Lab13

Evaluation criteria

Category	Evaluation	
p13	100	
Total	100	

- *Use GCC **11** version*
- *No score will be given if the gcc version is different.*

Lab13 – Dijkstra's algorithm

- The deadline for lab13 submission is June 7 at 11:59 PM.

- Folder name : lab13
- code name: p13.c
- Each code will be tested by 5 different input files.
- 20 score for each input, if you don't get the answer you get 0 score.

Lab13 – Dijkstra's algorithm

Graph* createGraph(int X)

- Create a graph with nodes.

Graph* findShortestPath(Graph* G, int s)

- Find the shortest path using Dijkstra's algorithm

void printShortestPath(Graph* G)

- Print the shortest path for the given path

Lab13 – Dijkstra's algorithm

Heap* createMinHeap(int X)

- Create a min heap

void insert (Heap* H, Node N)

- Insert a new node to the min heap

Node deleteMin(Heap* H)

- Delete the node which has the smallest distance

void decreaseKey(Heap* H, Node N)

- Reconstruct the heap

Lab13 – Dijkstra's algorithm

- **Input.txt**

```
4
1 2 3
2 3 2
3 1 5
```

```
4
1 2 3
2 3 2
3 1 5
3 4 1
```

- Line1 : the number of the vertices
- Others : the edge information
  - sourceNode destinationNode edgeWeight

- **Output**

```
2<-1 cost: 3
3<-2<-1 cost: 5
4 can not be reached.
```

```
2<-1 cost: 3
3<-2<-1 cost: 5
4<-3<-2<-1 cost: 6
```

- The result is the shortest path and cost from the vertex 1 to the others

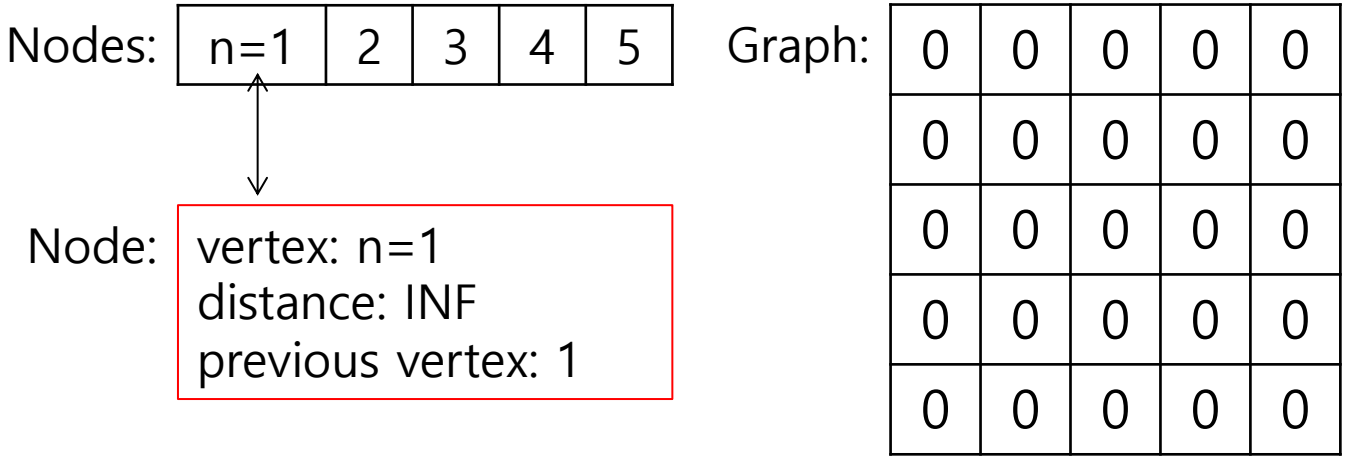
Lab13 – Dijkstra's algorithm

```c
#include<stdio.h>
#include<stdlib.h>

const int INF = (int)2e9;

typedef struct Node{
    int vertex;
    int dist;
    int prev;
}Node;

typedef struct Graph{
    int size;
    Node* nodes;
    int** matrix;
}Graph;

Graph* createGraph(int X);
Graph* findShortestPath(Graph* G, int s);
void printShortestPath(Graph* G);

typedef struct Heap{
    int capacity;
    int size;
    Node* elements;
}Heap;

Heap* createMinHeap(int X);
void insert(Heap* H, Node N);
Node deleteMin(Heap* H);
void decreaseKey(Heap* H, Node N);
```

```c
int main(int argc, char* argv[]){
    FILE *fi = fopen(argv[1], "r");
    int size;

    fscanf(fi, "%d", &size);

    Graph* G = createGraph(size);

    int node_s, node_d, weight;
    while(fscanf(fi, "%d %d %d", &node_s, &node_d, &weight) != EOF){
        G->matrix[node_s][node_d] = weight;
    }

    G = findShortestPath(G, 1);

    printShortestPath(G);

    return 0;
}
```

