1. PA01-1 친구 찾기

문제

(x, y)점들의 좌표가 주어졌을 때, 가장 가까운 두 점을 찾아 해당 좌표를 출력하고, 두 점 사이의 거리를 계산하는 문제이다.

풀이

- 모든 경우에 대하여 계산하는 경우 : 주어진 n개의 점에 대하여 가능한 모든 직선을 계산해서 최단거리를 찾으면, nC2 = n*(n-1), 즉 O(n^2)의 시간복잡도로 답을 구할 수 있다.

- divide and conquer 방법을 적용하는 경우 : 주어진 n개의 점에 대하여 x값 기준으로 정렬한 후, 상대적으로 왼쪽에 있는 n/2개의 점(group L)과 상대적으로 오른쪽에 있는 n/2개의 점(group R)으로 그룹을 지을 수 있다. 그렇다면 두 점을 이용하여 만들 수 있는 선분은 다음과 같이 3가지 경우로 나뉜다.
  1) group L의 두 점을 이용하여 선분을 만드는 경우
  2) group R의 두 점을 이용하여 선분을 만드는 경우
  3) group L에서 점 하나, group R에서 점 하나를 이용하여 선분을 만드는 경우
  하지만 1), 2) 경우는 n/2개의 점에서 다시 최단거리를 찾는 경우와 같으므로 재귀적으로 처리 할 수 있고, 3)의 경우에는 group L에서 점을 뽑는 경우의 수가 n/2, group R에서 점을 뽑는 경우의 수가 n/2임으로 총 경우의 수는 $\frac{2}{n}\times\frac{2}{n}$= $\frac{n^2}{4}$가지이다. 이 경우 따라서, $T(n)=2T(\frac{n}{2})+\frac{n^2}{4}$로, 마찬가지로 시간 복잡도는 O(n^2)을 가지게 된다.

- 3)에서 경우의 수를 줄이는 방법 : 앞의 divide and conquer 방법에서 시간복잡도를 줄이기 위하여 3)의 경우의 수를 줄이는 방법을 생각해보면 다음과 같다. 1)과 2)에서 나온 최단 거리(d)를 기준으로 3)에서 경우의 수를 줄일 수 있다. group L에서 가장 오른쪽 점과 group R에서 가장 왼쪽 점의 x좌표의 평균(mid_x)을 구하여 해당 값에 대하여 x축에 수직인 직선(x = mid_x)를 그린다. 그런 다음 (mid_x - d, mid_x + d)의 범위에 있는 점에 대하여만 계산하면 된다. (해당 범위의 점들을 group M이라 칭함) 왜냐하면 x값의 거리차이가 d이상이면 당연히 두 점 사이의 거리는 d이상일 수 밖에 없기 때문이다.

x의 범위에 대하여 값들을 걸렀다면, 걸러진 좌표에 대하여 다시 한 번 y의 값에 대하여 거를 수 있다. group M에서 하나의 점을 잡는다면, 그 점에서 y값의 차가 d이하인 범위의 점들만 추가로 계산해주면 된다. 이렇게 할 수 있는 이유는 앞에서와

같다. 또한, group L, group R 사이에서는 거리가 d이상인 점들이기에, 이렇게 x축과 y축에 대하여 자른 범위에 대한 점들의 개수는 상수로 제한된다. 아래의 소스코드에서는 group M에 대한 각 점당 최대 5개의 점에 대하여만 거리를 계산하면 된다.

시간복잡도

$T(n) = 2T(\frac{n}{2}) + Cn$으로 식을 세울 수 있으며 이를 계산하면 O(nlgn)에 포함된다.

소스코드

```
int distance_line(pair <int, int > a, pair <int, int > b){
  int dx = a.first - b.first;
  int dy = a.second - b.second;
  return dx *dx + dy *dy;
}

class line{
public:
  pair <int, int > coo1;
  pair <int, int > coo2;
  int distance;
  line(pair <int, int > a, pair <int, int > b){
    this ->coo1 = a;
    this ->coo2 = b;
    this ->distance = distance_line(a, b);
  }
  line(){
    this ->coo1 =make_pair(-1, -1);
    this ->coo2 =make_pair(-1, -1);
    distance = INT_MAX;
  }
  bool operator <(line & l){
    if(this ->distance < l.distance)
      return true;
    return false;
  }
  line(const line & l){
    this ->coo1 = l.coo1;
    this ->coo2 = l.coo2;
    this ->distance = l.distance;
  }
};
```

