1005 列车停放站题解

Gospel_rock

1 题意

给定 n 辆列车的停放区间 $[l_i, r_i]$。m 次询问，每次给出区间 $[x, y]$，求在 $[x, y]$ 内最多能停放多少辆列车。选中的列车区间必须完全包含在 $[x, y]$ 内，且互不重叠（端点可以相切）。

2 贪心策略

忽略多次询问，先考虑单次查询 $[x, y]$：从所有满足 $x \le l_i$ 且 $r_i \le y$ 的列车中，选出最多的互不重叠列车。这是经典的**区间调度最大化**问题，贪心策略为：

> 维护一个指针 p（初始 $p = x$）。每次从所有 $l_i \ge p$ 且 $r_i \le y$ 的列车中，选择**右端点** r_i **最小**的那辆，然后令 $p \leftarrow r_i$，重复此过程直到无法再选。

为什么总是选右端点最小的？设有两辆可选列车 $A = [l_A, r_A]$ 和 $B = [l_B, r_B]$，其中 $r_A < r_B$。选完 A 后，剩余可用空间为 $[r_A, y]$；选完 B 后，剩余空间为 $[r_B, y]$。因为 $r_A < r_B$，前者严格包含后者，所以任何在 $[r_B, y]$ 中可行的后续方案，在 $[r_A, y]$ 中同样可行，反之不然。因此选 A 一定不比选 B 差。

图 1 以样例数据（查询 $[1, 4]$）演示了这一过程。

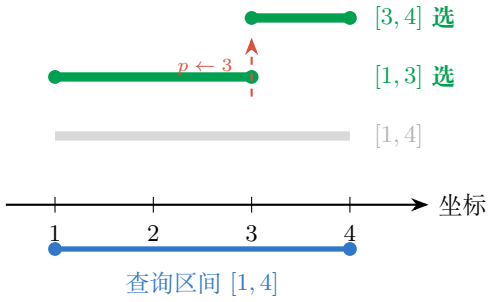

Figure 1: 样例查询 $[1, 4]$ 的贪心过程。初始 $p = 1$，先选右端点最小的 $[1, 3]$，更新 $p = 3$；再选 $[3, 4]$，答案为 2。$[1, 4]$ 因右端点更大而被 $[1, 3]$ 取代。

3 从暴力到倍增

3.1 朴素贪心的瓶颈

对单次查询，贪心过程需逐个选取列车，最多选 $O(n)$ 辆。面对 m 次查询，总时间 $O(nm)$，在 $n, m \le 10^5$ 下不可接受。瓶颈在于：每次查询都要”一步步跳”。能否预处理后一次跳很多步？

3.2 确定性的”下一步”与倍增

贪心过程有一个关键性质：**下一步选哪辆列车，只取决于当前指针 p 的位置**。y 只决定何时停止跳跃，不影响每一步的选择方向。

因此我们可以把这个”下一步”预处理出来，进而用**倍增**加速。定义二维数组 $\texttt{nxt}[i][k]$：

> $\texttt{nxt}[i][k]$：从离散化位置 i 出发，贪心放入 2^k 辆列车后，到达的右端点位置。

基础层 $\texttt{nxt}[i][0]$ 即”从位置 i 出发放 1 辆列车到达的最小右端点”，递推层则将两个半步合并：

$$\texttt{nxt}[i][k] = \texttt{nxt}[\texttt{nxt}[i][k-1]][k-1]$$

含义直观：先跳 2^{k-1} 辆到达中间位置，再从中间位置跳 2^{k-1} 辆，合计 2^k 辆。
图 2 以样例展示了倍增跳跃如何将两步合并为一步。

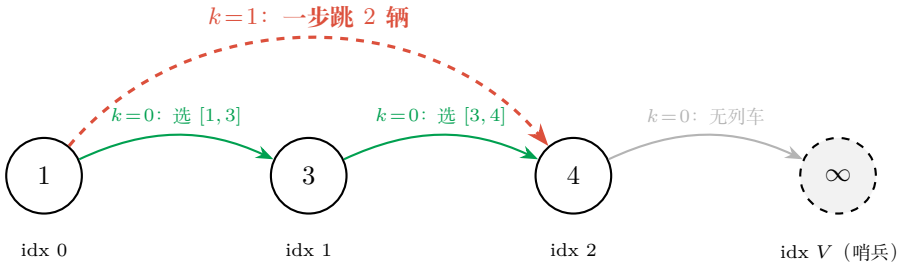

Figure 2: 样例的倍增表。$k=0$ 层（绿色实线）为单步贪心跳跃；$k=1$ 层（红色虚线）将两步合并为一步。查询 $[1,4]$ 时直接用 $k=1$ 一步到位，得到答案 2。

