5.1.3 DP的设计和实现

|  |
| --- |
| 例5-1. Bone Collector hdu 2602 |

（1）递推代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include<bits/stdc++.h>  using namespace std;  const int N = 1011;  int w[N], c[N]; // 物品的价值和体积  int dp[N][N];  int solve(int n, int C){  for(int i=1; i<=n; i++)  for(int j=0; j<=C; j++){  if(c[i] > j) dp[i][j] = dp[i-1][j]; //第i个物品比背包还大，装不了  else dp[i][j] = max(dp[i-1][j],dp[i-1][j-c[i]]+w[i]); //第i个物品能装  }  return dp[n][C];  }  int main(){  int T; cin>>T;  while(T--){  int n,C; cin >> n >> C;  for(int i=1;i<=n;i++) cin>>w[i];  for(int i=1;i<=n;i++) cin>>c[i];  memset(dp,0,sizeof(dp)); //清0，置初值为0  cout << solve(n, C) << endl;  }  return 0;  } |

（2）记忆化代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | int solve(int i, int j){ //前i个物品，放进容量j的背包  if (dp[i][j] != 0) return dp[i][j]; //记忆化  if(i == 0) return 0;  int res;  if(c[i] > j) res = solve(i-1,j); //第i个物品比背包还大，装不了  else res = max(solve(i-1,j), solve(i-1,j-c[i])+w[i]); //第i个物品可以装  return dp[i][j] = res;  } |

5.1.4 滚动数组

1. 交替滚动

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | // hdu 2602（滚动数组代码1）  int dp[2][N]; //替换 int dp[][];  int solve(int n, int C){  int now = 0, old = 1; //now指向当前正在计算的一行，old指向旧的一行  for(int i=1; i<=n; i++){  swap(old,now); //交替滚动。now始终指向最新的一行  for(int j = 0; j <= C; j++){  if(c[i] > j) dp[now][j] = dp[old][j];  else dp[now][j] = max(dp[old][j], dp[old][j-c[i]]+w[i]);  }  }  return dp[now][C]; //返回最新的行  } |

2. 自我滚动

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | // hdu 2602（滚动数组代码2）  int dp[N];  int solve(int n, int C){  for(int i=1; i<=n; i++)  for(int j=C; j>=c[i]; j--) //反过来循环  dp[j] = max(dp[j],dp[j-c[i]]+w[i]);  return dp[C];  } |

5.2 经典线性DP问题

|  |
| --- |
| 例5-2. **ACboy needs your help** hdu 1712 |

。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include<bits/stdc++.h>  using namespace std;  const int N=105;  int w[N][N],c[N][N]; //物品的价值、体积  int dp[N];  int n,m;  int main(){  while(scanf("%d%d",&n,&m) && n && m){ //输入：n门课=n组，m天=容量m  for(int i=1;i<=n;i++) //第i门课，即第i组  for(int k=1;k<=m;k++){ //m天=容量m；在这题中，m也是第i组的物品个数  scanf("%d",&w[i][k]); //第i组第k个物品的价值  c[i][k] = k; //第i组第k个物品的体积。学k天才能得分，体积就是k  }  memset(dp,0,sizeof(dp));  for(int i=1;i<=n;i++) //n门课=n组。遍历每门课，即遍历每个组  for(int j=m;j>=0;j--) //容量=m  for(int k=1;k<=m;k++) //用k遍历第i组的所有物品  if(j >= c[i][k]) //第k个物品能装进容量j的背包  dp[j] = max(dp[j], dp[j-c[i][k]] + w[i][k]);//第i组第k个  printf("%d\n",dp[m]);  }  } |

