常用高阶数据结构算法 只因版

1. 区间动态规划  
   1274：【例9.18】合并石子 （前缀和+区间动态规划）

#include<bits/stdc++.h>

using namespace std;

const int N=105;

int n,a[N],sum[N],dp[N][N];

int main(){

cin>>n;

memset(dp,0x3f,sizeof dp);

for(int i=1;i<=n;i++){

cin>>a[i];

sum[i]=sum[i-1]+a[i];

dp[i][i]=0;

}

for(int l=2;l<=n;l++){

for(int i=1;i<=n-l+1;i++){

int j=i+l-1;

for(int k=i;k<=j-1;k++){

dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]+sum[j]-sum[i-1]);

}

}

}

cout<<dp[1][n];

return 0;

}

1. 状态压缩动态规划

//1删除i的最后一位 i>>=1 i/=2

//2统计i中有几个1 // while(i){ // sum+=(i&1); //sum+=(i%2);// i>>=1; //i/=2//}

//3判断状态i(i表示的集合)中第j位是否为1 (i&(1<<j))!=0

//4判断状态j是否是集合i的子集 (i|j)==i (j&i)==j

//5把第k位添加到集合i里 i|(1<<k)

//6从集合i里取出(删除)第j位 i^(1<<j)

//7集合i和集合j没有冲突位 (i&j)==0

//8状态i没有相邻位被选择 (i&(i>>1))==0

//9枚举状态i的子集for(int t=i;t;t=(t-1)&i)

//10删掉最后一个 1 x&=(x-1)

P5911 [POI2004] PRZ

#include<bits/stdc++.h>

using namespace std;

int W,n,t[20],w[20],\_t[1<<20],\_w[1<<20];

int dp[1<<20];

int main(){

cin>>W>>n;

for(int i=0;i<n;i++){

cin>>t[i]>>w[i];

}

//111

for(int i=0;i<(1<<n);i++){

for(int j=0;j<n;j++){

if((1<<j)&i){

\_t[i]=max(\_t[i],t[j]);

\_w[i]+=w[j];

}

}

}

memset(dp,0x3f,sizeof dp);

dp[0]=0;

for(int i=0;i<(1<<n);i++){

for(int j=i;j;j=((j-1)&i)){

if(\_w[j]<=W) dp[i]=min(dp[i],\_t[j]+dp[i^j]);

}

}

cout<<dp[(1<<n)-1];

return 0;

}

1. 三分法  
   1435：【例题3】曲线

#include<bits/stdc++.h>

using namespace std;

const int N=1e4+5;

int T,n;

double a[N],b[N],c[N],l,r,m1,m2;

double f(double x){

double maxx=-0x7fffffff;

for(int i=1;i<=n;i++){

maxx=max(maxx,a[i]\*x\*x+b[i]\*x+c[i]);

}

return maxx;

} int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>T;

for(int i=1;i<=T;i++){

cin>>n;

for(int j=1;j<=n;j++) cin>>a[j]>>b[j]>>c[j];

l=0,r=1000;

while(r-l>1e-9){

m1=l+(r-l)/3;

m2=r-(r-l)/3;

if(f(m1)<f(m2)) r=m2;

else l=m1;

}

cout<<fixed<<setprecision(4)<<f(l)<<endl;

}

return 0;

}

1. 深搜剪枝策略  
   可行性剪枝->最优性剪枝->重复性剪枝 奇偶性剪枝
2. 双向广搜  
   1450：【例 3】Knight Moves

#include<bits/stdc++.h>

using namespace std;int T;//T组数据

struct P{

int x,y;

};queue<P> q[2];//q[0]正向搜索队列 q[1]反向搜索队列

bool book[2][305][305];//book[0][x][y]正向搜索时(x,y)是否走过(第一个队列)//book[1][x][y]反向搜索时(x,y)是否走过(第二个队列)

int step[2][305][305];//step[0][x][y]正向搜索时到(x,y)的最小步数//step[1][x][y]反向搜索时到(x,y)的最小步数

int nxt[8][2]={1,2,2,1,-1,-2,-2,-1,1,-2,-1,2,2,-1,-2,1};//方向数组

int sx,sy,ex,ey,n,ans;bool expand(int k){

int oldx=q[k].front().x,oldy=q[k].front().y;//用oldx oldy存储队首元素的坐标 准备拓展新点

int s=step[k][oldx][oldy];//s存到老点的步数

for(int i=0;i<=7;i++){

int nx=oldx+nxt[i][0];//计算新落脚点

int ny=oldy+nxt[i][1];

if(nx>=0&&nx<=n-1&&ny>=0&&ny<=n-1&&book[k][nx][ny]==0){

book[k][nx][ny]=1; //标记这个点走过了

q[k].push({nx,ny}); //让新点nx ny 入队

step[k][nx][ny]=s+1; //计算到nx ny的步数

if(book[1-k][nx][ny]){

ans=step[k][nx][ny]+step[1-k][nx][ny];

return 1;