4 实现

4.1 坐标离散化

列车端点坐标范围高达 10^9，但不同端点值至多 $2n$ 个。代码中使用 `Compress_` 模板完成离散化：收集所有列车的 l_i, r_i，排序去重后映射为连续索引。

查询坐标不参与离散化——通过 `projectLr` 方法将原始查询区间 $[x, y]$ 投影到离散化空间。`lower_bound` 找到第一个 $\geq x$ 的位置，`upper_bound` -1 找到最后一个 $\leq y$ 的位置；若投影为空（`ok = false`），直接返回 0：

```
ProjectResult projectLr(T l, T r) {
  int li = (int)(lower_bound(liLs.begin(), liLs.end(), l) - liLs.begin()) + shift;
  int ri = (int)(upper_bound(liLs.begin(), liLs.end(), r) - liLs.begin()) - 1 + shift;
  return {li <= ri, li, ri};
}
```

4.2 构建基础层 dp_mn_r

$\texttt{dp\_mn\_r}[i]$ 表示所有左端点索引 $\geq i$ 的列车中，右端点索引的最小值。这就是贪心的”下一步”——从位置 i 出发放 1 辆列车能到达的最优位置。

构建方式：将离散化后的列车按左端点排序，对每个位置 i 用二分查找定位该位置的所有列车，取右端点最小值，并从右向左维护后缀最小值：

```
void gen_dp_l_mn_r() {
  dp_mn_r.resize(liSZ + 1, INF);
  auto lr = liLr;
  sort(all(lr));
  for (int i = liSZ - 1; i >= 0; --i) {
    ll bg = lower_bound(all(lr), array<int, 2>{i, 0}) - lr.begin();
    ll ed = lower_bound(all(lr), array<int, 2>{i, INF}) - lr.begin();
    dp_mn_r[i] = dp_mn_r[i + 1];
    for (int j = bg; j < ed; ++j) {
      dp_mn_r[i] = min(dp_mn_r[i], lr[j][1]);
    }
  }
}
```

4.3 构建倍增表 nxt

基础层直接来自 dp_mn_r，递推层将两个半步合并：

```
void gen_nxt() {
  nxt.resize(liSZ, vector<int>(lgSZ + 1, INF));
  for (int lay = 0; lay <= lgSZ; ++lay) {
    for (int i = 0; i < liSZ; ++i) {
      if (lay == 0) {
        nxt[i][lay] = dp_mn_r[i];
      } else {
        if (nxt[i][lay - 1] == INF) continue;
        nxt[i][lay] = nxt[nxt[i][lay - 1]][lay - 1];
      }
    }
  }
}
```

4.4 回答查询

对每个查询 $[x, y]$，先用 projectLr 投影到离散化空间 $[l, r]$，然后从大到小枚举层级 k：若跳 2^k 辆后不超过 r，则执行跳跃并累加答案：

```
int reply_query(int bg, int ed) {
  auto [ok, l, r] = compress.projectLr(bg, ed);
  if (!ok) return 0;
  int cur = l;
  int res = 0;
  for (int lay = lgSZ; lay >= 0; --lay) {
    if (nxt[cur][lay] <= r) {
      cur = nxt[cur][lay];
      res += (1 << lay);
    }
  }
  return res;
}
```

5 复杂度分析

- **时间复杂度**：离散化 $O(n \log n)$；gen_dp_l_mn_r 中排序 $O(n \log n)$，每个位置二分查找后遍历，均摊 $O(n \log n)$；倍增表构建 $O(V \cdot \log V)$（$V \le 2n$ 个位置）；每次查询 $O(\log V)$。单组数据总计 $O((n+m) \log n)$。
- **空间复杂度**：倍增表 $O(V \cdot \log V)$。

