Motion Planning

Graph Based Methods

Jana Kosecka
Department of Computer Science

- Discrete planning, graph search, shortest path, A* methods
- Road map methods
- Configuration space

Slides thanks to http://cs.cmu.edu/~motionplanning, Jyh-Ming Lien

1

State space

- Set of all possible states is represented as graph
- Nodes states, links possible transitions between the states
- We will consider setting now where
- State space X is finite or countable infinite

- Transition function $x' = f(x, u)$
- Set of action in each state $U(x)$
- Initial and goal state

- Feasible planning – generate a set of actions, if the solution exists it must be found in the finite time
- Search must be systematic, explore unexplored states, keep track what has been visited
- Strategy: used graph search algorithms

2

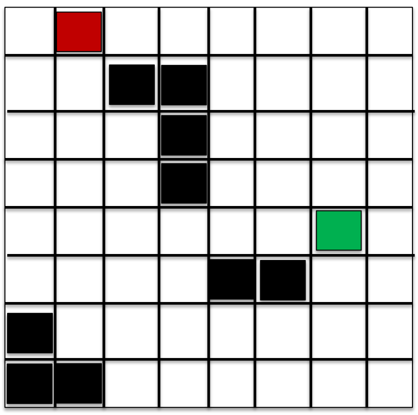
Motion Planning Problem

- Task planning – doing the dishes
- Motion planning problem how to go from pose one to pose two
- Example PACMAC

3

Planning on a grid

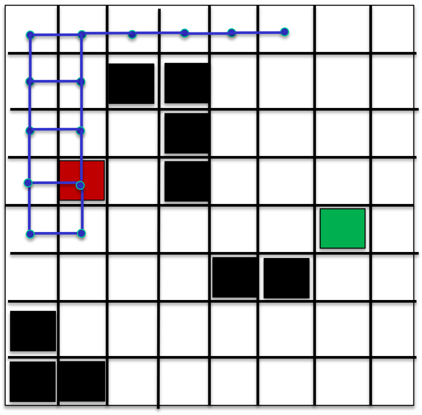
- Robot can move between adjacent cells
- Obstacles - black
- Goal – red
- Start - green



4

Planning on a grid

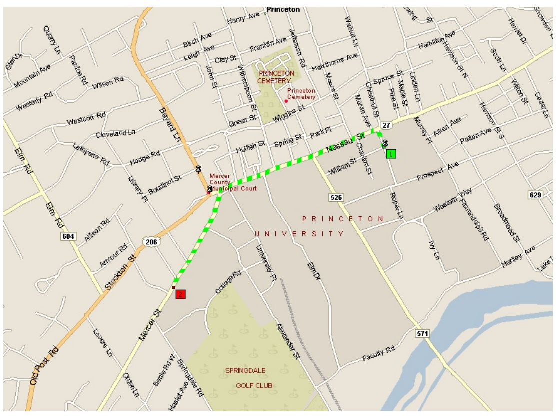
- Robot can move between adjacent cells
- Obstacles - black
- Goal – red
- Start – green
- Graph structure
- Set of nodes V
- Set of edges E
- G = (V, E)



5

Planning on a graph

- G = (V, E)

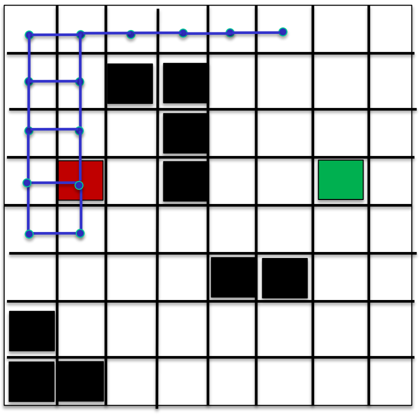


6

Planning on a grid

- Robot can move between adjacent cells
- Obstacles - black
- Goal – red
- Start – green
- Graph structure
- Set of nodes V
- Set of edges E
- G = (V, E)
- Here unit distance

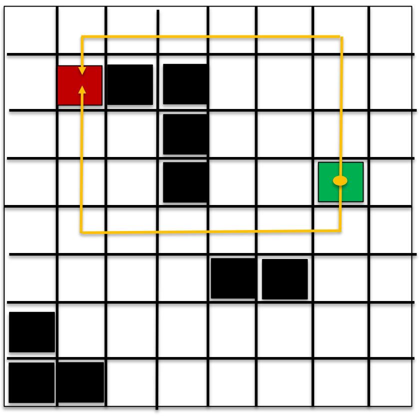
Between the nodes



7

Planning on a grid

- Find a path that minimizes cost
- Minimizes distance
- Shortest path



8

Grassfire Algorithm

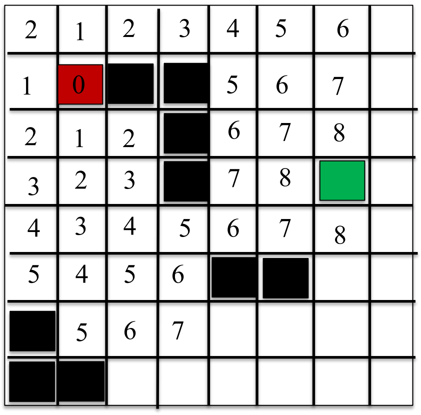
- Mark goal with 0
- Mark all neighbours +1
- Until you reach start node

- Create an empty list

add - goal to the list
While list not empty
Remove the first node
if n.dist = infinity
  n.dist = n.dist +1
add n to the back of the list

- To move to the goal just

move to the node with smallest distance values



9

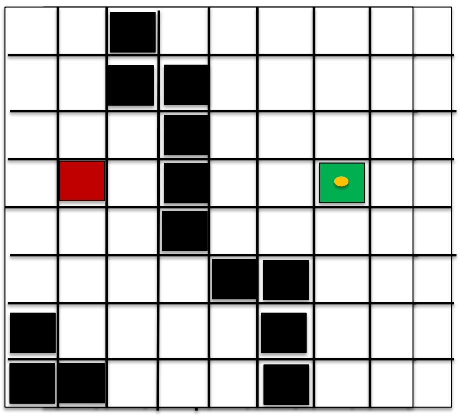
Grassfire Algorithm

- Mark goal with 0
- Mark all neighbours +1
- Until you reach start node

- Another example
- Terminates

- If the start node has not

been found, algorithm terminates.



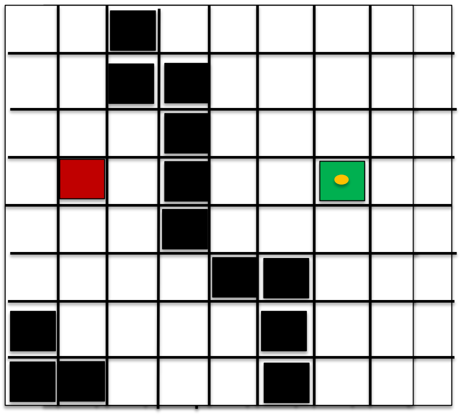
10

Computational Complexity

- Grassfire algorithm
- Complexity linear in the number of nodes

- In 2D grid
- 100x100 = 10^4
- In 3D 100x100x100 = 10^6
- In 6D 10^{12}

- Problem –
- Exponential complexity in terms of number of degrees of freedom



11

Dijkstra's Shortest Path Algorithm

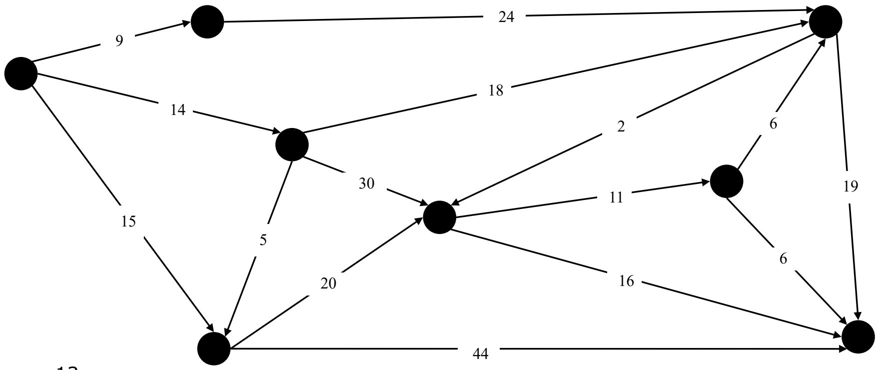
Single shortest path – single destination t (single source)
Given pair of vertices – what is the shortest path from u to v
Positive weights

Example:

12

Dijkstra's Shortest Path Algorithm

- Find shortest path from s to t.



