DP

树上背包

> 有一棵点数为 n 的树，树边有边权。给你一个在 $0 \sim n$ 之内的正整数 k ，你要在这棵树中选择 k 个点，将其染成黑色，并将其他的 $n-k$ 个点染成白色。将所有点染色后，你会获得黑点两两之间的距离加上白点两两之间的距离的和的收益。问收益最大值是多少。

解：

每条边对答案的贡献为 边权 * (左边黑点 * 右边黑点 + 左边白点 * 右边白点)

设 $dp[i][j]$ 表示 i 及子树中选 j 个黑点对答案的贡献，树上背包转移即可

```
int n,m;
int siz[N],dp[N][N];
void dfs(int x,int f){
    siz[x]=1;
    dp[x][0]=dp[x][1]=0;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to,val=edge[i].val;
        if(y==f)continue;
        dfs(y,x);
        for(int j=min(siz[x],m);j>=0;j--){
            for(int k=0;k<=min(j,siz[y]);k++){
                int sum=k*(m-k)+(siz[y]-k)*(n-m-siz[y]+k);
                // 这里是 j+k
                dp[x][j+k]=max(dp[x][j+k],dp[x][j]+dp[y][k]+val*sum);
            }
        }
        siz[x]+=siz[y];     //边算siz边转移，保证 O(n^2) 复杂度
    }
}
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++)dp[i][j]=INT_MIN;
    for(int i=1;i<n;i++){
        int u,v,x;
        cin>>u>>v>>x;
        add(u,v,x);add(v,u,x);
    }
    dfs(1,1);
    cout<<dp[1][m]<<endl;
}
```

基环树上dp

给一个基环森林，每个点有个权值，相连两点不能同时选，求最大权值

解：

树形 dp，$dp[i][0/1]$ 表示这一点选/不选能获得的最大价值

在每棵基环树上，选环上任意一条边断开，以两个端点分别做一次树形 dp，取 $max(dp[u][0],dp[v][0])$ 作为答案即可

```
int a[N],dp[N][2],vis[N];
int pos1,pos2;
void dfs(int x,int f){
    dp[x][0]=0,dp[x][1]=a[x];
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f||edge[i].val==0)continue;
        dfs(y,x);
        dp[x][0]+=max(dp[y][0],dp[y][1]);
        dp[x][1]+=dp[y][0];
    }
}
void dfs2(int x,int f,int id){
    if(vis[x]){
        pos1=f;pos2=x;
        edge[id].val=0;
        edge[id^1].val=0;
        return;
    }
    vis[x]=1;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f)continue;
        dfs2(y,x,i);
    }
}
void solve(){
    cntt=1;     //便于删边
    int n,ans=0;
    cin>>n;
    for(int i=1;i<=n;i++){
        int x;
        cin>>a[i]>>x;
        add(x,i,1);
        add(i,x,1);
    }
    map<int,int>mp;
    for(int i=1;i<=n;i++){
        if(vis[i]==0){
            dfs2(i,i,0);
            mp[pos1]=pos2;
        }
    }
    for(auto i:mp){
        int x=i.first,y=i.second;
        dfs(x,x);
        long long tmp=dp[x][0];
        dfs(y,y);
        ans+=max(tmp,dp[y][0]);
    }
    cout<<ans<<endl;
}
```

数位dp

题目描述

不含前导零且相邻两个数字之差至少为 2 的正整数被称为 windy 数。windy 想知道，在 a 和 b 之间，包括 a 和 b，总共有多少个 windy 数?

