[CSE3081(2 반)] 알고리즘 설계와 분석 과제 4 보고서

20151623 한상구

1. Single-pair shortest path in DAG, Subset sum 각각에 대한 설명
    a. Single-pair shortest path in DAG

    주어진 입력에 대해 인접리스트를 구현 ($O(|V| + |E|)$), 이를 이용하여 topological sort 를 구현한 뒤 ($O(|V| + |E|)$), 얻어지는 리스트를 통해 greedy approach 를 이용하여 shortest path 를 구한다 ($O(|V| + |E|)$)

    b. Subset sum

    주어진 입력에 대해 Exact (exponential time) 함수를 이용하여 optimal solution 을 구하고, Approximation 을 통해 얻은 근사해를 이와 비교한다. Merge-List 와 주어진 상한 값 L 에 대해 지워내는 함수가 기본으로 사용되며, Approxiamtion 의 경우 Trim 이라는 함수가 추가된다.

2. 구현 기법 상세 설명
    a. Single-pair shortest path in DAG

    문제에서 요구하는 사항은 1. 주어진 그래프가 Directed Acyclic Graph 인지 판단하고, 2. 주어진 시작점에서 도착점까지의 경로가 존재하는지 판단하며, 3. 존재한다면 그 경로의 최단거리를 구하라. 의 총 3 가지입니다.

    2, 3 번은 최단경로를 구하는 알고리즘 이후, 도착점까지의 거리가 초기에 설정한 inf 값과 같다면 (즉, 갱신되지 않았다면) 경로가 없는 것이므로 한 가지 문제로 묶을 수 있습니다. 곧, 1 번과 (2,3)번 두가지로 분류가 되게 됩니다.

    여기서 최단경로를 구하기 위해선, topological sort 가 선행되어야 하는데, Topological sort 는 DFS 기반으로, 스택에서 pop 된 노드의 순서를 기반으로 정렬하기 때문에 DFS 를 잘 사용하면 1 번을 처리하며 topological sort 를 구현할 수 있습니다. Adjacency list 에서 DFS 를 수행하는 시간복잡도는 $O(|V| + |E|)$ 이므로 topological sort 를 수행하는 비용도 이렇게 될 것입니다.

    DFS 를 하며, 아직 방문하지 않은 노드를 white, 방문은 하였으나 아직 스택에서 pop 되지 않은 노드를 grey, 스택에서 pop 된, 방문이 완료된 노드를 black 으로 색칠한다고 가정하겠습니다. 그렇다면 cycle 의 존재 유무는 간단하게 파악할 수 있습니다. 앞으로 방문할 노드의 색이 grey 인 경우입니다.

    이러한 구현을 통해 1 번을 처리하였고, (2,3)번의 경우는 topological sort 이후 주어진 list 를 이용, greedy approach 를 통해 shortest path 를 구했으며 이 이후 도착점의 최단경로가 inf 값과 다른지를 본 뒤 경로의 존재

유무를 파악했습니다. 이는 topological sort 이후 얻은 list 에서 인접한 edge 를 보며 구하기 때문에, 시간복잡도는 역시 $O(|V| + |E|)$ 일 것입니다.

곧 총 시간복잡도는 $O(|E|) + O(|V| + |E|) + O(|V| + |E|) = O(|V| + |E|)$ 가 됩니다. (Adjacency list 구축, topological sort, find shortest path)

b. Subset sum

아직까지 polynomial time 에 solution 을 구하는 방법이 알려지지 않은 문제이기 때문에, 주어진 명세서에서는 exponential time algorithm 과 주어진 ϵ을 통해 근사시켜 polynomial time 에 solution 을 구하는 두 알고리즘을 요구하고 있습니다. 주어진 n 의 범위가 $[10, 30]$이기 때문에 구한 optimal solution 의 인덱스 추적은 int 형 변수를 이용하여 bit masking 을 통해 해결했고, 각 알고리즘의 구현은 수업시간을 토대로 진행하였습니다.

Merge-List 및 Trim 의 경우 vector 를 이용하여 진행하였고,

명세서에 상세한 설명을 요구하는 항목인 만큼, 다음 항목에서 다루도록 하겠습니다.