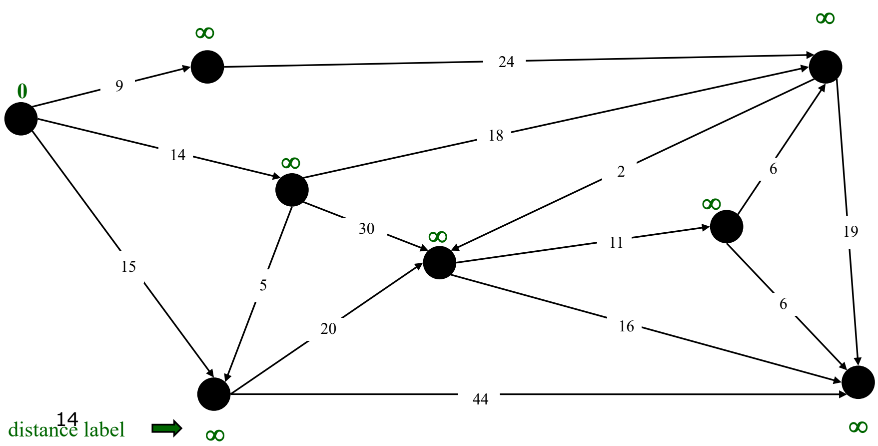
13

13

Dijkstra's Shortest Path Algorithm

S = { }

PQ = { s, 2, 3, 4, 5, 6, 7, t }

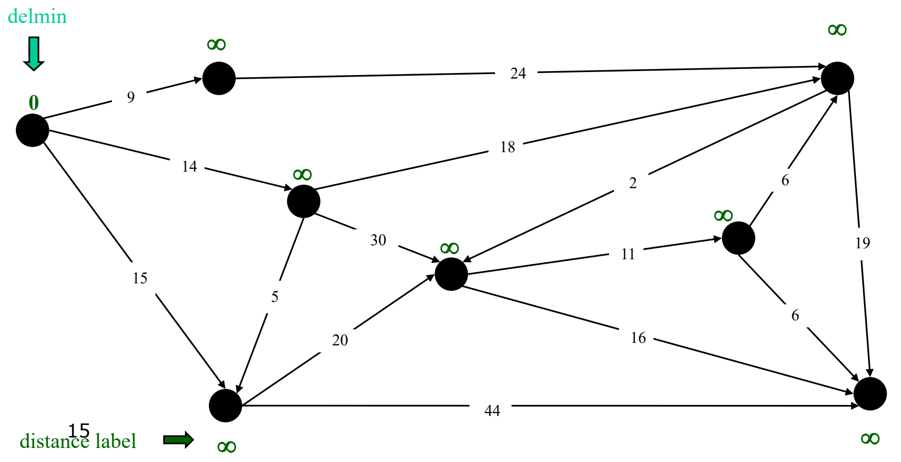


14

Dijkstra's Shortest Path Algorithm

S = { }

PQ = { s, 2, 3, 4, 5, 6, 7, t }

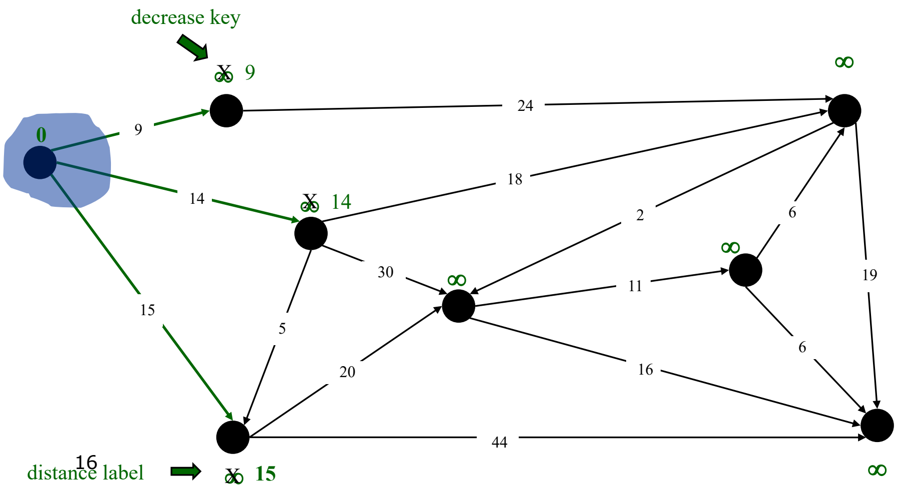


15

Dijkstra's Shortest Path Algorithm

S = { s }

PQ = { 2, 3, 4, 5, 6, 7, t }

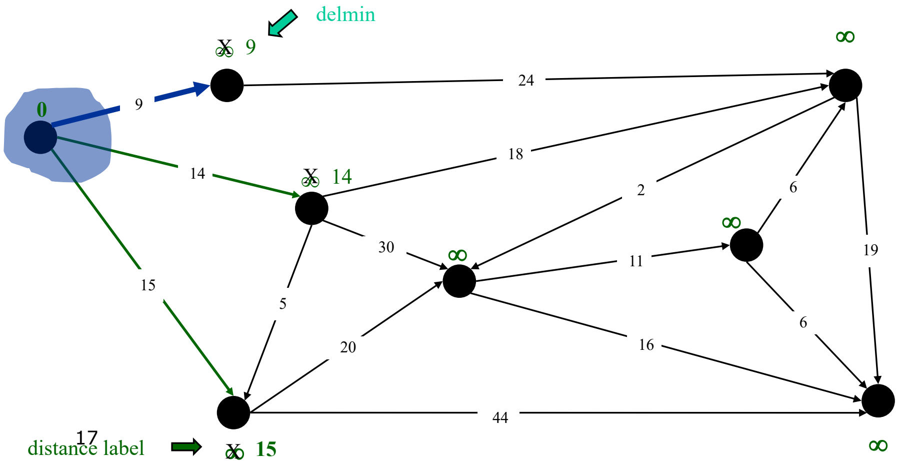


16

Dijkstra's Shortest Path Algorithm

S = { s }

PQ = { 2, 3, 4, 5, 6, 7, t }

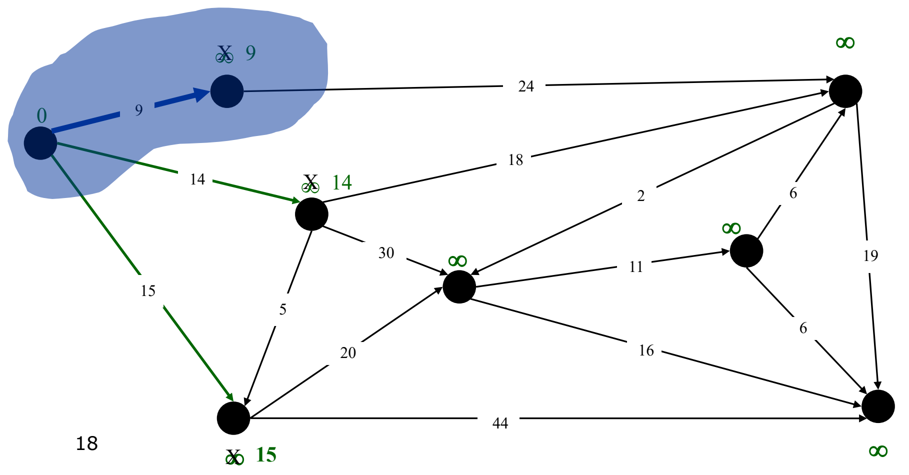


17

Dijkstra's Shortest Path Algorithm

S = { s, 2 }

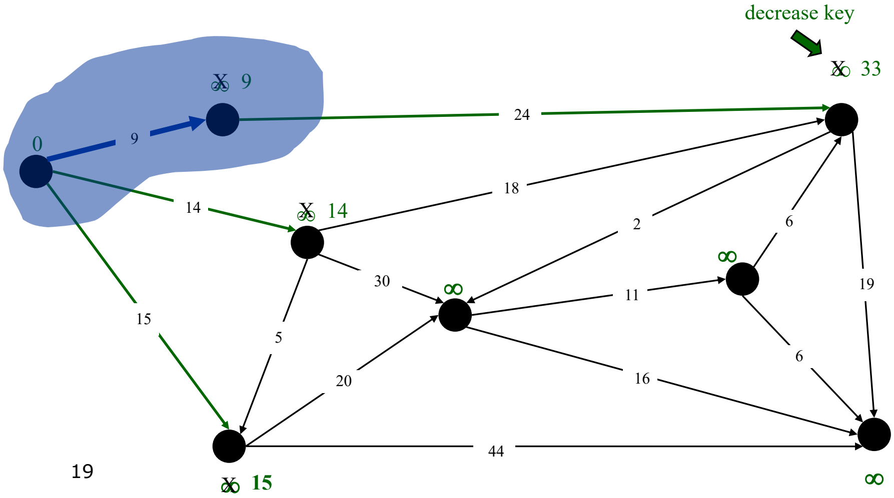
PQ = { 3, 4, 5, 6, 7, t }



18

Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }

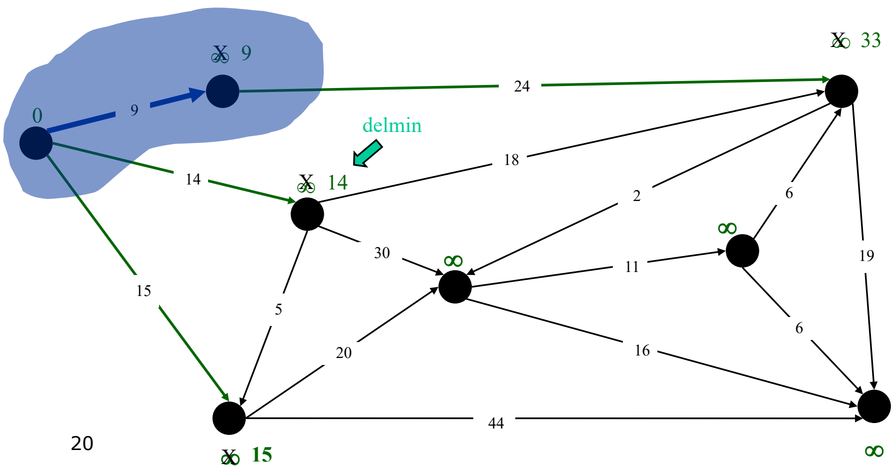


19

19

Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



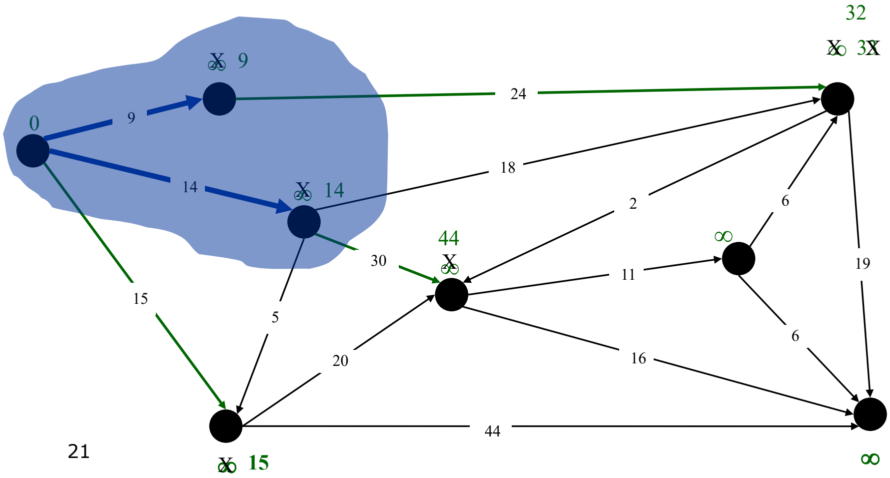
20

20

Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

PQ = { 3, 4, 5, 7, t }



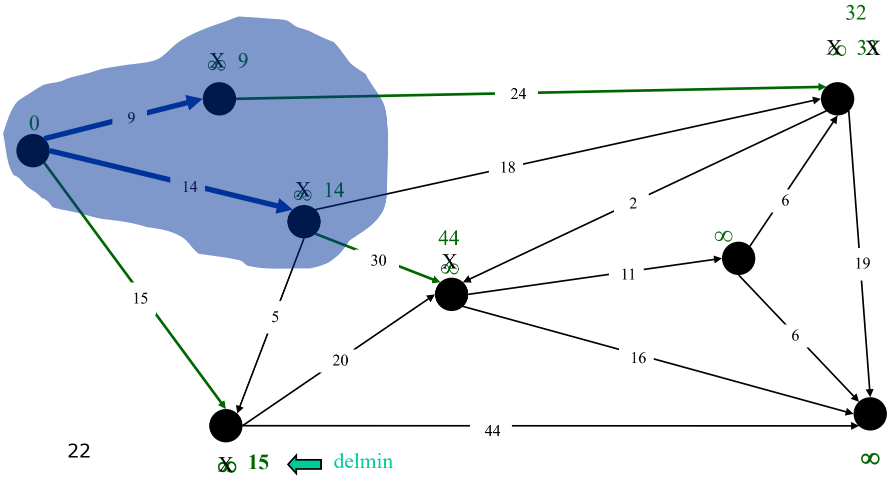
21

21

Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

PQ = { 3, 4, 5, 7, t }



