Speedrun

Calvin Fung

August 30, 2025

1 Problem

You are playing a video game. The game has n quests that need to be completed. However, the j-th quest can only be completed at the beginning of hour h_j of a game day. The game day is k hours long. The hours of each game day are numbered $0, 1, ..., k-1$. After the first day ends, a new one starts, and so on.

Also, there are dependencies between the quests, that is, for some pairs (a_i, b_i) the b_i-th quest can only be completed after the a_i-th quest. It is guaranteed that there are no circular dependencies, as otherwise the game would be unbeatable and nobody would play it.

You are skilled enough to complete any number of quests in a negligible amount of time (i. e. you can complete any number of quests at the beginning of the same hour, even if there are dependencies between them). You want to complete all quests as fast as possible. To do this, you can complete the quests in any valid order. The completion time is equal to the difference between the time of completing the last quest and the time of completing the first quest in this order.

Find the least amount of time you need to complete the game.

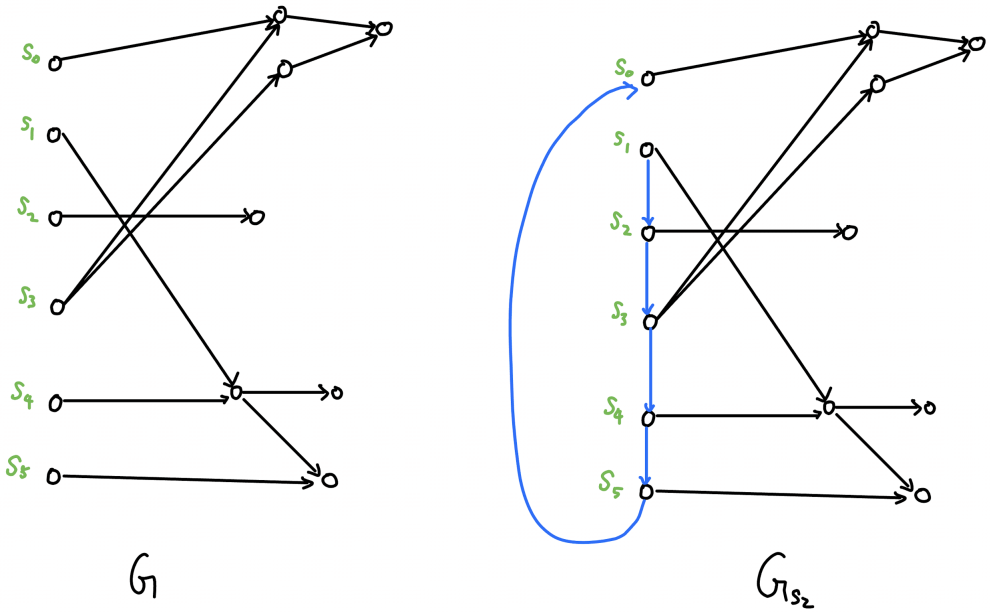
2 Solution

We present an $O(m + n \lg n)$ time solution, the main idea is the same as the naive $O(n(n + m))$ brute force approach except that we combine with a technical trick. The time is dominated by sorting, all other routines run in linear time.

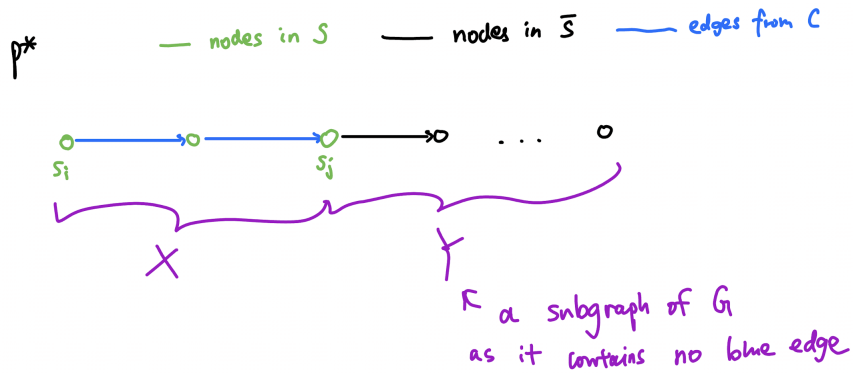
2.1 analysis

Consider the dependency graph G where the vertices are the quests and there's a directed edge (a, b) iff b can only be completed after quest a. After completion of a, b has to wait at least $w(a, b) = (h_b - h_a) \bmod k$ time before it can begin.

