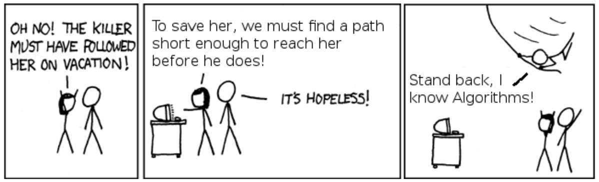
CSE 421: Algorithms

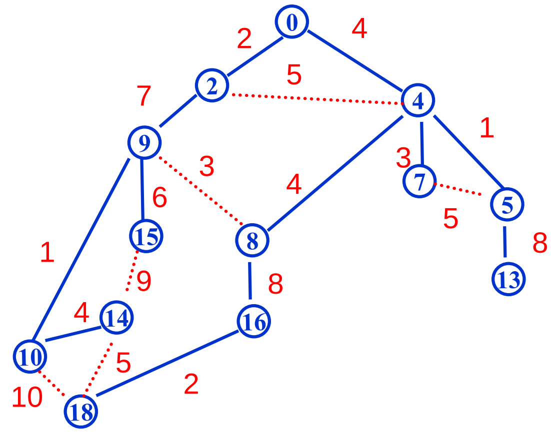
Winter 2014

Lecture 10: Dijkstra's algorithm / Divide & Conquer

Reading: Sections 5.1-5.4



single-source shortest paths



a greedy algorithm

Dijkstra's Algorithm:

- **Maintain a set S of vertices whose shortest paths are known**

  initially S={s}
- **Maintaining current best lengths of paths that only go through S to each of the vertices in G**

  path-lengths to elements of S will be right, to V-S they might not be right
- **Repeatedly add vertex v to S that has the shortest tentative distance of any vertex in V-S**

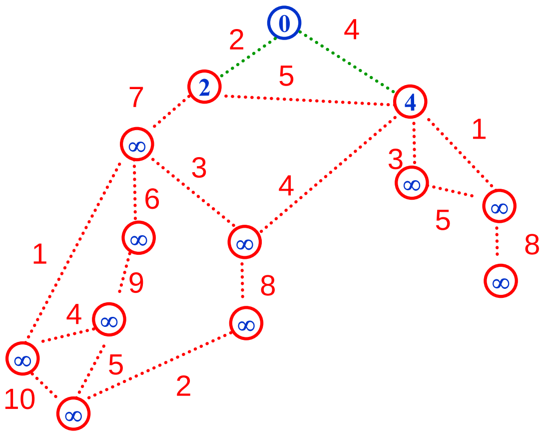
  update path lengths based on new paths through v

Dijkstra's Algorithm

Dijkstra(**G**,**w**,**s**)
  S←{**s**}
  d[**s**]←**0**
  while S≠**V do**
    of all edges **e**=(**u**,**v**) s.t. **v**∉**S** and **u**∈**S** select* one with the minimum value of **d**[**u**]+**w**(**e**)
      S←**S**∪ {**v**}
      d[**v**]←**d**[**u**]+**w**(**e**)
      pred[**v**]←**u**

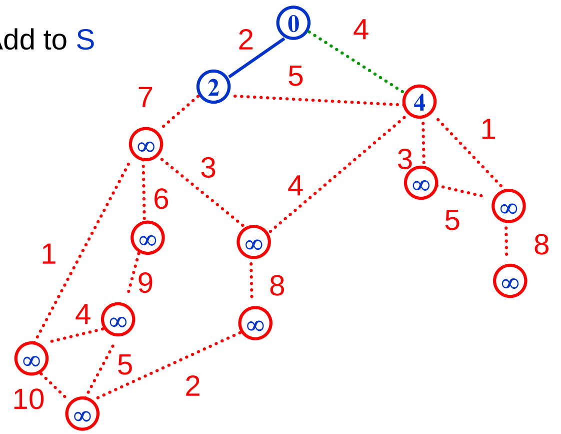
*For each v∉S maintain d'[v]=minimum value of d[u]+w(e) over all vertices u∈S s.t. e=(u,v) is in of G