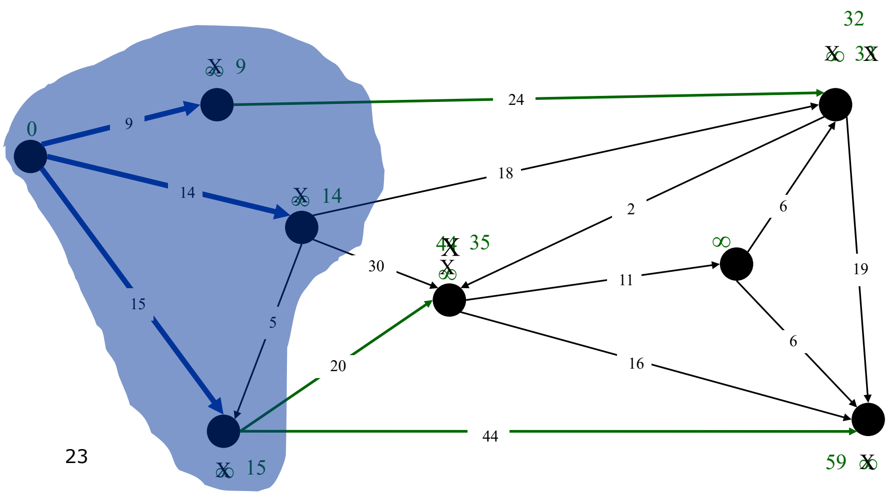
22

22

Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

PQ = { 3, 4, 5, t }



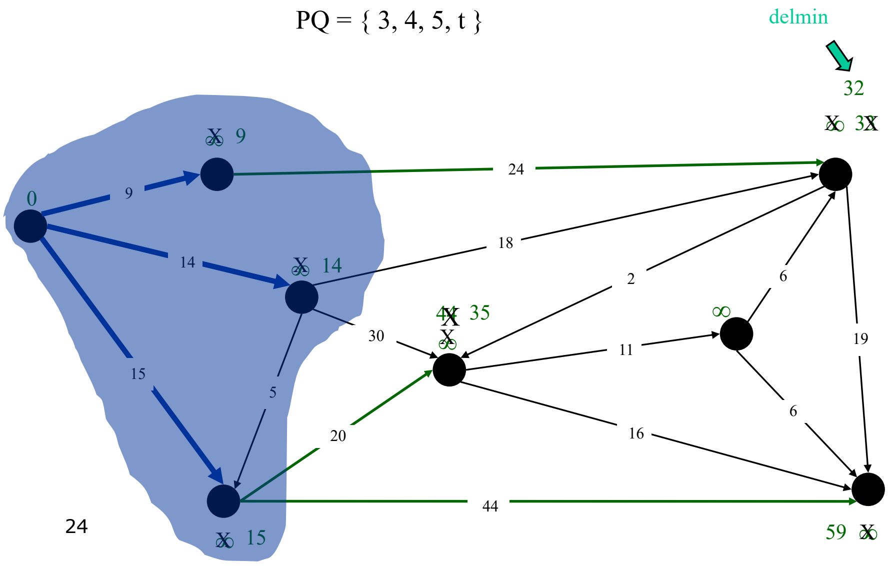
23

23

Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

PQ = { 3, 4, 5, t }



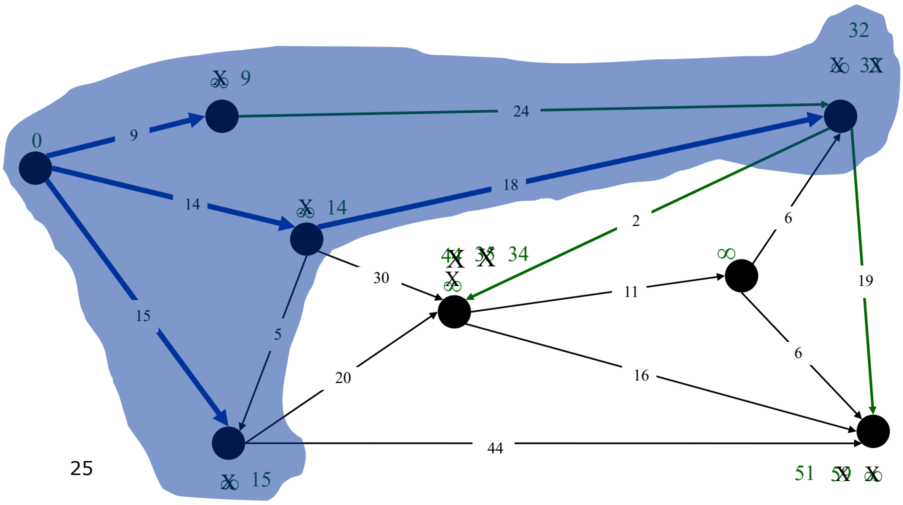
24

24

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 6, 7 }

PQ = { 4, 5, t }

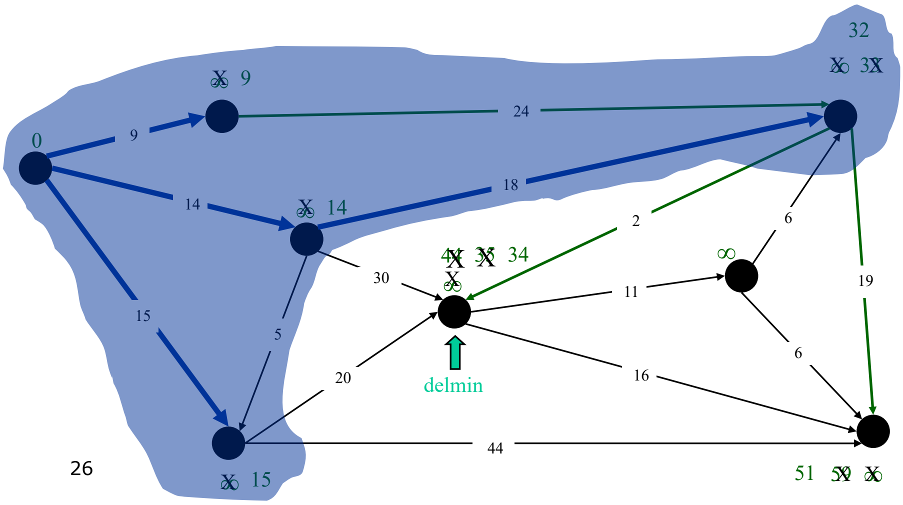


25

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 6, 7 }

PQ = { 4, 5, t }

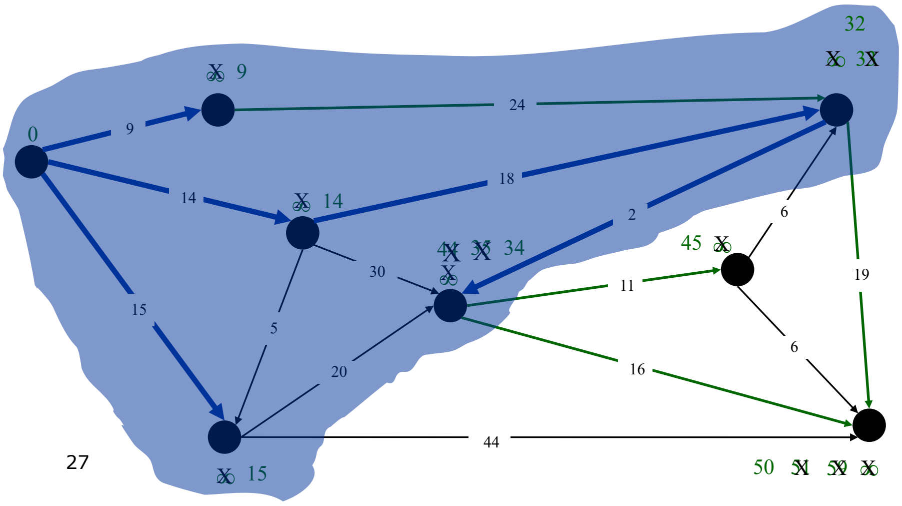


26

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 5, 6, 7 }

PQ = { 4, t }



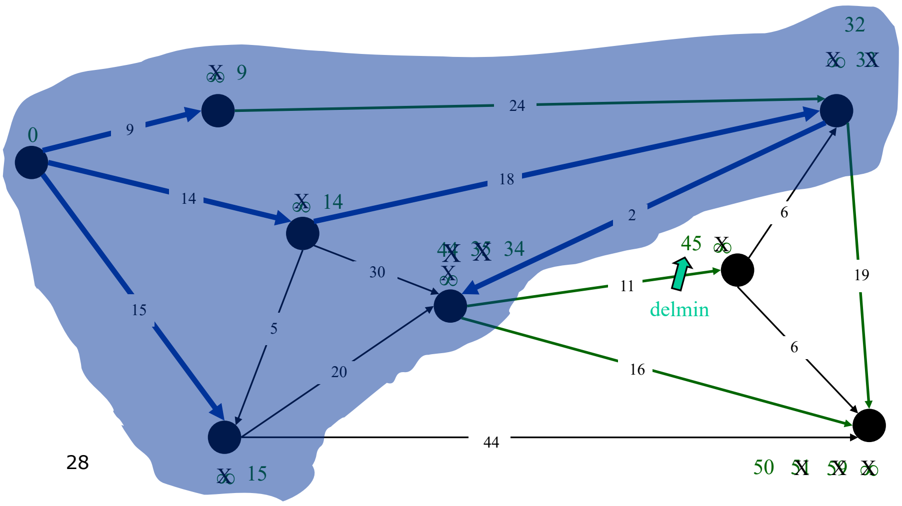
27

27

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 5, 6, 7 }

PQ = { 4, t }



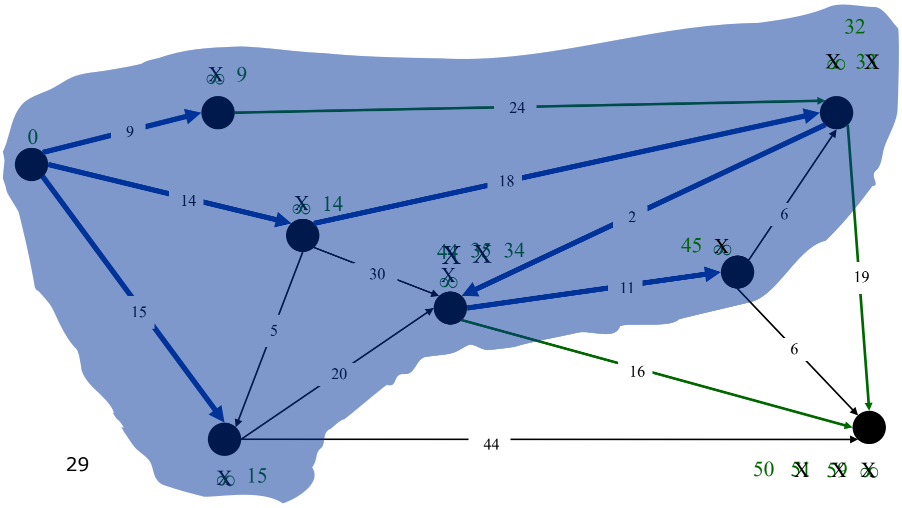
28

28

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7 }

PQ = { t }

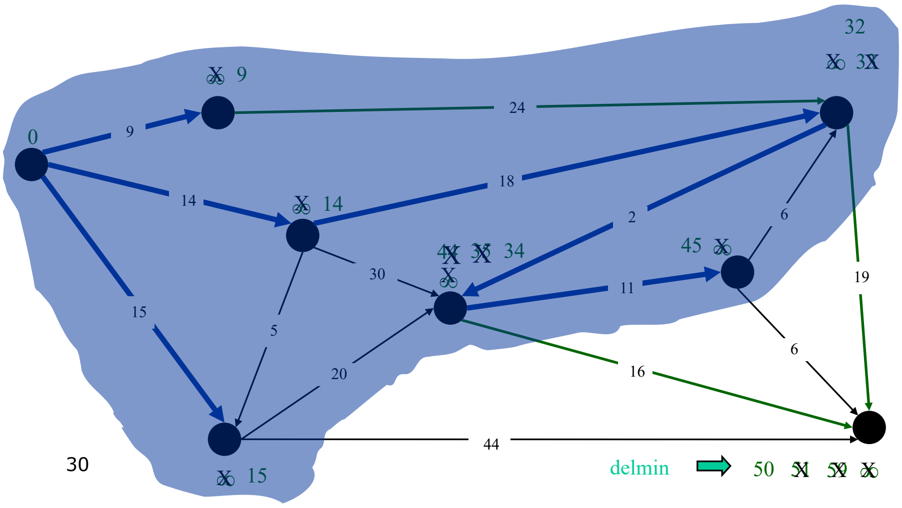


29

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7 }

PQ = { t }

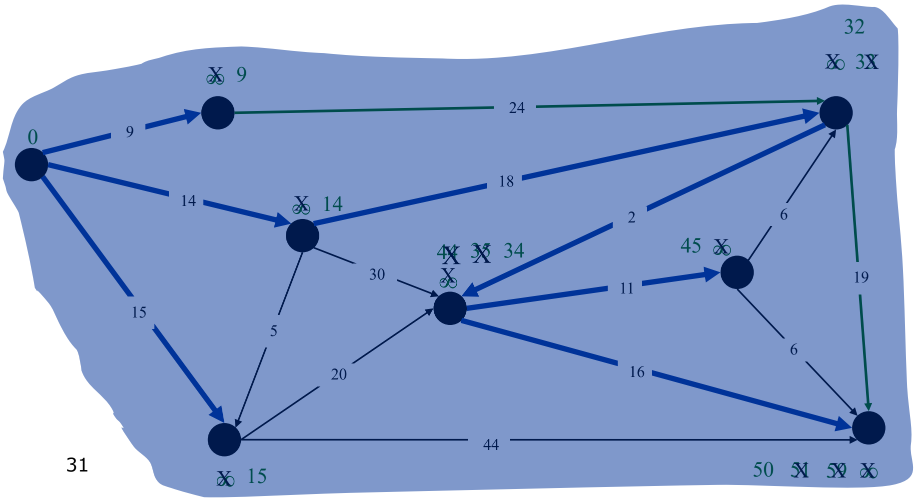


30

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7, t }

