkstra(A,B)
for each vertex v:
  v.visited = false;
  v.prev = -1;
p_queue Q = {(A,0,-1)};

while(!Q.empty())
  v = Q.pop();
  if(v.node == B) return v.cost;
  if(v.node.visited) continue;
  v.node.visited = true;
  v.node.prev = v.prev;
  for each neighbor w:
    if(!w.visited)
      Q.push((w, v.dist + d_{vw},
               v.node));
return infinity;
Dijkstra’s Algorithm

What’s the run time?
Dijkstra’s Algorithm

What’s the run time? $O(|E| \log |E|)$
What’s the run time? $O(|E| \log |E|)$

(can be improved to $O(|E| + |V| \log |V|)$)