Bitset

bitset是一种类似数组的数据结构，它的每一个元素只能是0或1，每个元素只用1bit的空间。可以使用只包含‘0’或‘1’的字符串构造。

优点

支持所有的位运算。

空间占用非常小，也可用于优化时间。

声明方式

```
#include<bitset>
bitset<30>bi;

string s="100101";
bitset<10>bs(s); //长度为10，前面用0补充
cout<<bs<<endl; //0000100101
cout<<bs[0]<<endl; //下标是从右到左的，可以像数组一样访问每一位
```

方法

```
bi.size() //返回大小，即位数
bi.count() //返回1的个数
bi.any() //返回是否有1
bi.none() //返回是否没有1
bi.set() //全部变成1
bi.set(p)//将第p+1位变成1
bi.set(p,x)//将p+1位变成x
bi.reset() //全部变成0
bi.reset(p) //将p+1位变成0
bi.flip() //全部取反
bi.flip(p) //将p+1位取反
bi.test(p) //返回i的索引，如果不存在则返回0
bi.to_ulong() //返回它转换为unsigned long的结果，如果超出范围则报错
bi.to_ullong() //返回它转换为unsigned long long的结果
bi.to_string() //返回它转换位string的结果
```

bitset优化传递闭包

[JSOI2010]连通数

```
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
#define int long long
using namespace std;
const int maxn=2007;

bitset<maxn>bi[maxn];
int n,ans=0;
string str;

signed main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>str;
        for(int j=1;j<=n;j++)
            bi[i][j]=(str[j-1]=='1'||i==j)?1:0;
    }
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(bi[j].test(i)) bi[j]|=bi[i];
    for(int i=1;i<=n;i++) ans+=bi[i].count();
    cout<<ans<<endl;
    return 0;
}
```

CSP-S模拟赛 世界线

题目大意：给你个有向图，问加多少条边可以使得不出现u能到达v, v能到达w, u不能达到w的情况。

这道题难在卡空间，传递闭包不能开n^2的空间，因此我们要把bitset拆开处理，即先处理了前半部分的闭包，再处理后半部分的闭包。当然这道题可以分很多份，分两份就可以卡过去了。

```
#include<bits/stdc++.h>
using namespace std;
const int N=60007;
const int mod=1e9+7;

int read(){ int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}

int n,m,u,v,res=0,vis[N];
vector<int>G[N];
bitset<30005>bi[N];

void dfs(int x,int flag){
    vis[x]=1;
    if(x<=n/2&&!flag) bi[x][x]=1;
    if(x>n/2&&flag) bi[x][x-n/2]=1;
    for(auto to:G[x]){
        if(!vis[to]) dfs(to,flag);
        bi[x]|=bi[to];
    }
    res+=bi[x].count();
}

signed main(){
    freopen("worldline.in","r",stdin);
    freopen("worldline.out","w",stdout);
    n=read();m=read();
    for(int i=1;i<=m;i++){
        u=read();v=read();
        G[u].push_back(v);
    }
    for(int i=1;i<=n;i++) if(!vis[i]) dfs(i,0);
    for(int i=1;i<=n;i++) vis[i]=0,bi[i].reset();
    for(int i=1;i<=n;i++) if(!vis[i]) dfs(i,1);
    printf("%d\n",res-m-n);
    return 0;
}
```

bitset优化Dp转移

转移式$dp[i][k]| = dp[i-1][k-j*j]$

```
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
```

```
#define int long long
using namespace std;
const int N=105;
bitset<N*N*N>dp[N];
int n,ans=0;
int l[N],r[N];

signed main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>l[i]>>r[i];
    }
    dp[0].set(0);
    for(int i=1;i<=n;i++){
        for(int j=l[i];j<=r[i];j++){
            dp[i]|=dp[i-1]<<(j*j);
        }
    }
    cout<<dp[n].count()<<"\n";
    return 0;
}
```