Dijkstra's Algorithm



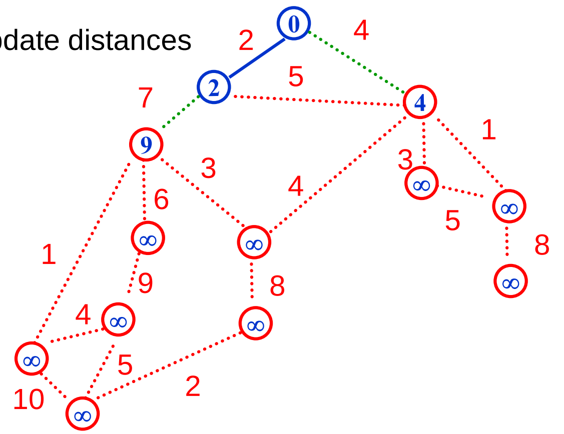
Dijkstra's Algorithm

Add to S



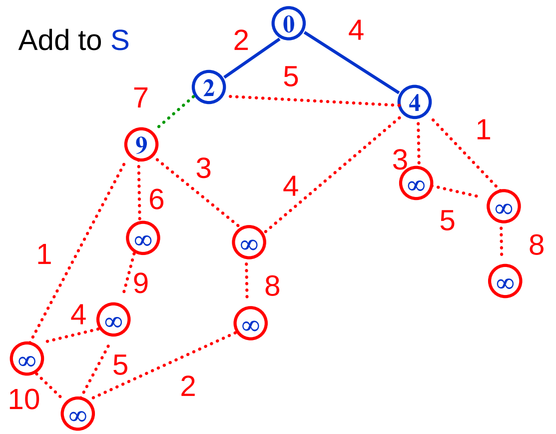
Dijkstra's Algorithm

Update distances



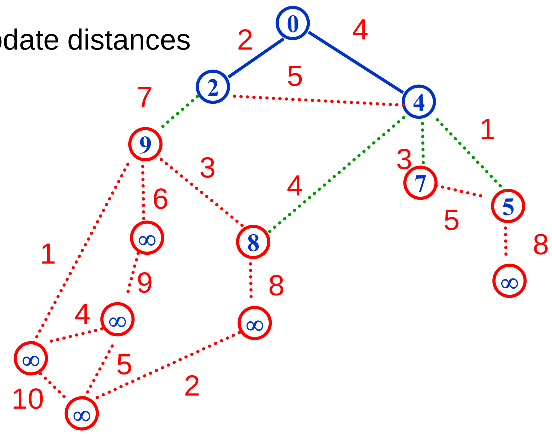
Dijkstra's Algorithm

Add to S



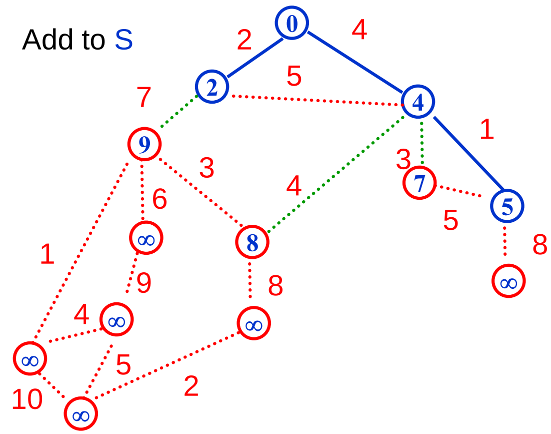
Dijkstra's Algorithm

Update distances



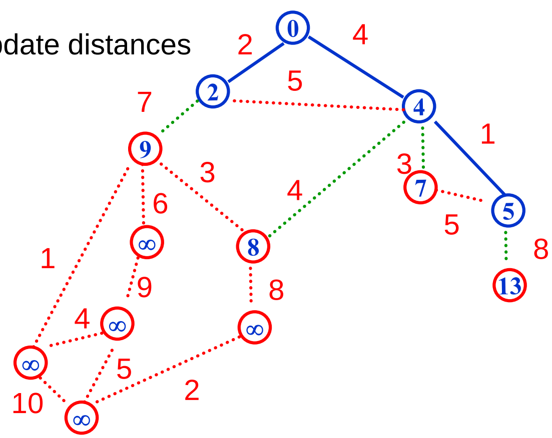
Dijkstra's Algorithm

Add to S



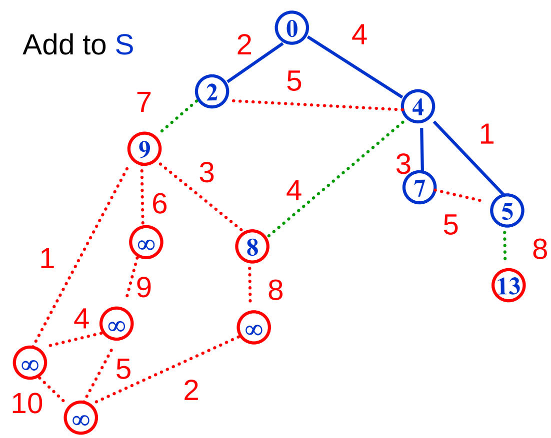
Dijkstra's Algorithm

Update distances



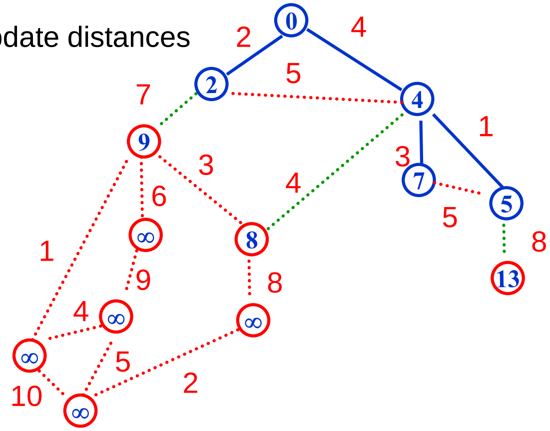
Dijkstra's Algorithm

Add to S



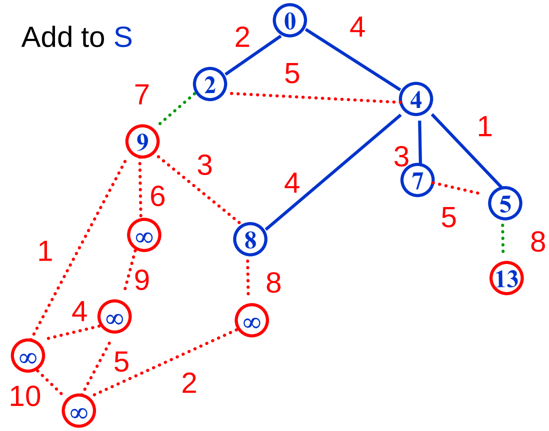
Dijkstra's Algorithm

Update distances



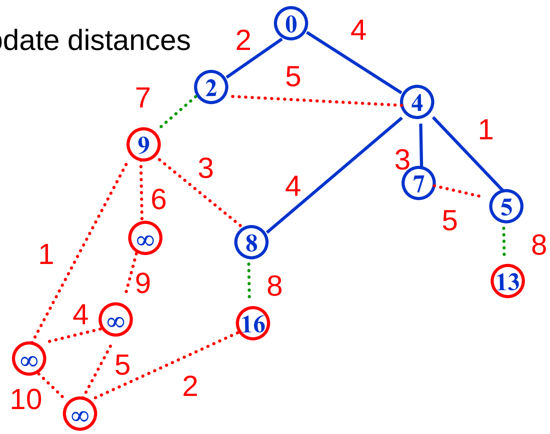
