A Matrix Product Approach to Weighted Graph Problems

Virginia Vassilevska

Carnegie Mellon University

June 16, 2007

Introduction

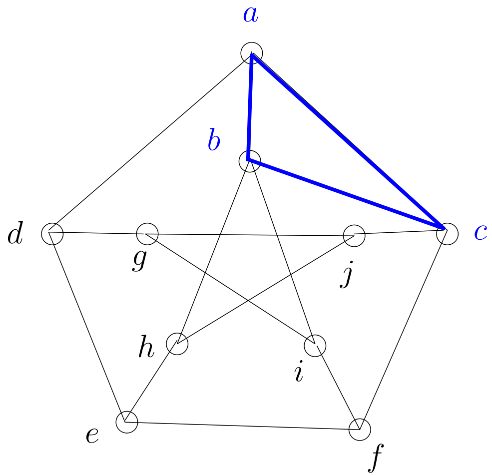
Introduction

Using fast matrix multiplication one can often obtain faster algorithms.

Introduction

Using fast matrix multiplication one can often obtain faster algorithms.

E.g., in a graph $G = (V, E)$ to find a TRIANGLE (a, b, c) look at the diagonal of the cube of the adjacency matrix. [Itai and Rodeh, 1978]



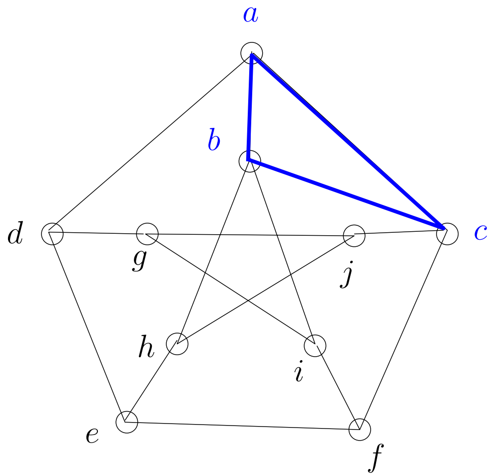
$$G^3 = \begin{pmatrix} 2 & \cdot & \cdot & \cdots \\ \cdot & 2 & \cdot & \cdots \\ \cdot & \cdot & 2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Naiive algorithm: $O(n^3)$, matrix mult.: $O(n^\omega) = O(n^{2.38})$.

Introduction

Using fast matrix multiplication one can often obtain faster algorithms.

E.g., in a graph $G = (V, E)$ to find a TRIANGLE (a, b, c) look at the diagonal of the cube of the adjacency matrix. [Itai and Rodeh, 1978]



$$G^3 = \begin{pmatrix} 2 & \cdot & \cdot & \cdots \\ \cdot & 2 & \cdot & \cdots \\ \cdot & \cdot & 2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Naiive algorithm: $O(n^3)$, matrix mult.: $O(n^\omega) = O(n^{2.38})$.

Other examples: *LP, exact algorithms for NP-hard problems, graph perfect matching, unweighted APSP.*

What about weighted problems?

What about weighted problems?

Itai and Rodeh's paper ends with:

"A related problem is finding a minimum weighted circuit in a weighted graph. It is unclear to us whether our methods can be modified to answer this problem too."

What about weighted problems?

Itai and Rodeh's paper ends with:

"A related problem is finding a minimum weighted circuit in a weighted graph. It is unclear to us whether our methods can be modified to answer this problem too."

In general it is not clear how to speed-up weighted versions of problems in a similar way.

Example open problems include: *maximum weighted matching, finding minimum weighted triangles and other patterns, weighted APSP*.

Matrix product approach

Instead of matrix multiplication we use other matrix products to speed-up weighted problems: dominance product, MaxMin product, $(\min, \leq)$-product .

We demonstrate the approach on *finding minimum weighted triangles, computing bits of the distance product, all pairs bottleneck paths, all pairs nondecreasing paths*.

Talk outline

1. Some definitions
2. Maximum weighted triangle
3. Computing bits of the distance product
4. All pairs bottleneck paths
5. All pairs nondecreasing paths
6. Open problems

Various Matrix Products: definitions

Various Matrix Products: definitions

Algebraic Product:

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Dominance Product:

$$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Dominance Product:

$$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$$

$(\min, \leq)$-Product:

$$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product:

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$ $\quad n^{\omega}$

Distance Product:

$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$

MaxMin Product:

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$

Dominance Product:

$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$

$(\min, \leq)$-Product:

$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}.$

Various Matrix Products: definitions

Algebraic Product:

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}$. n^{ω}

Distance Product:

$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}$.

MaxMin Product:

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$.

Dominance Product:

$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|$. $n^{\frac{3+\omega}{2}}$

$(\min, \leq)$-Product:

$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}$.

Various Matrix Products: definitions

Algebraic Product:

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$ n^{ω}

Distance Product:

$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$

MaxMin Product:

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$ $n^{2+\frac{\omega}{3}}$

Dominance Product:

$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$ $n^{\frac{3+\omega}{2}}$

$(\min, \leq)$-Product:

$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}.$ $n^{2+\frac{\omega}{3}}$

Various Matrix Products: definitions

Algebraic Product:

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$ n^{ω}

Distance Product:

$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$ subcubic?

