TD: Priority Heaps

Algorithmique et Programmation

Pierre Senellart
pierre.senellart@ens.fr

3 November 2016

The goal of this exercise session is to study implementations of priority queues, binary heaps and Fibonacci heaps, and their applications to graph algorithms.

A *priority queue* is a data structure to store items along with their *priority value* (some real number; the lower the value, the higher the priority) and that supports the following operations:

empty() Construct an empty priority queue.

insert(item, priority) Add an item to the queue with its priority value.

pop() Retrieve the item with lowest priority value in the queue.

decrease(item, priority) Update the priority value of an item, with the condition that the new priority value must be lower than the existing one.

1 Priority Queues in Graph Algorithms

Priority queues are a basic component of many graph algorithms, including:

- Dijsktra's algorithm for single-source shortest paths in directed graphs with nonnegative weights.
- Prim's algorithm for minimum spanning trees in weighted undirected graphs.

Let G be some weighted graph, v its number of vertices, e its number of edges.

We denote by $c_{\mathsf{i}}(n)$, $c_{\mathsf{p}}(n)$, and $c_{\mathsf{d}}(n)$, respectively, the cost of the insert, pop, and decrease operations on a priority queue holding n elements.

1a. What is the worst-case asymptotic complexity of Dijkstra's algorithm, in terms of v, e, c_{i}, c_{p}, and c_{d}?

1b. What is the worst-case asymptotic complexity of Prim's algorithm, in terms of v, e, c_{i}, c_{p}, and c_{d}?

2 Naïve Implementation

We consider a naïve implementation of a priority queue as a linked list of pairs (item, priority) sorted by increasing order of priority.

2a. What is the worst-case asymptotic complexity of insert in such an implementation?

2b. What is the worst-case asymptotic complexity of pop in such an implementation?

2c. What is the worst-case asymptotic complexity of decrease in such an implementation?

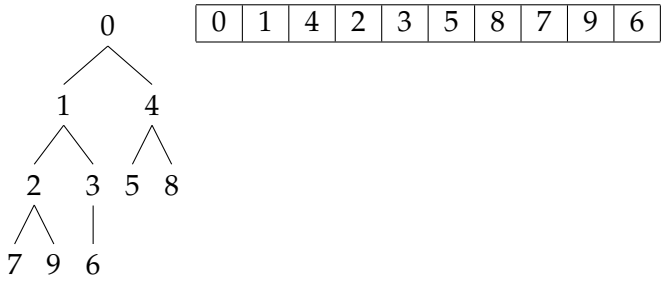
2d. What are the worst-case asymptotic complexities of Dijkstra's and Prim's algorithms with such a naïve implementation for the priority queue?

3 Binary Heaps

A *binary heap* is an implementation of a priority queue as an ordered binary tree. The binary tree is full on all levels except possibly for the last (i.e., if the tree is of depth d, every node of depth $\leqslant d-2$ has exactly 2 children). On the last level, existing leaves occupy the leftmost positions. We say that a heap is a *priority heap* (or that it verifies the *priority heap condition*) if the priority of a node is always less than or equal to the priority of its descendants.

A binary tree of n nodes is stored in an array A of size n in the following manner: $A[1]$ stores the root node; if a node u is stored in $A[k]$ and has children u_1 and u_2, then $A[2k]$ stores node u_1 and $A[2k+1]$ stores node u_2.

As an example, here is a binary heap represented as a binary tree and stored as an array:



One core operation used in implementing binary heaps is the `Heapify` procedure, used to correct a single violation of the priority heap condition. It takes as input an array A and an index i, such that the subtrees rooted at $2i$ and $2i+1$ in the array A are priority heaps, but the priority of node i may be greater than that of its children. It returns an array A such that the subtree rooted at i is a priority heap.

3a. Propose an implementation of the `Heapify` procedure that runs in time $O(h)$ where h is the height of the input node.

3b. Propose implementations of the empty, insert, pop, and decrease operations on binary heaps as efficient as possible, relying on the `Heapify` procedure when needed. What are their asymptotic complexities (using common assumptions on the complexities on operations over arrays)?

3c. What are the asymptotic complexities of Dijkstra's and Prim's algorithms with a binary heap implementation for the priority queue?

3d. What other common data structures could be used to implement a priority queue with the same operation complexities? What advantage do binary heaps have over these other data structures?

