作业反馈3-9

25.1.5.

25.2.4.

problem 25.2

25.1-5

Show how to express the single-source shortest-paths problem as a product of matrices and a vector. Describe how evaluating this product corresponds to a Bellman-Ford-like algorithm (see Section 24.1).

Define $R^{(1)}$ as the column vector corresponding to the source s of the matrix W. Define recursively that

$$R^{(i+1)} = R^{(i)} W .$$

The vector $R^{|V|-1}$ gives the result.

Ignore infinite entries in the matrix W, and each step of matrix multiplication runs in $\Theta(|E|)$ time. Thus the algorithm runs in $\Theta(|E| \log |V|)$ time.

Bellman-Ford algorithm is the same as doing (n - 1) multiplications.

25.2-4

As it appears above, the Floyd-Warshall algorithm requires $\Theta(n^3)$ space, since we compute $d_{ij}^{(k)}$ for $i, j, k = 1, 2, \ldots, n$. Show that the following procedure, which simply drops all the superscripts, is correct, and thus only $\Theta(n^2)$ space is required.

```
FLOYD-WARSHALL'(W)
n = W.rows
D = W
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            d_ij = min(d_ij, d_ik + d_kj)
return D
```

```
FLOYD-WARSHALL(W)
n = W.rows
D^(0) = W
for k = 1 to n
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    for i = 1 to n
        for j = 1 to n
            d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

根据这个动规方程我们可以得到 $d^{(k)}_{ij}$ 只和 $d^{(k-1)}_{ij}$ 有关，所以我们最多只用保存上一状态就可以了。

而在当前 k 的整个循环中对于任意的(i,j)d_{ik} 和 d_{jk} 是不会发生变化的，所以可以直接修改 d_{ij} 不会发生错误

所以空间复杂度只有$\Theta(n^2)$

25-2 *Shortest paths in ϵ-dense graphs*

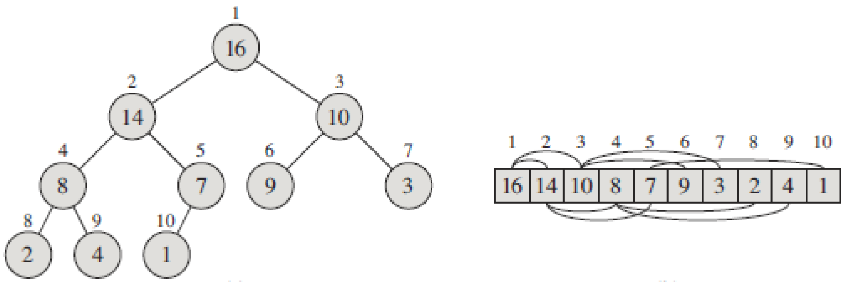
A graph $G = (V, E)$ is ***ϵ-dense*** if $|E| = \Theta(V^{1+\epsilon})$ for some constant ϵ in the range $0 < \epsilon \leq 1$. By using d-ary min-heaps (see Problem 6-2) in shortest-paths algorithms on ϵ-dense graphs, we can match the running times of Fibonacci-heap-based algorithms without using as complicated a data structure.

a. What are the asymptotic running times for INSERT, EXTRACT-MIN, and DECREASE-KEY, as a function of d and the number n of elements in a d-ary min-heap? What are these running times if we choose $d = \Theta(n^\alpha)$ for some constant $0 < \alpha \leq 1$? Compare these running times to the amortized costs of these operations for a Fibonacci heap.

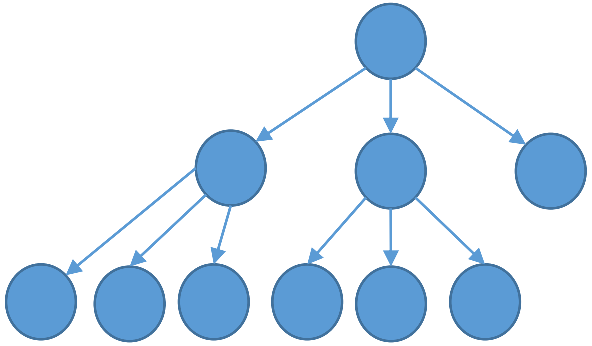
b. Show how to compute shortest paths from a single source on an ϵ-dense directed graph $G = (V, E)$ with no negative-weight edges in $O(E)$ time. (*Hint:* Pick d as a function of ϵ.)

c. Show how to solve the all-pairs shortest-paths problem on an ϵ-dense directed graph $G = (V, E)$ with no negative-weight edges in $O(VE)$ time.

d. Show how to solve the all-pairs shortest-paths problem in $O(VE)$ time on an ϵ-dense directed graph $G = (V, E)$ that may have negative-weight edges but has no negative-weight cycles.



(a) (b)



Insert: $O(\log_d n)$
Extract-Min: $O(d \log_d n)$
Decrease-Key: $O(\log_d n)$

$$d = n^\alpha$$

Insert: $O(\log_d n) = O(1/\alpha) = O(1)$

Extract-Min: $O(d \log_d n) = O\left(\frac{n^\alpha}{\alpha}\right) = O(n^\alpha)$

Decrease-Key: $O(\log_d n)$=$O(1/\alpha) = O(1)$

25-2 ***Shortest paths in ϵ-dense graphs***