解:

利用前缀和，处理 $[l,r]$ 数字个数时，设 $f(x)$ 为 $1-x$ 的合法数字个数，直接 $f(r)-f(l-1)$ 即可

对于每个 x 的处理，我们预处理出一个数组 $dp[i][j]$ 表示 长度为 i 的以 j 开头的合法数字的个数

然后枚举 x 的每一位，设当前位为 i，则当前位为 $0-(i-1)$ 的数字个数可直接得到，为 i 的数字个数接着枚举即可

注意第 1 位的枚举要从 1 开始，因为预处理时把 $01xx$ 标记为非法情况，我们计算的只有不带前导0的个数

所以最后要重新加上带有前导0的个数

```
int dp[N][N];   // 第 i 位，上一位是 j
int pos[N],len=0;
int dfs(int now,int lead,int limit,int pre){
    if(now==0)return 1;
    if(!limit && !lead && dp[now][pre]!=-1)return dp[now][pre];
    int ans=0;
    int mx=limit?pos[now]:9;
    for(int i=0;i<=mx;i++){
        if(abs(i-pre)<2)continue;
        if(lead && i==0)ans+=dfs(now-1,1,limit && i==mx,-2);
        else ans+=dfs(now-1,0,limit && i==mx,i);
    }
    if(!lead && !limit)dp[now][pre]=ans;
    return ans;
}
int cal(int x){
    len=0;
    while(x){
        pos[++len]=x%10;
        x/=10;
    }
    return dfs(len,1,1,-2);
}
void solve(){
    for(int i=0;i<=100;i++){
        for(int j=0;j<=9;j++)dp[i][j]=-1;
    }
    int l,r;
    cin>>l>>r;
    cout<<cal(r)-cal(l-1)<<endl;
}
```

斜率优化

先搞成 $y=kx+b$ 的形式，用 $dp[j]$ 更新 $dp[i]$ 时

$y=dp[j]+f[j]$，$k=a[i]$，$x=b[j]$，$b=dp[i]+g[i]$

把 $dp[i]$ 放到截距上，若取 min 维护下凸壳，取 max 维护上凸壳

若 k 具有单调性，用单调队列维护，否则数组维护凸壳，二分查找决策点

若 x 不具有单调性，用 cdq 分治维护

```
// 用来转移的每个状态相当于坐标平面的一个点
// k, b 与被转移的点相关, k 是定值, b 是关于 dp[i] 的方程
long double eps=1e-8;
inline long double X(int x){return c[x];}          //若卡精度，用long double
inline long double Y(int x){return dp[x]-S*c[x];}
bool check(int x,int y,int z){
    long double x1=X(y)-X(x),y1=Y(y)-Y(x);
    long double x2=X(z)-X(x),y2=Y(z)-Y(x);
    return x1*y2-x2*y1<eps;
}
long double slope(int x,int y){return 1.0*(Y(y)-Y(x))/(X(y)-X(x));}

void solve(){
    cin>>n>>L;
    for(int i=1;i<=n;i++){
        cin>>l[i];
        pre[i]=pre[i-1]+l[i];
        a[i]=pre[i]+i;
        b[i]=pre[i]+i+L+1;
    }
    b[0]=L+1;        //一定记得赋值0的各自值
    int hh=0,tt=-1;
    q[++tt]=0;
    for(int i=1;i<=n;i++){      //找第一个大于2a的斜率
        while(hh<tt&&slope(q[hh+1],q[hh])<=2*a[i])hh++;
        dp[i]=dp[q[hh]]+(a[i]-b[q[hh]])*(a[i]-b[q[hh]]);
        while(hh<tt&&check(q[tt-1],q[tt],i))tt--;
        q[++tt]=i;
    }
    cout<<dp[n]<<endl;
}
```