MaxMin Product:

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$ $n^{2+\frac{\omega}{3}}$

Dominance Product:

$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$ $n^{\frac{3+\omega}{2}}$

$(\min, \leq)$-Product:

$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}.$ $n^{2+\frac{\omega}{3}}$

Maximum node weighted triangle

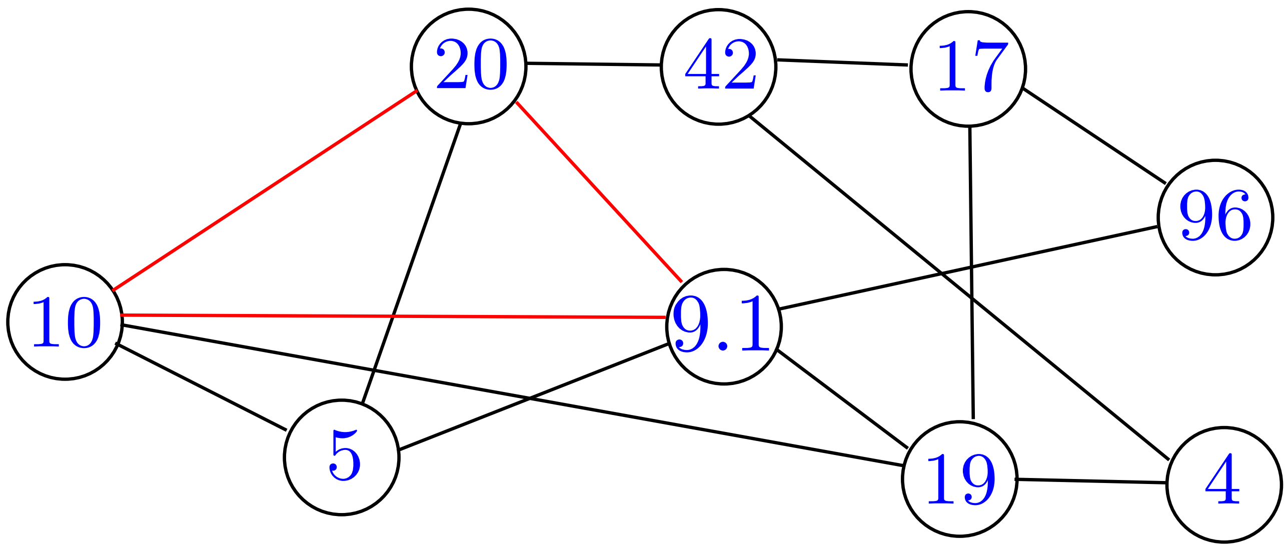
Input: Graph with real-number weights on the nodes

Task: Find a triangle of maximum weight sum

Maximum node weighted triangle

Input: Graph with real-number weights on the nodes

Task: Find a triangle of maximum weight sum



Maximum edge weighted triangle

Input: Graph with real-number weights on the edges

Task: Find a triangle of maximum weight sum

Maximum edge weighted triangle

Input: Graph with real-number weights on the edges

Task: Find a triangle of maximum weight sum

(Reduce Node-Weighted Triangle to Edge-Weighted Triangle):

Push weights from nodes to edges: $w(u,v) = (w(u) + w(v))/2$

Folklore Result

Folklore Result

Recall the **distance product** of A and B is

$$(A \star B)[i, j] = \min_k \{A[i, k] + B[k, j]\}$$

Folklore Result

Recall the **distance product** of A and B is

$$(A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}$$

Observation: Distance Product can solve Max Weighted Triangle

Folklore Result

Recall the **distance product** of A and B is

$$(A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}$$

Observation: Distance Product can solve Max Weighted Triangle

$\longrightarrow$ Compute $\quad MAX_{i,j}\{-((-A) \star (-A))[i,j] + A[i,j]\}$

(Min Weight Triangle: $MIN_{i,j}\{(A \star A)[i,j] + A[i,j]\}$)

”Easy” Weighted Triangle Algorithms

"Easy" Weighted Triangle Algorithms

- [Zwick, '02] $O(M \cdot n^{\omega})$ distance product algorithm, M is the largest weight of an edge

  $\Longrightarrow$ Max Weight Triangle in $O(M \cdot n^{\omega})$ (Pseudopolynomial)

"Easy" Weighted Triangle Algorithms

- [Zwick, '02] $O(M \cdot n^{\omega})$ distance product algorithm, M is the largest weight of an edge

  $\Longrightarrow$ Max Weight Triangle in $O(M \cdot n^{\omega})$ (Pseudopolynomial)

- [Chan, '07] $O(n^3 \log\log^3 n / \log^2 n)$ distance product

  $\Longrightarrow$ Max Weighted Triangle in $O(n^3 \log\log^3 n / \log^2 n)$

"Easy" Weighted Triangle Algorithms

- [Zwick, '02] $O(M \cdot n^{\omega})$ distance product algorithm, M is the largest weight of an edge

  $\Longrightarrow$ Max Weight Triangle in $O(M \cdot n^{\omega})$ (Pseudopolynomial)

- [Chan, '07] $O(n^3 \log\log^3 n / \log^2 n)$ distance product

  $\Longrightarrow$ Max Weighted Triangle in $O(n^3 \log\log^3 n / \log^2 n)$

Truly Sub-Cubic Algorithm for Max Weighted Triangle?

Using Dominance Product we get:

- Deterministic Algorithm [VW06]

  $O(B \cdot n^{(3+\omega)/2}) \leq O(B \cdot n^{2.688})$, where B is the bit precision

- Randomized (Strongly Polynomial) Algorithm [VW06]

$$O(n^{(3+\omega)/2} \log n) \leq O(n^{2.688})$$

Using Dominance Product we get:

- Deterministic Algorithm [VW06]

  $O(B \cdot n^{(3+\omega)/2}) \leq O(B \cdot n^{2.688})$, where B is the bit precision

- Randomized (Strongly Polynomial) Algorithm [VW06]

$$O(n^{(3+\omega)/2} \log n) \leq O(n^{2.688})$$

Aside: It is already known how to find a max node weighted triangle in $O(n^{\omega})$ [CzumajLingas07].

We can get for *all edges* the max node weighted triangle including the edge in $O(n^{2.58})$ time [VWY06].

Deterministic Algorithm: Outline

Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least K?

   $\longrightarrow$ dominance product instance.

Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least K?
   $\longrightarrow$ dominance product instance.
2. Do binary search on K to find the maximum weight W of a triangle.

Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least K?
   $\longrightarrow$ dominance product instance.
2. Do binary search on K to find the maximum weight W of a triangle.
3. Find a triangle of weight W.

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \longrightarrow$

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \to$

- row vector $A[i, ;] = (A[i, 1], \ldots, A[i, n])$ s.t.

$$A[i,j] = \begin{cases} K - w(i) & \text{if there is an edge from } i \text{ to } j \\ \infty & \text{otherwise.} \end{cases}$$

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \longrightarrow$

- row vector $A[i,;] = (A[i,1], \ldots, A[i,n])$ s.t.

$$A[i,j] = \begin{cases} K - w(i) & \text{if there is an edge from } i \text{ to } j \\ \infty & \text{otherwise.} \end{cases}$$

- column vector $B[;,i] = (B[1,i], \ldots, B[n,i])$ s.t.

$$B[j,i] = \begin{cases} w(i) + w(j) & \text{if there is an edge from } i \text{ to } j \\ -\infty & \text{otherwise.} \end{cases}$$

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \longrightarrow$

- row vector $A[i, ;] = (A[i, 1], \ldots, A[i, n])$ s.t.

$$A[i,j] = \begin{cases} K - w(i) & \text{if there is an edge from } i \text{ to } j \\ \infty & \text{otherwise.} \end{cases}$$

- column vector $B[; , i] = (B[1, i], \ldots, B[n, i])$ s.t.

$$B[j,i] = \begin{cases} w(i) + w(j) & \text{if there is an edge from } i \text{ to } j \\ -\infty & \text{otherwise.} \end{cases}$$

$A[i,j] \leq B[j,k] \iff K \leq w(i) + w(k) + w(j)$ and $(i,j),(j,k) \in E$

Step 1 cont.

Recall $C[i,j] = (A \odot B)[i,k] = |\{j \ : \ A[i,j] \leq B[j,k]\}|$.

Step 1 cont.

Recall $C[i,j] = (A \odot B)[i,k] = |\{j \; : \; A[i,j] \leq B[j,k]\}|$.

$(A \odot B)[i,k] \neq 0$ iff

$\exists j$ such that there is a path $i \to j \to k$ and $w(i) + w(k) + w(j) \geq K$

Step 1 cont.

Recall $C[i, j] = (A \odot B)[i, k] = |\{j \ : \ A[i, j] \leq B[j, k]\}|$.

$(A \odot B)[i, k] \neq 0$ iff

$\exists j$ such that there is a path $i \to j \to k$ and $w(i) + w(k) + w(j) \geq K$

Hence to check whether there is a triangle of weight at least K, compute $C = A \odot B$ and check for an entry $C[i, j] \neq 0$ such that $(i, j) \in E$.

Runtime

Runtime

Let B be the max number of bits needed to represent a weight.

Runtime

Let B be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$.

Runtime

Let B be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$.

But this algorithm is not strongly polynomial because of the binary search.

Runtime

Let B be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$.

But this algorithm is not strongly polynomial because of the binary search.

Can use random sampling of weighted triangles to obtain a $O(n^{\frac{3+\omega}{2}} \log n)$ strongly polynomial randomized algorithm.

Talk outline

1. Some definitions
2. Maximum weighted triangle
3. Computing bits of the distance product
4. All pairs bottleneck paths
5. All pairs nondecreasing paths
6. Open problems

The distance product

Recall $(A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}$.

The distance product

Recall $(A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}$.

The distance product is used to compute APSP.

The distance product

Recall $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

The distance product is used to compute APSP.

The complexity of computing the distance product of two $n \times n$ matrices is the same as that of computing all pairs shortest distances in an n vertex graph.

The distance product

Recall $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

The distance product is used to compute APSP.

The complexity of computing the distance product of two $n \times n$ matrices is the same as that of computing all pairs shortest distances in an n vertex graph.

The current best algorithm for arbitrary real weights is by Chan in $O(n^3 \log\log^3 n / \log^2 n)$.

Computing bits of the distance product

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Compute $D(K) = (A(K) \odot B)$

and $C(K)[i,j] = \begin{cases} 1 & \text{if } D(K)[i,j] = n \\ 0 & \text{otherwise.} \end{cases}$

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Compute $D(K) = (A(K) \odot B)$
$\rightarrow D(K)[i,j] \neq n \iff \exists k. K - A[i,k] > B[k,j]$

and $C(K)[i,j] = \begin{cases} 1 & \text{if } D(K)[i,j] = n \\ 0 & \text{otherwise.} \end{cases}$

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Compute $D(K) = (A(K) \odot B)$
$\rightarrow D(K)[i,j] \neq n \iff \exists k. K - A[i,k] > B[k,j]$

and $C(K)[i,j] = \begin{cases} 1 & \text{if } D(K)[i,j] = n \\ 0 & \text{otherwise.} \end{cases}$

Then $C(K)[i,j] = 1 \iff \min_k(A[i,k] + B[k,j]) \geq K$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Compute $D(K) = (A(K) \odot B)$
$\rightarrow D(K)[i,j] \neq n \iff \exists k. K - A[i,k] > B[k,j]$

and $C(K)[i,j] = \begin{cases} 1 & \text{if } D(K)[i,j] = n \\ 0 & \text{otherwise.} \end{cases}$

Then $C(K)[i,j] = 1 \iff \min_k(A[i,k] + B[k,j]) \geq K$.

Most significant bit is then $C(\frac{W}{2})$ where W is the *smallest power of* 2 *larger than the largest distance*.