luoguP1537 弹珠

01背包+bitset优化，效率非常优秀。

题意是：有1~6大小的若干个弹珠，要求能否平分总数。

```
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;

int read(){ int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}

int sum,s[10],idx=0;
bitset<20010>frog;

void clear(){
    sum=0;
    frog.reset();
}

signed main(){
    while(1){
        clear();
        for(int i=1;i<=6;i++) s[i]=read();
        for(int i=1;i<=6;i++) sum+=s[i]*i; //记录总数
        if(!sum) break;
        frog.set(0);
        for(int i=1;i<=6;i++){
            for(int j=1;j<=s[i];j++) frog|=(frog<<i);
        }
        printf("Collection #%d:\n",++idx);
        if(frog.test(sum>>1)&&sum%2==0) puts("Can be divided.\n");
```

```
        else puts("Can't be divided.\n");
    }
    return 0;
}
```

luoguP5020 [NOIP2018 提高组] 货币系统

bitset优化完全背包

题意：给定货币系统(n,a)，求等价的货币系统(m,b)，令m最小（等价说明他们能表示的面额一样）

```
#include<bits/stdc++.h>
using namespace std;

int read(){ int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}

int t,n;
int a[105];
bitset<25005>s;

signed main(){
    t=read();
    while(t--){
        s.reset();
        n=read();
        for(int i=1;i<=n;i++) a[i]=read();
        sort(a+1,a+1+n);
        s[0]=1;
        int ans=0;
        for(int i=1;i<=n;i++){ //只要用最小的数表示a就可以了
            if(!s[a[i]]){
                ++ans;
                int x=a[i];
                while(x<=a[n]){ //第i个面额可以表示出的数额
                    s|=s<<x;
                    x<<=1;
                }
            }
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

bitset优化区间操作

```
//luoguP3674 小清新人渣的本愿
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+7;
const int mod=1e9+7;
typedef long long ll;

namespace IPT{
    const int L=1000000;
    char buf[L],*front=buf,*end=buf;
    char GetChar(){
        if(front==end){
            end=buf+fread(front=buf,1,L,stdin);
            if(front==end) return -1;
        }
        return *(front++);
    }
}

template<typename T>
inline void qr(T &x){
    char ch=IPT::GetChar(),lst=' ';
    while(ch>'9'||ch<'0') lst=ch,ch=IPT::GetChar();
    while(ch>='0'&&ch<='9') x=(x<<1)+(x<<3)+(ch^48),ch=IPT::GetChar();
    if(lst=='-') x=-x;
}

template<typename T>
inline void ReadDb(T &x){
    char ch=IPT::GetChar(),lst=' ';
    while(ch>'9'||ch<'0') lst=ch,ch=IPT::GetChar();
    while(ch>='0'&&ch<='9') x=x*10+(ch^48),ch=IPT::GetChar();
    if(ch=='.'){
        ch=IPT::GetChar();
        double base=1;
        while(ch>='0'&&ch<='9') x+=(ch^48)*((base*=0.1)),ch=IPT::GetChar();
    }
    if(lst=='-') x=-x;
}

namespace OPT{
    char buf[120];
}

template<typename T>
inline void qw(T x,const char aft,const bool pt){
    if(x<0){x=-x,putchar('-');}
    int top=0;
    do{OPT::buf[++top]=x%10+'0';}while(x/=10);
    while(top) putchar(OPT::buf[top--]);
    if(pt) putchar(aft);
}

const int maxn=100010;
```

```
bitset<maxn>s1,s2; //第i为表示i这个数字有无出现，再建一个表示N-x是否出现

int n,m;
int MU[maxn],bel[maxn],oc[maxn];

struct Ask{
    int opt,l,r,v,id;
    bool ans;
    inline bool operator <(const Ask &_others)const{
        if(bel[this->l]!=bel[_others.l]) return this->l < _others.l;
        if(bel[this->l]&1) return this->r < _others.r;
        return this->r > _others.r;
    }
}ask[maxn];