2. 多重背包

|  |
| --- |
| 例5-3. 宝物筛选 洛谷 P1776 |

这里给出3种解法。

1）简单方法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | //洛谷 P1776：滚动数组版本的多重背包（超时TLE）  #include <bits/stdc++.h>  using namespace std;  const int N=100010;  int n,C,dp[N];  int w[N],c[N],m[N]; //物品i的价值、体积、数量  int main(){  cin >> n >> C; //物品数量，背包容量  for(int i=1;i<=n;i++) cin>>w[i]>>c[i]>>m[i];  //以下是滚动数组版本的多重背包  for(int i=1;i<=n;i++) //枚举物品  for(int j=C;j>=c[i];j--) //枚举背包容量  for(int k=1; k<=m[i] && k\*c[i]<=j; k++)  dp[j] = max(dp[j],dp[j-k\*c[i]]+k\*w[i]);  cout << dp[C] << endl;  return 0;  } |

2）“二进制拆分”优化

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | //洛谷 P1776：二进制拆分+滚动数组  #include <bits/stdc++.h>  using namespace std;  const int N=100010;  int n,C,dp[N];  int w[N],c[N],m[N];  int new\_n; //二进制拆分后的新物品总数量  int new\_w[N],new\_c[N],new\_m[N]; //二进制拆分后新物品  int main(){  cin >> n >>C;  for(int i=1;i<=n;i++) cin>>w[i]>>c[i]>>m[i];  //以下是二进制拆分  int new\_n = 0;  for(int i=1;i<=n;i++){  for(int j=1;j<=m[i];j<<=1) { //二进制枚举：1,2,4...  m[i]-=j; //减去已拆分的  new\_c[++new\_n] = j\*c[i]; //新物品  new\_w[new\_n] = j\*w[i];  }  if(m[i]){ //最后一个是余数  new\_c[++new\_n] = m[i]\*c[i];  new\_w[new\_n] = m[i]\*w[i];  }  }  //以下是滚动数组版本的0/1背包  for(int i=1;i<=new\_n;i++) //枚举物品  for(int j=C;j>=new\_c[i];j--) //枚举背包容量  dp[j] = max(dp[j],dp[j-new\_c[i]]+new\_w[i]);  cout << dp[C] << endl;  return 0;  } |

5.3 数位统计DP

|  |
| --- |
| 例5-4. 数字计数 洛谷P2602，poj 2282 |

5.3.1 数位统计DP的递推实现

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | //改写自：https://www.luogu.com.cn/blog/mak2333/solution-p2602  #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  const int N=15;  ll ten[N],dp[N];  ll cnta[N],cntb[N]; //cnt[i]，统计数字“i”出现了多少次  int num[N];  void init(){ //预计算dp[]  ten[0] = 1; //ten[i]: 10的i次方  for(int i=1;i<=N;i++){  dp[i] = i\*ten[i-1]; //或者写成：dp[i] = dp[i-1]\*10+ten[i-1];  ten[i] = 10\*ten[i-1];  }  }  void solve(ll x, ll \*cnt){  int len = 0; //数字x有多少位  while(x){ //分解x，num[i]是x的第i位数字  num[++len] = x%10;  x=x/10;  }  for(int i=len;i>=1;i--){ //从高到低处理x的每一位  for(int j=0;j<=9;j++) cnt[j] += dp[i-1]\*num[i];  for(int j=0;j<num[i];j++) cnt[j] += ten[i-1]; //特判最高位比num[i]小的数字  ll num2 = 0;  for(int j=i-1;j>=1;j--) num2 = num2\*10+num[j];  cnt[num[i]] += num2+1; //特判最高位的数字num[i]  cnt[0] -= ten[i-1]; //特判前导0  }  }  int main(){  init();  ll a,b; cin >> a>>b;  solve(a-1, cnta), solve(b, cntb);  for(int i=0;i<=9;i++) cout << cntb[i]-cnta[i] <<" ";  } |