A graph $G = (V, E)$ is ***ϵ-dense*** if $|E| = \Theta(V^{1+\epsilon})$ for some constant ϵ in the range $0 < \epsilon \leq 1$. By using d-ary min-heaps (see Problem 6-2) in shortest-paths algorithms on ϵ-dense graphs, we can match the running times of Fibonacci-heap-based algorithms without using as complicated a data structure.

a. What are the asymptotic running times for INSERT, EXTRACT-MIN, and DECREASE-KEY, as a function of d and the number n of elements in a d-ary min-heap? What are these running times if we choose $d = \Theta(n^\alpha)$ for some constant $0 < \alpha \leq 1$? Compare these running times to the amortized costs of these operations for a Fibonacci heap.

c. Show how to solve the all-pairs shortest-paths problem on an ϵ-dense directed graph $G = (V, E)$ with no negative-weight edges in $O(VE)$ time.

d. Show how to solve the all-pairs shortest-paths problem in $O(VE)$ time on an ϵ-dense directed graph $G = (V, E)$ that may have negative-weight edges but has no negative-weight cycles.

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

The algorithm calls both **INSERT** and **EXTRACT-MIN** once per vertex

calls **DECREASE-KEY** at most |E| times

$$|\boldsymbol{V}| * (\boldsymbol{c}(\boldsymbol{Insert}) + \boldsymbol{c}(\boldsymbol{ExtMin})) + |\boldsymbol{E}| * \boldsymbol{c}(\boldsymbol{DK})$$

$$\boldsymbol{n} * (O(\log_d n) + O(d \log_d n)) + \boldsymbol{n}^{\boldsymbol{1+\epsilon}} * O(\log_d n)$$
$$= O(n^{1+\epsilon} \log_d n) = O(|E| \log_d n)$$
$$= O(|E|)$$

$$O(\log_d n) = O(1)$$

$d = n^\alpha$可满足

25-2 *Shortest paths in ϵ-dense graphs*

A graph $G = (V, E)$ is ***ϵ-dense*** if $|E| = \Theta(V^{1+\epsilon})$ for some constant ϵ in the range $0 < \epsilon \leq 1$. By using d-ary min-heaps (see Problem 6-2) in shortest-paths algorithms on ϵ-dense graphs, we can match the running times of Fibonacci-heap-based algorithms without using as complicated a data structure.

a. What are the asymptotic running times for INSERT, EXTRACT-MIN, and DECREASE-KEY, as a function of d and the number n of elements in a d-ary min-heap? What are these running times if we choose $d = \Theta(n^\alpha)$ for some constant $0 < \alpha \leq 1$? Compare these running times to the amortized costs of these operations for a Fibonacci heap.

b. Show how to compute shortest paths from a single source on an ϵ-dense directed graph $G = (V, E)$ with no negative-weight edges in $O(E)$ time. (*Hint:* Pick d as a function of ϵ.)

```
JOHNSON(G, w)
  compute G′, where G′.V = G.V ∪ {s},
      G′.E = G.E ∪ {(s, v) : v ∈ G.V}, and
      w(s, v) = 0 for all v ∈ G.V
  if BELLMAN-FORD(G′, w, s) == FALSE
      print "the input graph contains a negative-weight cycle"
  else for each vertex v ∈ G′.V
          set h(v) to the value of δ(s, v)
              computed by the Bellman-Ford algorithm
      for each edge (u, v) ∈ G′.E
          ŵ(u, v) = w(u, v) + h(u) − h(v)
      let D = (d_uv) be a new n × n matrix
      for each vertex u ∈ G.V
          run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
          for each vertex v ∈ G.V
              d_uv = δ̂(u, v) + h(v) − h(u)
      return D
```