Fibonacci Heaps [Introduction to Algorithms by CLRS]

dual purpose:

1) supports a set of operations that constitutes a "mergeable heap".
2) several operations run in constant amortized time.

mergeable heap:

- MAKE-HEAP()
- INSERT(H, x) inserts element x, whose key has already been filled in, into heap H.
- MINIMUM(H) returns a pointer to the element in heap H whose key is minimum.
- EXTRACT-MIN(H)
- UNION(H_1, H_2)

Fibonacci heaps also support:

+ DECREASE-KEY(H, x, k) assigns to element x within H the new key value k, which we assume to be no greater than its current key value.
+ DELETE(H, x)

Procedure	Binary heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

desirable when number of these operations are small

application:

1) counting minimum spanning trees
2) single-source shortest paths.

drawbacks:

- large constant factors
- programming complexity

Fibonacci heap:

A Fibonacci heap is a collection of rooted trees that are min-heap ordered.

min-heap property:

the key of a node is greater than or equal to the key of its parent

Potential function

$t(H)$ the number of trees in the root list
$m(H)$ the number of marked nodes in H.

$$\Phi(H) = t(H) + 2m(H)$$

- assume that a unit of potential can pay for a constant amount of work

Maximum degree

assume that we know an upper bound $D(n)$ on the maximum degree of any node in an n-node Fibonacci heap.

$$D(n) \leq \lfloor \lg n \rfloor$$

when we support DECREASE-KEY and DELETE, $D(n) = O(\lg n)$.

Inserting a node

- just add it to the root list
- the increase in potential $t(H') = t(H)+1$, $m(H') = m(H)$ is 1,
  actual cost $O(1)$, amortized cost $O(1)+1 = O(1)$

Uniting two Fibonacci heaps

- change in potential

$$\Phi(H) - (\Phi(H_1) + \Phi(H_2))$$
$$= (t(H) + 2m(H)) - ((t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2)))$$
$$= 0$$

The amortized cost of FIB-HEAP-UNION is equal to its $O(1)$ actual cost.

Extracting the minimum node

```
FIB-HEAP-EXTRACT-MIN(H)
    z = H.min
    if z ≠ NIL
        for each child x of z
            add x to the root list of H
            x.p = NIL
        remove z from the root list of H
        if z == z.right
            H.min = NIL
        else H.min = z.right      ← not necessarily going to be the new minimum node.
            CONSOLIDATE(H)
        H.n = H.n - 1
    return z
```

```
CONSOLIDATE(H)                                    need to know upper bound.
  let A[0..D(H.n)] be a new array   // keep track of roots according to their degrees
  for i = 0 to D(H.n)
      A[i] = NIL
  for each node w in the root list of H
      x = w
      d = x.degree
      while A[d] ≠ NIL
          y = A[d]          // another node with the same degree as x
          if x.key > y.key
              exchange x with y
          FIB-HEAP-LINK(H, y, x)
          A[d] = NIL
          d = d+1
      A[d] = x
  H.min = NIL
  for i = 0 to D(H.n)
      if A[i] ≠ NIL
          if H.min == NIL
              create a root list of H containing just A[i]
              H.min = A[i]
          else insert A[i] into H's root list
              if A[i].key < H.min.key
                  H.min = A[i]
```

loop invariant: At the start of each iteration of the while loop d = x.degree

② the total amount of work in the for loop is at most proportional to $D(n) + t(H)$

```
FIB-HEAP-LINK(H, y, x)
1. remove y from the root list of H
2. make y a child of x, incrementing x.degree
3. y.mark = FALSE
```

amortized cost : try to show it is $O(D(n))$.

① $O(D(n))$ contribution comes from FIB-HEAP-EXTRACT-MIN processing at most $D(n)$ children of the minimum node,

Thus, the total actual work is ①② in $O(D(n) + t(H))$.

potential before extracting minimum $t(H) + 2m(H)$

potential afterwards is at most $(D(n)+1) + 2m(H)$

can scale up the units of potential to dominate the constant.

$$O(D(n) + t(H)) + ((D(n)+1) + 2m(H)) - (t(H) + 2m(H)) = O(D(n)) + \underline{O(t(H)) - t(H)} = O(D(n))$$

Bounding the maximum degree

to show the upper bound of $D(n)$ is $O(\lg n)$.