4 Fibonacci Heaps

A Fibonacci heap is a complex data structure used to implement priority queues. Formally, it is a collection of n unranked ordered rooted trees (i.e., each node can have an arbitrary degree) that verify the priority heap condition, with some additional information recorded:

- The collection is stored as a circular doubly linked list of the roots of the trees.
- In each tree, each node has a pointer to one of its children if any and to its parent if any, and to its left and right siblings (as a circular doubly linked list, so that the right sibling of the rightmost child of a node is the leftmost child).
- Each node of a tree has an additional *marker* which is a Boolean variable initially set to false. The marker is set to true if that node *has lost a child since the last time it changed parent*.
- A special variable *min* holds a pointer to the tree whose root node has the minimum priority value.
- The number n of trees and the degree $\delta(u)$ of each node u are also kept in memory.

4a. Propose an implementation of the empty operation.

4b. One consider the following *potential function* of a Fibonacci heap: the potential of a Fibonacci heap H is $\varphi(H) = \gamma \times (t + 2m)$ where t is the number of trees in the heap, m is the number of marked nodes, and γ is a positive constant to be defined further. In what follows we are going to use this potential function to carry out an *amortized complexity* analysis of the cost of the different operations.

4bα) What is the potential of the empty Fibonacci heap?

4bβ) For an operation x that transforms a Fibonacci heap H into a new Fibonacci heap $x(H)$ with cost $c_x(H)$, consider

$$\hat{c}_x(H) \coloneqq c_x(H) + \varphi(x(H)) - \varphi(H).$$

Show that for any sequence of operations $x_1 \dots x_k$ starting from the empty Fibonacci heap H_0, with $H_i \coloneqq x_i(H_{i-1})$:

$$\frac{1}{k}\sum_{i=1}^{k} c_{x_i}(H_{i-1}) \leqslant \frac{1}{k}\sum_{i=1}^{k} \hat{c}_{x_i}(H_{i-1}).$$

We will therefore call in the following $\hat{c}_x(H)$ the *amortized cost* of operation x and use it to characterize the complexity of operations on Fibonacci heaps.

4c. Propose a simple implementation of the insert operation whose asymptotic amortized complexity (and worst-case complexity) is $O(1)$.

4d. pop on Fibonacci heaps works as follows: we remove the node pointed to by the *min* pointer and make each of its children the root of a new tree. Then, the list of trees is modified to ensure that the degree $\delta(r)$ of every root r of a tree is distinct: to ensure this, every time two root nodes have the same degree, one is made a child of the other (respecting the priority heap condition), repeatedly until no two root nodes have the same degree. Markers are updated as needed throughout the procedure.

Show that, by choosing an appropriate value for γ, the amortized complexity of pop on a Fibonacci heap H can be made to be $O\left(\max\left(\max_{u \text{ node in } H} \delta(u), \max_{u \text{ node in } \mathsf{pop}(H)} \delta(u)\right)\right)$.

4e. decrease on Fibonacci heaps works as follows: if after updating the priority of a node the priority condition is violated, this node u is cut from its parent p and put as a new root node of a tree in the Fibonacci heap. Subsequently, a *cascading procedure* is applied to the node p as follows: if the node was already marked (meaning it had already lost another child), it is cut from its own parent p' and added as a new root node of a tree in the Fibonacci heap; otherwise, it is marked. The cascading procedure is applied recursively on p'.

Show that, by choosing an appropriate value for γ (compatible with the choice already made for the pop operation), the amortized complexity of decrease on Fibonacci heaps can be made to be $O(1)$. It can be useful to introduce the number of times the cascading procedure has been called.

4f. We are now going to show that the maximum degree of any node in a Fibonacci heap of n nodes is $O(\log n)$.

4fα) Show that for any node u in a Fibonacci heap with children $u_1, u_2, \dots u_k$ ordered in the chronological order they were linked to u, for any $2 \leqslant i \leqslant k$, $\delta(u_i) \geqslant i - 2$.

4fβ) Let F_k be the k-th term of the Fibonacci sequence:

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_k = F_{k-1} + F_{k-2} \quad \text{for } k \geqslant 2. \end{cases}$$

Show that for any node in a Fibonacci heap, $\sigma(u) \geqslant F_{\delta(u)+2}$ where $\sigma(u)$ is the size of the subtree rooted at u.

4fγ) Show that for any k, $F_{k+2} \geqslant \left(\frac{1+\sqrt{5}}{2}\right)^k$.

4fδ) Conclude that the maximum degree of any node in a Fibonacci heap is $O(\log n)$.

4g. What are the worst-case asymptotic complexities of Dijkstra's and Prim's algorithms with a Fibonacci heap implementation for the priority queue?