6 AC 代码

```
// teamname: Gospel_rock
/**
 * Problem: 列车停放站
 * Contest:
 * Judge: HDOJ
 * URL: https://acm.hdu.edu.cn/contest/problem?cid=1200&pid=1005
 * Created: 2026-04-12 15:26:47
 * Author: Gospel_rock
 * My blog: https://znzryb.com/
 *
 * Powered by AutoCp https://github.com/Pushpavel/AutoCp
 */

#include <bits/stdc++.h>
```

```cpp
#define all(vec) vec.begin(),vec.end()
#define lson(o) (o<<1)
#define rson(o) (o<<1|1)
#define SZ(a) ((long long) a.size())
#define debug(var) cerr << #var <<" = ["<<var<<"]"<<"\n";
#define debug1d(a)    \
cerr << #a << " = [";   \
for (int i = 0; i < (int)(a).size(); i++) \
cerr << (i ? ", " : "") << a[i]; \
cerr << "]\n";
#define debug2d(a)  \
cerr << #a << " = [\n";  \
for (int i = 0; i < (int)(a).size(); i++)  \
{   \
cerr << "  [";  \
for (int j = 0; j < (int)(a[i]).size(); j++) \
cerr << (j ? ", " : "") << a[i][j];   \
cerr << "]\n";   \
}  \
cerr << "]\n";
#define cend cerr<<"\n-----------\n"
#define fsp(x) fixed<<setprecision(x)

using namespace std;

using ll = long long;
using ull = unsigned long long;
using DB = double;
using i128 = __int128;
using CD = complex<double>;

static constexpr ll MAXN = (ll) 1e6 + 10;
const int INF = (ll) 1e9 + 7;
static constexpr ll mod = 998244353; // (ll)1e9+7;
static constexpr double eps = 1e-8;
const double PI = acos(-1.0);

ll lT, testcase;

/*
 *
 */

/* 参考了 https://github.com/hh2048/XCPC/blob/484e9e0e6e4f127f187eb3df5bc419e9ec863795/02%20-%20%E6%89%93%E5%8D%B0%E7%A8%BF%E6%A8%A1%E6%9D%BF%E6%B1%87%E6%80%BB/08%20-%20%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84.md#L737
 * AC https://acm.hdu.edu.cn/contest/view-code?cid=1200&rid=14607
 * 上面这个测试了这个 projectLr,get_i 应该是没什么问题
 */
template<typename T>
struct Compress_ {
  // shift=0 就是采用 0-based 下标
  // shift=1 就是采用 1-based 下标
  int n, shift = 0;
  vector<T> liLs;

  Compress_() {
  }

  // 预估大小
  Compress_(ll n_) {
    liLs.reserve(n_);
  }
```

```
Compress_(vector<T> in) : liLs(in) {
  init();
}

void add(T x) {
  liLs.emplace_back(x);
}

template<typename... Args>
void add(T x, Args... args) {
  add(x), add(args...);
}

void init() {
  liLs.emplace_back(numeric_limits<T>::max());
  sort(liLs.begin(), liLs.end());
  liLs.resize(unique(liLs.begin(), liLs.end()) - liLs.begin());
  this->n = liLs.size();
}

int size() {
  return n;
}

int get_i(T x) {
  // 返回 x 元素的新下标
  return lower_bound(liLs.begin(), liLs.end(), x) - liLs.begin() + shift;
}

struct ProjectResult {
  bool ok;
  int l, r;
};

ProjectResult projectLr(T l, T r) {
  // 将区间 [l, r] 投影到离散化空间
  // li: 第一个 >= l 的离散化位置
  // ri: 最后一个 <= r 的离散化位置
  int li = (int) (lower_bound(liLs.begin(), liLs.end(), l) - liLs.begin()) + shift;
  int ri = (int) (upper_bound(liLs.begin(), liLs.end(), r) - liLs.begin()) - 1 + shift;
  return {li <= ri, li, ri};
}

T operator[](int idx) {
  // 根据新下标返回原来元素
  assert(idx - shift < n);
  return idx - shift < n ? liLs[idx - shift] : -1;
}

bool contains(T x) {
  // 查找元素 x 是否存在
  return binary_search(liLs.begin(), liLs.end(), x);
}

void reserve(ll n_) {
  liLs.reserve(n_);