Dijkstra's Algorithm

Add to S



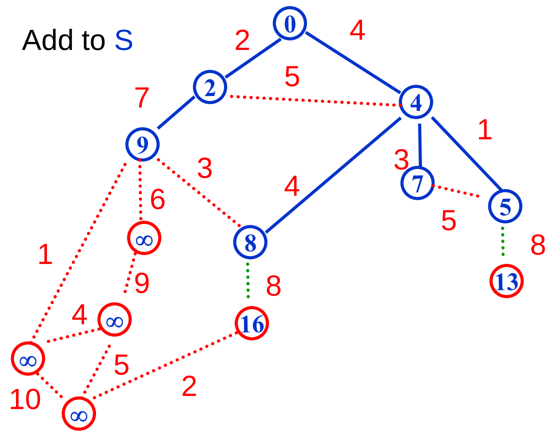
Dijkstra's Algorithm

Update distances



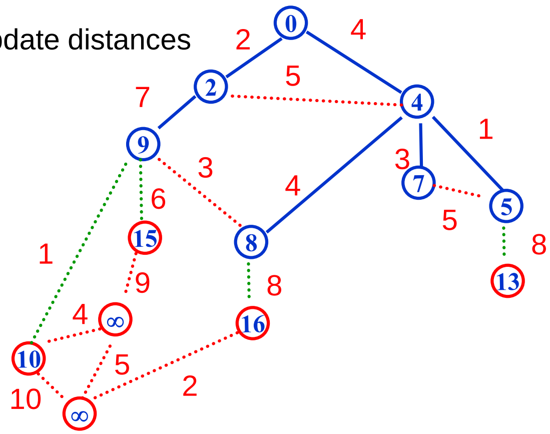
Dijkstra's Algorithm

Add to S



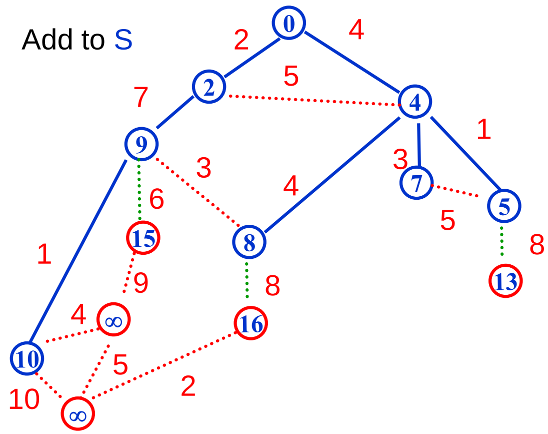
Dijkstra's Algorithm

Update distances



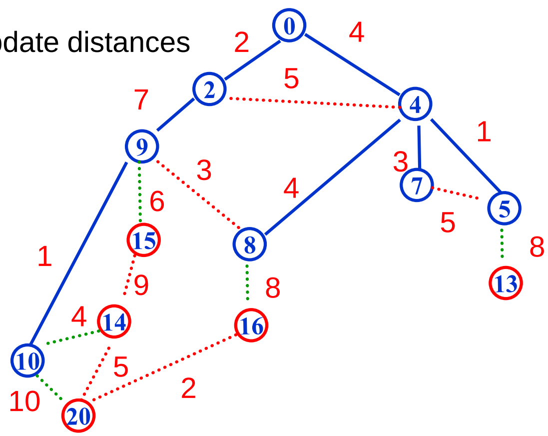
Dijkstra's Algorithm

Add to S



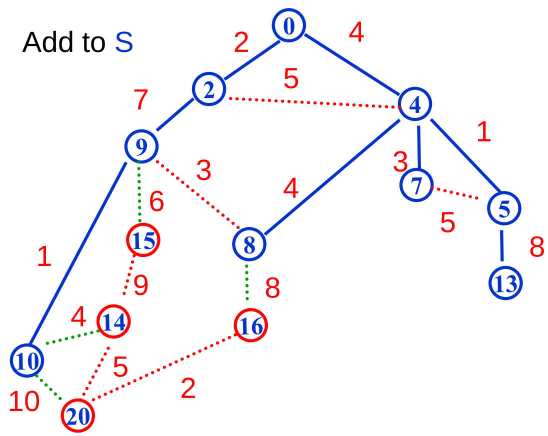
Dijkstra's Algorithm

Update distances



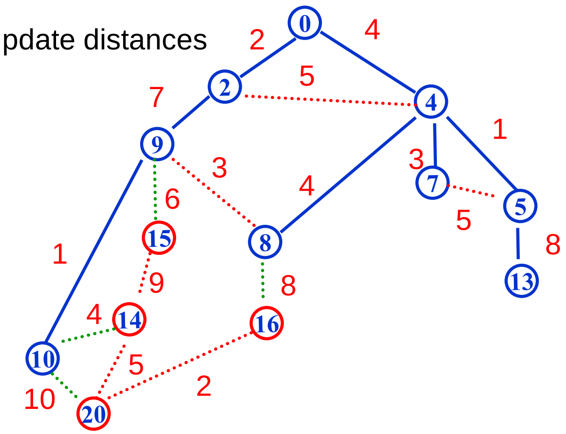
Dijkstra's Algorithm

Add to S



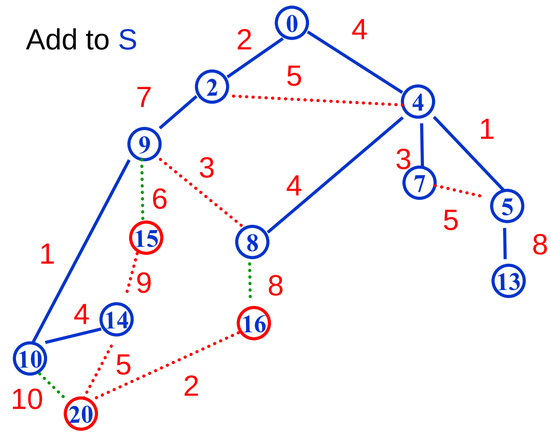
Dijkstra's Algorithm

Update distances



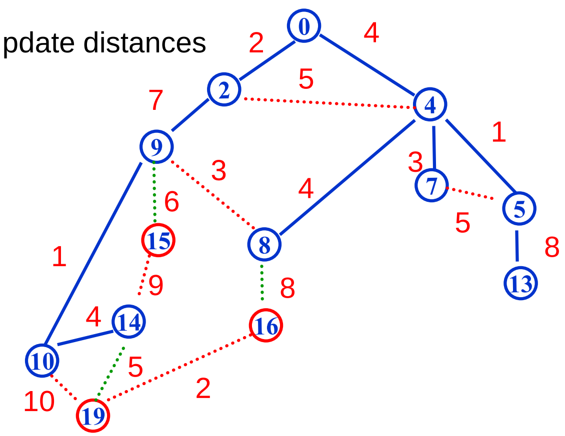
Dijkstra's Algorithm

Add to S