Computing bits of the distance product

Computing bits of the distance product

$$C(K)[i,j] = 1 \iff \min_k(A[i,k] + B[k,j]) \geq K$$

The second most significant bit of $(A \star B)[i,j]$ is

$$(\neg C(W)[i,j] \wedge C(\frac{3W}{4})[i,j]) \vee (\neg C(\frac{W}{2})[i,j] \wedge C(\frac{W}{4})[i,j]).$$

Only compute 4 dominance products.

Computing bits of the distance product

$$C(K)[i,j] = 1 \iff \min_k(A[i,k] + B[k,j]) \geq K$$

The second most significant bit of $(A \star B)[i,j]$ is

$$(\neg C(W)[i,j] \wedge C(\frac{3W}{4})[i,j]) \vee (\neg C(\frac{W}{2})[i,j] \wedge C(\frac{W}{4})[i,j]).$$

Only compute 4 dominance products.

The ℓth bit is

$$\bigvee_{s=0}^{2^{\ell-1}-1} [\neg C(W(1 - \frac{s}{2^{\ell-1}}))[i,j] \wedge C(W(1 - \frac{s}{2^{\ell-1}} - \frac{1}{2^{\ell}}))[i,j]].$$

Here need $O(2^{\ell})$ dominance products.

Computing bits of the distance product

Thm. The first $\mathcal{B}$ most significant bits of the distance product of two $n \times n$ matrices can be computed in $O(2^{\mathcal{B}} n^{\frac{3+\omega}{2}})$ time.

One can compute $(\frac{3-\omega}{2} - \varepsilon) \log n$ bits in $O(n^{3-\varepsilon})$ time.

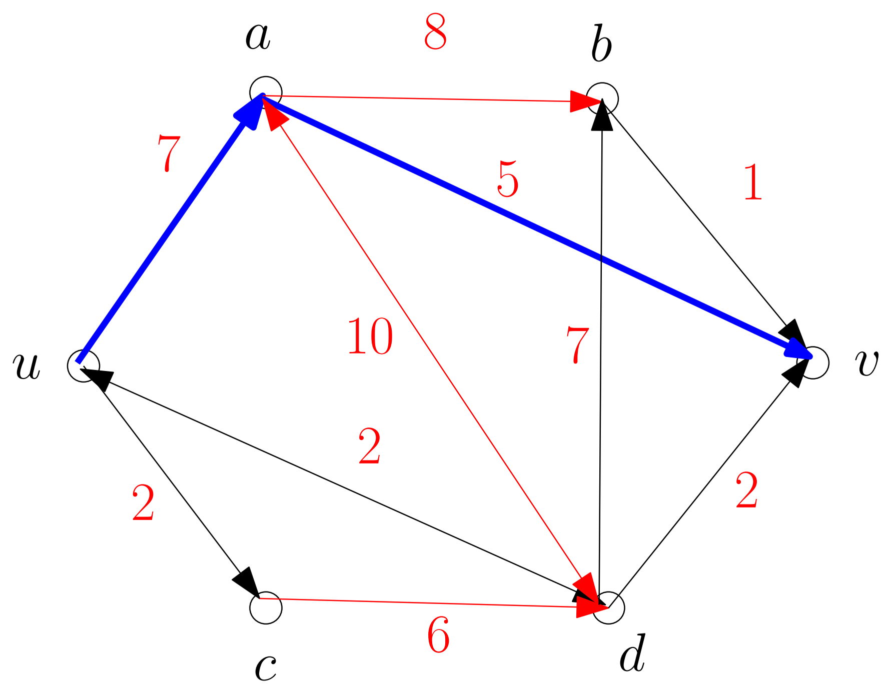
Talk outline

1. Some definitions
2. Maximum weighted triangle
3. Computing bits of the distance product
4. All pairs bottleneck paths
5. All pairs nondecreasing paths
6. Open problems

Bottleneck paths

The bottleneck edge of a path in a graph from vertex u to vertex v is the edge of smallest weight.

In many applications (e.g. max flow), the path of maximum bottleneck is needed.



$ab : 8$	$bu : -\infty$
$ac : 2$	$ub : 7$
$ad : 10$	$cd : 6$
$au : 2$	$dc : 2$
$av : 5$	$ud : 7$
$bc : -\infty$	$du : 2$
$cb : 6$	$uv : 5$

In this talk we will consider the all pairs max bottlenecks problem.

Bottleneck paths – related work

single source:

- Folklore: in $O(m + n \log n)$ by Dijkstra, using Fibonacci heaps.

all pairs:

- Pollack 1960: introduced the problem, first cubic algorithm.
- Hu 1961: undirected, edge weighted using max spanning tree. Now $O(n^2)$.
- Shapira, Yuster, Zwick 2007: directed, node weighted in $O(n^{2.58})$.
- V., Williams, Yuster 2007: directed, edge weighted in $O(n^{2.79})$.

MaxMin product

The MaxMin product of two $n \times n$ matrices A and B is

$$(A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

MaxMin product

The MaxMin product of two $n \times n$ matrices A and B is

$$(A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Adjacency matrix for weighted graph $G = (V, E, w)$: $A[i,j] = w_{ij}$.

$(A \bullet A)[i,j]$ is the maximum bottleneck edge weight over all paths of length 2 from i to j.

$\underbrace{A \bullet A \bullet \ldots \bullet A}_{n \text{ times}}$: the maximum bottleneck weights for all vertex pairs.

MaxMin product

MaxMin product

The MaxMin product is used to compute all pairs maximum bottleneck paths (APBP), similar to how one uses distance product for APSP.

MaxMin product

The MaxMin product is used to compute all pairs maximum bottleneck paths (APBP), similar to how one uses distance product for APSP.

Computing the MaxMin product of two $n \times n$ matrices takes the same time as computing all pairs bottleneck distances in an n vertex graph. [AhoHopcroftUllman74]

Computing the MaxMin product faster

$C = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$

We use the dominance product again:

$(A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|$.