5.3.2 数位统计DP的记忆化搜索实现

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  const int N = 15;  ll dp[N][N];  int num[N],now; //now：当前统计0~9的哪一个数字  ll dfs(int pos,int sum,bool lead, bool limit){ //pos：当前处理到第pos位  ll ans = 0;  if(pos == 0) return sum; //递归到0位数，结束，返回  if(!lead && !limit && dp[pos][sum]!=-1) return dp[pos][sum]; //记忆化搜索  int up = (limit ? num[pos] : 9); //这一位的最大值，例如324的第3位是up = 3  for(int i=0;i<=up;i++){ //下面以now=2为例：  if(i==0 && lead) ans += dfs(pos -1, sum, true, limit&&i==up); // 计算000~099  else if(i == now) ans += dfs(pos -1, sum+1,false,limit&&i==up); // 计算200~299  else if(i != now) ans += dfs(pos -1, sum, false,limit&&i==up); // 计算100~199  }  if(!lead && !limit) dp[pos][sum] = ans; //状态记录：无前导0  return ans;  }  ll solve(ll x){  int len = 0; //数字x有多少位  while(x){  num[++len] = x%10;  x/=10;  }  memset(dp,-1,sizeof(dp));  return dfs(len,0,true,true);  }  int main(){  ll a,b; cin>>a>>b;  for(int i=0;i<10;i++) now = i, cout << solve(b)-solve(a-1)<<" ";  return 0;  } |

5.3.3 数位统计DP例题

|  |  |  |  |
| --- | --- | --- | --- |
| 例5-5. Windy数 洛谷P2657 | | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include<bits/stdc++.h>  using namespace std;  int dp[15][15], num[15];  int dfs(int pos, int last, bool lead, bool limit){  int ans = 0;  if(pos == 0) return 1;  if(!lead && !limit && dp[pos][last]!=-1) return dp[pos][last];  int up = (limit?num[pos]:9);  for(int i=0;i<=up;i++){  if(abs(i-last)<2) continue; //不是windy数  if(lead && i==0) ans += dfs(pos-1,-2,true, limit&&i==up);  else ans += dfs(pos-1,i,false, limit&&i==up);  }  if(!limit && !lead) dp[pos][last] = ans;  return ans;  }  int solve(int x){  int len = 0;  while(x) { num[++len]=x%10; x/=10;}  memset(dp,-1,sizeof(dp));  return dfs(len,-2,true,true);  }  int main(){  int a,b; cin >>a>>b;  cout << solve(b)-solve(a-1);  return 0;  } | |
| 例5-6. 手机号码 洛谷P4124 | | | |
|  | |  | |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | //改写自：www.luogu.com.cn/blog/yushuotong-std/solution--4124  #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  ll dp[15][11][11][2][2][2];  int num[15];  ll dfs(int pos,int u,int v,bool state,bool n8,bool n4,bool limit){  ll ans=0;  if(n8 && n4) return 0; //8和4不能同时出现  if(!pos) return state;  if(!limit && dp[pos][u][v][state][n8][n4]!=-1) return dp[pos][u][v][state][n8][n4];  int up = (limit?num[pos]:9);  for(int i=0;i<=up;i++)  ans += dfs(pos-1,i,u,state||(i==u&&i==v),n8||(i==8),n4||(i==4),limit&&(i==up));  if(!limit) dp[pos][u][v][state][n8][n4]=ans;  return ans;  }  ll solve(ll x){  int len = 0;  while(x){num[++len]=x%10; x/=10;}  if(len!=11) return 0;  memset(&dp,-1,sizeof(dp));  ll ans = 0;  for(int i=1;i<=num[len];i++) //最高位1~9，避开前导0问题  ans+= dfs(len-1,i,0,0,i==8,i==4,i==num[len]);  return ans;  }  int main(){  ll a,b; cin >> a>>b;  cout <<solve(b)-solve(a-1);  } |