四边形不等式优化

$dp[i]=max(a[j]+w(i,j))$

$w(i,j)$ 满足 $w(i,j+1)+w(i+1,j)\geq w(i,j)+w(i+1,j+1)$，即 包含大于交叉 时，考虑使用决策单调性优化

维护一个单调队列，储存每个决策的支配区域，插入时二分反超点

```
inline int w(int j,int i){return a[j]+sqrt(i-j);}
struct Q{
    int l,r,p;
```

```
}q[N];
void solve(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    int hh=0,tt=0;
    q[0]={1,n,1};        //队列保存[l,r]的最优决策是p
    for(int i=2;i<=n;i++){
        if(q[hh].r<i)hh++;     //左边决策完了
        else q[hh].l=i;          //更新
        dp[i]=w(q[hh].p,i);     // 用最优决策更新当前点
        while(hh<=tt&&w(i,q[tt].l)>=w(q[tt].p,q[tt].l))tt--;    //当前决策比队尾优
        int pos=n+1;
        if(hh>tt)pos=i;     //集合为空
        else{
            int l=q[tt].l,r=q[tt].r;    //二分找到第一个反超点（决策优于队尾决策）
            while(l<r){
                int mid=(l+r)>>1;
                if(w(i,mid)>=w(q[tt].p,mid)){       // 反超
                    r=mid,pos=mid;
                }
                else l=mid+1;
            }
        }
        if(pos==(n+1))continue;     // 当前决策不优
        q[tt].r=pos-1;      // 更新队尾决策
        q[++tt]={pos,n,i};      // 入队
    }
}
```

倍增优化

题意翻译

给定一个 $1 \times n$ 的地图，在里面玩 2048，每次可以合并相邻两个（数值范围 $1 \sim 40$），问序列中出现的最大数字的值最大是多少。注意合并后的数值并非加倍而是 $+1$，例如 2 与 2 合并后的数值为 3。

解：

暴力dp：考虑区间dp，设 $dp[l][r][k]$ 表示能否把 $l-r$ 内的元素合并成 k

转移方程为 $\quad dp[l][r][k] \ \mid= (dp[l][i][k-1] \& dp[i+1][r][k-1])$ ，$i \in [l, r-1]$

枚举 l, r, i, k ，复杂度为 $O(n^4)$

倍增优化：设 $dp[i][j]$ 表示从第 i 个点开始合并出 j 的右端点的右边一个点

初始状态为 $\quad dp[i][a[i]] = i+1$

转移方程为 $\quad dp[i][j] = dp[dp[i][j-1]][j-1]$

枚举 i, j ，复杂度为 $O(n^2)$

```
int dp[N][N],a[N];
void solve(){
    int n;
    cin>>n;
```

```
    int ans=0;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        ans=max(a[i],ans);
        dp[i][a[i]]=i+1;
    }
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            if(!dp[i][k])dp[i][k]=dp[dp[i][k-1]][k-1];
            if(dp[i][k])ans=max(ans,k);
        }
    }
    cout<<ans<<endl;
    return;
}
```

闵可夫斯基和优化

维护差分数组，每次归并排序/区间加，一般维护平衡树，然后启发式合并

```
vector<int>G[N];
int n,k;
priority_queue<int>p1[N];   // 大根堆
priority_queue<int,vector<int>,greater<>>p2[N];     // 小根堆
int add1[N],add2[N];
void adjust(int x){     // 对前 k/2 个区间-2
    while(p1[x].size()>k/2)
        p2[x].push(p1[x].top()+add1[x]-add2[x]),p1[x].pop();
}
void merge(int x,int y){
    // 把 x 启发式合并到 y 上
    if(p1[x].size()+p2[x].size()>p1[y].size()+p2[y].size())
        swap(p1[x],p1[y]), swap(p2[x],p2[y]), swap(add1[x],add1[y]),
swap(add2[x],add2[y]);
    while(!p1[x].empty())p1[y].push(p1[x].top()+add1[x]-add1[y]),p1[x].pop();
    while(!p2[x].empty())p2[y].push(p2[x].top()+add2[x]-add2[y]),p2[x].pop();
}
void dfs(int x,int f){
    p1[x].push(0-add1[x]);
    for(auto y:G[x]){
        if(y==f)continue;
        dfs(y,x);
        merge(y,x);
        adjust(x);
    }
    if(x!=f){
        add1[x]-=2;
        if(k%2==0)add2[x]+=2;
        else if(!p2[x].empty()){
            int v=p2[x].top()+add2[x];
            p2[x].pop();
            add2[x]+=2;
            p2[x].push(v-add2[x]);
        }
    }
```

```
}
void solve(){
    cin>>n>>k;
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1,1);
    int ans=(n-1)*k;
    for(int i=1;i<=k;i++){
        if(!p1[1].empty())ans+=p1[1].top()+add1[1],p1[1].pop();
        else ans+=p2[1].top()+add2[1],p2[1].pop();
    }
    cout<<ans<<endl;
}
```