ArosTemplate

(2013.03.14 updated)

目录

InSkill

Ubuntu 下 CodeBlocks 更改调试终端

在环境设置里进行如下设置：把 Terminal to launch console programs 那个选项改成“gnome-terminal -t $TITLE –x”，原来是“xterm -T $TITLE –e”。

HDU 上的 DFS 爆栈问题的简易解决方法

在文件 gui 头处加上这么一句“#pragma comment(linker, "/STACK:1024000000,1024000000")”后面两个数字随便写，你觉得能过就好，另外不要超了栈内存的真正上限。基于 VC++的编译预处理命令，这个代码必须拿 C++来提交，所以 C++会出现的那种 long long 和__int64 的问题也要注意到。

通过内嵌汇编把堆空间作为栈空间使用_hdu_4118

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100000+5, MAXM = 200000+5;
int T, N, X, Y, Z;
int e, head[MAXN], next[MAXM], v[MAXM];
int cnt[MAXN];
long long w[MAXM], ans;
void addedge(int x, int y, int z)
{
  v[e] = y; w[e] = z;
  next[e] = head[x]; head[x] = e++;
}
void dfs(int u, int fa = 0)
{
  cnt[u] = 1;
  for (int i = head[u]; i != -1; i = next[i]) if (v[i] != fa)
  {
    dfs(v[i], u);
    ans += min(cnt[v[i]], N-cnt[v[i]])*2*w[i];
    cnt[u] += cnt[v[i]];
  }
}

void call_dfs()
{
  const int STACK_SIZE = 1<<23;
  static char stack[STACK_SIZE];
  int bak;
  __asm__ __volatile__
  (
    "movl %%esp, %0\n"
    "movl %1, %%esp\n":
    "=g"(bak):
    "g"(stack+STACK_SIZE-1):
  );

  dfs(1);

  __asm__ __volatile__
  (
    "movl %0, %%esp\n":
    :
    "g"(bak):
  );
}

int main()
{
  scanf("%d", &T);
  for (int cas = 1; cas <= T; cas++)
  {
    e = 0;
    memset(head, -1, sizeof(head));
    scanf("%d", &N);
    for (int i = 1; i < N; i++)
    {
      scanf("%d%d%d", &X, &Y, &Z);
      addedge(X, Y, Z);
      addedge(Y, X, Z);
    }
    ans = 0;
    call_dfs();
    printf("Case #%d: %I64d\n", cas, ans);
  }
  return 0;
}
```

Graph

spfa

```cpp
#include<cstdio>
#include<cstring>
#include<queue>
using namespace std;
const int MAXN = 1000+5, MAXM = 1000+5;
const int INF = 0x3f3f3f3f;
```

```
int n, m, e, s;
int v[MAXM], next[MAXM], head[MAXN];
int w[MAXM], d[MAXN];
int inq_cnt[MAXN]; //存在负权回路时需要
bool inq[MAXN];
queue<int> Q;
void addedge(int x, int y, int z)
{
  v[e] = y; w[e] = z;
  next[e] = head[x]; head[x] = e;
  e++;
}
bool spfa()
{
  for (int i = 1; i <= n; i++)
    d[i] = (i == s ? 0 : INF);
  memset(inq, 0, sizeof(inq));
  memset(inq_cnt, 0, sizeof(inq_cnt));
  while (!Q.empty()) Q.pop();
  Q.push(s);
  inq[s] = 1;
  inq_cnt[s]++;
  while (!Q.empty())
  {
    int u = Q.front(); Q.pop();
    inq[u] = 0;
    for(int e = head[u]; e != -1; e = next[e])
      if(d[v[e]] > d[u]+w[e])
      {
        d[v[e]] = d[u]+w[e];
        if(!inq[v[e]])
        {
          Q.push(v[e]);
          inq[v[e]] = 1;
          inq_cnt[v[e]]++;
          if (inq_cnt[v[e]] > n)
            return 0;
        }
      }
  }
  return 1;
}
int main()
{
//  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
  memset(head, -1, sizeof(head));
  e = 0;