5.4.1 引子

|  |  |  |
| --- | --- | --- |
| 例5-7. 最短Hamilton路径 https://www.acwing.com/problem/content/description/93/ | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <bits/stdc++.h>  using namespace std;  int n, dp[1<<20][21];  int dist[21][21];  int main(){  memset(dp,0x3f,sizeof(dp)); //初始化最大值  cin>>n;  for(int i=0; i<n; i++) //输入图  for(int j=0; j<n; j++)  cin >> dist[i][j]; //输入点之间的距离  dp[1][0]=0; //开始：集合中只有点0，起点和终点都是0  for(int S=1; S<(1<<n); S++) //从小集合扩展到大集合，集合用S的二进制表示  for(int j=0; j<n; j++) //枚举点j  if((S>>j) & 1) //(1): 这个判断与下面的(2)一起起作用  for(int k=0; k<n; k++) //枚举到达j的点k，k属于集合S-j  if((S^(1<<j)) >> k & 1) //(2): k属于集合S-j。S-j用(1)保证  //把(1)和(2)写在一起，像下面这样，更容易理解，但是效率低一点：  //if( ((S>>j) & 1) && ((S^(1<<j)) >> k & 1) )  dp[S][j] = min(dp[S][j],dp[S^(1<<j)][k] + dist[k][j]);  cout << dp[(1<<n)-1][n-1]; //输出：路径包含了所有的点，终点是n-1  return 0;  } |

5.4.2 状态压缩DP的原理

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include<bits/stdc++.h>  int main(){  int a = 213, b = 21; // a = 1101 0101, b = 0001 0101  printf("a & b = %d\n",a & b); // AND = 21, 二进制0001 0101  printf("a | b = %d\n",a | b); // OR = 213, 二进制1101 0101  printf("a ^ b = %d\n",a ^ b); // XOR = 192, 二进制1100 0000  printf("a << 2 = %d\n",a << 2); // a\*4 = 852, 二进制0011 0101 0100  printf("a >> 2 = %d\n",a >> 2); // a/4 = 53, 二进制0011 0101  int i = 5; //(1)a的第i位是否为1  if((1 << (i-1)) & a) printf("a[%d]=%d\n",i,1); //a的第i位是1  else printf("a[%d]=%d\n",i,0); //a的第i位是0  a = 43, i = 5; //(2)把a的第i位改成1。a = 0010 1011  printf("a=%d\n",a | (1<<(i-1))); //a=59, 二进制0011 1011  a = 242; //(3)把a最后的1去掉。 a = 1111 0010  printf("a=%d\n", a & (a-1)); //去掉最后的1。 =240, 二进制1111 0000  return 0;  } |

5.4.3 状态压缩DP例题

|  |
| --- |
| 例5-8. **Mondriaan's Dream poj 2411** |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | #include <iostream>  #include <cstring>  using namespace std;  long long dp[2][1<<11];  int now,old; //滚动数组，now指向新的一行，old指向旧的一行  int main(){  int n,m;  while( cin>>n>>m && n ){  if(m>n) swap(n,m); //复杂度O(nm\*2^m), m较小有利  memset(dp,0,sizeof(dp));  now=0,old=1; //滚动数组  dp[now][(1<<m)-1]=1;  for(int i=0;i<n;i++) //n行  for(int j=0;j<m;j++){ //m列  swap(now,old); //滚动数组，now始终指向最新的一行  memset(dp[now],0,sizeof(dp[now]));  for(int k=0;k<(1<<m);k++){ //k：轮廓线上的m格  if(k & 1<<(m-1)) //情况(1)。要求k5=1  dp[now][(k<<1) & (~(1<<m))] += dp[old][k];  //原来的k5=1移到了第m+1位，置0  if(i && !(k & 1<<(m-1) ) ) //情况(2)  //i不等于0，即i不是第一行。另外要求k5=0  dp[now][(k<<1)^1] += dp[old][k];  if(j && (!(k&1)) && (k & 1<<(m-1)) ) //情况(3)  //j不等于0，即j不是第一列。另外要求k0=0, k5=1  dp[now][((k<<1) | 3) & (~(1<<m))] += dp[old][k];  //k末尾置为11，且原来的k5移到了第m+1位，置0  }  }  cout << dp[now][(1<<m)-1]<<endl;  }  return 0;  } |

