Data Structures and Graph Algorithms

Shortest Paths

Kurt Mehlhorn
Max-Planck-Institut für Informatik

Contents

1. the worst case running time of many graph algorithms can be improved by clever data structures
   - priority queues for Dijkstra's shortest path algorithm $O(n^2) \Longrightarrow O(m + n \log n)$
   - dynamic trees for maxflow algorithms $O(n^2 \sqrt{m}) \Longrightarrow O(nm)$
   - mergeable priority queues for general weighted matchings

     $$O(n^3) \Longrightarrow O(nm \log n)$$

2. what is the effect on "actual" running times on synthetic and real inputs
   - priority queues for Dijkstra's shortest path algorithm
   - dynamic trees for maxflow algorithms
   - mergeable priority queues for general weighted matchings

3. how large are the gains and can we explain them ???

Dijkstra's Single Source Shortest Path Algorithm

$G = (V,E)$ directed graph, $s \in V$ source node, $c : E \mapsto \mathbb{R}_{\geq 0}$ edge costs

Dijkstra's Algorithm

$d(s) = 0$ and $d(v) = \infty$ for $v \neq s$; tentative distances

declare all nodes unscanned;

while there is an unscanned node

{ let u be the unscanned node with minimal tentative distance;

  forall edges $e = (u,v)$ out of u

  { $C = d(u) + c(e)$;

    if ($C < d(v)$) set $d(v) = C$;

  }

  declare u scanned;

}

Dijkstra iterated over all nodes to find the unscanned u with minimal $d(u)$

running time $\Theta(n^2 + m)$ it is Θ and not just O !!!!!

Dijkstra's Algorithm with Priority Queues

the unscanned nodes u with $d(u) < \infty$ are stored in a priority queue

```
define a priority queue for the nodes of G;                                   init
set d(s) = 0 and d(v) = ∞ for v ≠ s and declare all nodes unscanned
PQ.insert(s,0);                                                               insert
while (! PQ.is_empty( ) )                                                     is_empty
{ select u ∈ PQ with d(u) minimal and remove it; declare u scanned            extract_min
  forall edges e = (u,v)
  { if ( D = d(u) + c(e) < d(v) )
    { if ( d(v) == ∞ )
      { PQ.insert(v,D);   // v has been reached  }                            insert
      else
      { PQ.decrease_p(v,D);                      }                            decrease_p

      d(v) = D;
    }
  }
}
```

Dijkstra's Algorithm with Priority Queues

```
define a priority queue for the nodes of G;                                   init
set d(s) = 0 and d(v) = ∞ for v ≠ s and declare all nodes unscanned
PQ.insert(s,0);                                                           1 insert
while (! PQ.is_empty( ) )                                               n is_empty
{ select u ∈ PQ with d(u) minimal and remove it; declare u scanned   n extract_min
  forall edges e = (u,v)
  { if ( D = d(u) + c(e) < d(v) )
    { if ( d(v) == ∞ )
      { PQ.insert(v,D);    // v has been reached   }                n − 1 insert
      else
      { PQ.decrease_p(v,D);                        }    up to m − (n − 1) decrease_p

      d(v) = D;
    }
  }
}
```

$$\text{time} = \Theta(n + m + T_{init} + n \cdot (T_{is\_empty} + T_{extract\_min} + T_{insert})) + O(m \cdot T_{decrease\_p})$$