In particular, $D(n) \leq \lfloor \log_\phi n \rfloor$

$\phi = \frac{1+\sqrt{5}}{2}$

Lemma 19.1

Let x be any node in a Fibonacci heap, and suppose that $x.degree = k$. Let $y_1, y_2, \dots, y_k$ denote the children of x in the order in which they were linked to x from the earliest to the latest. Then $y_1.degree \geq 0$ and $y_i.degree \geq i-2$ for $i = 2, 3, \dots k$.

Lemma 19.2

For all integers $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

$$F_k = \begin{cases} 0 & , k=0 \\ 1 & , k=1 \\ F_{k-1} + F_{k-2} & , k \geq 2 \end{cases}$$

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fibonacci number satisfies $F_{k+2} \geq \phi^k$

inductive step:

$$\begin{aligned} F_{k+2} &= F_{k+1} + F_k \\ &\geq \phi^{k-1} + \phi^{k-2} \quad \text{(by the inductive hypothesis)} \\ &= \phi^{k-2}(\phi+1) \\ &= \phi^{k-2} \cdot \phi^2 \quad (\phi \text{ is the positive root of equation } x^2 = x+1) \\ &= \phi^k \end{aligned}$$

Lemma 19.4

Let x be any node in a Fibonacci heap, and let $k = x.degree$. Then $size(x) \geq F_{k+2} \geq \phi^k$, where $\phi = \frac{1+\sqrt{5}}{2}$

Proof. let s_k denote minimum possible size of any node of degree k in any Fibonacci heap.

$s_k \leq size(x)$

$$\begin{aligned} size(x) &\geq s_k \\ &\geq 2 + \sum_{i=2}^{k} s_{y_i.degree} \\ &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &\geq 2 + \sum_{i=2}^{k} F_i \\ &= 1 + \sum_{i=0}^{k} F_i \\ &= F_{k+2} \quad \text{(by Lemma 19.2)} \\ &\geq \phi^k \quad \text{(by Lemma 19.3)} \end{aligned}$$

$\therefore size(x) \geq s_k \geq F_{k+2} \geq \phi^k$.

Corollary 19.5

The maximum degree $D(n)$ of any node in an n-node Fibonacci heap is $O(\lg n)$.

$n \geq size(x) \geq \phi^k$ where $k = x.degree$

$k \leq \lfloor \log_\phi n \rfloor$

Maximum Flows and Parametric Shortest Paths in Planar Graphs

by Jeff Erickson jeffe@cs.uiuc.edu

let $G = (V, E)$ be a directed plane graph. s, t be vertices of G

let $c : E \longrightarrow \mathbb{R}$ be a nonnegative capacity function

Goal: compute a maximum (s,t)-flow in G.

Assume WLOG the reversal of any directed edge in G is also an edge in G

$\longrightarrow$ implies that both G and its dual G^* are strongly connected.

Venkatesan's Reduction

Idea: compute a feasible (s,t)-flow with fixed value λ, or correctly report that no such flow exists, by reduction to a single source shortest path problem in an appropriately weighted dual graph G^*.

$\pi(e)$: Fixed an arbitrary directed path P from s to t, and let $\pi : E \to \mathbb{R}$ denote the unit flow through P:

$$\pi(e) := \begin{cases} 1 & \text{if } e \in P \\ -1 & \text{if } \mathrm{rev}(e) \in P \\ 0 & \text{otherwise} \end{cases}$$

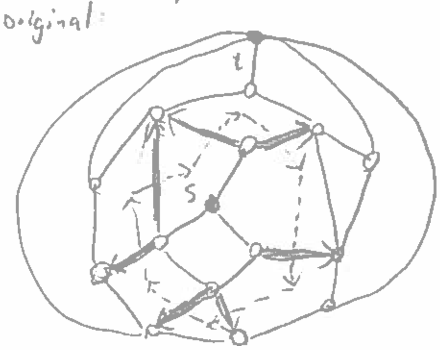
For any subset $E' \subseteq E$, let $\pi(E') = \sum_{e \in E'} \pi(e)$.

cocycle: A subgraph C of G is called a cocycle if the corresponding dual subgraph C^* is a simple directed cycle in G^*.

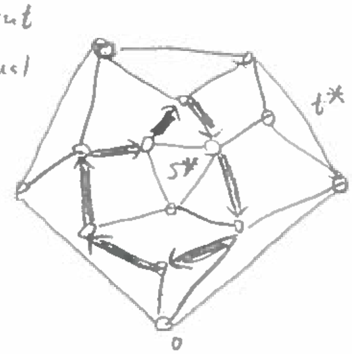
crossing number: For any cycle C^* in G^*, we call $\pi(C)$ the crossing number of C^*.

