DIJKSTRA'S ALGORITHM

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Greedy algorithms

Greedy algorithms: constructs a solution through a sequence of steps, each expanding a partially constructed solution

On each step, the choice made must be:

- Feasible: satisfies the problem's constraints
- Locally optimal: best feasible local choice
- Irrevocable: cannot be changed later

Dijkstra's algorithm

Let G be a weighted graph and $v \in V$ (*source*), finds the shortest path from v to all other nodes in V

- Single-source shortest paths

Applications

- Transport planning
- Communication networks
- Social networks
- Robotics
- Pathfinding
- Puzzles
- etc.

Dijkstra's algorithm: cannot be used on weighted graphs with negative weights

Agenda

Dijkstra's algorithm

First, find the closest node to v (itself)

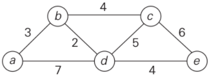
On the i-th step:

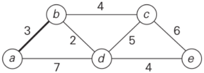
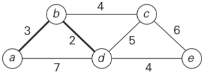
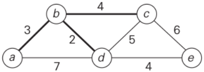
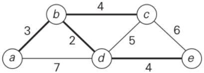
- Knows the $(i-1)$-th closest nodes to v (they form a tree)
- Since there are no negative weights, the next closest one is adjacent to one of the $i-1$ closest nodes to v
- After chosing the $i-th$ closest node (w), updates the possible shortest paths to yet unchosen nodes (u) if $d_w + weight(w,u) < d_u$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Dijkstra's algorithm[1]



Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

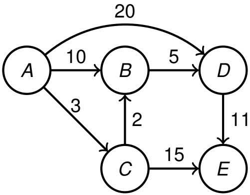
[1] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  $D[i] \leftarrow \infty$; $P[i] \leftarrow -$;
  $setMark(G, i, UNVISITED)$;
$H[1] \leftarrow (s, s, 0)$; $D[s] \leftarrow 0$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  repeat
    $(p, v) \leftarrow removemin(H)$;
    if $v = NULL$ **then return**;
  until $getMark(G, v) = UNVISITED$;
  $setMark(G, v, VISITED)$; $P[v] \leftarrow p$;
  $w \leftarrow first(G, v)$;
  while $w < n(G)$ **do**
    if $getMark(G, w) \neq VISITED \wedge$
     $D[w] > D[v] + weight(G, v, w)$ **then**
      $D[w] \leftarrow D[v] + weight(G, v, w)$;
      $insert(H, (v, w, D[w]))$;
    $w \leftarrow next(G, v, w)$;

Let $s = A$



	A	B	C	D	E
Mark	×	×	×	×	×
Distance	0	∞	∞	∞	∞
Parent	–	–	–	–	–

(A,A,0)

Centro de Informática UFPE

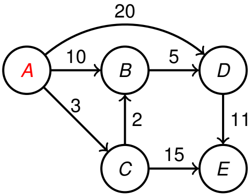
Universidade Federal de Pernambuco

Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  $D[i] \leftarrow \infty$; $P[i] \leftarrow -$;
  $setMark(G, i, UNVISITED)$;
$H[1] \leftarrow (s, s, 0)$; $D[s] \leftarrow 0$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  repeat
    $(p, v) \leftarrow removemin(H)$;
    if $v = NULL$ **then return**;
  until $getMark(G, v) = UNVISITED$;
  $setMark(G, v, VISITED)$; $P[v] \leftarrow p$;
  $w \leftarrow first(G, v)$;
  while $w < n(G)$ **do**
    if $getMark(G, w) \neq VISITED \wedge D[w] > D[v] + weight(G, v, w)$ **then**
      $D[w] \leftarrow D[v] + weight(G, v, w)$;
      $insert(H, (v, w, D[w]))$;
    $w \leftarrow next(G, v, w)$;

Let $s = A$



	A	B	C	D	E
Mark	✓	×	×	×	×
Distance	0	∞	∞	∞	∞
Parent	A	–	–	–	–

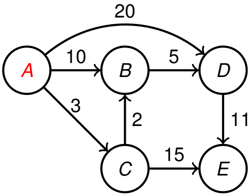
–

Dijkstra's algorithm

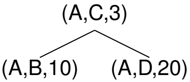
Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	×	×	×	×
Distance	0	10	3	20	∞
Parent	A	–	–	–	–

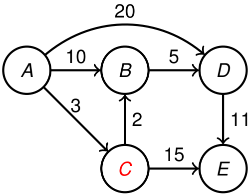


Dijkstra's algorithm

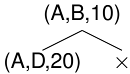
Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	×	✓	×	×
Distance	0	10	3	20	∞
Parent	A	–	A	–	–

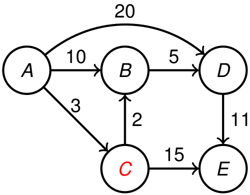


Dijkstra's algorithm

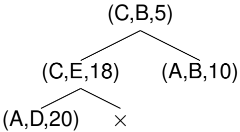
Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	×	✓	×	×
Distance	0	5	3	20	18
Parent	A	–	A	–	–



Centro de Informática UFPE

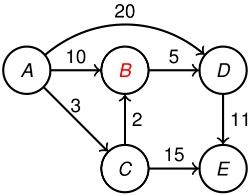
Universidade Federal de Pernambuco

Dijkstra's algorithm

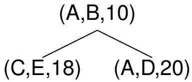
Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	×	×
Distance	0	5	3	20	18
Parent	A	C	A	–	–