2

|  |  |
| --- | --- |
| 例5-9. 排兵布阵 hdu 4539 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | //代码改写自：https://blog.csdn.net/jzmzy/article/details/20950205  #include <bits/stdc++.h>  using namespace std;  int mp[105][12]; //地图  int dp[105][200][200];  int n,m;  int sta[200]; //预计算一行的合法情况。m = 10时，只有169种合法情况  int init\_line(int n){ //预计算出一行的合法情况  int M = 0;  for(int i = 0; i < n; i ++)  if((i&(i>>2))==0 && (i&(i<<2))==0)//左右间隔2的位置没人，就是合法的  sta[M++] = i;  return M; //返回合法情况有多少种  }  int count\_line(int i, int x){ //计算第i行的士兵数量  int sum = 0;  for(int j=m-1; j>=0; j--) { //x是预计算过的合法安排  if(x&1) sum += mp[i][j]; //把x与地形匹配  x >>= 1;  }  return sum;  }  int main(){  while(~scanf("%d%d",&n,&m)) {  int M = init\_line(1<<m); //预计算一行的合法情况，有M种  for(int i = 0; i < n; i ++)  for(int j = 0; j < m; j ++)  scanf("%d",&mp[i][j]); //输入地图  int ans = 0;  memset(dp, 0, sizeof(dp));  for(int i = 0; i < n; i ++) //第i行  for(int j = 0; j < M; j ++) //枚举第i行的合法安排  for(int k = 0; k < M; k ++) { //枚举第i-1行的合法安排  if(i == 0) { //计算第1行  dp[i][j][k] = count\_line(i, sta[j]);  ans = max(ans, dp[i][j][k]);  continue;  }  if((sta[j]&(sta[k]>>1)) || (sta[j]&(sta[k]<<1)))  continue; //第i行和第i-1行冲突  int tmp = 0;  for(int p = 0; p < M; p ++){ //枚举第i-2行合法状态  if((sta[p]&(sta[k]>>1)) || (sta[p]&(sta[k]<<1))) continue;  //第i-1行和第i-2行冲突  if(sta[j]&sta[p]) continue; //第i行和第i-2行冲突  tmp = max(tmp, dp[i-1][k][p]); //从i-1递推到i  }  dp[i][j][k] = tmp + count\_line(i, sta[j]);  //加上第i行的士兵数量  ans = max(ans, dp[i][j][k]);  }  printf("%d\n",ans);  }  return 0;  } |

5.4.4 三进制状态压缩DP

|  |
| --- |
| 例5-10. hdu 3001 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | #include<bits/stdc++.h>  const int INF = 0x3f3f3f3f;  using namespace std;  int n,m;  int bit[12]={0,1,3,9,27,81,243,729,2187,6561,19683,59049};  //三进制每一位的权值。与二进制的0, 1, 2, 4, 8...对照理解  int tri[60000][11];  int dp[11][60000];  int graph[11][11]; //存图  void make\_trb(){ //初始化，求所有可能的路径  for(int i=0;i<59050;++i){ //共3^10 = 59050种路径状态  int t=i;  for(int j=1; j<=10; ++j){ tri[i][j]=t%3; t/=3; }  }  }  int comp\_dp(){  int ans = INF;  memset(dp, INF, sizeof(dp));  for(int j=0;j<=n;j++)  dp[j][bit[j]]=0; //初始化：从第j个城市出发，只访问j，费用为0  for(int i=0;i<bit[n+1];i++){ //遍历所有路径，每个i是一个路径  int flag=1; //所有的城市都遍历过1次以上  for(int j=1;j<=n;j++){ //遍历城市，以j为起点  if(tri[i][j] == 0){ //是否有一个城市访问次数是0  flag=0; //还没有经过所有点  continue;  }  for(int k=1; k<=n; k++){ //遍历路径i-j的所有城市  int l=i-bit[j]; //l:从路径i中去掉第j个城市  dp[j][i]=min(dp[j][i],dp[k][l]+graph[k][j]);  }  }  if(flag) //找最小费用  for(int j=1; j<=n; j++)  ans = min(ans,dp[j][i]); //路径i上，最小的总费用  }  return ans;  }  int main(){  make\_trb();  while(cin>>n>>m){  memset(graph,INF,sizeof(graph));  while(m--){  int a,b,c; cin>>a>>b>>c;  if(c<graph[a][b]) graph[a][b]=graph[b][a]=c;  }  int ans = comp\_dp();  if(ans==INF) cout<<"-1"<<endl;  else cout<<ans<<endl;  }  return 0;  } |