We will proceed as follows:

Computing the MaxMin product faster

$$C = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$$

We use the dominance product again:

$(A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|$.

We will proceed as follows:

1. compute for all i, j, $a_{ij} = \max_k\{A[i,k] \mid A[i,k] \leq B[k,j]\}$,
2. compute for all i, j, $b_{ij} = \max_k\{B[k,j] \mid B[k,j] \leq A[i,k]\}$,

Computing the MaxMin product faster

$C = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$

We use the dominance product again:

$(A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|$.

We will proceed as follows:

1. compute for all i, j, $a_{ij} = \max_k\{A[i,k] \mid A[i,k] \leq B[k,j]\}$,
2. compute for all i, j, $b_{ij} = \max_k\{B[k,j] \mid B[k,j] \leq A[i,k]\}$,
3. set for all i, j, $C[i,j] = \max\{a_{ij}, b_{ij}\}$.

Computing the MaxMin product faster

$C = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$

We use the dominance product again:

$(A \odot B)[i,j] = |\{k \;:\; A[i,k] \leq B[k,j]\}|$.

We will proceed as follows:

1. compute for all i, j, $a_{ij} = \max_k\{A[i,k] \mid A[i,k] \leq B[k,j]\}$,
2. compute for all i, j, $b_{ij} = \max_k\{B[k,j] \mid B[k,j] \leq A[i,k]\}$,

   $(\max, \leq)$-Product!, $(\min, \leq)$-Product analogous.
3. set for all i, j, $C[i,j] = \max\{a_{ij}, b_{ij}\}$.

Computing the MaxMin product faster

We want $a_{ij} = \max_k \{A[i,k] \mid A[i,k] \leq B[k,j]\}$.

1. Take the rows of A and sort the entries of each row.
2. Bucket the entries of each row of A, in their sorted order into s roughly equal buckets.

$$A = \begin{pmatrix} 10 & -1.1 & 5.1 & 3.2 \\ 2 & 3 & 7 & 1 \\ 0 & -1 & -2 & -3 \\ 7 & 2.1 & 4 & 2.1 \end{pmatrix}$$

$$\begin{array}{lllll} \text{row } 1: & A[1,2], & A[1,4], & A[1,3], & A[1,1] \\ \text{row } 2: & A[2,4], & A[2,1], & A[2,2], & A[2,3] \\ \text{row } 3: & A[3,4], & A[3,3], & A[3,2], & A[3,1] \\ \text{row } 4: & A[4,4], & A[4,2], & A[4,3], & A[4,1] \end{array}$$

Computing the MaxMin product faster

3. For each bucket b create a matrix $A(b)$ containing only the elements in bucket b and ∞ in all other entries.

$$A(1) = \begin{pmatrix} \infty & -1.1 & \infty & 3.2 \\ 2 & \infty & \infty & 1 \\ \infty & \infty & -2 & -3 \\ \infty & 2.1 & \infty & 2.1 \end{pmatrix} \quad A(2) = \begin{pmatrix} 10 & \infty & 5.1 & \infty \\ \infty & 3 & 7 & \infty \\ 0 & -1 & \infty & \infty \\ 7 & \infty & 4 & \infty \end{pmatrix}$$

Computing the MaxMin product faster

4. Compute $A(b) \odot B$ for each bucket b.

$$A(2) \odot A = \begin{pmatrix} 10 & \infty & 5.1 & \infty \\ \infty & 3 & 7 & \infty \\ 0 & -1 & \infty & \infty \\ 7 & \infty & 4 & \infty \end{pmatrix} \odot \begin{pmatrix} 10 & -1.1 & 5.1 & 3.2 \\ 2 & 3 & 7 & 1 \\ 0 & -1 & -2 & -3 \\ 7 & 2.1 & 4 & 2.1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 2 & 1 & 2 & 2 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

This tells us for every bucket b and each i, j, the number of coords k such that $A[i,k]$ is in bucket b and $A[i,k] \leq B[k,j]$.

This step takes $O(sn^{\frac{3+\omega}{2}})$.

Computing the MaxMin product faster

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i,k]$ such that $A[i,k] \le B[k,j]$.

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i, k]$ such that $A[i, k] \le B[k, j]$.

   For each i, j, search that bucket for k - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i, k]$ such that $A[i, k] \leq B[k, j]$.

   For each i, j, search that bucket for k - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

6. The overall runtime is maximized for $s = n^{\frac{3-\omega}{4}}$ and the runtime is then $O(n^{\frac{9+\omega}{4}}) = O(n^{2.85})$.

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i,k]$ such that $A[i,k] \leq B[k,j]$.

   For each i, j, search that bucket for k - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

6. The overall runtime is maximized for $s = n^{\frac{3-\omega}{4}}$ and the runtime is then $O(n^{\frac{9+\omega}{4}}) = O(n^{2.85})$.

7. You can do slightly better by using sparse dominance $\longrightarrow O(n^{2.79})$.

Talk outline

1. Some definitions
2. Maximum weighted triangle
3. Computing bits of the distance product
4. All pairs bottleneck paths
5. All pairs nondecreasing paths
6. Open problems

Nondecreasing paths

A path from s to t in a weighted graph G is nondecreasing if the consecutive weights on the path are nondecreasing:

$$s \xrightarrow{1} u_1 \xrightarrow{20} u_2 \xrightarrow{30} t$$

A minimum nondecreasing path from s to t is the path with minimum last edge over all nondecreasing paths.

Nondecreasing paths

Why do we want the min last edge?