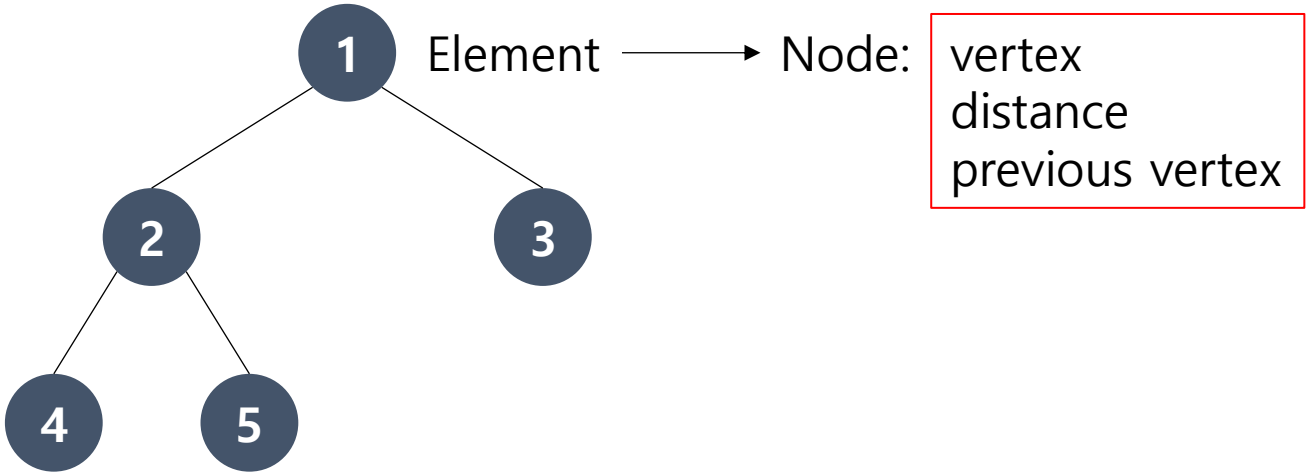
Lab13 – Dijkstra's algorithm

- createGraph

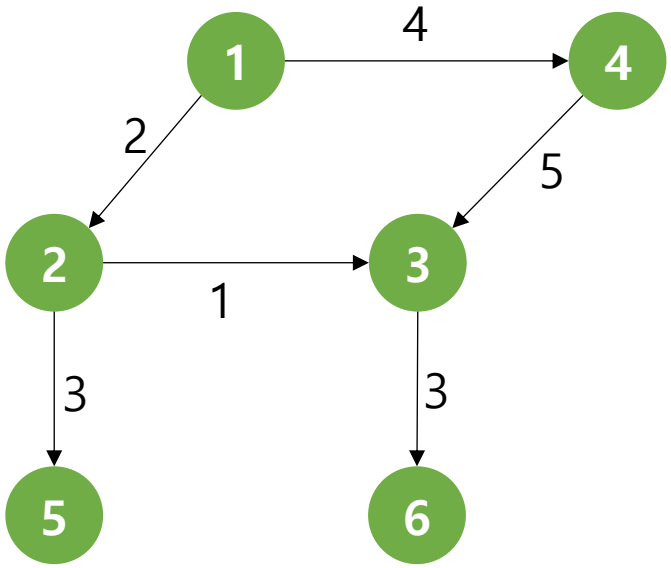


If the vertex is n, it enters the node which index is n.

