奶酪

读数据

```
typedef struct node_ {
  long long int x;
  long long int y;
  long long int z;
  int near[MAXNSIZE];
  int nearnum;
  int step;
} node;
```

```
for (int d = 0; d < n; d++) {
      scanf("%lld %lld %lld", &Node[d].x, &Node[d].y, &Node[d].z);
    }
```

排序

排序减少后面建边时候的时间

```
int cmp(const void *c, const void *d) {
  node *A = (node *)c;
  node *B = (node *)d;
  return (int)(A->z - B->z);
}
```

```
qsort(Node, n, sizeof(Node[0]), cmp);
```

建边

```
for (int i = 0; i < n; i++) {
      if (Node[i].z - r <= 0) {
        start[startnum++] = i;
        visit[i] = true;
      }

      for (int j = i + 1; j < n; j++) {
        if (Node[j].z - Node[i].z > 2 * r) {
          break;
        }
        if (isCross(Node[i], Node[j]) == true) {
          N + 1ode[i].near[Node[i].nearnum++] = j;
          Node[j].near[Node[j].nearnum++] = i;
        }
      }
    }
```

BFS

```
int rear, front;
    rear = front = 0;
    memset(queue, 0, MAXQSIZE);
    for (int i = 0; i < startnum; i++) {
      queue[rear] = start[i];
      rear = (rear + 1) % MAXQSIZE;
    }

    while (rear != front) {
      int pop = queue[front];
      front = (front + 1) % MAXQSIZE;
      int step_temp = Node[pop].step + 1;
      if (Node[pop].z + r >= h) { // out
        flag = true;
        printf("%d\n", step_temp);
        break;
      } else {
        for (int i = 0; i < Node[pop].nearnum; i++) {
          if (visit[Node[pop].near[i]] == true) {
            continue;
          } else {
            queue[rear] = Node[pop].near[i];
            visit[Node[pop].near[i]] = true;
            rear = (rear + 1) % MAXQSIZE;
            Node[Node[pop].near[i]].step = step_temp;
          }
        }
      }
    }
    if (flag == false) {
      printf("-1\n");
    }
  }
```

源代码

```
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXQSIZE 20005
#define MAXNSIZE 1010

typedef struct node_ {
  long long int x;
  long long int y;
  long long int z;
  int near[MAXNSIZE];
  int nearnum;
  int step;
} node;

int cmp(const void *c, const void *d) {
  node *A = (node *)c;
  node *B = (node *)d;
  return (int)(A->z - B->z);
}

int queue[MAXQSIZE];
int r;
```

```c
bool visit[MAXNSIZE];
node Node[MAXNSIZE];
int start[MAXNSIZE];

bool isCross(node A, node B) {
  double dist = sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y) +
                     (A.z - B.z) * (A.z - B.z));
  if (dist <= 2 * r)
    return true;
  else
    return false;
}

int main() {
  //freopen("input.txt", "r", stdin);
  int T;
  scanf("%d", &T);
  for (int g = 0; g < T; g++) {
    int n = 0, h = 0;
    bool flag = false; // exist out way?

    scanf("%d %d %d", &n, &h, &r);
    memset(Node, 0, sizeof(struct node_) * MAXNSIZE);
    memset(visit, false, sizeof(bool) * MAXNSIZE);

    for (int d = 0; d < n; d++) {
      scanf("%lld %lld %lld", &Node[d].x, &Node[d].y, &Node[d].z);
    }

    qsort(Node, n, sizeof(Node[0]), cmp);

    memset(start, 0, MAXNSIZE);
    int startnum = 0;

    for (int i = 0; i < n; i++) {
      if (Node[i].z - r <= 0) {
        start[startnum++] = i;
        visit[i] = true;
      }

      for (int j = i + 1; j < n; j++) {
        if (Node[j].z - Node[i].z > 2 * r) {
          break;
        }
        if (isCross(Node[i], Node[j]) == true) {
          Node[i].near[Node[i].nearnum++] = j;
          Node[j].near[Node[j].nearnum++] = i;
        }
      }
    }

    int rear, front;
    rear = front = 0;
    memset(queue, 0, MAXQSIZE);

    for (int i = 0; i < startnum; i++) {
      queue[rear] = start[i];
      rear = (rear + 1) % MAXQSIZE;
    }
    while (rear != front) {
      int pop = queue[front];
      front = (front + 1) % MAXQSIZE;
      int step_temp = Node[pop].step + 1;
      if (Node[pop].z + r >= h) { // out
        flag = true;
        printf("%d\n", step_temp);
        break;
      } else {
        for (int i = 0; i < Node[pop].nearnum; i++) {
          if (visit[Node[pop].near[i]] == true) {
            continue;
          } else {
            queue[rear] = Node[pop].near[i];
            visit[Node[pop].near[i]] = true;
            rear = (rear + 1) % MAXQSIZE;
            Node[Node[pop].near[i]].step = step_temp;
          }
        }
      }
    }
    if (flag == false) {
      printf("-1\n");
    }
  }

  return 0;
}
```