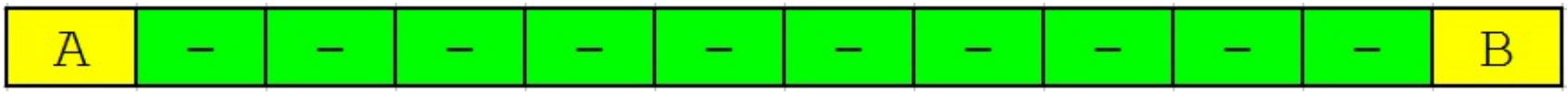
倍增与 ST 表

墨宇

2020 年 8 月 3 日

引子

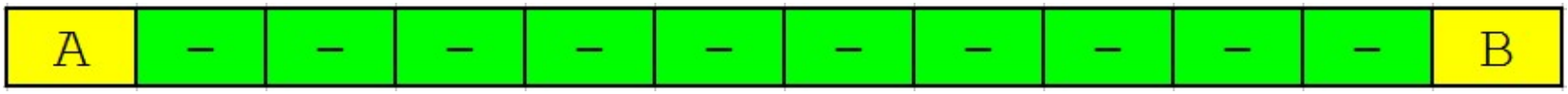
从前，有三只可爱得不得了的小白兔，它想从 A 地去往遥远的 B 地。



2B 小白兔：
向右边跳一步，左边跳一步，再向右边跳很多步，再……（对不起，这个太脑残了）

引子

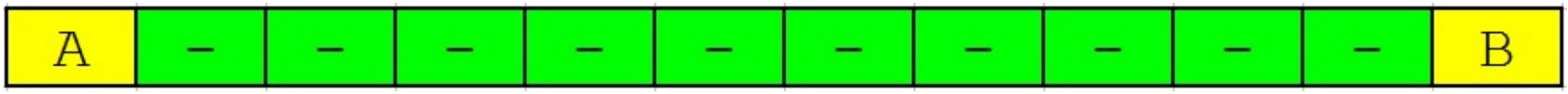
从前，有三只可爱得不得了的小白兔，它想从 A 地去往遥远的 B 地。



2B 小白兔：
向右边跳一步，左边跳一步，再向右边跳很多步，再……（对不起，这个太脑残了）
普通小白兔：
向右边跳一步，再跳一步，再跳一步……再跳一步，哇，到了！好开心！

引子

从前，有三只可爱得不得了的小白兔，它想从 A 地去往遥远的 B 地。



2B 小白兔：
向右边跳一步，左边跳一步，再向右边跳很多步，再……（对不起，这个太脑残了）
普通小白兔：
向右边跳一步，再跳一步，再跳一步……再跳一步，哇，到了！好开心！
超级小白兔：
向右边跳一大步，一步跳到 B，然后默默回头，鄙视一下那只一步一步跳的小白兔.

引子

我相信作为一个正常人，是不会考虑到 2B 小白兔的这种做法的，因为它太脑残了。
同时我也相信，作为一个正常人，也不会考虑到超级小白兔的这种做法的，因为……

引子

我相信作为一个正常人，是不会考虑到 2B 小白兔的这种做法的，因为它太脑残了。
同时我也相信，作为一个正常人，也不会考虑到超级小白兔的这种做法的，因为……

“我擦！你什么时候说可以这样跳了！(愤怒)”
“我什么时候说不可以了！(卖萌)”

引子

我相信作为一个正常人，是不会考虑到 2B 小白兔的这种做法的，因为它太脑残了。
同时我也相信，作为一个正常人，也不会考虑到超级小白兔的这种做法的，因为……

> “我擦！你什么时候说可以这样跳了！(愤怒)”
> “我什么时候说不可以了！(卖萌)”

但是你不得不承认，超级小白兔还是有两把刷子的，因为它真的是太厉害了，厉害得你想都没有想到。

又来

从前，有三只可爱得不得了的小白兔，它想从 A 地去往遥远的 B、C、D、E、F 这几个让它魂牵梦萦的地方。(不要问我从哪里来，我的梦想在远方)

2B 小白兔：
(对不起，我的生命有限，我不想再提到它了)

又来

从前，有三只可爱得不得了的小白兔，它想从 A 地去往遥远的 B、C、D、E、F 这几个让它魂牵梦萦的地方。(不要问我从哪里来，我的梦想在远方)

