T1 小苹果

题意：n个苹果，每轮从第一个苹果开始，隔两个拿走一个苹果。
剩下的苹果会合成一个新的序列，重复上述操作，直到全部拿完。

问多久能拿走全部的苹果，以及第几轮拿走第n个苹果。

思路：

考虑第一问，注意到每次取走一些苹果后会变成另一个子问题，且 $n' = n - \lceil \frac{n}{3} \rceil$。这样每次n会变成 $\frac{2}{3}n$，故暴力维护n的大小即可，最多只会有 $\mathcal{O}(\log n)$ 轮。

对于第二问，用同样的办法，n永远都是最后一个。当$n\%3 = 1$时，最后一个苹果被拿走。

```
#include<cstdio>
using namespace std;
int n;

int main()
{
    scanf("%d", &n);
    int id = 0;
    int t = 0;
    while(n) {
        ++t;
        if((n + 2) % 3 == 0 && !id) id =  t;
        n = n - ((n + 2) / 3);
    }
    printf("%d %d\n", t, id);
    return 0;
}
```

T2 公路

题意：有n个加油站，每个加油站有一个油价，油箱无限大，问从第一个加油站出发，到第n个加油站，最少需要多少钱。

考虑贪心。每次要走到下一个加油站的时候，所用的油价是前面的加油站的最便宜的油价。因此，对于每个加油站，记录下油价的前缀min把它作为走到下一个加油站的油价。由于距离不一定整除单位体积的油可以移动的距离，因此需要新增一个变量记录有多少距离是可以免费走的。

```
#include<cstdio>
#include <algorithm>
using namespace std;
#define ll long long
#define maxn 100005
int n, d;

ll v[maxn];
ll a[maxn];
```

```
int main()
{
    scanf("%d%d", &n, &d);
    for(int i = 2;i <= n;i++) scanf("%lld", &v[i]);
    for(int i = 1;i <= n;i++) scanf("%lld", &a[i]);
    ll mi = a[1], freed = 0;
    ll ans = 0;
    ll t, k;
    for(int i = 2;i <= n;i++) {
        if(freed >= v[i]) {
            freed -= v[i];
        }
        else {
            t = v[i] - freed;
            freed = 0;
            k = t / d;
            if(t % d) k++;
            ans += k * mi;
            freed = k * d - t;
        }
        mi = min(mi, a[i]);
    }
    printf("%lld\n", ans);
    return 0;
}
/*
5 4
10 10 10 10
9 8 9 6 5
*/
```

T3 一元二次方程

按题意要求模拟即可，需要注意的点：

1. 当 delta 是完全平方数时，没有 sqrt 项
2. 当有 sqrt 项时，第一项是 0 无需输出
3. 当有 sqrt 项时，第二项分子是 1 无需输出
4. 分母是 1 时，无需输出

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

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  std::cout.tie(nullptr);
  int testcase, upperbound;
  std::cin >> testcase >> upperbound;
```

```cpp
  while (testcase--) {
    int A, B, C;
    std::cin >> A >> B >> C;
    int D = B * B - 4 * A * C;
    if (D < 0) {
      std::cout << "NO" << std::endl;
      continue;
    }
    // x=-B/2A+sqrt(D)/2A
    //  =p1/q1+p2/q2*sqrt(r)
    auto simplify = [&](int& p, int& q) {
      int g = std::gcd(p, q);
      p /= g;
      q /= g;
      if (q < 0) {
        p = -p;
        q = -q;
      }
    };

    int p1 = -B, q1 = 2 * A;

    int p2 = 1, q2 = 2 * A, r = 0;
    for (int t = 1; t * t <= D; ++t) {
      if (D % (t * t)) continue;
      r = D / (t * t);
      p2 = t;
    }

    if (r == 1 || r == 0) {
      int p = p1 + p2 * r, q = 2 * A;
      simplify(p, q);
      std::cout << p;
      if (q > 1) std::cout << "/" << q;
      std::cout << std::endl;
      continue;
    }

    simplify(p1, q1);
    simplify(p2, q2);
    if (p1) {
      std::cout << p1;
      if (q1 > 1) std::cout << "/" << q1;
      std::cout << "+";
    }
    if (p2 > 1) std::cout << p2 << "*";
    std::cout << "sqrt(" << r << ")";
    if (q2 > 1) std::cout << "/" << q2;
    std::cout << std::endl;
  }
  return 0;
}
```

T4 旅游巴士

考虑拆点，把一个点 u 拆成 $(u, 0 \sim k-1)$，若原图存在一条 $u \to v$ 的边，那么新图上则有 $(u, x) \to (v, (x+1) \bmod k)$。

那么问题转化成求 $(1, 0) \to (n, 0)$ 的最短路。

考虑边的限制怎么解决，注意到我们可以任选开局的时机，因此如果在一个位置被卡住了，可以通过不断加 k 直到满足边的时间限制，走回头路一定不优，这个过程用一次 dij 实现即可实现。

时间复杂度 $\mathcal{O}(nk\log(mk))$。

此外，注意到到达时间越晚，关于边的限制也就越松，因此可以拆点后二分到达时间然后倒着 bfs。

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

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  std::cout.tie(nullptr);
  int n, m, k;
  std::cin >> n >> m >> k;
  std::vector<std::vector<std::pair<int, int>>> G(n);
  while (m--) {
    int u, v, a;
    std::cin >> u >> v >> a;
    --u, --v;
    G[u].emplace_back(v, a);
  }

  constexpr long long inf = 1e18;
  std::vector<long long> dis(n * k, inf);
  std::priority_queue<std::pair<long long, int>,
                      std::vector<std::pair<long long, int>>,
                      std::greater<std::pair<long long, int>>>
      q;
  q.emplace(dis[0] = 0, 0);
  while (q.size()) {
    auto [disu, idu] = q.top();
    q.pop();
    if (dis[idu] < disu) continue;
    int u = idu / k, ru = idu % k;
    assert(disu % k == ru);
    for (auto [v, a] : G[u]) {
      int rv = (ru + 1) % k;
      int idv = v * k + rv;
      int shifted_disu = std::max((a + k - 1ll - ru) / k * k + ru, disu);
      assert(shifted_disu >= a);
      assert(shifted_disu >= disu);
      assert(shifted_disu % k == ru);
      if (dis[idv] <= shifted_disu + 1) continue;
```

```
      q.emplace(dis[idv] = shifted_disu + 1, idv);
    }
  }

  if (dis[(n - 1) * k] < inf)
    std::cout << dis[(n - 1) * k] << std::endl;
  else
    std::cout << -1 << std::endl;
  return 0;
}
```