Let $S = s_0, s_1, ...$ be the sequence of quests with indegree 0 sorted in increasing h value. Consider a weighted circle $C = \{(s_i, s_{(i+1) \bmod |S|}) : i = 0, ..., |S|-1\}$ around S where $w(s_i, s_{(i+1) \bmod |S|}) = (h_{s_{(i+1) \bmod |S|}} - h_{s_i}) \bmod k$ is the time that $s_{(i+1) \bmod |S|}$ must wait after completion of s_i before it can begin (if it is not the first quest that we start with).



Clearly we must start at a quest in S. Once we know which quest $s_i \in S$ to start with, it's clear that greedily complete all quests as soon as possible will be optimal. Therefore, consider a new DAG G_{s_i} where we add edges $C \setminus (s_{(i-1) \bmod |S|}, s_i)$ to G. Observe that the minimum completion time is the longest path P^* starting at s_i in G_{s_i}.



P^* has a special structure. Consider the first node in P^* that's not in S and let s_j be the node right infront of it in P^*. We can then break of P^* into two paths, the first path X from s_i to s_j and the second path Y from s_j to the end. Notice in Y there cannot be another node in S other than s_j as there's no edge from $\bar{S}$ to S in G_{s_i}. It follows that no edge in Y is from C and Y must the a longest path starting at s_j in G. Letting $dp[s_j]$ be length of a longest path starting at s_j in G, then the length of P^* is simply $W_{i,j} + dp[s_j]$ where

$$W_{i,j} = \begin{cases} 0 & i = j \\ \sum_{q=i+1}^{j} w(s_{q-1 \bmod |S|}, s_q) & i < j \\ \sum_{q=i+1}^{|S|-1} w(s_{q-1 \bmod |S|}, s_q) + \sum_{q=0}^{j} w(s_{q-1 \bmod |S|}, s_q) & \text{otherwise} \end{cases}$$

is the weight of the path from s_i to s_j in C. Since we don't know what is s_j, we have to try all possiblities and the minimum completion time given we start at s_i is $\max_{j=0,\ldots,|S|-1} W_{i,j} + dp[s_j]$.

Overall, since we also don't know what is optimal start s_i, we have to try all possiblities and the minimum completion time is $\min_{i=0,\ldots,|S|-1} \max_{j=0,\ldots,|S|-1} W_{i,j} + dp[s_j]$.

The $dp[v]$ values can easily be computed in reverse topological order of G in time $O(n+m)$ ie. starting at nodes with outdegree 0.

At the first glance it seems like computing $\min_{i=0,...,|S|-1} \max_{j=0,...,|S|-1} W_{i,j} + dp[s_j]$ would take quadratic time. However, by using a trick we can compute it in linear time (given S which is a sorted sequence):

Initially let constant $w(C) = W_{0,|S|-1} + w(s_{|S|-1}, s_0)$ (weight of C) and variables $offset = 0, max = 0, minimax = \infty$.

Compute for $i = 0$ by brute force and let $A[j] = W_{0,j} + dp[s_j]$, $mx = max_j A[j]$. Notice $mx - offset$ is the minimum completion time given we start at s_0. Therefore we update $minimax = \min(minimax, mx - offset)$.

