T1

本质上就是括号匹配。

`R` 视为左括号，`L` 视为右括号，成功匹配的左右括号之间的距离，就是这两个位置的消失时间。

无法完成匹配的位置，就永远无法消失。

T2

直接模拟。

如何判断一个串是否合法?

- 先出现 3 个 `.` 再出现 1 个 `:`。
- 串首尾一定是数字。
- `.` 和 `:` 不会出现在相邻位置。
- 若数字的首位为 `0` 则下一位必须是 `.` 或 `:`。

当确认了串的形式为 `a.b.c.d:e` 后，可以通过格式化输入得到数字的值。

```
// x 是读入的 string 类型的串
sscanf(x.c_str(),"%lld.%lld.%lld.%lld:%lld",&a,&b,&c,&d,&e);
```

接下来，再判断数字是否是允许范围内的值。

对于连接状态的判定，利用 `map` 可以轻松解决。

T3

10 分:

分类讨论。

50 分:

区间动态规划问题。

$f[l][r]$ 表示合并区间 $[l,r]$ 内所有小球为一个小球时，该小球的最大权值。

只考虑合并相邻小球时，$f[l][r]=OR(f[l][k]\ AND\ f[k+1][r]),l\leq k<r$。

考虑合并三个小球时，
$f[l][r]=OR(f[l][k1]\ AND\ f[k2][r]),l<=k1<k2<=r,k1+1\leq k2,f[l][k1]=f[k2][r]$
。

时间复杂度 $O(n^4)$。

100 分:

上一档部分分中，$k1$ 右移时，$l\sim k1$ 的值单调不降；$k2$ 左移时，$k2\sim r$ 的值单调不降。

因此，可以使用双指针法维护 $k1,k2$ 两个变量。

具体来说：

- $l\sim k1$ 的和小于 $k2\sim r$ 的和，则 $k1$ 右移。
- $l\sim k1$ 的和大于 $k2\sim r$ 的和，则 $k2$ 左移。
- 两侧相等时，若 $f[l][k1]\ AND\ f[k1+1][k2-1]\ AND\ f[k2][r]$ 为真，则 $l\sim r$ 可以合并为一个小球，结束当前区间的判断；否则 $k1$ 继续右移。
- 当 $k1>k2$ 时仍未找到可行方案，则当前区间不能合并为一个小球。

时间负责度 $O(n^3)$。

最终答案为 $max\ \{f[l][r]\times(sum[r]-sum[l-1])\},1\leq l\leq r\leq n$。

T4

20 分:

留给最朴素的 dfs。

40 分:

留给强一点的暴力做法。

100 分:

最多的花费尽量少，考虑二分。

问题转化为，对于一个二分出来的 mid，是否能够在经过不超过 k 条边权大于 mid 的边的情况下，从 1 号点走到 n 号点。

此时，若将边权大于 mid 的边视为 1，其余的边视为 0，则原图可以转化为一个仅由 0 和 1 两种边权的边。于是，问题又可以转化为新图（可以不直接建出来）上的最短路问题。我们需要检验 1 到 n 的最短路长度是否不超过 k。

这里介绍 01BFS 的做法，可以在 $O(n+m)$ 的时间复杂度内求解边权只有 0 和 1 的图上的单源最短路问题。

首先，建立一个双端队列，可以用 `deque`。初始化除起点外所有节点的最短路径长度为无穷大，起点的最短路径为 0。

- 起点入队。
- 若队列不为空，则令队首节点 x 出队（pop_front()）。若 x 是第一次从队列中取出，则记录当前距离为起点到 x 的最短路，否则跳过点 x，重新取出新的队首节点。
- 对于第一次取出的点 x，遍历其出边更新其他节点。设当前被更新的节点为 y。若是边权为 0 的边更新了 y，则将 y 从队首入队（push_front()）；若是边权为 1 的边更新了 y，则将 y 从队尾入队（push_back()）。

外层二分，内层 01BFS，时间复杂度为 $O((n+m)log(w))$。