  return 0;
}
```

二维最短路_hdu_4396

```
/*
题意：求至少经过 K 条边，到达终点的最短路（K<=50）。
思路：因为 K<=500，所以每个节点最多扩展成 50 个节点，最后一个节点表示到达该节点时经过的边数（收集到的木材/10）已经满足 K 值对应的要求。然后 spfa，每个节点表示为(编号,经过的边数)。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 5000+5, MAXM = 200000+5, MAXK = 50+5;
const int INF = 0x3f3f3f3f;
int N, M, A, B, C, S, T, K, mk;
int e, head[MAXN], next[MAXM], v[MAXM];
int d[MAXN][MAXK], w[MAXM];
bool inq[MAXN][MAXK];
queue<pair<int, int> > Q;
void addedge(int x, int y, int z)
{
  v[e] = y; w[e] = z;
  next[e] = head[x]; head[x] = e++;
}
void spfa(int s)
{
  for (int i = 1; i <= N; i++)
    for (int j = 1; j <= mk; j++)
      d[i][j] = INF;
  Q.push(make_pair(s, 0));
  while (!Q.empty())
  {
    int u = Q.front().first, k = Q.front().second;
    Q.pop();
    inq[u][k] = 0;
    for (int i = head[u]; i != -1; i = next[i])
    {
      int l = k+(k < mk ? 1 : 0);
      if (d[u][k]+w[i] < d[v[i]][l])
      {
        d[v[i]][l] = d[u][k]+w[i];
        if (!inq[v[i]][l])
        {
          Q.push(make_pair(v[i], l));
          inq[v[i]][l] = 1;
        }
      }
    }
  }
}
void init()
{
  e = 0;
  memset(head, -1, sizeof(head));
}
int main()
{
  while (scanf("%d%d", &N, &M) != EOF)
  {
    init();
    for (int i = 0; i < M; i++)
    {
      scanf("%d%d%d", &A, &B, &C);
      addedge(A, B, C);
      addedge(B, A, C);
    }
    scanf("%d%d%d", &S, &T, &K);
    mk = (K-1)/10+1;
    spfa(S);
    printf("%d\n", d[T][mk] < INF ? d[T][mk] : -1);
  }
  return 0;
}
```

找环_hdu_4337

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
using namespace std;
const int MAXN = 150+5, MAXM = 22500+5, MAXP = 50+5;
int N, M, a, b;
int e, head[MAXN], next[MAXM], v[MAXM];
int mark[MAXN];
vector<int> vec;
void Init()
{
  e = 0;
  memset(head, -1, sizeof(head));
  memset(mark, -1, sizeof(mark));
  vec.clear();
}
void addedge(int x, int y)
{
  v[e] = y;
  next[e] = head[x]; head[x] = e++;
}
bool dfs(int u, int step = 0)
{
  mark[u] = step;
  vec.push_back(u);
  for (int i = head[u]; i != -1; i = next[i])
  {
    if (mark[v[i]] == -1)
    {
      if (dfs(v[i], step+1))
        return 1;
    }
    else if (step-mark[v[i]]+1 == N)
      return 1;
  }
  mark[u] = -1;
  vec.pop_back();
  return 0;
}
int main()
{
  while (scanf("%d%d", &N, &M) != EOF)
  {
    Init();
    for (int i = 0; i < M; i++)
    {
      scanf("%d%d", &a, &b);
      addedge(a, b);
      addedge(b, a);
    }
    if (dfs(1))
    {
      for (int i = 0; i < (int)vec.size(); i++)
      {
        if (i)
          printf(" ");
        printf("%d", vec[i]);
      }
      printf("\n");
    }
    else
      printf("no solution\n");
  }
  return 0;
}
```

最小生成树的最佳替换边_hdu_4126

```
/*
题意：给定一个图 G，有 q 次询问（相互独立），每次询问(u,v,w)，表示将<u,v>这条边的边权更改为 w，求此时的最小生成树的值。
算法：先求得最小生成树，对于每次询问(u,v,w)，分两种情况讨论：
1、若<u,v>是非最小生成树上的边，那么不用考虑，最小生成树仍是原来的值。
2、若<u,v>是最小生成树上的边，那么我们就需要在这条边所导致的两个集合中分别选出一个点 i,j 并且 g[i][j]最小来替代那条被增加的边，那么反过来考虑，对于一条非最小生成树上的边<u,v>，它可以替代哪些边呢？就是 u->x1->x2->...->xk->v 这条路径（因为是树，所以这条路径唯一）上的边。那么现在要求的就是对于每条树上的边<u,v>，得到一个 best[u][v]表示去掉它，最小的替代边的权值。这个可以用 dfs 来做，以每个点为起点做 dfs，遍历整个最小生成树，得到 best[i][j]，这样复杂度就是 O(N^2)的，然后对于每次询问就可以 O(1)回答了。这个 dfs 的写
```

```cpp
法还是有点技巧的，具体就见代码吧。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 3000+5, MAXM = 6000+5;
const int INF = 0x3f3f3f3f;
int N, M, Q, X, Y, C;
int g[MAXN][MAXN], best[MAXN][MAXN], dis[MAXN], pre[MAXN];
int e, head[MAXN], next[MAXM], v[MAXM];
bool vis[MAXN];
void addedge(int x, int y)
{
  v[e] = y;
  next[e] = head[x]; head[x] = e++;
}
void init()
{
  e = 0;
  memset(head, -1, sizeof(head));
  for (int i = 0; i < N; i++)
    for (int j = 0; j < i; j++)
      g[i][j] = g[j][i] = best[i][j] = best[j][i] = INF;
}
int prim()
{
  for (int i = 0; i < N; i++)
    vis[i] = 0, pre[i] = -1, dis[i] = INF;
  int res = 0;
  dis[0] = 0;
  for (int j = 0; j < N; j++)
  {
    int u = -1;
    for (int i = 0; i < N; i++)
      if (!vis[i] && (u == -1 || dis[i] < dis[u]))
        u = i;
    vis[u] = 1;
    res += dis[u];
    if (pre[u] != -1)
    {
      addedge(u, pre[u]);
      addedge(pre[u], u);
    }
    for (int i = 0; i < N; i++)
      if (!vis[i] && g[u][i] < dis[i])
      {
        dis[i] = g[u][i];
        pre[i] = u;
      }
  }
  return res;
}
int dfs(int st, int u, int fa)
{
  int mini = INF;
  for (int i = head[u]; i != -1; i = next[i]) if (v[i] != fa)
  {
    int cur = dfs(st, v[i], u);
    mini = min(mini, cur);
    best[u][v[i]] = best[v[i]][u] = min(best[u][v[i]], cur);
  }
  if (st != fa)
    mini = min(mini, g[st][u]);
  return mini;
}
int main()
{
  while (scanf("%d%d", &N, &M))
  {
    if (!N && !M)
      break;
    init();
    for (int i = 0; i < M; i++)
    {
      scanf("%d%d%d", &X, &Y, &C);
      g[X][Y] = g[Y][X] = C;
    }
    int mst = prim();
    for (int i = 0; i < N; i++)
      dfs(i, i, -1);
    scanf("%d", &Q);
    double ans = 0;
    for (int i = 0; i < Q; i++)
    {
      scanf("%d%d%d", &X, &Y, &C);
      if (pre[X] == Y || pre[Y] == X)
        ans += mst-g[X][Y]+min(C, best[X][Y]);
      else
        ans += mst;
    }
    ans /= Q;
    printf("%.4f\n", ans);
  }
  return 0;
}
```

最小树形图_hdu_4009

```cpp
/*
题意：有 n 个地方需要供水，每个地方都可以选择是自己挖井，还是从别的地方引水，根据方法不同和每个地方的坐标不同，花费也不同，现在给出每个地方的坐标，花费的计算方法，以及每个地方可以给哪些地方供水（即对方可以从这里引水），求给所有地方供水的最小花费。
思路：显然对于每个地方，只有一种供水方式就足够了，这样也能保证花费最小，而每个地方都可以自己挖井，所以是不可能出现无解的情况的，为了方便思考，我们引入一个虚拟点，把所有自己
```

```
挖井的都连到这个点，边权为挖井的花费，而如果 i 能从 j 处引水，则从 j 向 i 连边，边权为引水的
花费，然后对这个有向图，以虚拟点为根，求最小树形图即可（最小树形图即为有向图的最小生成树）。
*/
#include<cstdio>
#include<cstring>
#include<cmath>
#include<algorithm>
using namespace std;
const int MAXN = 1000+5, MAXM = 1001000+5;
const int INF = 0x3f3f3f3f;
int N, X, Y, Z, K, x[MAXN], y[MAXN], z[MAXN];
int e, u[MAXM], v[MAXM], w[MAXM];
int pre[MAXN], id[MAXN], vis[MAXN];
int in[MAXN];
int Directed_MST(int root,int NV,int NE) //number vertices from zero!!!
{
 int res = 0;
 for (;;)
 {
  //1.找最小入边
  for (int i = 0; i < NV; i++)
   in[i] = INF, id[i] = -1, vis[i] = -1;
  for (int i = 0; i < NE; i++)
  {
   int s = u[i], t = v[i];
   if (w[i] < in[t] && s != t)
   {
    pre[t] = s;
    in[t] = w[i];
   }
  }
  for (int i = 0; i < NV; i++)
  {
   if (i == root)
    continue;
   if (in[i] == INF)
    return -1;//除了跟以外有点没有入边,则根无法到达它
  }
  //2.找环
  int cntnode = 0;
  in[root] = 0;
  for (int i = 0; i < NV; i++)
  {//标记每个环
   res += in[i];
   int t = i;
   for (; vis[t] != i && id[t] == -1 && t != root; t = pre[t])
    vis[t] = i;
   if (t != root && id[t] == -1)
   {
    for (int s = pre[t] ; s != t ; s = pre[s])
     id[s] = cntnode;
    id[t] = cntnode++;
   }
  }
  if (!cntnode)
   break;//无环
  for (int i = 0; i < NV; i++)
   if (id[i] == -1)
    id[i] = cntnode++;
  //3.缩点,重新标记
  for (int i = 0; i < NE; i++)
  {
   int t = v[i];
   u[i] = id[u[i]];
   v[i] = id[v[i]];
   if (u[i] != v[i])
    w[i] -= in[t];
  }
  NV = cntnode;
  root = id[root];
 }
 return res;
}
void addedge(int x, int y, int z)
{
 u[e] = x; v[e] = y; w[e] = z;
 e++;
}
int main()
{
 while (scanf("%d%d%d%d", &N, &X, &Y, &Z))
 {
  if (!N && !X && !Y && !Z)
   break;
  e = 0;
  int root = 0;
  for (int i = 1; i <= N; i++)
  {
   scanf("%d%d%d", &x[i], &y[i], &z[i]);
   addedge(root, i, z[i]*X);
  }
  for (int i = 1; i <= N; i++)
  {
   scanf("%d", &K);
   for (int k = 1, j; k <= K; k++)
   {
    scanf("%d", &j);
    if (z[i] < z[j])
     addedge(i, j, (abs(x[i]-x[j])+abs(y[i]-y[j])+abs(z[i]-z[j]))*Y+Z);
    else
     addedge(i, j, (abs(x[i]-x[j])+abs(y[i]-y[j])+abs(z[i]-z[j]))*Y);
   }
  }
  printf("%d\n", Directed_MST(root, N+1, e));
  }
  return 0;
}
```

Network

最大流 ISAP_hdu_3879

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 55000+5, MAXM = 155000*2+5;
const int INF = 0x3f3f3f3f;
int N, M;
int n, s, t;
int e, v[MAXM], next[MAXM], head[MAXN];
int cap[MAXM];
int h[MAXN], gap[MAXN];
void init()
{
  e = 0;
  memset(head, -1, sizeof(head));
  memset(gap, 0, sizeof(gap));
  memset(h, 0, sizeof(h));
}
void addedge(int x, int y, int c)
{
  v[e] = y; cap[e] = c;
  next[e] = head[x]; head[x] = e++;
  v[e] = x; cap[e] = 0;
  next[e] = head[y]; head[y] = e++;
}
int sap(int u, int f)
{
  if (u == t)
   return f;
  int minh = n-1, rf = f;
  for (int i = head[u]; i != -1; i = next[i]) if (cap[i])
  {
    if (h[v[i]]+1 == h[u])
    {
      int cf = sap(v[i], min(cap[i], rf));
      cap[i] -= cf;
      cap[i^1] += cf;
      rf -= cf;
      if (h[s] >= n)
        return f-rf;
      if (!rf)
        break;
    }
    minh = min(minh, h[v[i]]);
  }
  if (rf == f)
  {
    gap[h[u]]--;
    if (!gap[h[u]])
      h[s] = n;
    h[u] = minh+1;
    gap[h[u]]++;
  }
  return f-rf;
}
int maxflow()
{
  int res = 0;
  gap[0] = n;
  while (h[s] < n)
   res += sap(s, INF);
  return res;
}
int main()
{
  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
  while (scanf("%d%d", &N, &M) != EOF)
  {
    init();
    n = N+M+2; s = N+M+1; t = s+1;
    for (int i = 1; i <= N; i++)
    {
      int P;
      scanf("%d", &P);
      addedge(i, t, P);
    }
    int tp = 0;
    for (int i = 1; i <= M; i++)
    {
      int x, y, z;
      scanf("%d%d%d", &x, &y, &z);
      tp += z;
      addedge(s, N+i, z);
      addedge(N+i, x, INF);
      addedge(N+i, y, INF);
    }
    int f = maxflow();
    printf("%d\n", tp-f);
  }
  return 0;
}
```

最大流-邻接表

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 1000+5, MAXM = 1000+5;
const int INF = 0x3f3f3f3f;
int e, s, t, n;
int v[MAXM], next[MAXM], head[MAXN];
int cap[MAXM], a[MAXN], f;
int pv[MAXN], pe[MAXN];
queue<int> Q;
void addedge(int u_, int v_, int c_)
{
  v[e] = v_; cap[e] = c_;
  next[e] = head[u_]; head[u_] = e;
  e++;
  v[e] = u_; cap[e] = 0;
  next[e] = head[v_]; head[v_] = e;
  e++;
}
void maxflow()
{
  f = 0;
  for (;;)
  {
    memset(a, 0, sizeof(a));
    a[s] = INF;
    Q.push(s);
    while (!Q.empty())
    {
      int u = Q.front(); Q.pop();
      for (int e = head[u]; e != -1; e = next[e])
        if(!a[v[e]] && cap[e])
        {
          Q.push(v[e]);
          a[v[e]] = min(a[u], cap[e]);
          pv[v[e]] = u; pe[v[e]] = e;
        }
    }
    if (!a[t]) break;
    for (int v = t; v != s; v = pv[v])
    {
      cap[pe[v]] -= a[t];
      cap[pe[v]^1] += a[t];
    }
    f += a[t];
  }
}
int main()
{
//  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
  memset(cap, 0, sizeof(cap));
  memset(head, -1, sizeof(head));
  e = 0;

  return 0;
}
```

最大流-邻接矩阵

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 1000+5;
const int INF = 0x3f3f3f3f;
int s, t, n;
int p[MAXN];
int cap[MAXN][MAXN], flow[MAXN][MAXN], a[MAXN], f;
queue<int> Q;
void addedge(int u_, int v_, int c_)
{
  cap[u_][v_] = c_;
}
void maxflow()
{
  f = 0;
  memset(flow, 0, sizeof(flow));
  for(;;)
  {
    memset(a, 0, sizeof(a));
    a[s] = INF;
    Q.push(s);
    while(!Q.empty())
    {
      int u = Q.front(); Q.pop();
      for(int v = 1; v <= n; v++)
        if(!a[v] && cap[u][v] > flow[u][v])
        {
          p[v] = u; Q.push(v);
          a[v] = min(a[u], cap[u][v]-flow[u][v]);
        }
    }
    if(a[t] == 0) break;
    for(int v = t; v != s; v = p[v])
    {
      flow[p[v]][v] += a[t];
      flow[v][p[v]] -= a[t];
    }
    f += a[t];
  }
}
int main()
{
//  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
  memset(cap, 0, sizeof(cap));

  return 0;
}
```

最小费用最大流-邻接表

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 1000+5, MAXM = 1000+5;
const int INF = 0x3f3f3f3f;
int e, s, t, n;
int v[MAXM], next[MAXM], head[MAXN];
int cap[MAXM], f;
int cost[MAXM], d[MAXN], c;
int pv[MAXN], pe[MAXN];
bool inq[MAXN];
queue<int> Q;
void addedge(int u_, int v_, int c_, int w_)
{
  v[e] = v_; cap[e] = c_; cost[e] = w_;
  next[e] = head[u_]; head[u_] = e;
  e++;
  v[e] = u_; cap[e] = 0; cost[e] = -w_;
  next[e] = head[v_]; head[v_] = e;
  e++;
}
void mincostflow()
{
  f = 0; c = 0;
  for (;;)
  {
    memset(inq, 0, sizeof(inq));
    for (int i = 1; i <= n; i++)
      d[i] = (i == s ? 0 : INF);
    Q.push(s); inq[s] = 1;
    while (!Q.empty())
    {
      int u = Q.front(); Q.pop();
      inq[u] = 0;
      for (int e = head[u]; e != -1; e = next[e])
        if(cap[e] && d[v[e]] > d[u]+cost[e])
        {
          d[v[e]] = d[u]+cost[e];
          if (!inq[v[e]])
            Q.push(v[e]), inq[v[e]] = 1;
          pv[v[e]] = u; pe[v[e]] = e;
        }
    }
    if (d[t] == INF) break;
    int a = INF;
    for (int v = t; v != s; v = pv[v])
      a = min(a, cap[pe[v]]);
    for (int v = t; v != s; v = pv[v])
    {
      cap[pe[v]] -= a;
      cap[pe[v]^1] += a;
    }
    f += a;
    c += d[t]*a;
  }
}
int main()
{
//  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
  memset(cap, 0, sizeof(cap));
  memset(cost, 0, sizeof(cost));
  memset(head, -1, sizeof(head));
  e = 0;

  return 0;
}
```

最小费用最大流-邻接矩阵

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 1000+5;
const int INF = 0x3f3f3f3f;
int s, t, n;
int cost[MAXN][MAXN], d[MAXN], c;
int cap[MAXN][MAXN], flow[MAXN][MAXN], f;
int p[MAXN];
bool inq[MAXN];
queue<int> Q;
void addedge(int u_, int v_, int c_, int w_)
{
  cap[u_][v_] = c_;
  cost[u_][v_] = w_; cost[v_][u_] = -w_;
}
void mincostflow()
```

```
{
  f = 0, c = 0;
  memset(flow, 0, sizeof(flow));
  for(;;)
  {
    for(int i = 1; i <= n; i++)
      d[i] = (i == s ? 0 : INF);
    memset(inq, 0, sizeof(inq));
    Q.push(s); inq[s] = 1;
    while(!Q.empty())
    {
      int u = Q.front(); Q.pop();
      inq[u] = 0;
      for(int v = 1; v <= n; v++)
        if(cap[u][v] > flow[u][v] && d[v] > d[u]+cost[u][v])
        {
          d[v] = d[u]+cost[u][v];
          if(!inq[v])
            Q.push(v), inq[v] = 1;
          p[v] = u;
        }
    }
    if (d[t] == INF) break;
    int a = INF;
    for(int v = t; v != s; v = p[v])
      a = min(a, cap[p[v]][v]-flow[p[v]][v]);
    for(int v = t; v != s; v = p[v])
    {
      flow[p[v]][v] += a;
      flow[v][p[v]] -= a;
    }
    c += d[t]*a;
    f += a;
  }
}
int main()
{
//  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
  memset(cap, 0, sizeof(cap));
  memset(cost, 0, sizeof(cost));

  return 0;
}
```

Number

组合数 C(N, R)

```
int com(int n, int r)
{// return C(n, r)
  if (n-r > r) r = n-r; // C(n, r) = C(n, n-r)
  int s = 1;
  for (int i = 0, j = 1; i < r; i++)
  {
    s *= (n-i);
    for(; j <= r && s%j == 0; j++)
      s /= j;
  }
  return s;
}
```

Structure

AC 自动机

AC 自动机_hdu_2222

```
/*
网络流上流传最广的 AC 自动机模板题，问你目标串中出现了几个模式串
如果一个结点是单词末尾的话 out 标记为 true,在 search 的时候对于每个结点都向 fail 指针找，找到 out 为 true 的就将其标记为 false,且 ans+=out
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 1000000+5, MAXM = 50+5;
const int MAX_NODE = 500000+5, MAX_CHD = 26;
int T, N;
int chd[MAX_NODE][MAX_CHD], fail[MAX_NODE], out[MAX_NODE];
int ID[1<<8], nv;
char key[MAXM], des[MAXN];
queue<int> Q;
namespace AC_Automaton
{
  void Initialize()
  {
    fail[0] = 0;
    for (int i = 0; i < MAX_CHD; i++)
      ID[i+'a'] = i;
  }
  void Reset()
  {
    memset(chd[0], 0, sizeof(chd[0]));
    nv = 1;
  }
  void Insert(char *pat)
  {
    int u = 0;
    for (int i = 0; pat[i]; i++)
    {
      int c = ID[pat[i]];
      if (!chd[u][c])
      {
        memset(chd[nv], 0, sizeof(chd[nv]));
        out[nv] = 0;
        chd[u][c] = nv++;
      }
      u = chd[u][c];
    }
    out[u]++;
  }
  void Construct()
  {
    for (int i = 0; i < MAX_CHD; i++)
      if (chd[0][i])
      {
        fail[chd[0][i]] = 0;
        Q.push(chd[0][i]);
      }
    while (!Q.empty())
    {
      int u = Q.front(); Q.pop();
      for (int i = 0; i < MAX_CHD; i++)
      {
        int v = chd[u][i];
        if (v)
        {
          Q.push(v);
          fail[v] = chd[fail[u]][i];
        }
        else
          chd[u][i] = chd[fail[u]][i];
      }
    }
  }
}
int main()
{
  AC_Automaton::Initialize();
  scanf("%d", &T);
  while (T--)
  {
    scanf("%d", &N);
    AC_Automaton::Reset();
    for (int i = 0; i < N; i++)
    {
      scanf("%s", key);
      AC_Automaton::Insert(key);
    }
    AC_Automaton::Construct();
    scanf("%s", des);
    int ans = 0;
    for (int i = 0, u = 0; des[i]; i++)
    {
      u = chd[u][ID[des[i]]];
      for (int t = u; t; )
      {
        ans += out[t];
        out[t] = 0;
        t = fail[t];
      }
    }
    printf("%d\n", ans);
  }
  return 0;
}
```

AC 自动机+DP_hdu_2825