5.5.2 区间DP例题

|  |  |  |  |
| --- | --- | --- | --- |
| 例5-12. String painter hdu 2476 | | | |
|  | |  | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #include <bits/stdc++.h>  using namespace std;  char A[105],B[105];  int dp[105][105];  const int INF = 0x3f3f3f3f;  int main() {  while(~scanf("%s%s", A+1, B+1)){  int n = strlen(A+1); //输入A, B  for(int i=1;i<=n;i++) dp[i][i]=1; //初始化  //先从空白串转换到B  for(int len=2; len<=n; len++)  for(int i=1; i<=n-len+1; i++){  int j = i + len-1;  dp[i][j] = INF;  if(B[i] == B[j]) //区间[i, j]两端的字符相同B[i] = B[j]  dp[i][j] = dp[i+1][j]; //或者 = dp[i][j-1])  else //区间[i, j]两端的字符不同B[i] ≠ B[j]  for(int k=i; k<j; k++)  dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j]);  }  //下面从A转换到B  for(int j=1; j<=n; ++j){  if(A[j] == B[j]) dp[1][j] = dp[1][j-1]; //字符相同不用转  else  for(int k=1; k<j; ++k)  dp[1][j] = min(dp[1][j], dp[1][k] + dp[k+1][j]);  }  printf("%d\n",dp[1][n]);  }  return 0;  } | |

5.5.3 二维区间DP

|  |  |
| --- | --- |
| 例5-14. CF1199 F. Rectangle Painting 1 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | //代码改写自：https://www.cnblogs.com/zsben991126/p/11643832.html  #include<bits/stdc++.h>  using namespace std;  #define N 55  int dp[N][N][N][N];  char mp[N][N]; //方格图  int main(){  int n; cin>>n;  for(int i=1;i<=n;i++)  for(int j=1;j<=n;j++) cin>>mp[i][j];  for(int i=1;i<=n;i++)  for(int j=1;j<=n;j++)  if(mp[i][j]=='.') dp[i][j][i][j]=0; //白格不用涂  else dp[i][j][i][j]=1; //黑格(i,j)涂成白色需1次  for(int lenx=1;lenx<=n;lenx++) //len从1开始，不是2。因为有x和y两个方向  for(int leny=1;leny<=n;leny++)  for(int x1=1;x1<=n-lenx+1;x1++)  for(int y1=1;y1<=n-leny+1;y1++){  int x2 = x1+lenx-1; //x1：x轴起点；x2：x轴终点  int y2 = y1+leny-1; //y1：y轴起点；y2：y轴终点  if(x1==x2 && y1==y2) continue; //lenx=1且leny=1的情况  dp[x1][y1][x2][y2] = max(abs(x1-x2),abs(y1-y2)) + 1; //初始值  for(int k=x1;k<x2;k++) //枚举x方向，y不变。区间[x1,k]+[k+1,x2]  dp[x1][y1][x2][y2] = min(dp[x1][y1][x2][y2], dp[x1][y1][k][y2]+dp[k+1][y1][x2][y2]);  for(int k=y1;k<y2;k++) //枚举y方向，x不变。区间[y1,k]+[k+1,y2]  dp[x1][y1][x2][y2] = min(dp[x1][y1][x2][y2], dp[x1][y1][x2][k]+dp[x1][k+1][x2][y2]);  }  cout << dp[1][1][n][n];  } |