PQ = { }



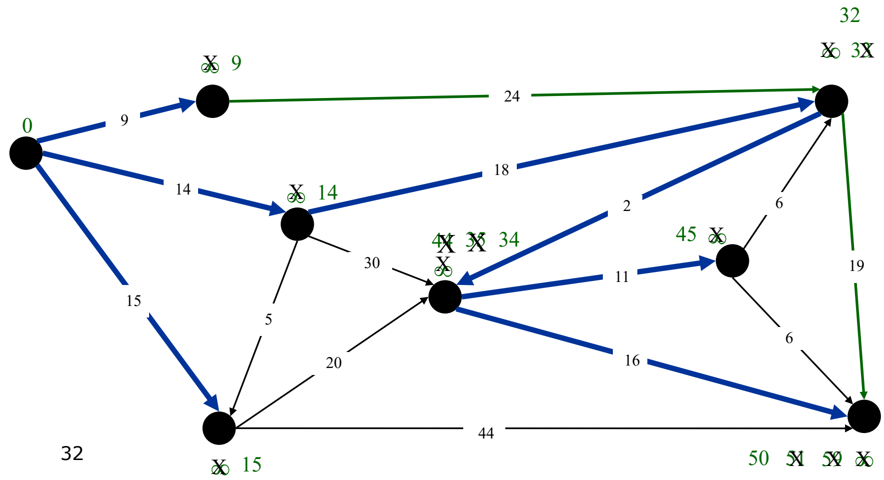
31

31

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7, t }

PQ = { }



32

32

Optimal Substructure

Cost of the path – sum of the weight of all edges

Observation 1: Shortest path has a optimal substructure Problem (we will see greedy and dynamic programming techniques)

If p is the shortest path from *v1* to *vk* , *v1, v2, … vi, …. vj, … vk.* The for any i, j the shortest path from *vi* to *vj* is contained in it.
Proof: if some subpath is not a shortest path then we can substitute it and obtain shorter path then the original (contradiction)

Notion of the shortest path is well defined only when there are No negative weight cycles – we will just consider positive weights

33

Relaxation

Idea: originally assign the path some upper bound and then keep on decreasing it as until you reach the cost of shortest path, *d[u] – attribute keeps upper bound on the cost of shortest path*

Dijkstra's Algorithm (G,w,s) %(with adjancency lists)
Initialize-Single-Source(G,s) % set cost to go to infinity – except
S := 0 % S start node