inline bool cmp(const Ask &_a,const Ask &_b){
    return _a.id<_b.id;
}

inline void add(int x){
    x=MU[x]; if(!(oc[x]++)) s1[x]=s2[n-x]=1;
}

inline void dlt(int x){
    x=MU[x]; if(!(--oc[x])) s1[x]=s2[n-x]=0;
}

signed main(){
    qr(n);qr(m);
    for(int i=1;i<=n;i++) qr(MU[i]);
    for(int i=1;i<=m;i++){
        Ask &now=ask[i];
        qr(now.opt);qr(now.l);qr(now.r);qr(now.v);
        now.id=i;
    }
    for(int i=1,sn=sqrt(n);i<=n;i++) bel[i]=i/sn;
    sort(ask+1,ask+1+m);
    int L=ask[1].l,R=L-1;
    for(int i=1;i<=m;i++){
        int l=ask[i].l,r=ask[i].r;
        while(L<l) dlt(L++);
        while(L>l) add(--L);
        while(R<r) add(++R);
        while(R>r) dlt(R--);
        int op=ask[i].opt,x=ask[i].v;
        if(op==1) ask[i].ans=(s1&(s1<<x)).any();//y和y+x同时存在等于s和s<<x同时存在一
个1
        else if(op==2) ask[i].ans=(s1&(s2>>(n-x))).any(); //n-i是否出现，右移(n-x)
位就是x-i是否出现
        else{
            for(int j=1;j*j<=x;j++) if(!(x%j)){
                if(s1[j]&&s1[x/j]){
                    ask[i].ans=1;
                    break;
                }
            }
        }
    }
```

```
    sort(ask+1,ask+1+m,cmp);
    for(int i=1;i<=m;i++) puts(ask[i].ans?"hana":"bi");
    return 0;
}
```

手写Bitset

用于解决bitset的过度封装导致的bitset的一些操作不能实现，比如两个二进制数求lowbit等问题。

```
#define Fusu_Bitset

#ifdef Fusu_Bitset
#include <cstddef>              //size_t used
#include <cstring>

#include <string>
#include <algorithm>
#endif

namespace Fusu {


template <size_t _len>
struct Bitset {
#define rg register
#define ci const int
#define cl const long long
    typedef long long int ll;
    typedef unsigned long long int ull;
    const static int _BitNum = 64;
    const static int _Size = _len / _BitNum + ((_len % _BitNum) == 0 ? 0 : 1);
    ull _Ary[_Size];
    ull _upceil;
    const static ull _INF = (1ull << 63) - 1 + (1ull << 63);
    Bitset() {                                          //constructed function
of std::string is left out because i dont know how to implement it
        memset(_Ary, 0, sizeof _Ary);
        int _firstsize = _len % _BitNum;
        for (rg int i = 0; i < _firstsize; ++i) _upceil |= 1ull << i;
        if (!_firstsize) _upceil = _INF;
    }

    void reset() {*this = Bitset();}
                                                        //operators

    void rtmve(const int &_v) {
        for (rg int i = _Size - 1; i >= _v; --i) this->_Ary[i] = this->_Ary[i -
_v];
        for (rg int i = _v - 1; ~i; --i) this->_Ary[i] = 0;
    }

    void lftmve(const int &_v) {
        for (rg int i = 0; (i + _v) < _Size; ++i) this->_Ary[i] = this->_Ary[i +
_v];
```

```
        for (rg int i = _Size - _v; i < _Size; ++i) this->_Ary[i] = 0;
    }

    Bitset& operator<<=(int _v) {
        if (_v < 0) {
            *this >>= -_v;
            return *this;
        }
        this->lftmve(_v / _BitNum);
        _v %= _BitNum;
        ull _tp = 0, _Pos = _BitNum - _v;
        for (rg int  i = 1; i <= _v; ++i) _tp |= 1ull << (_BitNum - i);
        ull _Lstv = 0;
        for (rg int i = _Size - 1; ~i; --i) {
            ull _Tp_Lstv = (_Ary[i] & _tp) >> _Pos;
            _Ary[i] <<= _v;
            _Ary[i] |= _Lstv;
            _Lstv = _Tp_Lstv;
        }
        this->_Ary[0] &= _upceil;
        return *this;
    }