- createMinHeap

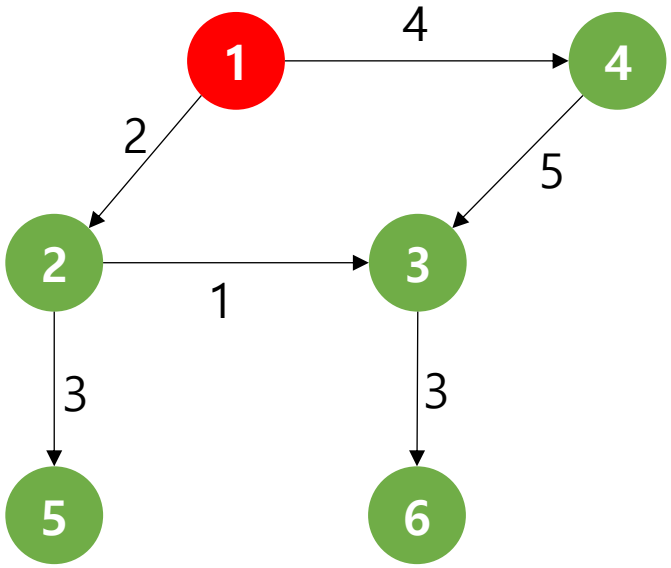


Lab13 – Dijkstra's algorithm

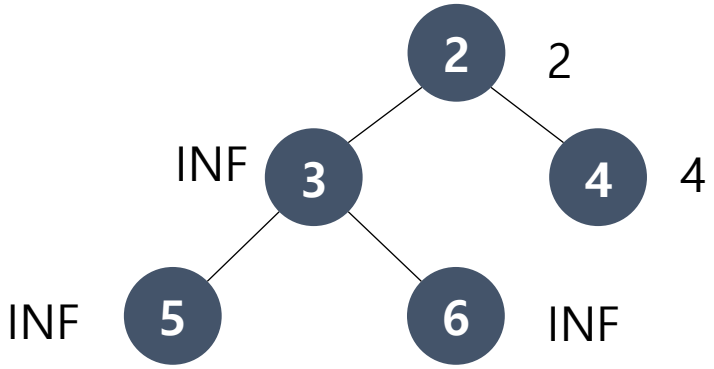


Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: INF prev: null	vertex: 2 dist: INF prev: null	vertex: 3 dist: INF prev: null	vertex: 4 dist: INF prev: null	vertex: 5 dist: INF prev: null	vertex: 6 dist: INF prev: null

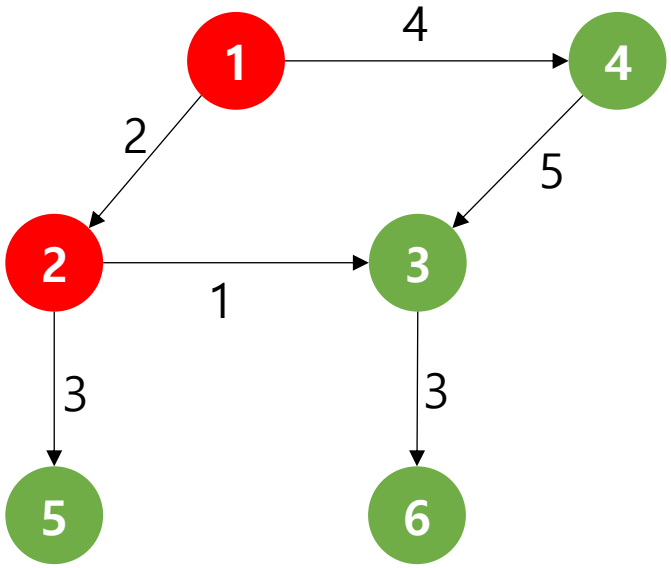
Lab13 – Dijkstra's algorithm



Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: INF prev: null	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: INF prev: null	vertex: 6 dist: INF prev: null