Q:= V[G] % priority queue put all vertices there
while *Q = 0* **do**
    u := extract_min(Q)
    S := S U {u}
    for each vertex *v in Adj[u]* **do**
        if *v in Q* **and** *d[u] + w(u,v) < d[v]* **then**
        d[v] := d[u] + w(u,v)
End

Relaxation

Example blackboard

34

Running time analysis

The values of the array are kept in priority queue (min-heap)

- Updating the heap structure takes at most *O(lg V)* and there are |V| of such operations. Building the heap takes *O(V).*
- Extract min , we need to fix heap structure |*V*| times, *V lg V*
- Also relaxation needs to **Decrease_key** for elements in the heap and still fix the heap structure (that can be done in *E lg*
- The total running time:

  O ((V+E) lg V) for sparse graphs *O (E lg V)*   *E ~ V sparse grap*

  O (V lg V + E) with Fibonacci heap decrease key amortized cost is *O(1)*

  If the priority queue is maintained as linear array then **Extract_min** takes *O(V)* and there are |*V*| of such operations Then **Extract_min** will take total *O(V^2)* + each edge in the Adj. List is examined, then total running time *O(V^2 + E)* Hence *O(V^2)* since *E* is *O(V^2).*

35

Running time

- Naïve version

$$O(|V|^2)$$

- Keeping list of nodes sorted by the cost in the priority queue

$$O(|E| + |V| \log |V|)$$

36

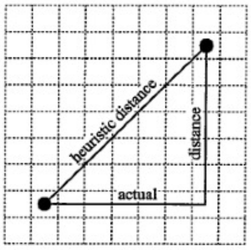
Graph Algorithms

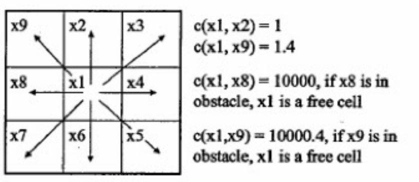
- Grassfire
- Dijstra's
- Both explore nodes based on the distance from the start node
- They produce similar behaviors
- Both visit all or most nodes in the graph
- How can we do it more quickly
- Next A* - best first search algorithm

37

A*

- Extension to Dijkstra, tries to reduce the number of states
- Using heuristic estimate of the cost to go
- Evaluation function f(n) = g(n) + h(n)
- Operating cost function g(n) – cost so far
- Heuristic function h(n)
  information used to find promising node to take next
  heuristics is admisible if it never overestimates the actual cost





Cost on a grid

38

A*

- Commonly used heuristic functions on grids
- Euclidean Distance

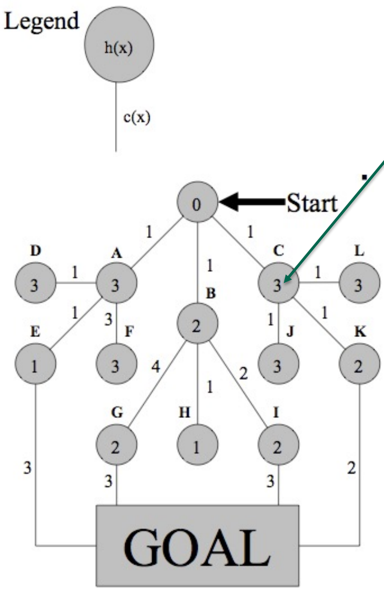
$$H(x_n, y_n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$