    Bitset& operator>>=(int _v) {
        if (_v < 0) {
            *this <<= -_v;
            return *this;
        }
        this->rtmve(_v / _BitNum);
        _v %= _BitNum;
        ull _tp = (1ull << _v)- 1;
        ull _Lstv = 0, __Pos = _BitNum - _v;
        for (rg int i = 0; i < _Size; ++i) {
            ull _Tp_Lstv = (_Ary[i] & _tp) << __Pos;
            _Ary[i] >>= _v;
            _Ary[i] |= _Lstv;
            _Lstv = _Tp_Lstv;
        }
        this->_Ary[0] &= _upceil;
        return *this;
    }

    Bitset operator&(const Bitset &_others) const {
        Bitset _ret;
        for (rg int i = _Size - 1; ~i; --i) {
            _ret._Ary[i] = this->_Ary[i] & _others._Ary[i];
        }
        return _ret;
    }

    Bitset operator|(const Bitset &_others) const {
        Bitset _ret;
        for (rg int i = _Size - 1; ~i; --i) {
            _ret._Ary[i] = this->_Ary[i] | _others._Ary[i];
        }
        return _ret;
    }
```

```
    Bitset operator^(const Bitset &_others) const {
        Bitset _ret;
        for (rg int i = _Size - 1; ~i; --i) {
            _ret._Ary[i] = this->_Ary[i] ^ _others._Ary[i];
        }
        return _ret;
    }

    Bitset operator~() const {
        Bitset _ret;
        for (rg int i = _Size - 1; ~i; --i) {
            _ret._Ary[i] = ~this->_Ary[i];
        }
        return _ret;
    }

    Bitset operator<<(const int &_v) const {
        Bitset x = *this;
        x <<= _v;
        return x;
    }

    Bitset operator>>(const int &_v) const {
        Bitset x = *this;
        x >>= _v;
        return x;
    }

                                                        //member functions
    inline void __GetPos(const ull &_pos, int &__Pos, int &_v) {
        __Pos = _Size - _pos / _BitNum - 1;
        _v = _pos % _BitNum;
    }

    void set() {
        for (rg int i = 0; i < _Size; ++i) this->_Ary[i] = _INF;
    }

    void set(const ull &_pos, const bool val = true) {
        int __Pos , _v;
        __GetPos(_pos,__Pos,_v);
        if (val) {
            this->_Ary[__Pos] |= (1ull << (_v));
        } else {
            this->_Ary[__Pos] |= (1ull << (_v));
            this->_Ary[__Pos] ^= (1ull << (_v));
        }
    }

    int test(const ull &_pos) {
        int __Pos , _v;
        __GetPos(_pos,__Pos,_v);
        return this->_Ary[__Pos] & (1ull << (_v)) ? 1 : 0;
    }

    bool any() {
        for (rg int i = _Size - 1; ~i; --i) if (this->_Ary[i]) return true;
        return false;
```

```
    }

    bool none() {
        return !this->any();
    }

    ull conut() {
        ull _cnt = 0;
        for (rg int i = _Size - 1; ~i; --i) _cnt += __builtin_popcount(this-
>_Ary[i]);
        /*
            *if u cant used double_underlined functions,
            *u can set a val to maintain the num of true
            *and change it in other operators which would change the num of true
        */
        return _cnt;
    }

    void flip() {
        *(this) = ~(*this);
    }

    void flip(const ull &_pos) {
        if(this->test(_pos)) this->set(_pos, false);
        else this->set(_pos, true);
    }

                                                        //changing functions
    std::string to_string() {
        std::string _ret;
        _ret.clear();
        for (rg int i = 0; i < _Size; ++i) {
            for (rg int j = _BitNum - 1; ~j; --j) _ret += (this->_Ary[i] & (1ull
<< j)) ? '1' : '0';
        }
        return _ret;
    }

    unsigned int to_ulong() {
        return this->_Ary[_Size - 1];
    }
};
}   //namespace
```

bitset优化LCS

复杂度单次$O(n^2/w)$

```
/*
 * Author : _Wallace_
 * Source : https://www.cnblogs.com/-Wallace-/
 * Problem : LOJ #6564. 最长公共子序列
 * Standard : GNU C++ 03
 * Optimal : -Ofast
 */
```

```cpp
#include <algorithm>
#include <cstddef>
#include <cstdio>
#include <cstring>

typedef unsigned long long ULL;

const int N = 7e4 + 5;
int n, m, u;

struct bitset {
  ULL t[N / 64 + 5];

  bitset() {
    memset(t, 0, sizeof(t));
  }
  bitset(const bitset &rhs) {
    memcpy(t, rhs.t, sizeof(t));
  }

  bitset& set(int p) {
    t[p >> 6] |= 1llu << (p & 63);
    return *this;
  }
  bitset& shift() {
    ULL last = 0llu;
    for (int i = 0; i < u; i++) {
      ULL cur = t[i] >> 63;
      (t[i] <<= 1) |= last, last = cur;
    }
    return *this;
  }
  int count() {
    int ret = 0;
    for (int i = 0; i < u; i++)
      ret += __builtin_popcountll(t[i]);
    return ret;
  }

  bitset& operator = (const bitset &rhs) {
    memcpy(t, rhs.t, sizeof(t));
    return *this;
  }
  bitset& operator &= (const bitset &rhs) {
    for (int i = 0; i < u; i++) t[i] &= rhs.t[i];
    return *this;
  }
  bitset& operator |= (const bitset &rhs) {
    for (int i = 0; i < u; i++) t[i] |= rhs.t[i];
    return *this;
  }
  bitset& operator ^= (const bitset &rhs) {
    for (int i = 0; i < u; i++) t[i] ^= rhs.t[i];
    return *this;
  }

  friend bitset operator - (const bitset &lhs, const bitset &rhs) {
    ULL last = 0llu; bitset ret;
```

```
    for (int i = 0; i < u; i++){
      ULL cur = (lhs.t[i] < rhs.t[i] + last);
      ret.t[i] = lhs.t[i] - rhs.t[i] - last;
      last = cur;
    }
    return ret;
  }
} p[N], f, g;

signed main() {
  scanf("%d%d", &n, &m), u = n / 64 + 1;
  for (int i = 1, c; i <= n; i++)
    scanf("%d", &c), p[c].set(i);
  for (int i = 1, c; i <= m; i++) {
    scanf("%d", &c), (g = f) |= p[c];
    f.shift(), f.set(0);
    ((f = g - f) ^= g) &= g;
  }
  printf("%d\n", f.count());
  return 0;
}
```

补充：使用__builtin进行位运算

```
#include<bits/stdc++.h>
#define lb(x) (x&(-x))
using namespace std;

signed main(){
    int a=20,b=65; //a=10100,1000001
    cout<<__builtin_ffs(a)<<" "<<__builtin_ffs(b)<<"\n"; //返回最后一位1是从后向前第几位
    cout<<__builtin_clz(a)<<" "<<__builtin_clz(b)<<"\n"; //返回前导0的个数
    cout<<__builtin_ctz(a)<<" "<<__builtin_ctz(b)<<"\n"; //返回后面的0个数
    cout<<__builtin_popcount(a)<<" "<<__builtin_popcount(b)<<"\n"; //返回二进制中1的个数
    cout<<__builtin_parity(a)<<" "<<__builtin_parity(b)<<"\n"; //返回奇偶校验位(1的个数模2的结果)
}
```

参考资料

https://www.cnblogs.com/magisk/p/8809922.html

https://blog.csdn.net/linkfqy/article/details/75578669

https://blog.csdn.net/aolian4963/article/details/101947150