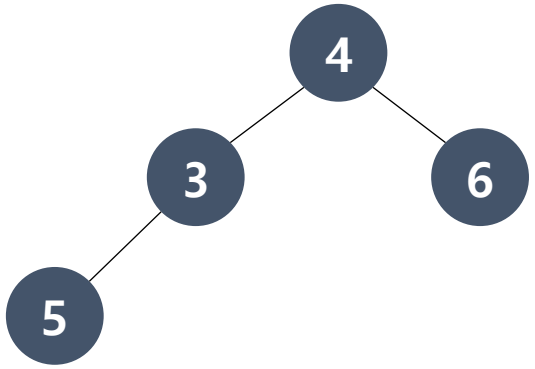
Lab13 – Dijkstra's algorithm



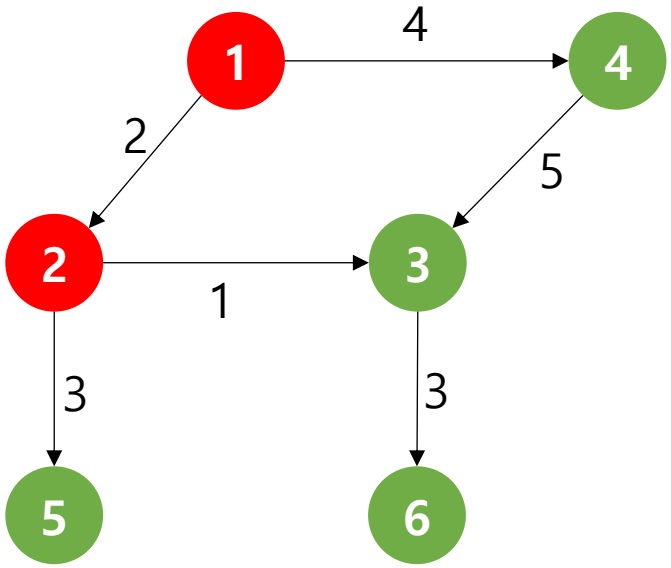
Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: INF prev: null	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: INF prev: null	vertex: 6 dist: INF prev: null

3

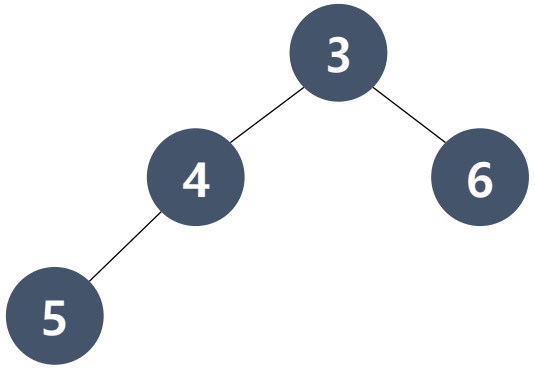
5



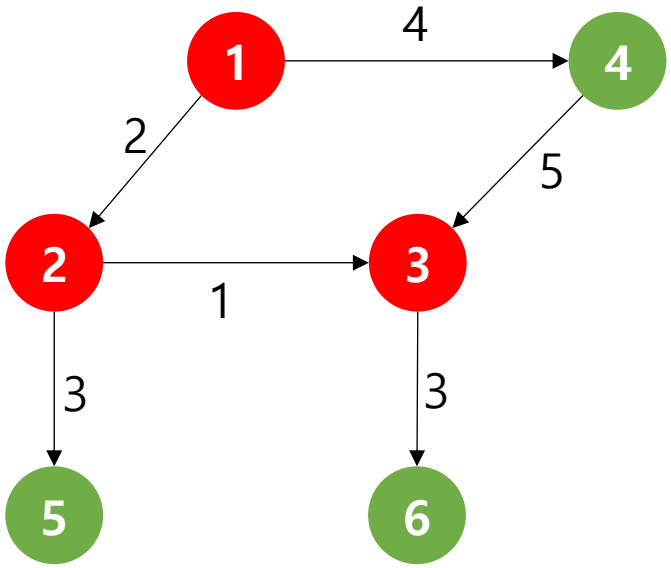
Lab13 – Dijkstra's algorithm



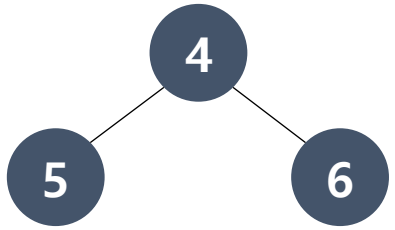
Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: 3 prev: 2	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: 5 prev: 2	vertex: 6 dist: INF prev: null