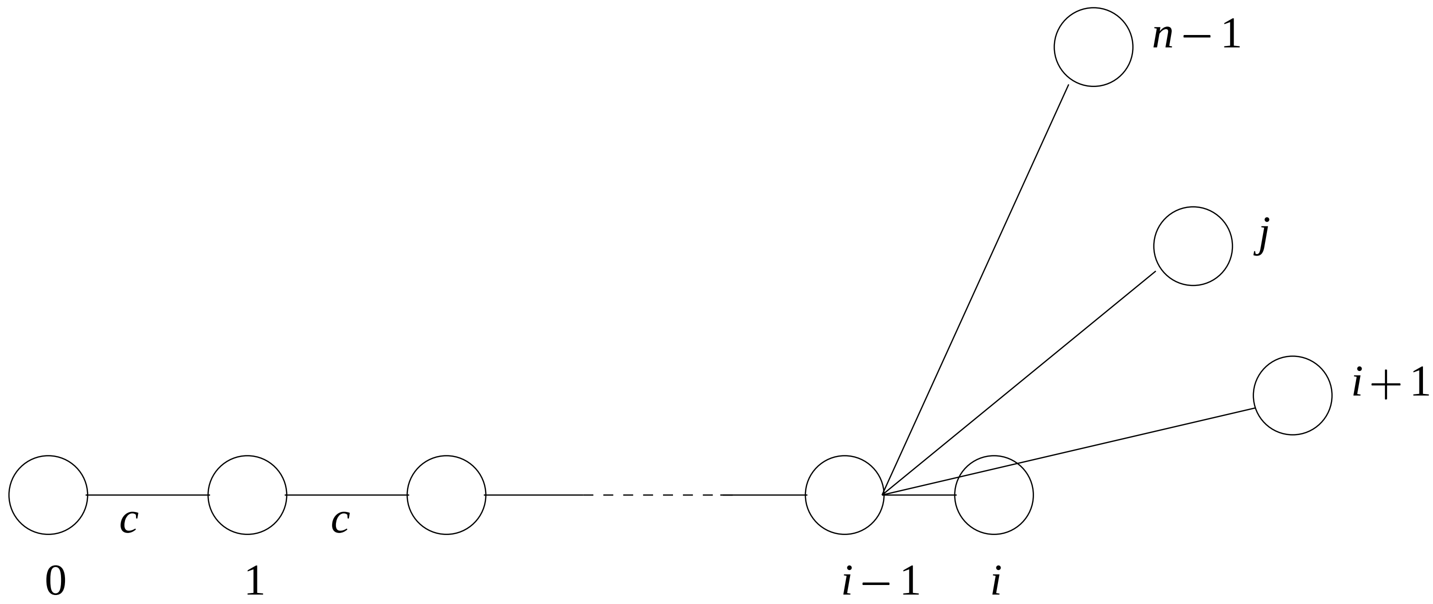
Priority Queue Implementations

$$\text{time} = \Theta(n + m + T_{init} + n \cdot (T_{is\_empty} + T_{extract\_min} + T_{insert})) + O(m \cdot T_{decrease\_p})$$

	insert	*extract_min*	*decrease_p*	worst-case T
no data structure	1	n	1	$\Theta(n^2 + m)$
binary heaps	$\log n$	$\log n$	$\log n$	$\Theta(n \log n) + O(m \log n)$
Fib heaps	$\log n$	$\log n$	1	$\Theta(n \log + m)$

Fib heaps have larger constant factors than bin heaps

A Worst-Case Example



A worst case graph for Dijkstra's algorithm. All edges $(i, i+1)$ have cost c and an edge (i, j) with $i+1 < j$ has cost $c_{i,j}$. The $c_{i,j}$ are chosen such that the shortest path tree with root 0 is the path $0, 1, \ldots, n-1$ and such that the shortest path tree that is known after removing node $i-1$ from the queue is as shown. Among the edges out of node $i-1$ the edge $(i-1, i)$ is the shortest, the edge $(i-1, n-1)$ is the second shortest, and the edge $(i-1, i+1)$ is the longest. Every decrease prio makes smallest key in PQ.

source: LEDAbook, Section on priority queues

Experiments [Cherkassky-Goldberg-Radzik, LEDAbook]

Instance	*f_heap*	*p_heap*	*k_heap*	*bin_heap*	*list_pq*	*r_heap*	*m_heap*
s,r,S	0.36	0.34	0.35	0.34	0.51	0.33	0.35
s,r,L	0.38	0.36	0.37	0.34	0.54	0.35	0.54
s,w,S	1.86	1.09	3.77	1.38	1	0.76	2.68
s,w,L	1.87	1.1	3.68	1.34	1	0.77	8.49
l,r,S	4.96	3.19	5.2	3.36	-	2.52	2.52
l,r,L	6.61	4.81	6.4	4.49	-	3.76	3.38
l,w,S	3.32	2.56	9.17	3.79	-	1.63	3.11
l,w,L	2.91	1.92	7.65	3.22	-	2.57	2.55

$m = 500000$ and $n = 2000$ (s), or $n = 200000$ (l) nodes.

random graphs (r) with random edge weights in $[0..M-1]$, where $M = 100$ (S) or $M = 100000$ (L),

worst case graphs (w) with $c = 0$ (S) or $c = 10000$ (L).

bin_heap $\ll$ *list_pq* and *bin_heap* $\ll$ *fib_heap* for random graphs and *f_heap* $\ll$ *bin_heap* for worst-case graphs with large n

Noshita's Average Case Analysis

- $G = (V, E)$ arbitrary directed graph, s source node
- for every $v \in V$ let $C(v)$ be a set of non-negative real numbers of cardinality $indeg(v)$.
- the assignment of the costs in $C(v)$ to the edges into v is made at random, i.e., probability space consists of $\prod_v indeg(v)!$ many assignments of edge costs to edges.
- **Theorem [Noshita]:** The expected number of *decrease_p* operations is $O(n \log(m/n))$.