5.6.1 树形DP的基本操作

|  |  |  |
| --- | --- | --- |
| **二叉苹果树** 洛谷P2015 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | //洛谷P2015的代码（邻接表存树）  #include<bits/stdc++.h>  using namespace std;  const int N = 200;  struct node{  int v, w; //v是子结点，w是边[u,v]的值  node(int v = 0, int w = 0) : v(v), w(w) {}  };  vector<node> edge[N];  int dp[N][N], sum[N]; //sum[i]记录以点i为根的子树的总边数  int n, q;  void dfs(int u, int father){  for(int i = 0; i < edge[u].size(); i++) { //用i遍历u的所有子结点  int v = edge[u][i].v, w = edge[u][i].w;  if(v == father) continue; //不回头搜父亲，避免循环  dfs(v, u); //递归到最深的叶子结点，然后返回  sum[u] += sum[v] + 1; //子树上的总边数  //for(int j=sum[u];j>=0;j--)  // for(int k=0;k<=j-1;k++) //两个for优化为下面的代码。不优化也行  for(int j = min(q, sum[u]); j >= 0; j--)  for(int k = 0; k <= min(sum[v], j - 1); k++)  dp[u][j] = max(dp[u][j], dp[u][j-k-1] + dp[v][k] + w);  }  }  int main(){  scanf("%d%d", &n, &q); //n个点，留q条树枝  for(int i = 1; i < n; i++){  int u, v, w; scanf("%d%d%d", &u, &v, &w);  edge[u].push\_back(node(v,w)); //把边[u,v]存到u的邻接表中  edge[v].push\_back(node(u,w)); //无向边  }  dfs(1, 0); //从根结点开始做记忆化搜索  printf("%d\n", dp[1][q]);  return 0;  } |

|  |
| --- |
| 例5-15. 没有上司的舞会 洛谷P1352 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | #include<bits/stdc++.h>  using namespace std;  const int N = 6005;  int val[N], dp[N][2], father[N];  vector <int> G[N];  void addedge(int from,int to){  G[from].push\_back(to); //用邻接表建树  father[to] = from; //父子关系  }  void dfs(int u){  dp[u][0] = 0; //赋初值：不参加宴会  dp[u][1] = val[u]; //赋初值：参加宴会  //for(int i=0;i<G[u].size();i++){ //遍历u的邻居v。逐一处理这个父结点的每个子结点  // int v = G[u][i];  for(int v : G[u]){ //这一行和上面两行的作用一样  dfs(v); //深搜子结点  dp[u][1] += dp[v][0]; //父结点选择，子结点不选  dp[u][0] += max(dp[v][0], dp[v][1]); //父结点不选，子结点可选可不选  }  }  int main(){  int n; scanf("%d",&n);  for(int i=1;i<=n;i++) scanf("%d",&val[i]); //输入快乐指数  for(int i=1;i<n;i++){  int u,v; scanf("%d%d",&u,&v);  addedge(v,u);  }  int t = 1;  while(father[t]) t = father[t]; //查找树的根结点  dfs(t); //从根结点开始，用dfs遍历整棵树  printf("%d\n", max(dp[t][0], dp[t][1]));  return 0;  } |

5.7 一般优化

|  |
| --- |
| 例5-17. 方伯伯的玉米田 洛谷P3287 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //代码改写自： www.luogu.com.cn/blog/361308/solution-p3287  #include<bits/stdc++.h>  using namespace std;  #define lowbit(x) ((x) & - (x))  int a[10005], dp[10005][505], t[505][5505], n, k;  void update(int x, int y, int d) { //更新区间  for (int i=x; i <= k + 1; i += lowbit(i))  for (int j=y; j <= 5500; j += lowbit(j))  t[i][j] = max(t[i][j], d);  }  int query(int x, int y) { //查区间最大值  int ans=0;  for (int i=x; i>0; i -= lowbit(i))  for (int j=y; j>0; j -= lowbit(j))  ans = max(ans, t[i][j]);  return ans;  }  int main() {  scanf("%d%d", &n, &k);  for (int i=1; i <= n; i++) scanf("%d", &a[i]);  for (int i=1; i <= n; i++)  for (int j=k; j>=0; j--){  dp[i][j] = query(j+1, a[i]+j) + 1;  update(j+1, a[i]+j, dp[i][j]);  }  printf("%d", query(k+1, 5500));  return 0;  } |