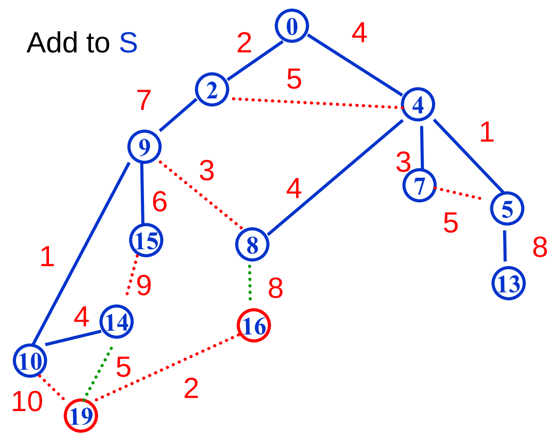
Dijkstra's Algorithm

Update distances



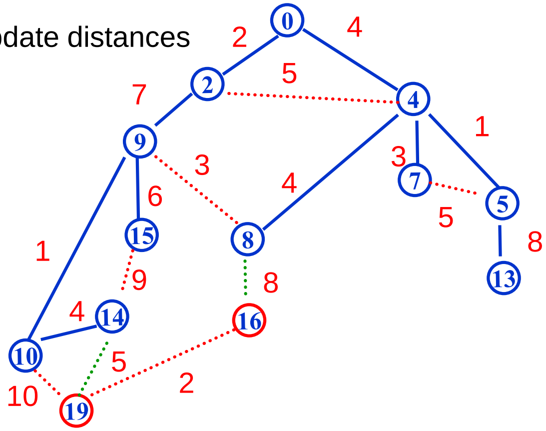
Dijkstra's Algorithm

Add to S



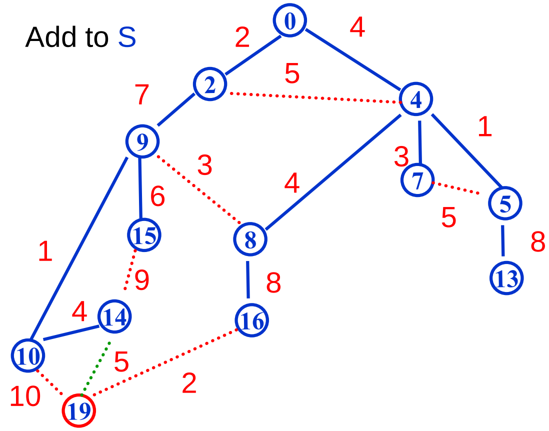
Dijkstra's Algorithm

Update distances



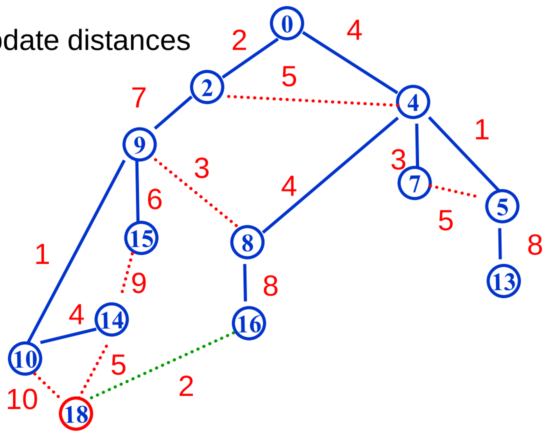
Dijkstra's Algorithm

Add to S



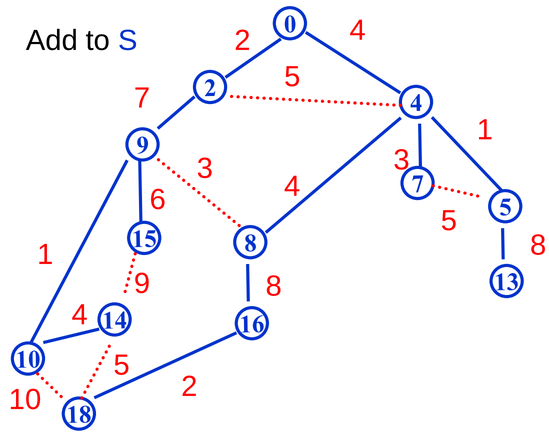
Dijkstra's Algorithm

Update distances



Dijkstra's Algorithm

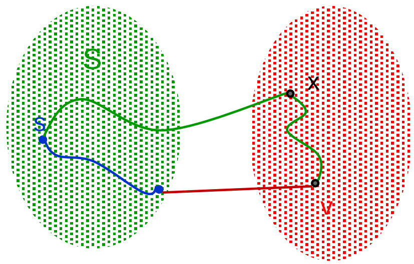
Add to S



Dijkstra's algorithm correctness

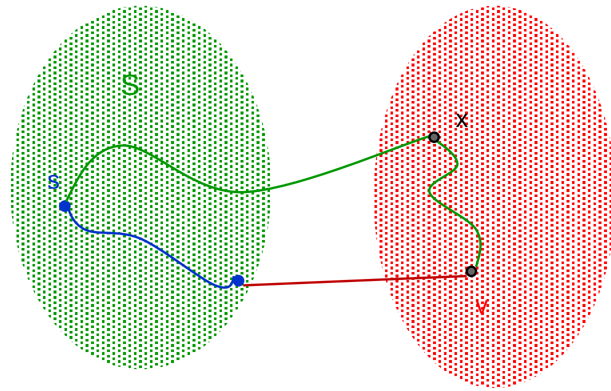
Suppose all distances to vertices in S are correct
and v has smallest current value in V-S

Distance value of vertex in V-S=length of shortest path from s
with only last edge leaving S



Therefore adding v to S keeps correct distances

Dijkstra's algorithm correctness



Dijkstra's algorithm

- Algorithm also produces a tree of shortest paths to v following pred links
  - From w follow its ancestors in the tree back to v
- If all you care about is the shortest path from v to w simply stop the algorithm when w is added to S

implementing Dijkstra's algorithm

Need to

- keep current distance values for nodes in V-S
- find minimum current distance value
- reduce distances when vertex moved to S

data structure review

- **Priority Queue:**
  - Elements each with an associated **key**
  - Operations
    - **Insert**
    - **Find-min**
      - Return the element with the smallest key
    - **Delete-min**
      - Return the element with the smallest key and delete it from the data structure
    - **Decrease-key**
      - Decrease the key value of some element
- Implementations
  - Arrays: **O(n)** time find/delete-min, **O(1)** time insert/ decrease-key
  - Heaps: **O(log n)** time insert/decrease-key/delete-min, **O(1)** time find-min

Dijkstra's algorithm with priority queues

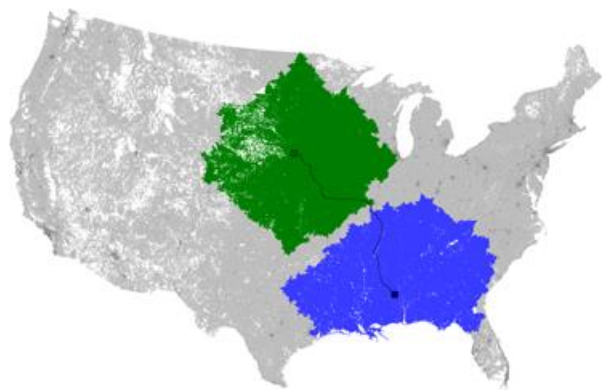
- For each vertex **u** not in tree maintain cost of current cheapest path through tree to **u**
  - Store **u** in priority queue with key = length of this path
- Operations:
  - **n-1** insertions (each vertex added once)
  - **n-1** delete-mins (each vertex deleted once)
    - pick the vertex of smallest key, remove it from the priority queue and add its edge to the graph
  - **<m** decrease-keys (each edge updates one vertex)

Dijkstra's algorithm with priority queues

Priority queue implementations

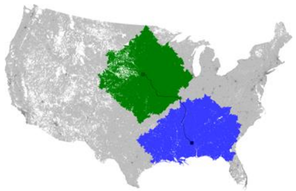
- **Array**
  - insert O(1), delete-min O(n), decrease-key O(1)
  - total $O(n+n^2+m)=O(n^2)$
- **Heap**
  - insert, delete-min, decrease-key all O(log n)
  - total O(m log n)
- **d-Heap (d=m/n)**
  - insert, decrease-key $O(\log_{m/n} n)$
  - delete-min $O((m/n) \log_{m/n} n)$
  - total $O(m \log_{m/n} n)$

computing point-to-point shortest paths

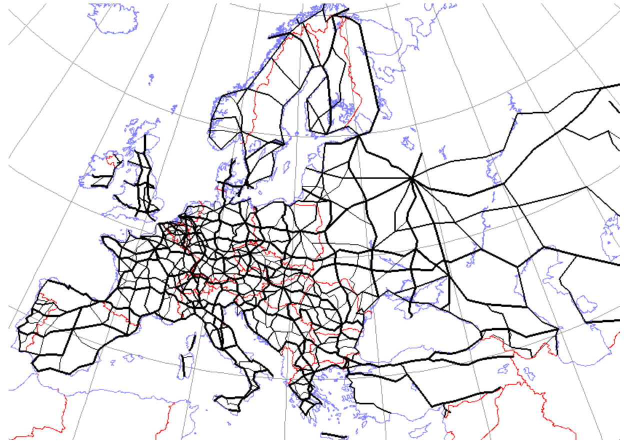


A^* algorithm

- Want to find shortest s-v path
- Since we do not care about all distances from s, would like our set S to “grow quickly toward v”
- For every node u, have a “heuristic” value $h(u)$ that gives a **lower bound** on the length of the shortest path from u to v
- Allows us to rule out certain nodes during the search!



A^* algorithm



A^* algorithm

- Want to find shortest s-v path
- For every node u, have a “heuristic” value $h(u)$ that gives a **lower bound** on the length of the shortest path from u to v
- If $d[u]$ is the current estimate on the distance from s to u, then we process the node with smallest value of $d[u] + h(u)$

divide & conquer

- Divide & Conquer
  - Reduce problem to one or more sub-problems of the same type
  - Typically, each sub-problem is **at most a constant fraction** of the size of the original problem

    e.g. Mergesort, Binary Search, Strassen’s Algorithm, Quicksort (kind of)

fast exponentiation

- Power(**a**,**n**)
  - **Input:** integer n and number a
  - **Output:** a^n
- Obvious algorithm
  - n-1 multiplications
- Observation:
  - if n is even, n=2m, then $a^n = a^m \cdot a^m$

divide & conquer qlgorithm

```
Power(a,n):
    if n=0 then
        return(1)
    else if n=1 then
        return(a)
    else
        x ← Power(a, ⌊n/2⌋)
        if n is even then
            return(x•x)
        else
            return(a•x•x)
```

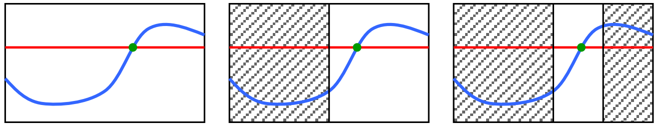
analysis

- Worst-case recurrence
  - $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2$ for $n \geq 1$
  - $T(1) = 0$
- Time:
- More precise analysis:
  $T(n) = \lceil \log_2 n \rceil +$ # of 1's in n's binary representation

practical application: RSA

- Instead of a^n want a^n mod N
  - a^{i+j} mod N = ((a^i mod N)•(a^j mod N)) mod N
  - same algorithm applies with each x•y replaced by
    ((x mod N)•(y mod N)) mod N
- In RSA cryptosystem (widely used for security)
  - need a^n mod N where a, n, N each typically have 1024 bits
  - Power: at most 2048 multiplies of 1024 bit numbers
    relatively easy for modern machines
  - Naive algorithm: 2^{1024} multiplies

binary search for roots (bisection method)



- **Given:**
  - continuous function f and two points a<b with f(a) ≤ 0 and f(b) > 0
- **Find:**
  - approximation to c s.t. f(c)=0 and a<c<b

bisection method

```
Bisection(a,b,ε):
   if (a-b) < ε then
      return(a)
   else
      c ←(a+b)/2
      if f(c) ≤ 0 then
           return(Bisection(c,b,ε))
      else
           return(Bisection(a,c,ε))
```

analysis

- At each step we halved the size of the interval
- It started at size b-a
- It ended at size ε
- # of calls to f is $\log_2\left(\frac{b-a}{\epsilon}\right)$