"星航计划" -- 十二月份 -- CSP-S全真模拟 -- 解析思路

T1 -- 平均(avg.cpp)

解法：

20pts

2^n 枚举所有可能的子集进行枚举，求解出其中的平均数计算答案，时间复杂度为 $O(2^n \times n)$。

40pts

将所有数排完序后，最终选择的一定是连续的一段区间，证明在之后给出。

选择一段区间后进行统计，时间复杂度为 $O(n^3)$。

枚举左端点 l，枚举右端点假设是 r，计算这个区间的平均数 $average$，统计区间内大于平均数的个数 $O(n^3)$。

60pts

选择一个左端点后，发现随着右端点往后移，平均数其实在不断增大，每个数在刚加进来时一定大于等于平均数，随着时间的推移，会比平均数小，且一旦比平均数小后就一直比平均数小，这个时间可以通过双指针快速计算，时间复杂度为 $O(n^2)$

100pts

假设最终选择的集合的平均数不超过 k。 为使平均数不超过 k，应将 $\leq k$ 的数全部选入，然后贪心选择 $> k$ 的部分中最小的若干个数，这也间接说明了肯定选择一个区间，且肯定是一个前缀，因此只需要对 $l = 1$ 的所有区间处理即可，套用 60 pts 的双指针做法，由于排序的原因，时间复杂度为 $O(n \log n)$ 。

假设最终的平均数不超过 k，意味着我们肯定想把所有 $\leq k$ 数都选进去，然后 $> k$ 的数肯定会贪心的选若干个最小的。

枚举一个数 k，$\leq k$ 都被选择了，$> k$ 的选了最小的若干个

选的数一定是一段前缀

$a_1, a_2, a_3, \ldots\ldots, a_i < k < a_{i+1}, a_{i+2}, a_{i+3}, a_j, \ldots\ldots\ldots, a_n$

把所有数从小到大排序，选的一定是连续的一段前缀。

枚举一个前缀，假设选择的是 $[1,i]$ 中的所有数，通过前缀和计算出它的平均数是 $average$，想知道有多少个 $> average$ 的数。

Code

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

using namespace std;

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);//关闭同步流
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  sort(a.begin(), a.end());
  long long sum=0,ans=0;
  for(int i=0;i<n;i++){//枚举 [1,i+1] 这一前缀
    sum+=a[i]; //表示的是前[1,i+1]个数之和
    //平均数:sum/(i+1)
    //计算前i+1个数中有多少个是>sum/(i+1)
    int l=0,r=i,pos=i+1;//计算第一个>sum/(i+1)的数的位置
    while(l<=r){
        int mid=(l+r)/2;
//      if(a[mid]>1.0*sum/(i+1))
//      平均数等于所有数的和 sum/所有数的个数 (i+1)
//      避免浮点数
        if(1ll*a[mid]*(i+1)>sum) pos=mid,r=mid-1;
        else l=mid+1;
    }
    //第一个大于平均数的位置是pos,最后一个大于平均数的位置是i
    //i-pos+1
    //大于平均数的一定是一段连续的区间，左端点是pos,右端点是i
    //[pos,i]   i-pos+1
    ans=max(ans,i+1-pos);
    //1ll  (long long)(1)   long long * int -> long long
  }
  cout << ans << "\n";
}