Nondecreasing paths

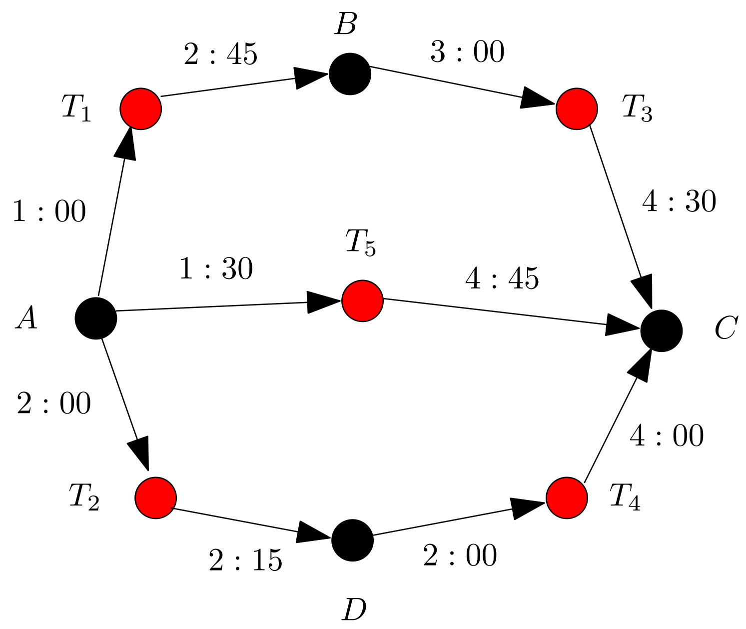
Why do we want the min last edge?

Train trip scheduling! You have a time table with train arrival, departure times, origins and destinations.

Want to know, for all origins s and destinations t, how to hop from one train to another to get to t as early as possible.

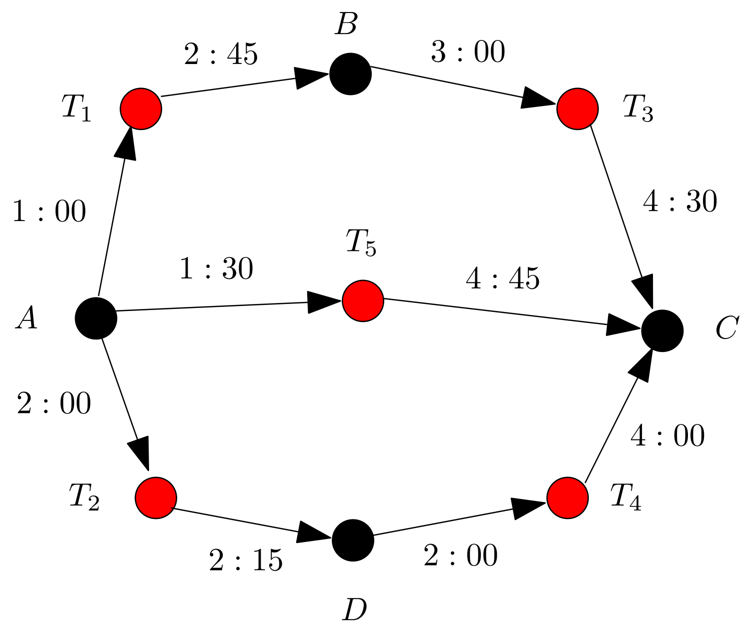
A train schedule graph

T_1	A	B	$1:00$	$2:45$
T_2	A	D	$2:00$	$2:15$
T_3	B	C	$3:00$	$4:30$
T_4	D	C	$2:00$	$4:00$
T_5	A	C	$1:30$	$4:45$



A train schedule graph

T_1	A	B	$1:00$	$2:45$
T_2	A	D	$2:00$	$2:15$
T_3	B	C	$3:00$	$4:30$
T_4	D	C	$2:00$	$4:00$
T_5	A	C	$1:30$	$4:45$



All pairs nondecreasing paths (APNP): for all pairs of nodes s, t find the minimum weight of a last edge over all nondecreasing paths from s to t.

Related work

Single source version has a long history; studied alongside SSSP.

Minty 1958, Moore 1959: single source version in $O(mn)$

Dijkstra + Fibonacci heaps: single source version in $O(m + n \log n)$.

The best running time in terms of n for APNP: $O(n^3)$.

$(\min, \leq)$-Product

Recall: $(\min, \leq)$-Product:

$$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] :\ A[i,k] \leq B[k,j]\}.$$

$(\min, \leq)$-Product

Recall: $(\min, \leq)$-Product:

$$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : A[i,k] \leq B[k,j]\}.$$

For edge weighted graph G, if

$A[i,j] = w_{ij}$, $w_{ii} = -\infty$, $w_{ij} = \infty$ if $(i,j) \notin E$:

$A \otimes A$ gives all pairs min nondecreasing paths of length ≤ 2.

$(\min, \leq)$-Product

Recall: $(\min, \leq)$-Product:

$$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] :\ A[i,k] \leq B[k,j]\}.$$

For edge weighted graph G, if

$A[i,j] = w_{ij}$, $w_{ii} = -\infty$, $w_{ij} = \infty$ if $(i,j) \notin E$:

$A \otimes A$ gives all pairs min nondecreasing paths of length ≤ 2.

$\underbrace{A \otimes A \otimes \ldots, \otimes A}_{k \text{ times}}$: all pairs min nondecreasing paths of length $\leq k$.

$(\min, \leq)$-Product

Recall: $(\min, \leq)$-Product:

$$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}.$$

For edge weighted graph G, if

$A[i,j] = w_{ij}$, $w_{ii} = -\infty$, $w_{ij} = \infty$ if $(i,j) \notin E$:

$A \otimes A$ gives all pairs min nondecreasing paths of length ≤ 2.

$\underbrace{A \otimes A \otimes \ldots, \otimes A}_{k \text{ times}}$: all pairs min nondecreasing paths of length $\leq k$.

Unclear how to compute transitive closure under $(\min, \leq)$-Product efficiently...

