자료구조 기말고사, 6/4, 2018

Open book, 70분

* 그런 일 없으리라 기대하지만 혹시 문제에 명시되지 않았는데 필요한 가정이 있으면 스스로 하라. 합리적인 가정이면 추가 점수를 받을 것이고, 미숙함의 결과로 도입된 가정이면 만점을 못 받을 것임.

1. (20점) 아래 검색트리 관련 물음에 답하라.
   (a) 2-3 tree에서 총 key가 2개이면 tree는 어떤 모양일까? 3개이면 어떤 모양일까?
   (b) B-tree에서 각 노드는 k개까지의 원소, 적어도 $\left\lfloor \frac{k}{2} \right\rfloor$ 개의 원소 조건이 있다. 이것을 r개까지의 분기, 적어도 $\left\lfloor \frac{r}{2} \right\rfloor$ 개의 분기라고 표현해도 같은 성질이 유지되는가? (Leaf node들은 생각하지 않아도 좋다. 즉, 가상의 empty node로 분기가 있다고 생각한다.) 정확하게 따져 설명해야 한다.

2. (20점) 해시 함수에 대해 다음 물음에 답하라.
(a) 한 학생이 다음과 같이 참신한 질문을 했다."더블 해싱에서 $h_i(x) = (h(x) + i * f(x)) \bmod m$ (여기서 $h(x) = x \bmod m$, $f(x) = x \bmod m'$)이라고 할 때, x가 m'의 배수인 경우, 해당 더블 해싱을 통해 x에 대해 probe 할 수 있는 slot은 단 한 개(Table[$h(x)$])가 되는데요, 이러한 문제는 어떻게 방지할 수 있나요? m'을 소수로 두어도, 2의 멱수여도, 어떻게 두어도 m'의 배수는 항상 존재하기 마련이기 때문에 ... 이런 경우 어떻게 해결할 수 있는지 모르겠습니다!" 이것은 교수도 교재를 집필할 때 했던 실수다. 이것을 어떻게 해결할 수 있겠는가?

(b) 다른 학생은 다음과 같이 또다른 참신한 아이디어를 냈다. "해시함수 $h(x) = h_0(x) = x \bmod m$에서 충돌이 일어날 때마다 아래와 같이 해시함수에 한 항씩 더하면 어떨까요? 만일 원소 a를 삽입할 때 충돌이 한번 이상 일어나서 결과적으로 $(h_0(a) + L_a) \bmod m$ 자리에 삽입되었다면 해시함수 $h(x)$에 $f(x-a)L_a$를 더해줍니다. 여기서 $f(i) = \begin{cases} 1, i = 0 \\ 0, otherwise \end{cases}$. 이런 식으로 충돌이 일어난 모든 원소에 대해 한 항씩 더해줍니다. 결과적으로 해시함수는 $h(x) = (h_0(x) + f(x-a)L_a + f(x-b)L_b + \dots) \bmod m$이 됩니다.($a, b, \dots$는 충돌이 일어난 원소들) 이렇게 하면 해시함수는 어떤 원소가 많은 충돌 끝에 빈자리를 찾았어도 나중에 검색할 때는 충돌 과정을 따라가지 않고 한 번에 제자리를 빨리 찾을 수 있지 않을까요?" 좋은 아이디어다. 그렇지만 문제가 있다. 기존 해시함수 대신 이것을 쓸 수 없는 이유가 뭘까?

3. (25점) 아래 topological sorting에 관한 질문에 답하라.

   (a) (15점) G는 directed acyclic graph이고 G의 한 vertex s에서 다른 모든 vertex로 가는 path가 존재한다. 즉, topological order로 따지면 1번 vertex이다. 이 vertex로부터 시작하여 아래와 같이 DFS 알고리즘을 약간 변형한 알고리즘 DFS-topSort(s)를 수행한다. 이 결과로 **print** 되는 vertex 순서의 역순이 topological order가 된다. 이유를 설명하라. 완벽한 증명이 되면 만점을 줄 것이고, 그렇지 못해도 핵심적인 이유를 파악하면 상당한 점수를 줌. 최초에는 모든 node가 UNVISITED로 마크되어 있다.

```
DFS-topSort(v) {
  mark[v] = VISITED;
  for all vertex w adjacent to v
        if (mark[w] = UNVISITED)
              DFS-topSort(w);
  print v;
```

}

(b) (10점) topological sorting을 뒤쪽부터 진행하는 아래 알고리즘을 배웠고 이것을 앞쪽부터 진행해도 된다고 배웠다.

```
topologicalSort(G)
// G: graph
{
        for i = 1 to n {
                Select a vertex v that has no successors;
                vertex[i] = v;
                Delete from G vertex v and the edges to v;
        }
}
```

이 결과 array vertex[1...n]에는 topological order의 역순으로 vertex들이 저장되어 있다. 이제 이것을 앞뒤 양쪽에서 동시에 진행하는 버전으로 바꾸어 보아라. 즉, predecessor(선행 vertex)가 없는 vertex 하나와 successor가 없는 vertex 하나를 동시에 고른다. 바뀐 알고리즘은 아래 for loop의 반복 횟수가 대략 반 정도 될 것이다. 알고리즘의 수행 결과 vertex[1...n]에는 topological order로 vertex들이 들어있도록 하라. 쉬운 문제이므로 간결하고 완결성 있는 알고리즘으로 표현해야 한다.

4. (20점) 아래는 directed graph G=(V, E)의 shortest path를 구하기 위한 Dijkstra's algorithm이다. 이 알고리즘에서 ①의 비교횟수와 ②의 수행횟수의 비율은 어떻게 되는가? 가능한 최솟값과 최대값을 밝히라. 이유도 간단히 설명해야 한다. 만일 버거우면 ①의 비교횟수와 ②의 수행횟수 각각에 대해 가능한 횟수를 최대한 생각해보라. 생각이 진행된 만큼 평가해줄 것임.

```
shortestPath (G, s) // s: the starting vertex의 index
{
        S = ∅;
        for v = 1 to G.size( )
                d[v] = ∞;
        d[s] = 0;
        for i = 1 to G.size( ) {
            Find the smallest d[v] s.t. v ∉ S
            S = S ∪{v};
            for each vertex u s.t. u ∉ S and u ∈ L(v) {
                if (d[u] > d[v] + weight(v, u))  ---------- ①
                    d[u] = d[v] + weight(v, u); ---------- ②
            }
        }
}
```

* 만일 필요하면 다음의 표현을 사용하라.
  |V|: vertex의 총수
  |E|: edge의 총수
  indegree(u): vertex u의 inbound degree (들어오는 edge의 개수)
  outdegree(u): vertex u의 outbound degree (나가는 edge의 개수)

5. (15점) Max-heap이 array A[1...n]에 저장되어 있다. 이 상태에서 A[i]의 값이 변했다. 이로 인해 heap 성질이 깨질 수 있다. 이를 고치는 알고리즘 repairHeap[i]를 제안하라. 수업 시간에 배운 함수 percolateDown(i)과 percolateUp(i)을 사용할 필요가 있으면 그 내용을 밝힐 필요없이 그냥 호출하면 된다.