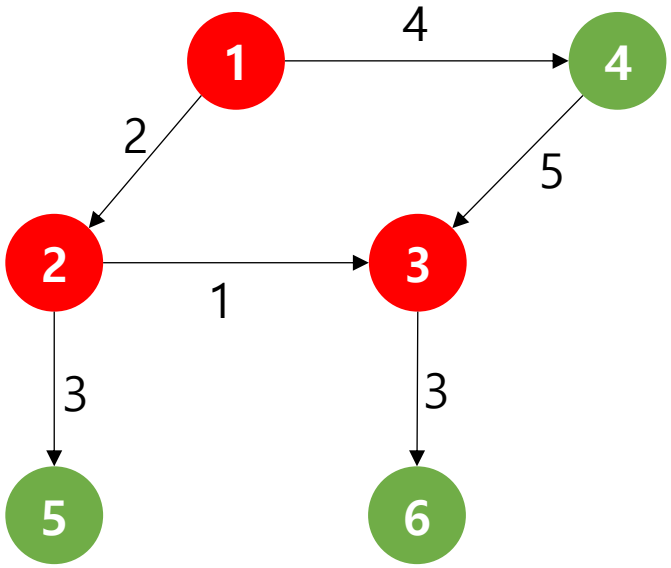
Lab13 – Dijkstra's algorithm



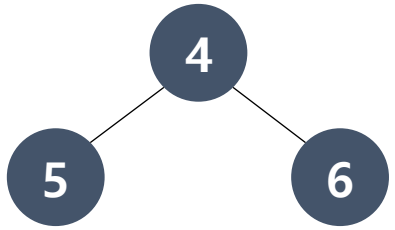
Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: 3 prev: 2	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: 5 prev: 2	vertex: 6 dist: INF prev: null



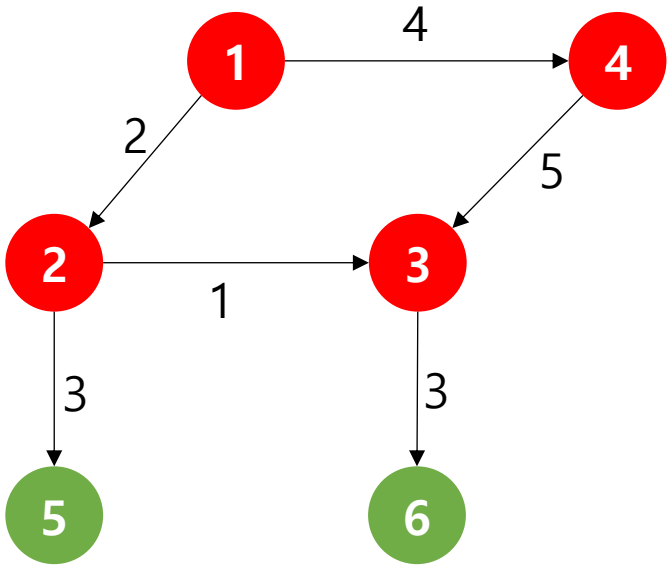
Lab13 – Dijkstra's algorithm



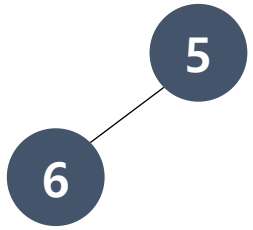
Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: 3 prev: 2	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: 5 prev: 2	vertex: 6 dist: 6 prev: 3