3. Subset sum

a. EXACT

i. Pseudo code

n <- | S |

L_0<- [0]

For i <- 1 to n {

L_i <- Merge($L_{i-1}, L_{i-1} + x_i$)

remove from L_i every element that is greater than L

}

return y^* (the largest element in L_n)

ii. 함수 원형

```
p exact(int n, int L, vector<int> S)
(typedef pair<unsigned int, int> p;)
```

pair 의 first 는 sum 을, second 는 index 추적을 위해 bit masking 한 int 형 변수입니다.

parameter 의 n 과 L 은 각각 |S|, sum 의 upper bound

vector<int> S 는 주어진 Set 을 받아옵니다.

코드 실행 후 제일 큰 first 를 갖는 pair 를 return 합니다.

b. APPROX

i. Pseudo code

n <- | S |

L_0<- [0]

For i <- 1 to n {

L_i <- Merge($L_{i-1}, L_{i-1} + x_i$)

L_i <- Trim($L_i, \epsilon / 2n$)

remove from L_i every element that is greater than L

}

return y^* (the largest element in L_n)

ii. 함수 원형

p approx(int n, double ep, int L, vector<int> S)

(typedef pair<unsigned int, int> p;)

Exact 와 동일한 parameter 입니다.

ep 라는 double 형 parameter 가 추가되었는데, 이는 주어진 ϵ 값입니다.

역시 코드 실행 후 제일 큰 first 를 갖는 pair 를 return 합니다.

c. 얼마나 큰 n 값까지 수행 가능한가?

Pair 를 <unsigned int, int>로 잡은 이유는 명세서에 기인하여 int 형, 즉 32bit 만으로도 충분이 인덱스를 역추적 할 수 있기 때문이었습니다. 실제로 n=32 일 때 까지 정상동작함을 확인할 수 있었습니다.

하지만 pair 를 <unsigned long long, long long> 로 정의하고 진행한다면, $1 \leq n \leq 64$ 의 범위에서 수행 가능 할 것으로 예상합니다. (사용 가능한 메모리가 충분하다는 가정 아래에, 수행 시간을 고려하지 않는다면)

시간을 고려한다면, n 이 33 보다 커지기만 해도 느려지는것이 체감되고, 더 크게 만든다면 유한한 시간 내에 확인할 수 없습니다. (Exact 의 경우)

4. Subset sum Test cases

a. N = 25

S = [16807, 14039793, 12037337, 11865130, 3258242, 449224, 364248, 15010302, 15937347, 27526221, 18258072, 8141909, 6099596, 7134178, 14144691, 30226247, 31996183, 16531729, 18072472, 9324884, 24129071, 31992593, 23303725, 18385669, 5190232]

T = 1<<25 = 2^25, ϵ = 0.05, 0.1, 0.2, 0.3, 0.4, 0.5

b. N = 31

S = [16807, 282475249, 1622650073, 984943658, 1144108930, 470211272, 101027544, 1457850878, 1458777923, 2007237709, 823564440, 1115438165, 1784484492, 74243042, 114807987, 1137522503, 1441282327, 16531729, 823378840, 143542612, 896544303, 1474833169, 1264817709, 1998097157, 1817129560, 1131570933, 197493099, 1404280278, 893351816, 1505795335, 1954899097]

T = 1<<31 = 2^31, ϵ = 0.05, 0.1, 0.2, 0.3, 0.4, 0.5

5. Subset sum 결과

a. N = 25, T = 2^25 = 33554432

```
139:4_2 uppo97$ ./a.out
Exact with n - 25, takes 4.461000ms
The optimal value is 33553651
Approx with n - 25, epsilon - 0.05, takes 1.907000ms
The optimal value is 33498669
(y*-z*)/z* is 0.001641
```

$\epsilon = 0.05$

```
139:4_2 uppo97$ ./a.out
Exact with n - 25, takes 4.341000ms
The optimal value is 33553651
Approx with n - 25, epsilon - 0.10, takes 1.299000ms
The optimal value is 33509041
(y*-z*)/z* is 0.001331
```

$\epsilon = 0.1$

```
139:4_2 uppo97$ ./a.out
Exact with n - 25, takes 5.732000ms
The optimal value is 33553651
Approx with n - 25, epsilon - 0.20, takes 1.348000ms
The optimal value is 33547378
(y*-z*)/z* is 0.000187
```

$\epsilon = 0.2$

```
139:4_2 uppo97$ ./a.out
Exact with n - 25, takes 4.490000ms
The optimal value is 33553651
Approx with n - 25, epsilon - 0.30, takes 0.781000ms
The optimal value is 33343832
(y*-z*)/z* is 0.006293
```

$\epsilon = 0.3$

```
Exact with n - 25, takes 3.627000ms
The optimal value is 33553651
Approx with n - 25, epsilon - 0.40, takes 0.615000ms
The optimal value is 33238934
(y*-z*)/z* is 0.009468
```

$\epsilon = 0.4$

```
Exact with n - 25, takes 4.558000ms
The optimal value is 33553651
Approx with n - 25, epsilon - 0.50, takes 0.627000ms
The optimal value is 33164413
(y*-z*)/z* is 0.011737
```

$\epsilon = 0.5$

b. N = 31, T = 2^31 = 2147483648

```
139:4_2 uppo97$ ./a.out
Exact with n - 31, takes 12.096000ms
The optimal value is 2147480366
Approx with n - 31, epsilon - 0.05, takes 2.786000ms
The optimal value is 2147463559
(y*-z*)/z* is 0.000008
```

$\epsilon = 0.05$

```
139:4_2 uppo97$ ./a.out
Exact with n - 31, takes 12.009000ms
The optimal value is 2147480366
Approx with n - 31, epsilon - 0.10, takes 1.502000ms
The optimal value is 2145350511
(y*-z*)/z* is 0.000993
```

$\epsilon = 0.1$

```
Exact with n - 31, takes 10.257000ms
The optimal value is 2147480366
Approx with n - 31, epsilon - 0.20, takes 1.229000ms
The optimal value is 2143950126
(y*-z*)/z* is 0.001647
```

$\epsilon = 0.2$

```
Exact with n - 31, takes 11.122000ms
The optimal value is 2147480366
Approx with n - 31, epsilon - 0.30, takes 0.871000ms
The optimal value is 2143950126
(y*-z*)/z* is 0.001647
```

$\epsilon = 0.3$

```
Exact with n - 31, takes 9.375000ms
The optimal value is 2147480366
Approx with n - 31, epsilon - 0.40, takes 0.943000ms
The optimal value is 2142599225
(y*-z*)/z* is 0.002278
```

$\epsilon = 0.4$

```
139:4_2 uppo97$ ./a.out
Exact with n - 31, takes 11.098000ms
The optimal value is 2147480366
Approx with n - 31, epsilon - 0.50, takes 0.736000ms
The optimal value is 2143950126
(y*-z*)/z* is 0.001647
```

$\epsilon = 0.5$

c. 번외, N = 35, T = 2^35/4 = 8589934592, $\epsilon = 0.1$

```
Exact with n - 35, takes 129138.113000ms
The optimal value is 8589934592
Approx with n - 35, epsilon - 0.10, takes 10.029000ms
The optimal value is 8589194957
(y*-z*)/z* is 0.000086
```

6. Subset sum 결과분석
   a. N 에 따라서

      N 이 커짐에 따라 Exact 의 수행시간이 exponential 하게 증가하는 것을 관찰할 수 있었습니다. (5 번항목 a, b, c, d 참조)

      L 에 따라서 살짝 바뀔 수 있을 것 같으나 (Cutting 되는 upper bound 가 작아지므로 List 의 크기가 더 작아질 것 이기 때문) 중점적으로 둔 것은 N 이기 때문에 T 는 고정한 채로 진행하였습니다.

      이에 반해, Approx 의 수행 시간은 polynomial 하게 증가하는 것 을 볼 수 있었는데, N 이 커짐에 따라 Exact 와의 수행시간 차이는 급격하게 벌어졌습니다.

   b. ϵ 에 따라서

      ϵ값은 optimal solution 과 approximate 한 optimal soultion 의 관계를 나타내는데, 이는 결과 마지막줄에서 명시 해 두었습니다. 전부 ϵ 이하의 비율을 보여주었습니다.

      ϵ값이 커짐에 따라, 보다 rough 한 approximate 가 가능해짐에 따라서 빠른 수행시간을 보여주었습니다.

7. 실험 환경
   a. OS : macOS Sierra Ver. 10.12

   b. CPU: 1.7 GHz Intel Core i7

   c. Ram: 8.00 GB

d. Compiler: gcc compiler Ver. 4.2.1 with –o optimization option