}

friend auto &operator<<(ostream &o, const auto &j) {
  cout << "{";
  for (auto it: j.alls) {
    o << it << " ";
  }
```

```
    return o << "}";
  }
};

bool cmp(const array<ll, 2> &a, const array<ll, 2> &b) {
  if (a[1] != b[1]) return a[1] < b[1];
  return a[0] < b[0];
}

struct Solve {
  int N;
  int M;
  vector<array<int, 2> > lrLs;
  vector<array<int, 2> > liLr;
  vector<array<int, 2> > queryLs;
  Compress_<int> compress;
  vector<vector<int> > nxt;
  vector<int> dp_mn_r;
  int liSZ, lgSZ;


  void gen_dp_l_mn_r() {
    // 我们这里是要求每个 l 所能搞到的最小 r
    dp_mn_r.resize(liSZ + 1, INF);
    auto lr = liLr;
    sort(all(lr));
    for (int i = liSZ - 1; i >= 0; --i) {
      ll bg = lower_bound(all(lr), array<int, 2>{i, 0}) - lr.begin();
      ll ed = lower_bound(all(lr), array<int, 2>{i, INF}) - lr.begin();
      dp_mn_r[i] = dp_mn_r[i + 1];
      for (int j = bg; j < ed; ++j) {
        dp_mn_r[i] = min(dp_mn_r[i], lr[j][1]);
      }
    }
#ifdef LOCAL
    debug1d(dp_mn_r);
#endif
  }

  void gen_nxt() {
    nxt.resize(liSZ, vector<int>(lgSZ + 1, INF));
    // 我们可不可以，就是对于每一个位置，我们都能知道，他跳 1<<i 辆列车，可以跳到的 r 点
    for (int lay = 0; lay <= lgSZ; ++lay) {
      for (int i = 0; i < liSZ; ++i) {
        if (lay == 0) {
          nxt[i][lay] = dp_mn_r[i];
        } else {
          if (nxt[i][lay - 1] == INF) continue;
          nxt[i][lay] = nxt[nxt[i][lay - 1]][lay - 1];
#ifdef LOCAL
          if (i == 0 || i == 1) {
            debug(i);
            debug(nxt[i][lay - 1]);
            cend;
          }
#endif
        }
      }
    }
#ifdef LOCAL
    debug2d(nxt);
#endif
  }
```

```
  int reply_query(int bg, int ed) {
    auto [ok,l,r] = compress.projectLr(bg, ed);
    if (!ok) return 0;
    int cur = l;
    int res = 0;
    for (int lay = lgSZ; lay >= 0; --lay) {
      if (nxt[cur][lay] <= r) {
        cur = nxt[cur][lay];
        res += (1 << lay);
      }
    }
    return res;
  }

  Solve() {
    cin >> N;
    lrLs.resize(N);
    compress.reserve(2 * (N + M));

    for (int i = 0; i < N; ++i) {
      cin >> lrLs[i][0] >> lrLs[i][1];
      compress.add(lrLs[i][0], lrLs[i][1]);
    }
    cin >> M;
    queryLs.resize(M);
    for (int i = 0; i < M; ++i) {
      cin >> queryLs[i][0] >> queryLs[i][1];
      // compress.add(queryLs[i][0], queryLs[i][1]);
    }
    compress.init();
    liSZ = compress.size(), lgSZ = __lg(compress.size());
    liLr.reserve(N);
    for (auto [l,r]: lrLs) {
      // auto [ok,bg,ed] = compress.projectLr(l, r);
      // if (ok) liLr.push_back({bg, ed});
      liLr.push_back({compress.get_i(l), compress.get_i(r)});
    }
    gen_dp_l_mn_r();
    gen_nxt();
    for (auto [l,r]: queryLs) {
      cout << reply_query(l, r) << "\n";
    }
  }
};

signed main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cout.tie(nullptr);
#ifdef LOCAL
  cout.setf(ios::unitbuf); // 无缓冲流，方便我们调试
#endif

  cin >> lT;
  for (testcase = 1; testcase <= lT; ++testcase)
    Solve solve;
  return 0;
}

/*
AC
https://acm.hdu.edu.cn/contest/view-code?cid=1200&rid=14607
```

```
*/
```