APNP

IDEA (GalilMargalit97, Zwick02 . . .): Handle short and long paths separately.

APNP

IDEA (GalilMargalit97, Zwick02 . . .): Handle short and long paths separately.

Short paths: at most s edges. Finding all pairs min nondecreasing paths on at most s edges:

$C_1 = A$

For $k = 2, \ldots, s$: $C_k = C_{k-1} \circledcirc A$.

This takes $O(sn^{2+\omega/3})$ time.

Also, using the witnesses keep track of actual paths of length at most s.

Long nondecreasing paths

Long nondecreasing paths

Long paths: Consider min nondecreasing path P, which is minimal but on at least s edges.

$$P = i \to u_1 \to u_2 \to \ldots \to u_s \to \ldots \to j.$$

Long nondecreasing paths

Long paths: Consider min nondecreasing path P, which is minimal but on at least s edges.

$$P = i \rightarrow u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_s \rightarrow \ldots \rightarrow j.$$

The subpath from i to u_s can be replaced WLOG with a minimum nondecreasing path from i to u_s of length s, without changing the minimality of the path from i to j.

Long nondecreasing paths

Long paths: Consider min nondecreasing path P, which is minimal but on at least s edges.

$$P = i \to u_1 \to u_2 \to \ldots \to u_s \to \ldots \to j.$$

The subpath from i to u_s can be replaced WLOG with a minimum nondecreasing path from i to u_s of length s, without changing the minimality of the path from i to j.

When computing C_s one will find a minimum nondecreasing path from i to u_s and it will be of length s by the minimality of P.

Long nondecreasing paths

A best nondecreasing path from i to j on at least s nodes can be obtained by *continuing* some path of length s obtained when computing C_s.

Long nondecreasing paths

A best nondecreasing path from i to j on at least s nodes can be obtained by *continuing* some path of length s obtained when computing C_s.

Consider all min nondecreasing paths of length s found when computing C_s (ignore shorter paths). We have at most n^2 such paths.

Long nondecreasing paths

A best nondecreasing path from i to j on at least s nodes can be obtained by *continuing* some path of length s obtained when computing C_s.

Consider all min nondecreasing paths of length s found when computing C_s (ignore shorter paths). We have at most n^2 such paths.

Lemma (Zwick02, Chan07...): Given a collection of $\leq n^2$ subsets of vertices, each of size s, one can find in $O(sn^2)$ time a set of $n \log n/s$ vertices, hitting every one of the subsets.

Long nondecreasing paths

A best nondecreasing path from i to j on at least s nodes can be obtained by *continuing* some path of length s obtained when computing C_s.

Consider all min nondecreasing paths of length s found when computing C_s (ignore shorter paths). We have at most n^2 such paths.

Lemma (Zwick02, Chan07...): Given a collection of $\leq n^2$ subsets of vertices, each of size s, one can find in $O(sn^2)$ time a set of $n \log n/s$ vertices, hitting every one of the subsets.

In $O(sn^2)$ time we obtain a vertex set S of size $n \log n/s$ hitting for every pair of vertices i, j some minimal long minimum nondecreasing path from i to j (if one exists).

Long nondecreasing paths

We have a set S of size $n \log n / s$. We want for all pairs of vertices i, j a minimum nondecreasing path from i to j going through S.

Long nondecreasing paths

We have a set S of size $n \log n/s$. We want for all pairs of vertices i, j a minimum nondecreasing path from i to j going through S.

We show that one can find all pairs min nondecreasing paths going through a given vertex in $O(n^2 \log n)$ time. So all pairs min nondecreasing paths through S can be found in $O((n^3 \log^2 n)/s)$ time.

APNP

All pairs min nondecreasing paths of length at most s can be found in $O(sn^{2+\omega/3})$ time.

Minimal best nondecreasing paths of length at least s can be found in $O((n^3 \log^2 n)/s)$ time.

To obtain APNP, take for all pairs the minimum of the short paths and long paths min weights.

Setting s to $\Theta(n^{\frac{1-\omega/3}{2}} \log n)$, compute APNP in $O(n^{\frac{15+\omega}{6}} \log n) = O(n^{2.9})$ time.

Open Problems

1. dominance product in n^{ω}?

2. remove bucketting?

3. truly subcubic distance product?

Thank You!

All pairs through a given vertex T

1. Find for each node u the minimum weight $W(u)$ of a last edge on a nondecreasing path from u to T

2. Find for each pair of nodes u, v the minimum weight of a last edge on a nondecreasing path from T to v starting with an edge of weight $\geq W(u)$. $\leftarrow$ use data structure.

3. Do all of this in $O(n^2 \log n)$ time.

Computing $W(u)$

1. For each u, sort inedges and store in binary search tree, so that successors can be found in $O(\log n)$ time.

2. start from T; For current vertex u, let w be the weight out of T used to get to u;

   If w is the first weight used to get to u, set $W(u) = w$.

3. Let w' be the weight used to enter u. In $O(\log n)$ time find the first inedge (v, u) of u in sorted order with weight $\geq w'$.

   Delete (v, u) from bintree and graph, recurse on v with w and $w(v, u)$.

4. This all takes $O(m \log n)$ and computes $W(u)$ for all u.

Last edge weights for paths from T to v

1. For each v, create a bintree $T(v)$ with the edges out of T as leaves.
   $\leftarrow O(n^2 \log n)$ time to create all $T(v)$.

2. Fill in two nums for each node:

   (a) min weight of leaf in subtree

   (b) min last weight edge on path from T to v starting with an edge in subtree;

Last edge weights for paths from T to v

1. For each v, create a bintree $T(v)$ with the edges out of T as leaves. $\leftarrow O(n^2 \log n)$ time to create all $T(v)$.

2. Fill in two nums for each node:

   (a) min weight of leaf in subtree Fill in at creation of tree.