//枚举一个前缀，假设选择的是 $[1,i]$ 中的所有数，通过前缀和计算出它的平均数是 $average$，想知道有多少个 $>average$ 的数。
```

T2 -- 操作(opn.cpp)

解法:

首先求出 a 序列的差分数组 b。

考虑操作 $[l,r,k]$ 等价于 $b_l := b_l + k, b_{r+1} := b_{r+1} - k$，使得 b 序列全部清零。

显然我们可以花 1 次操作的代价使得一个位置变为 0，但只有这个性质显然仍无法处理此题。

对于若干位置 $i_1, i_2, i_3, \ldots i_q$，它们的 b 之和为 0，注意到一次操作 $[l,r,k]$ 其实并不会修改 $b_l + b_{r+1}$ 的值，所以对于 q 个位置，它们之和为 0，只需要 $q-1$ 次操作可以将其中 $q-1$ 个改为 0，剩下一个自然就变成了 0。

因此问题等价于选择尽量多的没有交集的集合，每个集合的权值和均为 0。

考虑 $dp[S]$ 表示 S 集合内的数都已经被选择了，最多可以选出多少个集合。

我们可以枚举下一个集合的具体形态，这样的复杂度为 $O(3^n)$，可以拿到大部分分数，但仍无法通过此题。

我们可以考虑一个元素一个元素加入集合，即每次加入 S 集合外的一个元素x，转移到 $S|2^{x-1}$，每当 S 集合内的元素和为 0，则让 $dp[S] := dp[S] + 1$，时间复杂度为 $O(n \times 2^n)$。

Code

```
#include<bits/stdc++.h>
using namespace std;
int dp[1<<24];
int n,a[25];
int main(){
    cin>>n;for(int i=1;i<=n;i++) cin>>a[i];n++;
    for(int i=n;i>=1;i--) a[i]-=a[i-1];
    for(int i=1;i<1<<n;i++){
        long long sum=0;
        for(int j=1;j<=n;j++) if(i>>j-1&1) sum+=a[j];
        dp[i]=-n;
        for(int j=0;j<n;j++) if(i>>j&1) dp[i]=max(dp[i],dp[i^(1<<j)]);
        if(!sum) dp[i]++;
    }
    cout<<n-dp[(1<<n)-1]<<'\n';
}
```

T3 -- 选择排序(sort.cpp)

解法:

首先给出一个性质：如果一个位置上的数字在第 i 轮被交换过，且它不是i，那么它在第 $i+1$ 轮一定也会被交换。

证明：首先模拟一下交换的过程，我们可以知道在一轮交换中一个位置参与交换的充要条件是，在 此轮交换开始的时候，这个位置上的数是前缀最小值。在第i轮交换中一共有 $c[i]+1$个位置参与了交换，设这些位置从小到大为 $p_0,p_1,...,p_{c[i]}$,在这一轮交换前，这些位置上的数字为 $a_0,a_1,...,a_{c[i]}$ 。模拟交换的过程可以发现，$a_{c[i]}$ （也是全局最小值 i ）被交换到了最前面，从此不再参与任何交换。其他 p_i 上的数字都被移动到了 p_{i+1} 。因为在移动之前，a_{i+1} 是 a_i 之后的下一个前缀最小值，这说明在 (p_i,p_{i+1}) 之间没有数字比 a_i 更小，因此 a_i 移动到 p_{i+1} 之后，它仍然是一个前缀最小值。有了上述性质之后，我们考虑从第 n 轮开始往回倒推初始的状态。假设我们已经知道了第 $i+1$ 轮是哪些位置参与了交换，第 i 轮的交换位置就是此次新增的首位置 i，以及 $c[i]$ 个在第 $i+1$ 轮参与了交换的位 置。我们可以从 $c[i+1]+1$ 个位置中任意选出 $c[i]$ 个位置，根据交换过程，可以发现它们有唯一的合法方案：第 1 个位置上的数字移动到新增的首位置 i ，后面每个位置上的数都移动到前一个位置，新增的数字 i 放置到最后一个位置。因此假设我们已知了到第 $i+1$ 轮时合法方案数为 $dp[i+1]$ ，那么第 i 轮的方案数就是 $dp[i+1]*C(c[i+1]+1,c[i])$ 。时间复杂度 $O(n)$ 。

Code