2B 小白兔：
(对不起，我的生命有限，我不想再提到它了)
普通小白兔：
一步又一步，生命不息，跳跃不止。

又来

从前，有三只可爱得不得了的小白兔，它想从 A 地去往遥远的 B、C、D、E、F 这几个让它魂牵梦萦的地方。(不要问我从哪里来，我的梦想在远方)

2B 小白兔:
(对不起，我的生命有限，我不想再提到它了)
普通小白兔:
一步又一步，生命不息，跳跃不止。
超级小白兔:
一步到 B，再一步到 C，再一步到 D，再一步到 E，再一步到 F，完工。

又来

从前，有三只可爱得不得了的小白兔，它想从 A 地去往遥远的 B、C、D、E、F 这几个让它魂牵梦萦的地方。(不要问我从哪里来，我的梦想在远方)

2B 小白兔:
(对不起，我的生命有限，我不想再提到它了)
普通小白兔:
一步又一步，生命不息，跳跃不止。
超级小白兔:
一步到 B，再一步到 C，再一步到 D，再一步到 E，再一步到 F，完工。

又来

你不用解释，我深知你就想当那只超级小白兔，哼，你肯定是那样跳的。(神马？不是？兄弟你的智商我救不了了……)

又来

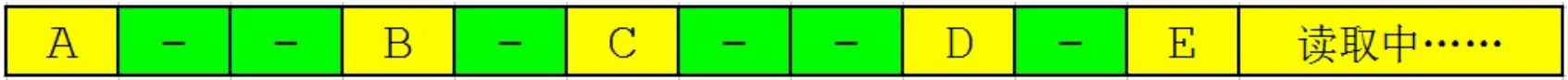
你不用解释，我深知你就想当那只超级小白兔，哼，你肯定是那样跳的。(神马？不是？兄弟你的智商我救不了了……)
是的，不这样跳的人对不起社会啊，浪费时间就是浪费青春，浪费青春就是犯罪。

又来

你不用解释，我深知你就想当那只超级小白兔，哼，你肯定是那样跳的。(神马？不是？兄弟你的智商我救不了了……)
是的，不这样跳的人对不起社会啊，浪费时间就是浪费青春，浪费青春就是犯罪。
好的，既然你是这样跳的，那你能告诉我你是怎么知道从 A 只要跳 2 个格子就会刚好到 B 的？难道你在空中使用了 GPS 全球卫星定位系统？你少来好么！主页君现在都没用过这种东西，你一只白白嫩嫩下酒菜的小兔子还用这个？笑死我吗？哈哈，你一定是出发前就偷偷学普通兔子一步一步跳过一遍，然后拿个本子做小抄，记录好从每个格子跳任意步会到达的地方，然后赶在天亮之前回来，风光的按照踩点计划大步的跳，让我们觉得你很厉害的样子，我没说错吧？不过看在你有这个诚心的份上，还是为你的聪明鼓掌吧。

还来

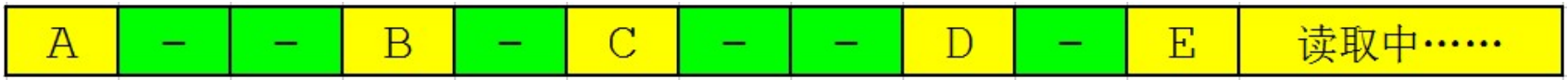
从前，有二只可爱得不得了的小白兔，它想从 A 地去往遥远的 1 (此处省略很多 0) 个地方，因为它真的是太没事情做了。



普通小白兔：
从离开家门的那天起，我就没有想过要放弃一步一步地跳往终点。(嗯，加油)

还来

从前，有二只可爱得不得了的小白兔，它想从 A 地去往遥远的 1 (此处省略很多 0) 个地方，因为它真的是太没事情做了。



普通小白兔：
从离开家门的那天起，我就没有想过要放弃一步一步地跳往终点。(嗯，加油)
超级小白兔：
轻轻松松，绝不多走一步。(哼哼)

还来

你想知道最后的结果吗？呵呵，好像还没有出结果……

还来

你想知道最后的结果吗？呵呵，好像还没有出结果……
写给普通小白兔的话：
亲爱的小白兔，我知道你勇毅，你质朴，但是，苦海无涯，回头是岸。

还来

你想知道最后的结果吗？呵呵，好像还没有出结果……
写给普通小白兔的话：
亲爱的小白兔，我知道你勇毅，你质朴，但是，苦海无涯，回头是岸。
写给超级小白兔的话：
我不知道你的小抄本是否还够用，我不知道你摸着黑就出门是为了什么，你不觉得你的行踪早就已经暴露了吗？你以为你很聪明吗？不，你错了，你就一下酒菜，永远都是，因为你不知道倍增算法，这是你失败的根源，再见，我心中永远不会逝去的蠢兔子。