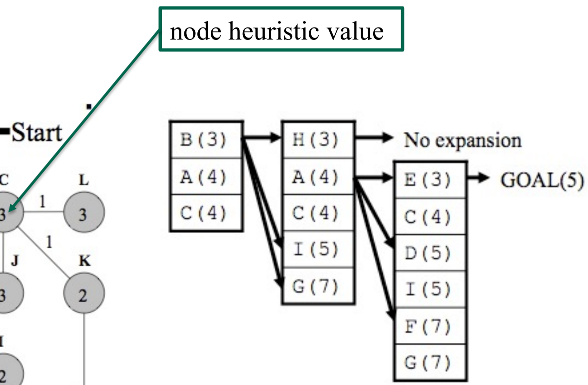
- Manhattan Distance

$$H(x_n, y_n) = |(x_n - x_g)| + |(y_n - y_g)|$$

39

A*



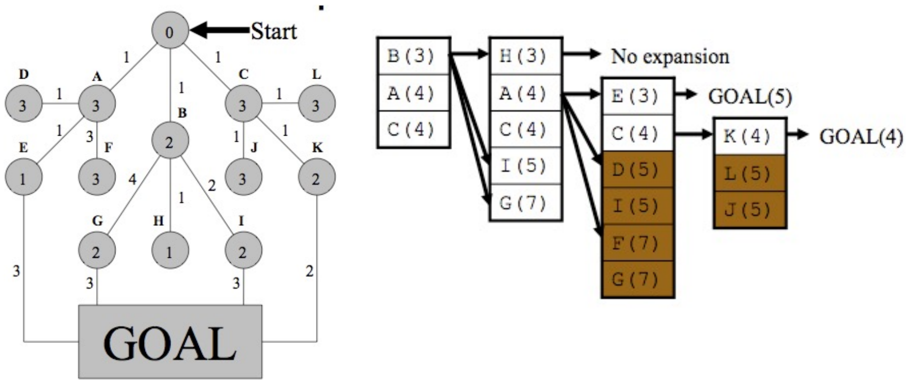


Start => A => E => goal

Nodes with higher or equal priority then goal can be pruned away
There can still be shorter path through the remaining nodes

40

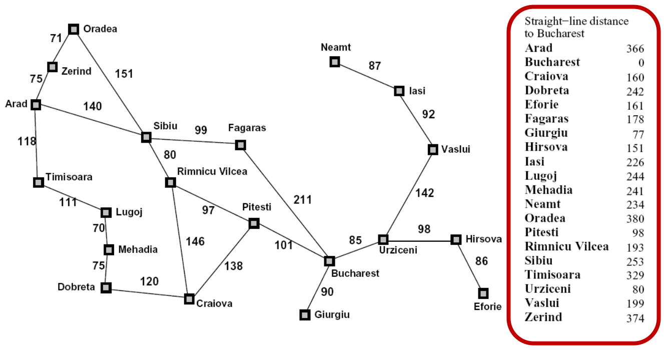
A*



Keep on expanding nodes with the priority level lower than goal
Path: Start => C = > K => Goal

41

Example: Heuristic Function



h(x)

42

A*

- Complete provided finite boundary condition
- Optimal in terms of path cost
- Memory inefficient
- Exponential growth of search space with respect to the length of the solution
- How can we use it in partially known, or dynamically changing environment D*

- Same as Dijkstra – different way of updating the cost of each node
- As long as heuristics under-estimates the cost to go – A* is optimal
- In many practical problems it is hard to find good heuristics

43

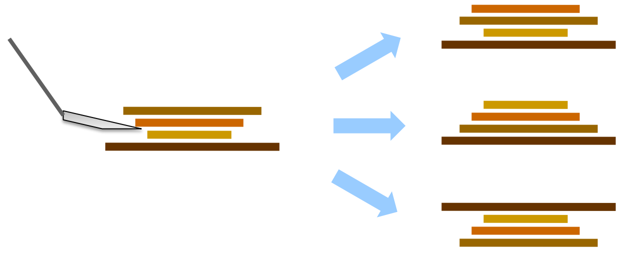
Planning Problems on Graphs

- Various performance measures of search:
- Optimality, completeness, time and space complexity

- Uninformed search – blind no information is gathered from the environment
  – wavefront algorithm, BFS, DFS
- Informed search
  – some evaluation function is used, Dijkstra, A*, D*

44

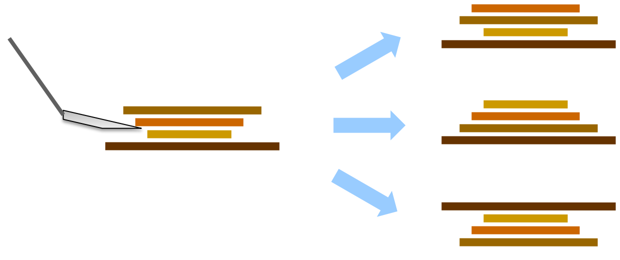
Example: Pancake Problem



Cost: Number of pancakes flipped

50

Example: Pancake Problem

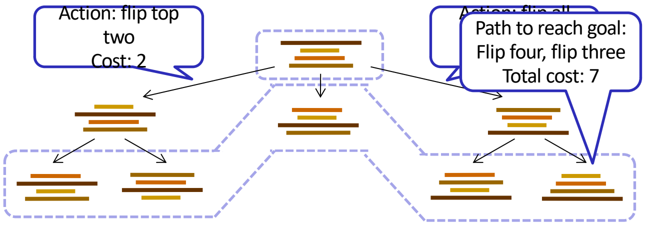


Cost: Number of pancakes flipped

51

General Tree Search

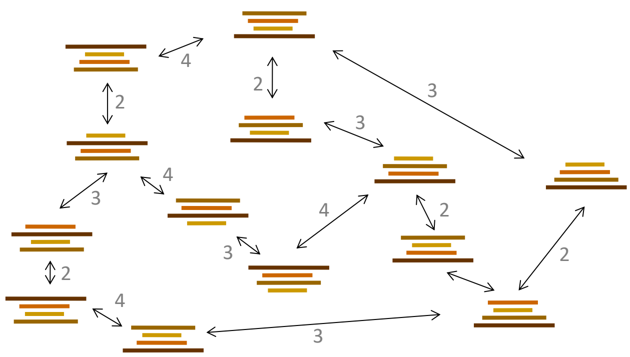
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```



52

Example: Pancake Problem

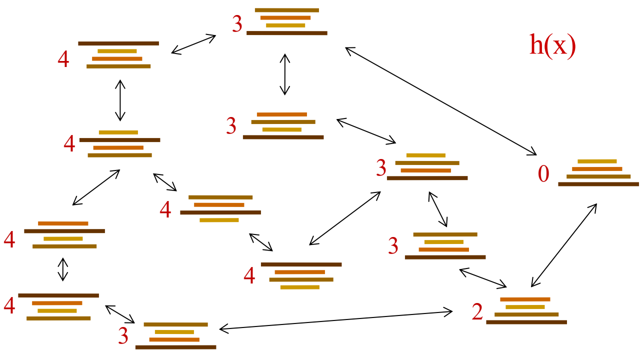
State space graph with costs as weights



53

Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place



54