line class에 대한 정의.

```
bool compare(pair <int, int > a, pair <int, int > b){
  if(a.second == b.second)
    return a.first < b.first;
  return a.second < b.second;
}
line crossDistnace(line l, vector <pair <int, int >>& coordinates,
double midLine){
  vector <pair <int, int >> candidates;
  for(int i =0; i < coordinates.size(); i ++){
    if(midLine - sqrt(l.distance) <= coordinates[i].first &&
coordinates[i].first <= sqrt(l.distance) + midLine){
      candidates.push_back(coordinates[i]);
    }
  }
  line result;
  if(candidates.size()==0) return result;
  sort(candidates.begin(), candidates.end(), compare);
  double limitDistance = sqrt(l.distance);
  for(int i =0; i < candidates.size()-1; i ++){
    for(int j = i +1; j < candidates.size(); j ++){
      if(candidates[j].second > candidates[i].second + limitDistance)
        break;
      if(result.distance > distance_line(candidates[i],
candidates[j]))
        result = line(candidates[i], candidates[j]);
    }
  }
  return result;
}
```

crossDistance함수는 midLine을 기준으로 group M에 대하여 최단 거리를 찾는 함수이다. 인자로 받는 line l은 group L, group R에서 측정한 최단거리인 선분이며, coordinates는 입력받은 점들의 vector이다.

```
line findMinDistance(vector <pair <int, int >>& coordinates, int
startIdx, int endIdx){
  line l;
  int midIdx = (startIdx + endIdx)/2;
  if(startIdx == endIdx){
    l.distance = INT_MAX;
    return l;
  }
  else if(startIdx +1 == endIdx){
    l = line(coordinates[startIdx], coordinates[endIdx]);
    return l;
  }
  else{
    line tmp = min(findMinDistance(coordinates, startIdx, midIdx),
findMinDistance(coordinates, midIdx +1, endIdx));
    double midLine_x =double(tmp.coo1.first + tmp.coo2.first)/2;
    line res = min(tmp, crossDistnace(tmp, coordinates, midLine_x));
    return res;
  }
}
```

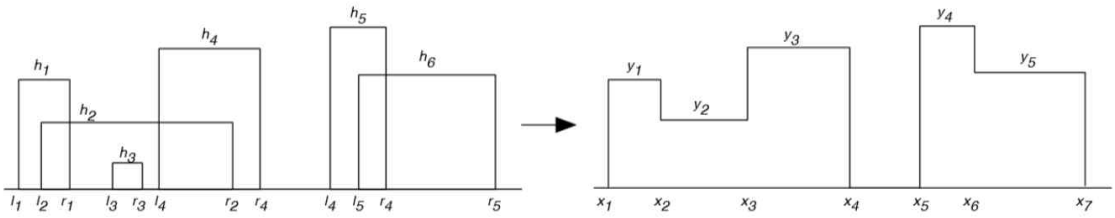
findMinDistance는 좌표값들을 저장한 coordinates를 인자로 받아 최단거리인 선분을 찾아 반환해주는 함수이다. 내부적으로 findMinDistance를 group L, group R에 대하여 실행한 다음, group M에 대하여도 계산하여 최단거리인 선분을 찾는다.

```
int main(void){
  int n;
  ifstream in("input.txt");
  ofstream out("PA01-1.out");
  vector <pair <int, int >> coordinates;
  in >> n;
  for(int i =0; i < n; i ++){
    int x, y;
    in >> x >> y;
    coordinates.push_back(make_pair(x, y));
  }
  sort(coordinates.begin(), coordinates.end());
  out << findMinDistance(coordinates, 0, coordinates.size()-1)
<<endl;
      return 0;
}
```

2. PA01-2 아파트 스카이라인 그리기

문제

건물들에 대한 정보가 주어지면 그림과 같이 건물들의 outline(스카이라인)을 찾는 문제이다.



건물들에 대한 정보는 left, height, right의 순서로 건물의 개수만큼 주어지며, skyline을 구한 다음, 출력은 스카이라인의 높이가 바뀔 때 마다 바뀌는 지점의 x좌표, y좌표를 순서대로 출력해야 한다.

풀이

- merge sort와 비슷하게 생각하여 적용할 수 있다. 크게 divide와 conquer로 과정을 나눌 수 있다. 먼저 divide는 하나의 건물에 대한 정보가 남을 때 까지 한다. 이후 combine은 나눠진 divide된 두 그룹을 합쳐서 스카이라인 정보를 저장하는 과정을 진행하면 된다.