}

}

}

return 0;

}void bfs(){

if(sx==ex&&sy==ey){

ans=0;

return ;

}

//起点入第一队 终点入第二队

q[0].push({sx,sy}); q[1].push({ex,ey});

//标记第一队的起点 标记第二队的终点

book[0][sx][sy]=1; book[1][ex][ey]=1;

while(!q[0].empty()&&!q[1].empty()){

if(q[0].size()<q[1].size()){//判断哪个队列里的元素比较少 就让谁扩展

if(expand(0)) return ;

q[0].pop();

}

if(q[0].size()>=q[1].size()){

if(expand(1)) return ;

q[1].pop();

}

}

} int main(){

cin>>T;

while(T--){

memset(book,0,sizeof book);//初始化数组

memset(step,0,sizeof step);

while(!q[0].empty()) q[0].pop();//清空队列

while(!q[1].empty()) q[1].pop();

cin>>n>>sx>>sy>>ex>>ey;//棋盘大小 起点 终点

bfs();

cout<<ans<<endl;

}

return 0;

}

1. 双端队列优化广搜  
   P4667 [BalticOI 2011 Day1] Switch the Lamp On

#include<bits/stdc++.h>

#define PII pair<int,int>

using namespace std;

const int N=505;

int dx[4]={1,-1,-1,1};//点的方向数组

int dy[4]={1,1,-1,-1};

char a[5]="\\/\\/";// \ / \ /

int nx[4]={0,-1,-1,0};//格子的方向数组

int ny[4]={0,0,-1,-1};

deque<PII> q,q1;//双端队列

char ch[N][N];//地图   
int step[N][N];//到每个点的最短步数

int n,m,T;//行列

void front\_and\_goback\_bfs(){

q=q1;

memset(step,0x3f,sizeof step);

q.push\_back(make\_pair(0,0));

step[0][0]=0;

while(!q.empty()){

PII old=q.front();

int xx=old.first,yy=old.second;

q.pop\_front();

for(int i=0;i<=3;i++){

int dnx=xx+dx[i];//新点坐标

int dny=yy+dy[i];

int inx=xx+nx[i];//格子坐标

int iny=yy+ny[i];

if(dnx>=0&&dnx<=n&&dny>=0&&dny<=m){

if(a[i]!=ch[inx][iny]){

int t=step[xx][yy]+1;

if(t<step[dnx][dny]){

q.push\_back(make\_pair(dnx,dny));

step[dnx][dny]=t;

}

}else{

int t=step[xx][yy];

if(t<step[dnx][dny]){

q.push\_front(make\_pair(dnx,dny));

step[dnx][dny]=t;

}

}

}

}

}

cout<<step[n][m]<<endl;

}int main(){

cin>>n>>m;

for(int i=0;i<n;i++){

cin>>ch[i];

}

if((n+m)%2) cout<<"NO SOLUTION"<<endl;

else front\_and\_goback\_bfs();

return 0;

}

1. 哈希表  
   1456：【例题2】图书管理

#include<bits/stdc++.h>

using namespace std;

typedef unsigned long long ull;

const int mod1=1e6+3,mod2=1e6+9,b1=47,b2=79,N=3e4+5;

int cnt=0,nxt[N],f[mod1+5],num[N+5];

int n,len,sum1,sum2;

char op[10],ch[205];

void insert(int x,int y){//x:哈希值 y:key

nxt[++cnt]=f[x];

f[x]=cnt;

num[cnt]=y;

}

int query(int x,int y){

for(int i=f[x];i;i=nxt[i]){

if(num[i]==y) return 1;

}

return 0;

}

int main(){

cin>>n;

while(n--){

cin>>op;

getchar();

fgets(ch,sizeof ch,stdin);

len=strlen(ch)-1;

sum1=0,sum2=0;

for(int i=0;i<len;i++){

sum1=(sum1\*b1+ch[i])%mod1;

sum2=(sum2\*b2+ch[i])%mod2;

}

if(op[0]=='a') insert(sum1,sum2);

else{

if(query(sum1,sum2)) cout<<"yes"<<endl;

else cout<<"no"<<endl;

}

}

return 0;

}