5.8.2 单调队列优化例题

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 例5-18. Mowing the Lawn 洛谷P2627 | | | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | //代码改写自：https://www.luogu.com.cn/blog/user21293/solution-p2627  #include <bits/stdc++.h>  using namespace std;  const int N=100005;  long long n,k,e[N],sum[N],dp[N];  long long ds[N]; //ds[j] = dp[j-1]-sum[j]  int q[N],head=0,tail=1; //递减的单调队列，队头最大  long long que\_max(int j){  ds[j] = dp[j-1]-sum[j];  while(head<=tail && ds[q[tail]]<ds[j]) tail--; //去掉不合格的队尾  q[++tail]=j; //j进队尾  while(head<=tail && q[head]<j-k) head++; //去掉超过窗口k的队头  return ds[q[head]]; //返回队头，即最大的dp[j-1]-sum[j]  }  int main(){  cin >> n >> k; sum[0] = 0;  for(int i=1;i<=n;i++){  cin >> e[i]; sum[i] = sum[i-1] + e[i]; //计算前缀和  }  for(int i=1;i<=n;i++) dp[i] = que\_max(i) + sum[i]; //状态转移方程  cout << dp[n];  } | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | | //洛谷 P1776：单调队列优化多重背包  #include<bits/stdc++.h>  using namespace std;  const int N=100010;  int n,C;  int dp[N],q[N],num[N];  int w,c,m; //物品的价值w、体积c、数量m  int main(){  cin >> n >> C; //物品数量n，背包容量W  memset(dp,0,sizeof(dp));  for(int i=1;i<=n;i++){  cin>>w>>c>>m; //物品i的价值w、体积c、数量m  if(m>C/c) m = C/c; //计算 min{m, j/c}  for(int b=0;b<c;b++){ //按余数b进行循环  int head=1, tail=1;  for(int y=0;y<=(C-b)/c;y++){ //y = j/c  int tmp = dp[b+y\*c]-y\*w; //用队列处理tmp = dp[b + xc] - xw  while(head<tail && q[tail-1]<=tmp) tail--;  q[tail] = tmp;  num[tail++] = y;  while(head<tail && y-num[head]>m) head++;  //约束条件y-min(mi,y)≤x≤y  dp[b+y\*c] = max(dp[b+y\*c],q[head]+y\*w); //计算新的dp[]  }  }  }  cout << dp[C] << endl;  return 0;  } |

5.9.4 例题

|  |
| --- |
| 例5-19. Print Article hdu 3507 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | #include <bits/stdc++.h>  using namespace std;  const int N = 500010;  int dp[N];  int q[N]; //单调队列  int sum[N];  int X(int x){ return 2\*sum[x]; }  int Y(int x){ return dp[x]+sum[x]\*sum[x]; }  //double slope(int a,int b){return (Y(a)-Y(b))/(X(a)-X(b));}//除法不好，改成下面的乘法  int slope\_up (int a,int b) { return Y(a)-Y(b);} //斜率的分子部分  int slope\_down(int a,int b) { return X(a)-X(b);} //斜率的分母部分  int main(){  int n,m;  while(~scanf("%d%d",&n,&m)){  for(int i=1;i<=n;i++) scanf("%d",&sum[i]);  sum[0] = dp[0] = 0;  for(int i=1;i<=n;i++) sum[i]+=sum[i-1];  int head=1,tail=1; //队头队尾  q[tail]=0;  for(int i=1;i<=n;i++){  while(head<tail &&  slope\_up(q[head+1],q[head])<=sum[i]\*slope\_down(q[head+1],q[head]))  head++; //斜率小于k，从队头弹走  int j = q[head]; //队头是最优点  dp[i] = dp[j]+m+(sum[i]-sum[j])\*(sum[i]-sum[j]); //计算dp[i]  while(head<tail &&  slope\_up(i,q[tail])\*slope\_down(q[tail],q[tail-1])  <= slope\_up(q[tail],q[tail-1])\*slope\_down(i,q[tail]))  tail--; //弹走队尾不合格的点  q[++tail] = i; //新的点进队尾  }  printf("%d\n",dp[n]);  }  return 0;  } |