Lemma 2.1 $\pi(C) \in \{-1, 0, 1\}$ for any cocycle C. Moreover, $\pi(C) = 1$ if and only if C is an (s,t)-cut

original:



dual



consider the flow $\lambda \cdot \pi$, which assigns value $\begin{cases} \lambda & \text{to every directed edge in path } P \\ -\lambda & \text{" " " " " } rev(P) \\ 0 & \text{to every other edge.} \end{cases}$

Let $G_\lambda := G_{\lambda \cdot \pi}$ denote the residual network of this flow

capacity function: $c(\lambda, e) := c(e) - \lambda \cdot \pi(e)$.

$\therefore$ $\lambda \cdot \pi$ is feasible if and only if $c(\lambda, e) \geq 0$ for every edge e in G.

Let G_λ^* denote the dual residual network, which is the directed dual graph G^* where every edge e^* has a cost $c(\lambda, e^*) = c(\lambda, e)$.

Lemma 2.2 There is a feasible (s,t)-flow in G with value λ if and only if the residual network G_λ^* does not contain a negative cycle.

let $dist(\lambda, p)$ denote the shortest path distance in G_λ^* from o to p. (arbitrary dual vertex called origin)

define $\quad \phi(\lambda, e) := dist(\lambda, head(e^*)) - dist(\lambda, tail(e^*)) + \lambda \cdot \pi(e)$.

because the duals of the edges leaving v define a directed cycle in G^* all the $dist(\lambda, \cdot)$ terms in the sum cancel out.

$\therefore$ $\phi(\lambda, \cdot)$ is a valid (s,t)-flow with value λ.

define the slack of each dual edge e^*:

$slack(\lambda, e^*) := dist(\lambda, tail(e^*)) + c(\lambda, e) - dist(\lambda, head(e^*))$

so $slack(\lambda, e^*) = c(e) - \phi(\lambda, e)$

Parametric Shortest Paths

Let λ_{max} denote the largest value of λ for which shortest paths in G_λ^* are well-defined. Lemma 2.2 implies that λ_{max} is also the value of the maximum flow.

For any particular value of λ, let T_λ denote the single-source shortest path tree in G_λ^* rooted at o.

High level algorithm

PLANARMAXFLOW (G, c, s, t):
Compute T_0
Maintain T_λ as λ increases continuously from 0 to λ_{max}
Compute $\phi(\lambda_{max}, \cdot)$ from $T_{\lambda_{max}}$

Genericity assumption

we assume that the capacity function is generic.

1) Our genericity assumption implies that T_λ is uniquely defined for all λ between 0 and λ_{max}, except for a finite set of critical values

2) Our genericity assumption implies that exactly one non-tree edge becomes tense at each critical value of λ.

Lemma 2.3 λ_{max} is the first critical value of λ whose pivot introduces a directed cycle into T_λ.

pivot At each critical value of λ, some non-tree dual edge $p \to q$ becomes tense and enters T_λ, replacing the previous edge $pred(\lambda, q) \to q$.

tense call a dual edge e^* tense if $slack(\lambda, e^*) = 0$.

Lemma 2.4 λ_{max} is the smallest critical value of λ whose pivot disconnects L_λ

loose call a primal edge e loose at λ if neither its dual e^* nor its reversed dual $rev(e^*)$ is tense at λ, and let L_λ be the graph of all loose edges.

active A dual edge is active at λ if its slack at λ is decreasing

LP_λ The primal spanning tree L_λ contains a unique directed path from s to t; call this loose path LP_λ.

Lemma 2.5 A dual edge e^* is active at λ if and only if e is an edge of LP_λ.

PLANARMAXFLOW(G, c, s, t):

```
Initialize the spanning tree L, predecessors and slacks
while s and t are in the same component of L
    LP ← the path from s to t
    p→q ← the edge in LP* with minimum slack
    Δ ← slack(p→q)
    for every edge e in LP
        slack(e*) ← slack(e*) − Δ
        slack(rev(e*)) ← slack(rev(e*)) + Δ
    delete (p→q)* from L
    if q≠o  << that is, if pred(q)≠∅ >>
        insert (pred(q)→q)* into L
    pred(q) ← p
for each edge e
    ϕ(e) ← c(e) − slack(e*)
```

mutate T_λ as λ increases by pivoting until λ_{max}

Compute the flow