```cpp
#include<bits/stdc++.h>
using namespace std;
const int o=2e5+10,MOD=998244353;
int n,a[o],fac[o],inv[o],ans=1;
inline int C(int x,int y){if(x<y||y<0) return 0;return fac[x]*1ll*inv[y]%MOD*inv[x-y]%MOD;}
int main(){
	scanf("%d",&n);inv[1]=1;
	for(int i=2;i<=n;++i) inv[i]=MOD-MOD/i*1ll*inv[MOD%i]%MOD;
	for(int i=fac[0]=inv[0]=1;i<=n;++i) fac[i]=fac[i-1]*1ll*i%MOD,inv[i]=inv[i-1]*1ll*inv[i]%MOD;
	for(int i=1;i<n;++i) scanf("%d",&a[i]);
	for(int i=n;--i;){
//		if(a[i+1]+1<a[i]) cout<<i<<"wwww"<<endl;
		ans=ans*1ll*C(a[i+1]+1,a[i])%MOD;
	}
	printf("%d",ans);
	return 0;
}
```

T4 --树 (tree.cpp)

解法：

对于一条实链，显然链底是最后操作的，其他的点之间顺序无所谓。 如果把实链缩成点，用链底的操作代表整个点的操作，则每个点必须比 其父亲点早操作。 上面两条性质是充分必要的，因此可以建出如下的树： 实链底向此实链的父亲实链底连边。 非实链底向实链底连边。 则每个点的出现时间要比他父亲的出现时间早。

问题转化为给定一棵树，要求在点上写一个排列，使得每个点比其父亲 的点小。

该答案为 $\frac{n!}{\prod_{i=1}^{n} size_i}$。

令 S 为实链顶的点集集合，可以得知原问题的答案为 $\frac{n!}{\prod_{i=1}^{n} size_i}$，如果暴力维护，可以做到$O(nq)$，可以拿到 30pts。

链的维护可以直接利用线段树等数据结构快速定位到链顶，模拟整个过程，时间复杂度为$O(q\log n)$，可以拿到 30pts。

对于 100pts，该答案的求解可以利用树链剖分或 LCT 直接处理，是树链剖分的经典模型，暴力维护整棵树的轻重链形态，如果利用树链剖分，假设 n, q 同阶，时间复杂度为 $O(n\log n^2)$，如果利用 LCT，时间复杂度为$O(n\log n)$。

Code

```
#include <bits/stdc++.h>
using namespace std;
const int N=2e6+7,p=998244353;
int n,q,k,op,g[N],f[N],df[N],S[N],fa[N],ft[N],v1[N],v0[N],f0[N],f1[N],sz[N],val[N],tg[N],sn[N],pos[N],L[N],R[N]; vector<int>v[N];
long long pows(long long u,int v){
    long long ans=1;
    while(v>0){
        if(v&1) ans=ans*u%p; u=u*u%p,v=v>>1;
    }
    return ans;
}
void dfs1(int u){
  sz[u]=1;
  for(int x:v[u]){
    dfs1(x),sz[u]+=sz[x];
    if(sz[g[u]]<sz[x]) g[u]=x;
  }

}
void dfs2(int u){
  if(!ft[u]) ft[u]=u;
  f[u]=++k,df[k]=u;
  // cout<<u<<","<<ft[u]<<endl;
  if(g[u]) ft[g[u]]=ft[u],dfs2(g[u]);
```

```
  for(int x:v[u]){
    if(x==g[u]) continue; dfs2(x);
  }
  v1[f[u]]=sz[u];
  v0[f[u]]=1;
  // cout<<f[u]<<':'<<v0[f[u]]<<","<<v1[f[u]]<<","<<sz[fa[u]]<<endl;
}
void pushdown(int t){
  if(tg[t]==-1) return;
  if(tg[t]==0){
    tg[t<<1]=tg[t<<1|1]=0;
    val[t<<1]=f0[t<<1],val[t<<1|1]=f0[t<<1|1],val[t]=f0[t];
  }
  else{
    tg[t<<1]=tg[t<<1|1]=1;
    val[t<<1]=f1[t<<1],val[t<<1|1]=f1[t<<1|1],val[t]=f1[t];
  }
  tg[t]=-1;
}
void build(int l,int r,int t){
  L[t]=l,R[t]=r;
  if(l==r){
    f0[t]=v0[l],f1[t]=v1[l],val[t]=f0[t],tg[t]=-1,pos[l]=t;
    return;
  }
  int d=(l+r)/2;
  build(l,d,t<<1),build(d+1,r,t<<1|1);
  f0[t]=1ll*f0[t<<1]*f0[t<<1|1]%p;
  f1[t]=1ll*f1[t<<1]*f1[t<<1|1]%p;
  val[t]=f0[t];
}
void cover0(int l,int r,int t,int ql,int qr){