思考

那么，如何让超级小白兔的小抄加速呢？当然是倍增算法啦。

思考

那么，如何让超级小白兔的小抄加速呢？当然是倍增算法啦。
我们可以不记录从每一点到任意长度的距离，而是从“2”入手。(2B 小白兔:???)
记录从每个点跳 $2\ 4\ 8\ 16\cdots$ 步后的点值即可。

LCA 介绍

最近公共祖先简称 LCA（Lowest Common Ancestor）。两个节点的最近公共祖先，就是这两个点的公共祖先里面，离根最远的那个。为了方便，我们记某点集 $S = v_1, v_2, \ldots, v_n$ 的最近公共祖先为 $\mathrm{LCA}(v_1, v_2, \ldots, v_n)$ 或 $\mathrm{LCA}(S)$ 。

LCA 性质

1. $\mathrm{LCA}(u) = u$;

LCA 性质

1. $\mathrm{LCA}(u) = u$；
2. u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u, v) = u$；

LCA 性质

1. $\mathrm{LCA}(u) = u$;
2. u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u, v) = u$;
3. 如果 u 不为 v 的祖先并且 v 不为 u 的祖先，那么 u, v 分别处于 $\mathrm{LCA}(u, v)$ 的两棵不同子树中;

LCA 性质

1. $\mathrm{LCA}(u) = u$；
2. u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u, v) = u$；
3. 如果 u 不为 v 的祖先并且 v 不为 u 的祖先，那么 u, v 分别处于 $\mathrm{LCA}(u, v)$ 的两棵不同子树中；
4. 前序遍历中，$\mathrm{LCA}(S)$ 出现在所有 S 中元素之前，后序遍历中 $\mathrm{LCA}(S)$ 则出现在所有 S 中元素之后；

LCA 性质

1. $\mathrm{LCA}(u) = u$ ；
2. u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u, v) = u$ ；
3. 如果 u 不为 v 的祖先并且 v 不为 u 的祖先，那么 u, v 分别处于 $\mathrm{LCA}(u, v)$ 的两棵不同子树中；
4. 前序遍历中，$\mathrm{LCA}(S)$ 出现在所有 S 中元素之前，后序遍历中 $\mathrm{LCA}(S)$ 则出现在所有 S 中元素之后；
5. 两点集并的最近公共祖先为两点集分别的最近公共祖先的最近公共祖先，即 $\mathrm{LCA}(A \cup B) = \mathrm{LCA}(\mathrm{LCA}(A), \mathrm{LCA}(B))$ ；

LCA 性质

1. $\mathrm{LCA}(u) = u$ ；
2. u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u, v) = u$ ；
3. 如果 u 不为 v 的祖先并且 v 不为 u 的祖先，那么 u, v 分别处于 $\mathrm{LCA}(u, v)$ 的两棵不同子树中；
4. 前序遍历中，$\mathrm{LCA}(S)$ 出现在所有 S 中元素之前，后序遍历中 $\mathrm{LCA}(S)$ 则出现在所有 S 中元素之后；
5. 两点集并的最近公共祖先为两点集分别的最近公共祖先的最近公共祖先，即 $\mathrm{LCA}(A \cup B) = \mathrm{LCA}(\mathrm{LCA}(A), \mathrm{LCA}(B))$ ；
6. 两点的最近公共祖先必定处在树上两点间的最短路上；

LCA 性质

1. $\mathrm{LCA}(u) = u$ ；
2. u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u, v) = u$ ；
3. 如果 u 不为 v 的祖先并且 v 不为 u 的祖先，那么 u, v 分别处于 $\mathrm{LCA}(u, v)$ 的两棵不同子树中；
4. 前序遍历中，$\mathrm{LCA}(S)$ 出现在所有 S 中元素之前，后序遍历中 $\mathrm{LCA}(S)$ 则出现在所有 S 中元素之后；
5. 两点集并的最近公共祖先为两点集分别的最近公共祖先的最近公共祖先，即 $\mathrm{LCA}(A \cup B) = \mathrm{LCA}(\mathrm{LCA}(A), \mathrm{LCA}(B))$ ；
6. 两点的最近公共祖先必定处在树上两点间的最短路上；
7. $d(u, v) = h(u) + h(v) - 2h(\mathrm{LCA}(u, v))$ ，其中 d 是树上两点间的距离，h 代表某点到树根的距离。

利用倍增思想解决 LCA 问题

这个思想可以解决 LCA 问题!