Now for $i = 1$, update $offset = offset + w(s_0, s_1)$ and $A[0] = A[0] + w(C)$. Notice now $A[j] - offset$ equals the desired $W_{1,j} + dp[s_j]$ for all j. As the new value of $A[0]$ is no smaller then the previous value, the maximum element in A is now $mx = \max(mx, A[0])$ and $mx - offset$ is the minimum completion time given we start at s_1. Therefore we update $minimax = \min(minimax, mx - offset)$.

The general inductive proof is that, suppose after the iteration for $i = q$, we have $A[j] - offset$ equals the desired $W_{q,j} + dp[s_j]$ for all j and mx is $\max_j A[j]$. Consider the next iteration where $i = q + 1 < |S|$, notice for $j \neq q$ we have $W_{q+1,j} + dp[s_j] = W_{q,j} - w(s_q, s_{q+1}) + dp[s_j] = A[j] - offset - w(s_q, s_{q+1})$ and $W_{q+1,q} + dp[s_q] = W_{q,q} + w(C) - w(s_q, s_{q+1}) + dp[s_q] = A[q] - offset + w(C) - w(s_q, s_{q+1})$. Therefore, after updating $A[q] = A[q] + w(C)$ and $offset = offset + w(s_q, s_{q+1})$ we have $A[j] - offset$ equals the desired $W_{q+1,j} + dp[s_j]$ for all j. Since the only update is at $A[q]$ and its new value is no smaller (as $w(C) \geq 0$), the maximum element in A is now $mx = max(mx, A[q])$ and $mx - offset$ is the minimum completion time given we start at s_q. Therefore we update $minimax = \min(minimax, mx - offset)$.

After running through $i = 0, ..., |S|$ the final value of $minimax$ will be the least amount of time we need to complete the game.

2.2 algorithm

Below is an implementation, the direction of edges and notations maybe different from above but the idea is the same.

```
#include<iostream>
#include <forward_list>
#include <queue>
#include <limits.h>
#include <algorithm>
#define ll long long
#define N 200005
using namespace std;

int T,n,m,k,h[N],odeg[N],ideg[N],s[N];
ll dp[N];
forward_list<int> G[N];

bool cmp(int i,int j) {return h[i]<h[j];}

void slv(){

    scanf("%d %d %d",&n,&m,&k);
    for(int i=1;i<=n;++i) scanf("%d",&h[i]), G[i].clear(), odeg[i] = 0, ideg[i] = 0, dp[i] = 0;
    for(int i=1,u,v;i<=m;++i)   scanf("%d %d",&u,&v), G[v].push_front(u), odeg[v]++, ideg[u]++;
    queue<int> Q;
    int possible_starts = 0;
    for(int i=1;i<=n;++i){
        if(ideg[i] == 0)    Q.push(i);
        if(odeg[i] == 0)    s[possible_starts++] = i;
    }
    while(!Q.empty()){
        int b = Q.front();
```

```
        Q.pop();
        for(auto a : G[b]){
            dp[a] = max(dp[a],(h[b] - h[a] + k)%k + dp[b]); //mod a negative num
            ideg[a]--;
            if(ideg[a] == 0)    Q.push(a);
        }
    }

    sort(s,s+possible_starts,cmp); //increasing time

    queue<ll> q;

    ll mx = dp[s[0]], acc = 0;
    q.push(dp[s[0]]);
    for(int i=1;i<possible_starts;++i){
        acc += (h[s[i]] - h[s[i-1]] + k)%k;
        ll time = dp[s[i]] + acc;
        if(time > mx)   mx = time;
        q.push(dp[s[i]] + acc);
    }

    acc += (h[s[0]] - h[s[possible_starts-1]] + k)%k;
    ll offset = 0, minimax = mx;
    for(int i=0;i<possible_starts-1;++i){
        offset += (h[s[i+1]] - h[s[i]] + k)%k;
        mx = max(mx, q.front() + acc);
        minimax = min(minimax, mx - offset);
        q.pop();
    }

    printf("%lld\n",minimax);

}

int main(){

    scanf("%d",&T);
    while(T--) slv();return 0;

}
```