Proof:

- Left-right maxima in a permutation

$$3 \quad 1 \quad 4 \quad 7 \quad 2 \quad 5 \quad 6$$

- Exp[# left-right maxima in a random permutation of length k] $= H_k \le \ln k$
- $\text{prob}(j\text{-th element is a maximum}) = 1/j$
- Exp[# left-right maxima] $= \sum_{1 \le j \le k} 1/j = H_k$

- Consider a fixed node v, let $k = indeg(v)$, let $e_1, \ldots, e_k$ be the order in which the edges into v are relaxed, and let $u_i = source(e_i)$.
- $d(u_1) \leq d(u_2) \leq \ldots \leq d(u_k)$ since nodes are scanned according to increasing d.
- Edge e_i causes a *decrease_p* iff $i \geq 2$ and $d(u_i) + c(e_i) < \min\left\{d(u_j) + c(e_j)\;;\; j < i\right\}$.
- number of *decrease_p*$(v, -)$ is bounded by the number of i such that

$$i \geq 2 \quad \text{and} \quad c(e_i) < \min\left\{c(e_j)\;;\; j < i\right\} .$$

- Since the order in which the edges into v are relaxed is independent of the costs assigned to them, the expected number of such i is simply the number of left-right maxima in a permutation of size k (minus 1, since $i = 1$ is not considered). Expectation = $H_k - 1$. Thus

$$\mathrm{E}[decrease\_p] \leq \sum_v H_{indeg(v)} - 1 \leq \sum_v \ln indeg(v) \leq n \ln(m/n)) \qquad \blacksquare$$

Consequence: expected running time of Dijkstra is $O(m + n\log(m/n)\log n)$ with the heap implementation of priority queues.

asymptotically more than $O(m + n\log n)$ only for $n = o(m)$ and $m = o(n \log n \log\log n)$.

Radix Heaps [Delgado-Fox, Ahuja-Mehlhorn-Orlin-Tarjan]

- edge costs are integers in $[0..C]$
- radix heaps exploit the binary representation of tentative distances.
- for numbers $a = \sum_{i \geq 0} \alpha_i 2^i$ and $b = \sum_{i \geq 0} \beta_i 2^i$ let

$$\text{(most distinguishing index)} \quad msd(a,b) = \begin{cases} \max\{i \,;\, \alpha_i \neq \beta_i\} & a \neq b \\ -1 & a = b \end{cases}$$

- If $a < b$ then a has a zero bit in position $i = msd(a,b)$ and b has a one bit.
- we assume that $msd(a,b)$ can be computed in $O(1)$ (can be removed)
- radix heap = sequence of buckets $B_{-1}, B_0, \ldots, B_K$ where $K = 1 + \lfloor \log C \rfloor$.
- *min* = tentative distance of node scanned most recently
- unscanned node v is stored in bucket B_i, where $i = \min(msd(min, d(v)), K)$.
- Buckets are organized as linear lists and every node keeps a handle to the list item representing it.

Operations on Radix Heaps

init create $K+1$ empty lists, time $O(K)$

insert$(v, d(v))$ inserts v into the appropriate list, time $O(1)$,

decrease_p$(v, d(v))$ removes v from the list containing it and inserts it into the appropriate queue, time $O(1)$

extract_min 1. find the minimum i such that B_i is non-empty.

2. time $O(1)$ if bit-vector of non-empty buckets is kept, $O(i)$ with linear search
3. if $i = -1$, extract an arbitrary element in B_{-1}. Time $O(1)$
4. if $i \geq 0$, iterate over B_i and set *min* to smallest tentative distance in B_i.
5. move elements in B_i to the appropriate new bucket.
6. total time for *extract_min* is $O(1)$ if $i = -1$ and $O(1 + |B_i|)$ if $i \geq 0$.
7. **Obs:** every node in bucket B_i moves to a bucket with smaller (!!!) index.
8. total time for searching for minimal i in all *extract_mins*: $O(n)$
9. total time for moving elements around in all *extract_mins*: $O(nK)$

Theorem 1 *With the Radix heap implementation of priority queues, Dijkstra's algorithm runs in time* $O(m + nK) = O(m + n \log C)$.

