这里首先要推个结论，最佳的策略必然没有哪个砖头的重量增加了3或者更多。

某一块砖头的初始重量为a，最佳策略中的最终重量为b，如果b-a>=3的话，

在这块砖头左右两侧的砖头重量都不能等于b，即使它们的重量都比b要小且不相等，也只能选择a,a+1,a+2这三个中的其中两个，那么剩下的那一个如果作为该砖头的重量仍然是满足条件的。而a,a+1,a+2这三种重量都是小与b的，也就是说这种策略是优于我们目前重量b的“最优策略的”，也就反证了选择重量b不是最优的策略。同理也证明了b-a>3的情况不是最优的。

推出这个结论后就可以使用dp来解决问题了，用dp[i][j]数组表示前i块砖头满足条件且第i块砖头增加j重量（这里j的范围为0-2）的最少花费。

转移方程为dp[i][k]=min（dp[i-1][j]+b[i]\*k，dp[i][k]），需要满足条件a[i-1]+j!=a[i]+k（相邻两块砖头的重量不可相等。

注意到这里dp数组每次转移只用到了相邻的两列，因此我们还可以把这个dp数组压缩成[2][3]

使用dp[n][3]的dp代码

#include<bits/stdc++.h>

#define ll long long

#define IOS ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

**using** **namespace** std;

**const** ll maxn=3e5+7;

#define llINF 9223372036854775807

ll dp[maxn][3];

ll a[maxn],b[maxn];

**int** main()

{

    IOS;

    ll t;

    cin>>t;

**while**(t--)

    {

        ll n;

        cin>>n;

**for**(ll i=1;i<=n;i++)

        {

            cin>>a[i]>>b[i];

        }

        dp[1][0]=0;

        dp[1][1]=b[1];

        dp[1][2]=b[1]\*2;

**for**(ll i=2;i<=n;i++)

        {

            dp[i][0]=dp[i][1]=dp[i][2]=llINF;

**for**(ll j=0;j<=2;j++)

            {

**for**(ll k=0;k<=2;k++)

                {

**if**(a[i-1]+j!=a[i]+k)

                    {

**if**(dp[i-1][j]+k\*b[i]<dp[i][k])

                        {

                            dp[i][k]=dp[i-1][j]+k\*b[i];

                        }

                    }

                }

            }

        }

        ll ans=llINF;

**for**(ll i=0;i<=2;i++)

            ans=min(ans,dp[n][i]);

        cout<<ans<<endl;

    }

}

使用dp[2][3]的dp代码

#include<bits/stdc++.h>

#define ll long long

#define IOS ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

**using** **namespace** std;

**const** ll maxn=3e5+7;

#define llINF 9223372036854775807

ll dp[2][3];

ll a[maxn],b[maxn];

**int** main()

{

    IOS;

    ll t;

    cin>>t;

**while**(t--)

    {

        ll n;

        cin>>n;

**for**(ll i=1;i<=n;i++)

        {

            cin>>a[i]>>b[i];

        }

        dp[1][0]=0;

        dp[1][1]=b[1];

        dp[1][2]=b[1]\*2;

**bool** flag=0;

**for**(ll i=2;i<=n;i++)

        {

            dp[flag][0]=dp[flag][1]=dp[flag][2]=llINF;

**for**(ll j=0;j<=2;j++)

            {

**for**(ll k=0;k<=2;k++)

                {

**if**(a[i-1]+j!=a[i]+k)

                    {

**if**(dp[!flag][j]+k\*b[i]<dp[flag][k])

                        {

                            dp[flag][k]=dp[!flag][j]+k\*b[i];

                        }

                    }

                }

            }

            flag=!flag;

        }

        ll ans=llINF;

**for**(ll i=0;i<=2;i++)

            ans=min(ans,dp[n&1][i]);

        cout<<ans<<endl;

    }

}