  pushdown(t);
  if(l==ql&&r==qr){
    val[t]=f0[t],tg[t]=0; return;
  }
  int d=(l+r)/2;
  if(ql<=d) cover0(l,d,t<<1,ql,min(d,qr));
  if(d+1<=qr) cover0(d+1,r,t<<1|1,max(d+1,ql),qr);
  val[t]=1ll*val[t<<1]*val[t<<1|1]%p;
  // cout<<l<<","<<r<<","<<ql<<","<<qr<<":"<<","<<val[t]<<","<<t<<","<<val[3]<<endl;
}
void cover1(int l,int r,int t,int ql,int qr){
    // cout<<l<<","<<r<<","<<ql<<","<<qr<<endl;
  pushdown(t);
  if(l==ql&&r==qr){
    val[t]=f1[t],tg[t]=1; return;
  }
  int d=(l+r)/2;
  if(ql<=d) cover1(l,d,t<<1,ql,min(d,qr));
  if(d+1<=qr) cover1(d+1,r,t<<1|1,max(d+1,ql),qr);
  val[t]=1ll*val[t<<1]*val[t<<1|1]%p;

```

```
}

int getpre(int u){
  if(S[pos[u]]) return u;
  u=pos[u];
  while(u>0){
    if(u&1){
      u=u>>1;
      if(S[u<<1]){
        u=u<<1;
        while(1){
          if(L[u]==R[u]) return L[u];
          if(S[u<<1|1]) u=u<<1|1;
          else u=u<<1;
        }
      }
    }
    else u=u>>1;
  }
  return 0;
}
void upd(int u){
  cover0(1,n,1,f[sn[u]],f[sn[u]]);
  int x=pos[f[u]];
  while(x>0) S[x]--,x=x>>1;
  sn[u]=0;
}
int main(){
  std::ios::sync_with_stdio(0),cin.tie(0);
  cin>>n>>q,op=1;
  for(int i=2;i<=n;i++)
    cin>>fa[i],v[fa[i]].push_back(i),op=1ll*op*i%p;
  dfs1(1),dfs2(1),v0[1]=v1[1]=1,build(1,n,1);
  for(int i=1;i<=n;i++) op=1ll*op*pows(sz[i],p-2)%p;
  // cout<<1ll*val[1]*op%p<<"www"<<op<<endl;
  while(q--){
    int u; cin>>u; int c=u;
    while(u>0){
      int a=ft[u],b=u,x=f[b];
      // cout<<val[1]<<"www"<<endl;

      if(g[u]>0) cover0(1,n,1,f[g[u]],f[g[u]]);

      while(1){
        x=getpre(x);
        if(x<f[a]) break;
        upd(df[x]),x--;
      }
      u=fa[ft[u]];
    }
    u=c;
    while(u>0){
      int a=ft[u],b=u;
      cover1(1,n,1,f[a],f[b]);
      if(ft[u]!=1){
```

```
            u=ft[u];
            sn[fa[u]]=u; int x=pos[f[fa[u]]];
            // cout<<S[x]<<","<<f[2]<<","<<x<<endl;
            while(x>0) S[x]++,x=x>>1;
            // cout<<S[9]<<"wwwasdasdasdasdasdas"<<endl;
          }
          u=fa[ft[u]];
          // cout<<tg[1]<<"asdasdasdas"<<val[3]<<","<<tg[3]<<endl;
        }
        cout<<1ll*op*val[1]%p<<'\n';
      }
      return 0;
    }
    // 7 2
    // 1 2 3 3 3 5
    // 5
    // 1
```

很荣幸，能为您提供便捷的解题思路

赛后补题很关键，及时优化代码，为自己知识宝库多积累一份珍贵资源

梦熊信竞平台，预祝您可以在2025年赛事中取得最佳战绩！