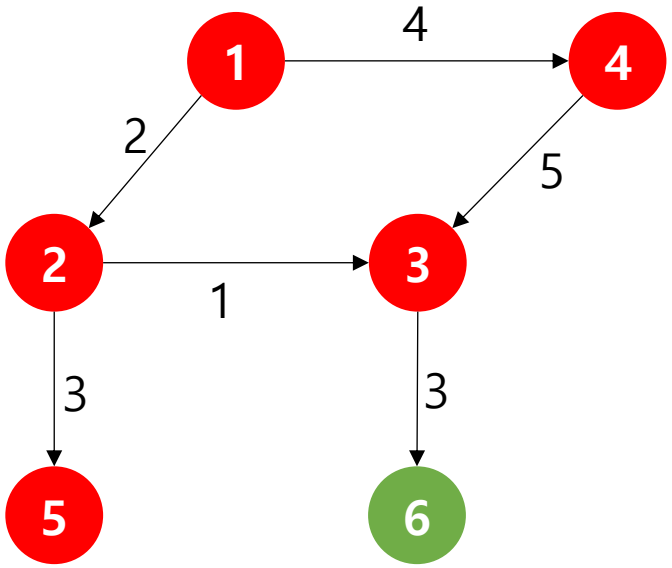
Lab13 – Dijkstra's algorithm



Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: 3 prev: 2	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: 5 prev: 2	vertex: 6 dist: 6 prev: 3



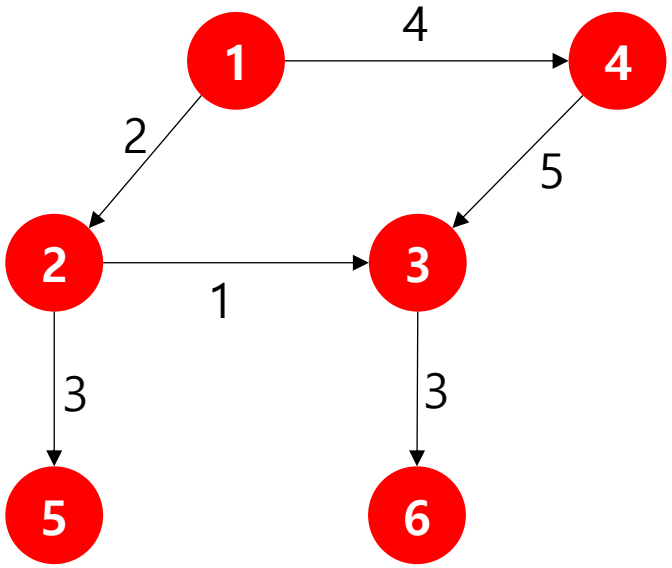
Lab13 – Dijkstra's algorithm



Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: 3 prev: 2	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: 5 prev: 2	vertex: 6 dist: 6 prev: 3

6

Lab13 – Dijkstra's algorithm



Node1	Node2	Node3	Node4	Node5	Node6
vertex: 1 dist: 0 prev: 1	vertex: 2 dist: 2 prev: 1	vertex: 3 dist: 3 prev: 2	vertex: 4 dist: 4 prev: 1	vertex: 5 dist: 5 prev: 2	vertex: 6 dist: 6 prev: 3

2<-1 cost: 2
3<-2<-1 cost: 3
4<-1 cost: 4
5<-2<-1 cost: 5
6<-3<-2<-1 cost: 6

Lab13 – Dijkstra's algorithm

- Input2.txt

```
5
1 2 3
2 3 3
1 3 6
3 4 2
4 5 1
3 5 3
1 5 6
```

- Input3.txt

```
5
1 2 10
1 3 20
1 4 30
1 5 40
2 3 8
2 4 11
2 5 15
3 4 9
3 5 4
4 5 2
```

- Out

```
2<-1 cost: 3
3<-1 cost: 6
4<-3<-1 cost: 8
5<-1 cost: 6
```

- Out

```
2<-1 cost: 10
3<-2<-1 cost: 18
4<-2<-1 cost: 21
5<-3<-2<-1 cost: 22
```