- combine 규칙 : 두 스카이라인을 combine은 하는 규칙은 다음과 같다. 합쳐야하는 것은 스카이라인임으로 스카이라인이 바뀌는 위치는 정보를 가지고 있는 두 그룹이다. 이 두 그룹을 합친 결과 그룹(res)을 새롭게 만들어야한다. 각 위치정보의 먼저 x의 정보를 먼저 비교한다. x값의 정보가 더 작은 쪽을 우선적으로 res에 넣는데, 선택된 x좌표의 그룹과 다른 그룹의 높이값(y값)을 비교하여 더 높은 스카이라인이 있는지 확인을 해야한다. 만약 더 높은 값이 있다면 res에 추가할때는 더 높은 y값을 택하여 (x, y)를 res에 넣으면 된다.

시간복잡도

$$T(n) = 2T(\frac{n}{2}) + n$$

combine하는 과정에서 만들어진 스카이라인 정보 전체를 traverse해야 함으로 n만큼이 소요되며, 이렇게 combine하는 과정은 총 lgn번 진행된다.
따라서 시간복잡도는 O(nlgn)으로 계산된다.

소스코드

```
typedef pair <int ,int > co;
ofstream&operator <<(ofstream & os, const co & p) {
	os << p.first <<", "<< p.second <<endl;
	return os;
}
ofstream&operator <<(ofstream & os, vector <co >& v){
  for(int i =0; i < v.size(); i ++)
    os << v[i];
  return os;
}
```

```
vector <co > merge_s(vector <co >& skyLines, int start, int end){
  vector <pair <int, int >> res;
  if(start ==end){
    res.push_back(skyLines[start *2]);
    res.push_back(skyLines[start *2 +1]);
  }
  if(start <end){
    int mid = (start +end)/2;
    vector <pair <int, int >> first = merge_s(skyLines, start, mid);
    vector <pair <int, int >> second = merge_s(skyLines, mid +1, end);
    res = merge(first, second);
  }
  return res;
}
int main(void){
  int n;
  ifstream in("input.txt");
  ofstream out("PA01-2.out");
  vector <co > skyLines;
  in >> n;
  for(int i =0; i < n; i ++){
    int right;
    int height;
    int left;
    in >> right >> height >> left;
    skyLines.push_back(make_pair(right, height));
    skyLines.push_back(make_pair(left, 0));
  }
  vector <co > res = merge_s(skyLines, 0, n -1);
  out << res <<endl;
	return 0;
}
```

main과 merge_s함수이다. main에서는 right, height, left값을 입력받아, 스카이라인 정보인 (right, height), (left, 0)로 고쳐서 skyLines라는 vector에 넣는다. 이후 스카이라인 정보들을 합쳐 최종적인 스카이라인 정보를 반환해주는 merge_s를 호출한다. merge_s라는 함수는 재귀적으로 호출되며 merge라는 함수를 호출하며 앞에서 말한 combine과정을 실행한다.

```
vector <co > merge(vector <co >& left, vector <co >& right){
  int l_walk =0;
  int r_walk =0;
  int l_height =0;
  int r_height =0;
  vector <co > total;
  while(l_walk < left.size() && r_walk < right.size()){
    if(left[l_walk].first < right[r_walk].first){
      l_height = left[l_walk].second;
      int height = max(left[l_walk].second, r_height);
      if(total.empty() || total.back().second != height){
        total.push_back(make_pair(left[l_walk].first, height));
      }
      l_walk++;
    }
    else if(left[l_walk].first > right[r_walk].first){
      r_height = right[r_walk].second;
      int height = max(right[r_walk].second, l_height);
      if(total.empty() || total.back().second != height){
        total.push_back(make_pair(right[r_walk].first, height));
      }
      r_walk++;
    }
    else{
      l_height = left[l_walk].second;
      r_height = right[r_walk].second;
      int height = max(l_height, right[r_walk].second);
      total.push_back(make_pair(left[l_walk].first, height));
    }
  }
  if(l_walk < left.size()){
    for(int walk = l_walk; walk < left.size(); walk ++)
      total.push_back(left[walk]);
  }
  if(r_walk < right.size()){
    for(int walk = r_walk; walk < right.size(); walk ++)
      total.push_back(right[walk]);
  }
  return total;
}
```

merge 함수는 combine 규칙에 따라 left, right라는 그룹을 인자를 받아 total이라는 그룹으로 만들어 반환해주는 함수이다.