Lemma 1 *Let i be minimal such that B_i is non-empty and assume $i \geq 0$. Let min be the smallest element in B_i. Then $msd(min, x) < i$ for all $x \in B_i$.*

- distinguish the cases $i < K$ and $i = K$.
- min' = the old value of min.
- assume $i < K$: i is the most significant distinguishing index of min' and any $x \in B_i$
  - min' has a zero in bit position i
  - all $x \in B_i$ have a one in bit position i.
  - they agree in all positions with index larger than i.
  - Thus the most significant distinguishing index for min and x is smaller than i.
- Let us next assume that $i = K$ and consider any $x \in B_K$. Then $min' < min \leq x \leq min' + C$. Let $j = msd(min', min)$ and $h = msd(min, x)$. Then $j \geq K$. We want to show that $h < K$. Observe first that $h \neq j$ since min has a one bit in position j and a zero bit in position h. Let $min' = \sum_l \mu_l 2^l$.

  Assume first that $h < j$ and let $A = \sum_{l>j} \mu_l 2^l$. Then $min' \leq A + \sum_{l<j} 2^l \leq A + 2^j - 1$ since the j-th bit of min' is zero. On the other hand, x has a one bit in positions j and h and hence $x \geq A + 2^j + 2^h$. Thus $2^h \leq C$ and hence $h \leq \lfloor \log C \rfloor < K$.

  Assume next that $h > j$ and let $A = \sum_{l>h} \mu_l 2^l$. We will derive a contradiction. min' has a zero bit in positions h and j and hence $min' \leq A + 2^h - 1 - 2^j$. On the other hand x has a one bit in position h and hence $x \geq A + 2^h$. Thus $x - min' > 2^j \geq 2^K \geq C$, a contradiction.

Linear Expected Time [Meyer 00, Goldberg 01]

- edge costs are random integers in $[0\,..C]$
- $\mathit{min\_in\_cost}(v)$ = minimum cost of any edge into v.
- split queue into two parts
  - F = all nodes whose tentative distance label is known to be exact
  - B = the other nodes in the queue. B is organized as a radix heap.
- also maintain a value *min*.
- scan nodes as follows:
  - when F is non-empty, scan an arbitrary node in F.
  - when F is empty, the minimum is selected from B and *min* is set to it.
  - the nodes in the first non-empty bucket B_i are redistributed if $i \geq 0$.
  - modified redistribution process: when v is moved and $d(v) \leq \mathit{min} + \mathit{min\_in\_cost}(v)$, move v to F.
  - Observe that any future relaxation of an edge into v cannot decrease $d(v)$ and hence $d(v)$ is know to be exact at this point.

Theorem 2 (Meyer, Goldberg) *Let G be an arbitrary graph and let c be a random function from E to $[0..C]$. Then alg above runs in expected time $O(n+m)$.*

- As before nodes start out in B_K.
- when v is moved to a new bucket B_j but not yet to F,

  $d(v) \geq min + min\_in\_cost(v)$ and hence $j \geq \log min\_in\_cost(v)$.
- We conclude that the total charge to nodes in *extract_min* ops is

$$\sum_v (K - \log min\_in\_cost(v) + 1) \leq n + \sum_e (K - \log c(e)) \ .$$

- $K - \log c(e)$ is the number of leading zeros in the binary representation of $c(e)$ when written as a K-bit number.
- our edge costs are uniform random numbers in $[0..C]$ and $K = 1 + \lfloor \log C \rfloor$
- thus the expected number of leading zeros is $O(1)$.
- total expected cost of *extract_min* is $O(n+m)$. Time outside is also $O(n+m)$.

Limited Randomness

Theorem 3 *Let G be an arbitrary graph, let $c : E \mapsto [0..C]$ be an arbitrary cost function, let $0 \le k \le K = 1 + \lfloor \log C \rfloor$, and let $\bar{c}$ be obtained from c by making the last k bits of each cost random. Then the single source shortest path problem can be solved in expected time $O(n(K-k)+m)$.*

- By the proof of the preceding theorem, the total cost is

$$O(n + m + \sum_v (K - \log min\_in\_cost(v) + 1)$$

- Next observe that $min\_in\_cost(v)$ is the minimum of $indeg(v)$ numbers of which the last k bits are random. Thus

$$\begin{aligned} \mathrm{E}[K - \log min\_in\_cost(v)] &\le K - k + \sum_{e=(u,v)} \#\text{ of leading zeros in random part of } \bar{c}(e) \\ &\le K - k + O(indeg(v)) \end{aligned}$$