利用倍增思想解决 LCA 问题

这个思想可以解决 LCA 问题!
可以记录每个节点跳 $2\ 4\ 8\ 16\cdots$ 步后的节点，比较并跳跃即可.
参考代码如下:

代码

```cpp
int fa[maxn][25];//fa[1][0] = -1
int dep[maxn];
//预处理
void dfs(int u)
{
    for(auto v : G[u])
    {
        fa[v][0] = u;
        dep[v] = dep[u] + 1;
        dfs(v);
    }
}
void init()
{
    fa[1][0] = -1;
    dep[1] = 1;
    dfs(1); //预处理father
    for(int j = 1;j <= 20;j++)
        for(int i = 1;i <= n;i++)
            dp[i][j] = dp[i][j-1] + dp[i+1<<(j-1)][j-1];
}
int lca(int x,int y)
{
    if(dep[x] < dep[y]) swap(x,y);
    if(dep[x] != dep[y])
        for(int i = 20;i >= 0;i--)
        {
            if(dep[fa[x][i]] < dep[y]) continue;
            x = fa[x][i];//跳到同一级别高度.
        }
    if(x == y) return x;
    for(int i = 20;i >= 0;i--)
    {
        if(fa[x][i] == fa[y][i]) continue;
        x = fa[x][i];
        y = fa[y][i];
    }
    if(x == y) return x;
    return fa[x][0];
}
```

介绍

RMQ

RMQ 是英文 Range Maximum/Minimum Query 的缩写，表示区间最大（最小）值。

利用倍增思想解决 RMQ 问题

倍增思想可以解决 RMQ 问题！

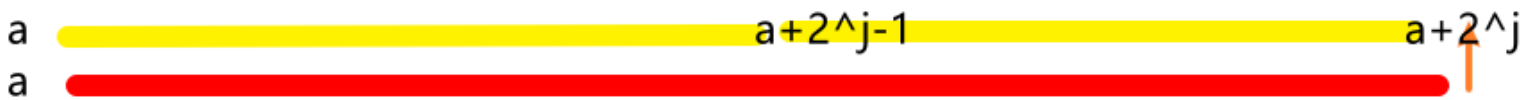
利用倍增思想解决 RMQ 问题

倍增思想可以解决 RMQ 问题!
可以记录每个点跳 $2\ 4\ 8\ 16\cdots$ 步后的最小值，转移方程为:

$$a[i][j] := a跳2^j步最小值 \tag{1}$$

$$a[i][j] = min(a[i][j-1], a[i+2^{j-1}][j-1]) \tag{2}$$

画出来差不多是这样的:



参考代码

```
int main() {
    log[1] = 0;
    for (int i = 2; i <= n; ++i)
        log[i] = log[i>>1] + 1;
    for (int i = 1; i <= n; ++i)
        f[0][i] = a[i];
    for (int j = 1; j < 17; ++j)
        for (int i = 1; i <= n; ++i)
            f[j][i] = min(f[j-1][i], f[j-1][i+(1<<(j-1))]);
    while (m--) {
        scanf("%d%d", &l, &r);
        int i = l;
        int s = 1e9;
        for (int j = 17-1; j > -1; --j)
            if (r-l+1 & 1<<j) {
                s = min(s, f[j][i]);
                i += 1<<j;
            }
        printf("%d\n", s);
    }
}
```

继续优化

该方法还可以进一步优化：
由于最小值可以重叠，答案就是：

$$min(a[l][log_2^{r-l+1}], a[r - log_2^{r-l+1} + 1][log_2^{r-l+1} + 1] \quad (3)$$

参考代码

```
int main() {
    log[1] = 0;
    for (int i = 2; i <= n; ++i)
        log[i] = log[i>>1] + 1;
    for (int i = 1; i <= n; ++i)
        f[0][i] = a[i];
    for (int j = 1; j < 17; ++j)
        for (int i = 1; i <= n; ++i)
            f[j][i] = min(f[j-1][i], f[j-1][i+(1<<(j-1))]);
    while (m--) {
        scanf("%d%d", &l, &r);
        // int s = log(r-l+1) / log(2);
        int s = log[r-l+1];
        printf("%d\n", min(f[s][l], f[s][r-(1<<s)+1]);
    }
}
```