   (b) min last weight edge on path from T to v starting with an edge in subtree;

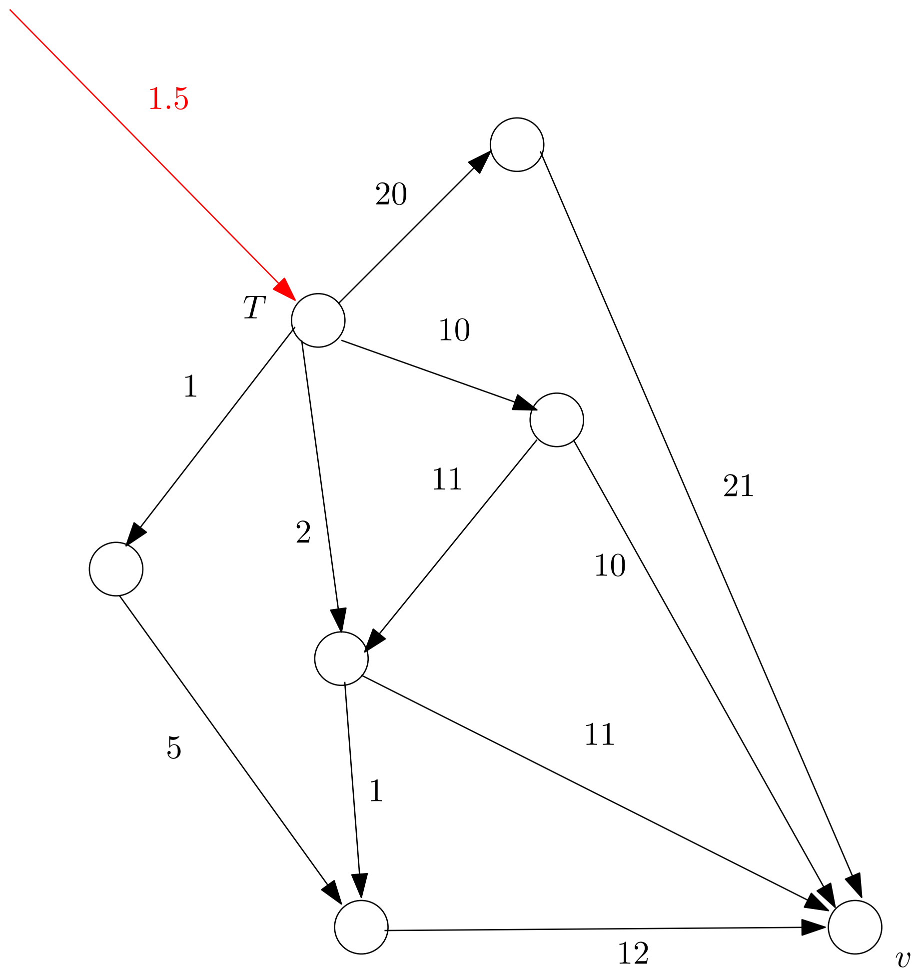
Last edge weights for paths from T to v

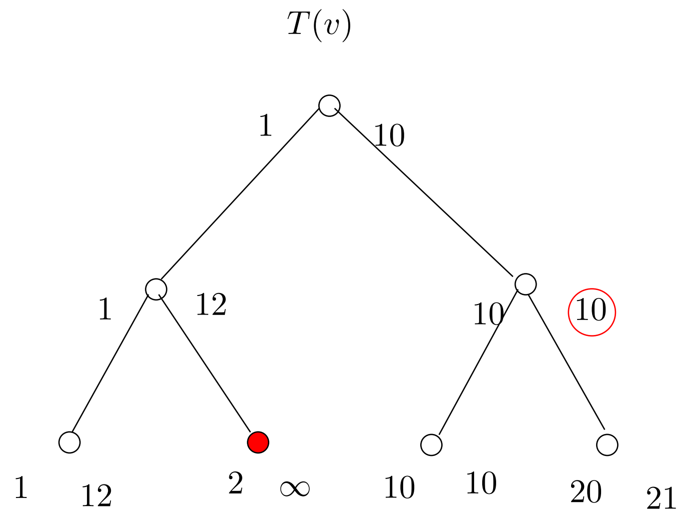
1. For each v, create a bintree $T(v)$ with the edges out of T as leaves.
   $\leftarrow O(n^2 \log n)$ time to create all $T(v)$.

2. Fill in two nums for each node:

   (a) min weight of leaf in subtree Fill in at creation of tree.

   (b) min last weight edge on path from T to v starting with an edge in subtree;Fill in with a second search.


1.5
T
20
10
1
2
11
21
10
5
11
1
12
v


T(v)
1
10
1
12
10
10
1
12
2
∞
10
10
20
21

Second search

store for each v the outedges in a binary tree $Out(v)$, sorted in nondecreasing order of weights

start from T and go thru its outedges (T, v) in nonincreasing order, running the following ALG($v, w(T, v), w(T, v)$)

ALG(v, w, w'):

let w be weight of edge out of T used to get to v; let w' be weight used to enter v. find w leaf in $T(v)$, if w' is current smallest weight into v then update second number of leaf,

go up path in tree and update second nums on path if necessary in $O(\log n)$ time.

then, for all edges (v, u) out of v with weights $\geq w'$, delete (v, u) from graph and $Out(v)$ in $O(\log n)$ per edge, recurse on u with w and

$w(u, v)$. this takes $O(n^2 \log n)$ over all.

for all pairs u, v do: get min weight w of last edge on nondec path from u to T. in tree for v, find w, then walk up path to root, checking right children of nodes to find min weight w' on a nondec path from T to v starting with a weight $\geq w$. this takes $O(\log n)$ time per pair u, v, so $O(n^2 \log n)$ overall.