Centro de Informática UFPE

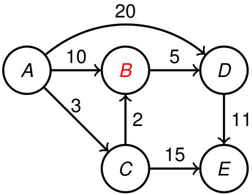
Universidade Federal de Pernambuco

Dijkstra's algorithm

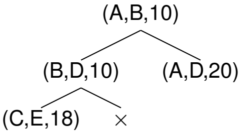
Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	×	×
Distance	0	5	3	10	18
Parent	A	C	A	−	−



Centro de Informática UFPE

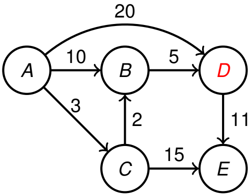
Universidade Federal de Pernambuco

Dijkstra's algorithm

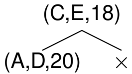
Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	✓	×
Distance	0	5	3	10	18
Parent	A	C	A	B	–

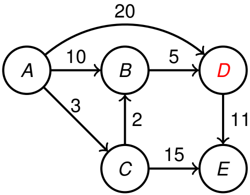


Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	✓	×
Distance	0	5	3	10	18
Parent	A	C	A	B	–

(C,E,18)

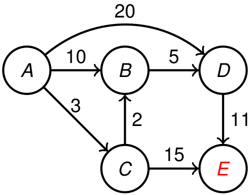
(A,D,20) ×

Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
    $D[i] \leftarrow \infty$; $P[i] \leftarrow -$;
    $setMark(G, i, UNVISITED)$;
$H[1] \leftarrow (s, s, 0)$; $D[s] \leftarrow 0$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
    repeat
        $(p, v) \leftarrow removemin(H)$;
        if $v = NULL$ **then return**;
    until $getMark(G, v) = UNVISITED$;
    $setMark(G, v, VISITED)$; $P[v] \leftarrow p$;
    $w \leftarrow first(G, v)$;
    while $w < n(G)$ **do**
        if $getMark(G, w) \neq VISITED \wedge D[w] > D[v] + weight(G, v, w)$ **then**
            $D[w] \leftarrow D[v] + weight(G, v, w)$;
            $insert(H, (v, w, D[w]))$;
        $w \leftarrow next(G, v, w)$;

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	✓	✓
Distance	0	5	3	10	18
Parent	A	C	A	B	C

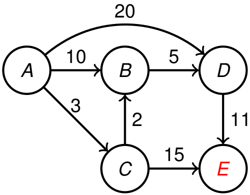
(A,D,20)

Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	✓	✓
Distance	0	5	3	10	18
Parent	A	C	A	B	C

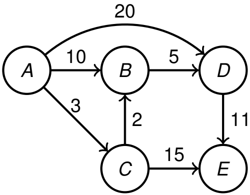
(A,D,20)

Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  $D[i] \leftarrow \infty$; $P[i] \leftarrow -$;
  $setMark(G, i, UNVISITED)$;
$H[1] \leftarrow (s, s, 0)$; $D[s] \leftarrow 0$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  repeat
    $(p, v) \leftarrow removemin(H)$;
    if $v = NULL$ **then return**;
  until $getMark(G, v) = UNVISITED$;
  $setMark(G, v, VISITED)$; $P[v] \leftarrow p$;
  $w \leftarrow first(G, v)$;
  while $w < n(G)$ **do**
    if $getMark(G, w) \neq VISITED \wedge$
     $D[w] > D[v] + weight(G, v, w)$ **then**
      $D[w] \leftarrow D[v] + weight(G, v, w)$;
      $insert(H, (v, w, D[w]))$;
    $w \leftarrow next(G, v, w)$;

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	✓	✓
Distance	0	5	3	10	18
Parent	A	C	A	B	C

(A,D,20)

Centro de Informática UFPE

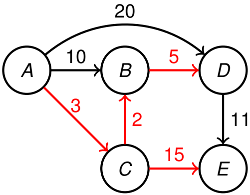
Universidade Federal de Pernambuco

Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Let $s = A$



	A	B	C	D	E
Mark	✓	✓	✓	✓	✓
Distance	0	5	3	10	18
Parent	A	C	A	B	C

(A,D,20)

Centro de Informática UFPE

Universidade Federal de Pernambuco

Dijkstra's algorithm

Algorithm: void Dijkstra(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞ ; P[i] ← −;
    setMark(G, i, UNVISITED);
H[1] ← (s, s, 0) ; D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        (p, v) ← removemin(H);
        if v = NULL then return;
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED) ; P[v] ← p;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧
          D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (v, w, D[w]));
        w ← next(G, v, w);
```

Time efficiency

- Matrix and no heap
  $\Theta(|V|^2 + E|) = \Theta(|V|^2)$, since $|E| \in O(|V|^2)$
  - Better for dense graphs
- List and heap
  $\Theta((|V| + |E|) \log |V|)$
  - Better for sparse graphs

Centro de Informática UFPE

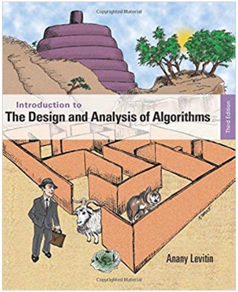
Universidade Federal de Pernambuco

Agenda

Centro de Informática UFPE

Universidade Federal de Pernambuco

Bibliography



Chapter 9 (pp. 333–337)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3rd edition. Pearson. 2011.

Chapter 11 (pp. 389–393)
Clifford Shaffer.
Data Structures and Algorithm Analysis. Dover, 2013.

DIJKSTRA'S ALGORITHM

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil