데이터 구조 및 실습

Written Report #3

(제출일 : 2018 년 12 월 11 일)

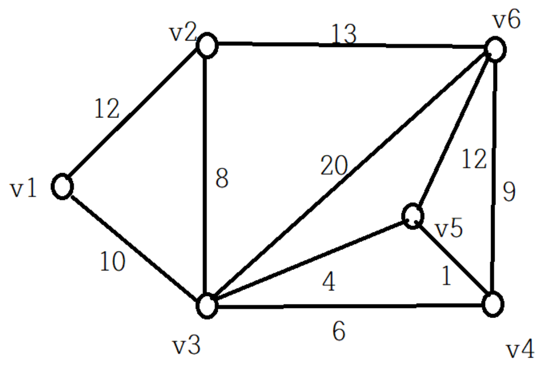
담당 교수 : 이상호

전공/학년 : 사이버보안 / 2
학번 : 17710076
이름 : 임은지

이메일 : 218926@naver.com
(긴급 연락처 : 010-6878-7807)

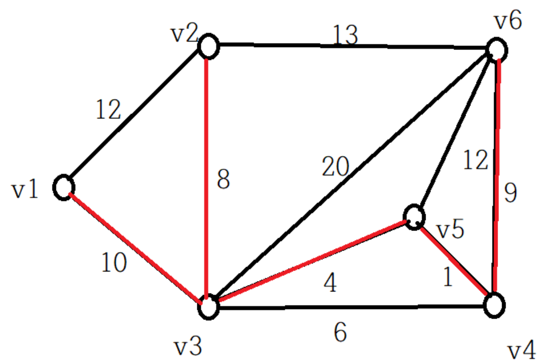
Written Report #3

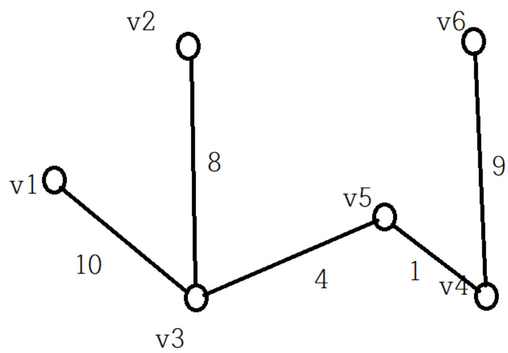
1. [최소 스패닝 트리(MST) 및 최단 경로 트리(SPT)]



(1) 다음 그래프 G의 MST를 Prim의 알고리즘에 의해 구하라. 단, Prim의 알고리즘에 의한 단계별 상태를 Lecture 10. 그래프-part2 강의노트 9쪽과 같은 테이블로 나타내고, 최종 결과를 그 가중치값과 함께 그림으로도 나타내야 함. (5점)

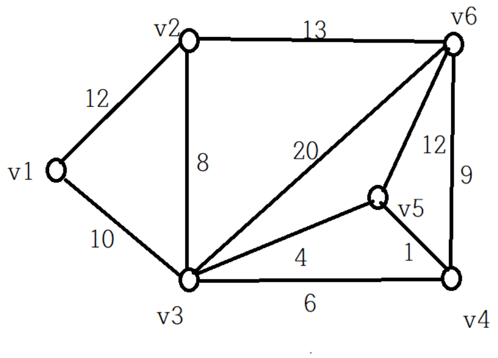
Pass:	Initially	1	2	3	4	5	6	weight	V1
Active Vertex :		V1	V3	V5	V4	V2	V6		
V1	0							0	-
V2	∞	12	8	8	8			8	V3
V3	∞	10						10	V1
V4	∞	∞	6	1				1	V5
V5	∞	∞	4					4	V3
V6	∞	∞	20	12	9	9		9	V4



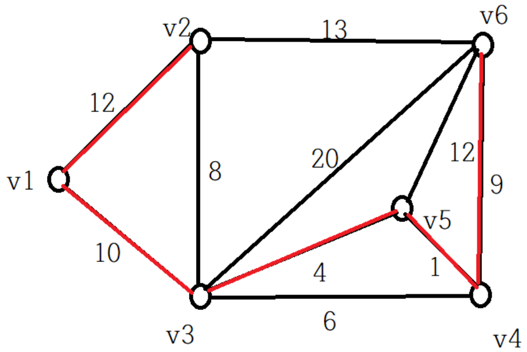


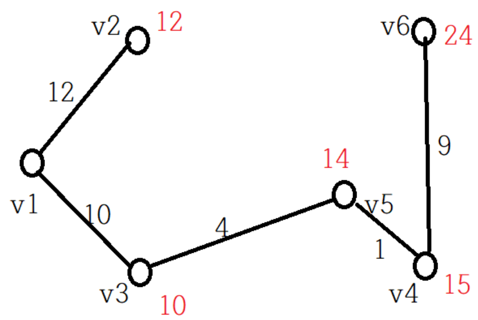
빨간색으로 표시된 영역(오른쪽 그림과 같음)이 Prim 알고리즘에 의한 최소 스패닝 트리이다.

(2) 정점 v_1으로부터 나머지 모든 정점들까지의 SPT를 Dijkstra의 알고리즘에 의해 구하라. 단, Dijkstra의 알고리즘에 의한 단계별 상태를 Lecture 10. 그래프-part2 강의 노트 9쪽과 같은 테이블로 나타내고, 최종 결과를 최단 경로의 거리(배열 distance[]이 최종값)와 함께 그림으로도 나타내야 함. (5 점)



Pass:	Initially	1	2	3	4	5	6	weight	V1
Vertex :		V1	V3	V2	V5	V4	V6		
V1	0							0	-
V2	∞	12	12					12	V1
V3	∞	10						10	V1
V4	∞	∞	16	16	15			15	V5
V5	∞	∞	14	14				14	V3
V6	∞	∞	30	25	25	24		24	V4





왼쪽의 빨간색으로 표시된 부분(오른쪽 그림)이 Dijkstra 알고리즘을 이용한 v1 에서 v2~v6 으로의 최단 경로이다.

distance[] =

0(v1)	1(v2)	2(v3)	3(v4)	4(v5)	5(v6)
0	12	10	15	14	24

distance 배열의 index 는 v1~v6 순서대로 0~5 로 붙였다.

2. [해싱]

크기가 11이고 슬롯이 하나인 해시테이블 및 해시함수 $h(k) = k \bmod 11$을 가정한다. 키(key)들이 다음과 같은 순서로 입력된다고 하면 아래의 각 경우에 대해 해시테이블을 그림으로 나타내어라.(5점)

12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5

h(12) = 1 mod 11, h(44) = 0 mod 11, h(13) = 2 mod 11, h(88) = 0 mod 11, h(23) = 1 mod 11, h(94) = 6 mod 11, h(11) = 0 mod 11, h(39) = 6 mod 11, h(20) = 9 mod 11, h(16) = 5 mod 11, h(5) = 5 mod 11

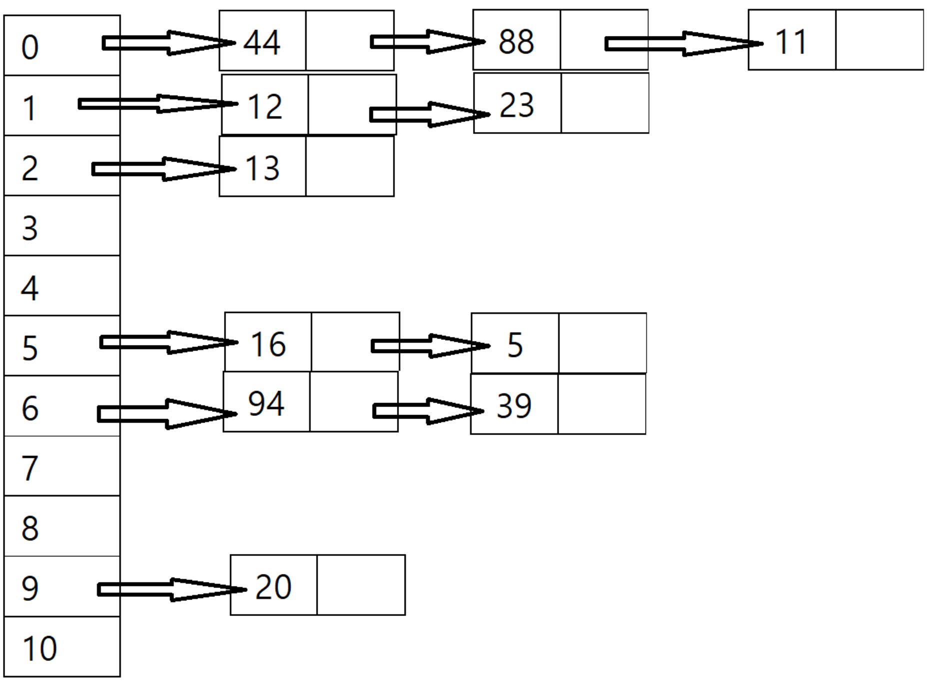
(1) 오버플로우를 선형조사법을 사용하여 처리하는 경우

버켓	1	2	3	4	5	6	7	8	9	10	11
0		44	44	44	44	44	44	44	44	44	44
1	12	12	12	12	12	12	12	12	12	12	12
2			13	13	13	13	13	13	13	13	13
3				88	88	88	88	88	88	88	88
4					23	23	23	23	23	23	23
5							11	11	11	11	11
6						94	94	94	94	94	94
7								39	39	39	39
8										16	16
9									20	20	20
10											5

(2) 오버플로우를 이중 해싱을 사용하여 처리하는 경우(단, $h'(k) = 7 - (k \bmod 7)$)

버켓	1	2	3	4	5	6	7	8	9	10	11
0		44	44	44	44	44	44	44	44	44	44
1	12	12	12	12	12	12	12	12	12	12	12
2			13	13	13	13	13	13	13	13	13
3				88	88	88	88	88	88	88	88
4								39	39	39	39
5									20	20	20
6					23	23	23	23	23	23	23
7											5
8										16	16
9							11	11	11	11	11
10						94	94	94	94	94	94

(3) 오버플로우를 체이닝을 사용하여 처리하는 경우



3. [탐색] 높이균형 이진탐색트리 T의 높이를 구하는 $O(\log_2 n)$-시간 알고리즘을 재귀적으로 작성하고, 여러분이 기술한 알고리즘이 왜 $O(\log_2 n)$-시간인지 밝혀라. 각 노드는 lchild, data, bf, rchild 필드로 이루어짐. (힌트 : bf(balance factor)의 값을 이용할 것) (5점)

root	lchild	data	bf	rchild

높이 = 1

루트 노드만 있을 때의 높이를 1 이라고 보았다.

```
get_height(root) {
        height = 0
        if (노드가 비어있지 않음)

                        if (왼쪽 자식이 더 높음)
                                height = (왼쪽 자식의 높이 + 1);
                        else //오른쪽 자식이 더 높음
                                height = (get_height(node->rchild) + 1);

        return height;
}
```

```
int get_height(TreeNode *node) {
        int height = 0;
        if (node != NULL) {
                        if (node->bf >= 0) { //bf:왼쪽 자식 트리 높이가 더 높으면 +
                                height = (get_height(node->lchild) + 1);
                                //왼쪽으로 내려간다.
                                //bf = 0으로 양쪽 자식 트리 높이가 같을 때는 어느
                        쪽으로 내려가도 상관없으므로 왼쪽으로 내려감.
                        }
                        else { //오른쪽 자식 트리가 더 높으면 -
                                height = (get_height(node->rchild) + 1);
                                //오른쪽으로 내려간다.
                        }
        }
```

```
        return height;
}
```

시간복잡도 계산

기본 연산: 덧셈 – height 계산

입력(전체 노드의 개수) : n

기본 연산인 덧셈은 get_height 함수를 호출하여 height 값을 갱신할 때마다 한번씩 늘어난다. 재귀적 기법을 통해 왼쪽 자식이 더 높으면 get_height(node->lchild)를 호출, 반대의 경우엔 get_height(node->rchild)를 호출한다. 따라서 처음에 호출한 함수가 get_height(root)라면, 전체 노드의 개수는 n 개, 다음에 호출 하는 함수가 get_height(node->lchild), get_height(node->rchild) 어떤 쪽이던 원래 트리 root 의 child 를 root 로 하는 트리의 노드의 전체 개수는 약 n/2 개 이다. 재귀적으로 함수를 호출할 때 마다 노드를 절반 나눠서 진행하기 때문에 입력값이 반복적으로 n/2 로 줄어든다.

이렇게 $\mathrm{n} = 2^k$ 이 될 때 까지 2 로 나누면서 진행하면, $k = \log_2 n$ 이고, 수식으로 표현하면 아래와 같다.

$$
\begin{aligned}
\mathrm{T}(n) &= 1 + T\left(\frac{n}{2}\right) \\
&= 1 + 1 + T\left(\frac{n}{2^2}\right) \\
&= 1 + 1 + 1 + T\left(\frac{n}{2^3}\right) \\
&= \cdots \\
&= 1 + 1 + 1 + \cdots + T\left(\frac{n}{2^k}\right) = (\log_2 n + 1) \in \mathrm{O}(\log_2 n)
\end{aligned}
$$

위의 식에서 1 은 k+1 개 있으므로 내가 정의한 시간 알고리즘은 $\mathrm{O}(\log_2 n)$ 시간을 따른다.