```
/*
求长度为 n 的字符串中包含至少 k 个给出的关键字的字符串的个数，结果模 MOD。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;

//MAX_NODE = StringNumber*StringLength
const int MAX_NODE = 100+5;
//字符集大小,一般字符形式的题 26 个
const int MAX_CHD = 26;
//每个节点的儿子,即当前节点的状态转移
int chd[MAX_NODE][MAX_CHD];
//记录题目给的关键数据(点的权值)
int out[MAX_NODE];
//传说中的 fail 指针
int fail[MAX_NODE];
//字母对应的 ID
int ID[1<<8];
//已使用节点个数
int nv;
```

```cpp
//队列,用于广度优先计算 fail 指针
queue<int> Q;

//特定题目需要
const int MAXN = 25+5;
const int MOD = 20090717;
int N, M, K, d[2][MAX_NODE][1<<10];

namespace AC_Automaton
{
  //初始化,计算字母对应的儿子 ID,如:'a'->0 ... 'z'->25
  void Initialize()
  {
    fail[0] = 0;
    for (int i = 0; i < MAX_CHD; i++)
      ID[i+'a'] = i;
  }
  //重新建树需先 Reset
  void Reset()
  {
    memset(chd[0], 0, sizeof(chd[0]));
    nv = 1;
  }
  //将权值为 key 的字符串 a 插入到 trie 中
  void Insert(char *pat, int key)
  {
    int u = 0;
    for (int i = 0; pat[i]; i++)
    {
      int c = ID[pat[i]];
      if (!chd[u][c])
      {
        memset(chd[nv], 0, sizeof(chd[nv]));
        out[nv] = 0;
        chd[u][c] = nv++;
      }
      u = chd[u][c];
    }
    out[u] = key;
  }
  //建立 AC 自动机,确定每个节点的权值以及状态转移
  void Construct()
  {
    for (int i = 0; i < MAX_CHD; i++)
      if (chd[0][i])
      {
        fail[chd[0][i]] = 0;
        Q.push(chd[0][i]);
      }
    while (!Q.empty())
    {
      int u = Q.front(); Q.pop();
      for (int i = 0; i < MAX_CHD; i++)
      {
        int &v = chd[u][i];
        if (v)
        {
          Q.push(v);
          fail[v] = chd[fail[u]][i];
          //以下一行代码要根据题目所给 out 的含义来写
          out[v] |= out[fail[v]];
        }
        else
          v = chd[fail[u]][i];
      }
    }
  }
}

//解题
int solve()
{
  int tot = (1<<M)-1, ans = 0, s = 0, t = 1;
  memset(d[t], 0, sizeof(d[t]));
  d[t][0][0] = 1;
  for (int i = 0; i < N; i++)
  {
    swap(s, t);
    memset(d[t], 0, sizeof(d[t]));
    for (int u = 0; u < nv; u++)
      for (int a = 0; a <= tot; a++) if (d[s][u][a])
        for (int k = 0; k < MAX_CHD; k++)
        {
          int v = chd[u][k], b = (a|out[v]);
          d[t][v][b] = (d[t][v][b]+d[s][u][a])%MOD;
        }
  }
  for (int a = 0; a <= tot; a++)
  {
    int cnt = 0;
    for (int i = 0; i < M; i++)
      if (a&(1<<i))
        cnt++;
    if (cnt >= K)
    {
      for (int u = 0; u < nv; u++)
        ans = (ans+d[t][u][a])%MOD;
    }
  }
  return ans;
}

int main()
{
  AC_Automaton::Initialize();
  while (scanf("%d%d%d", &N, &M, &K) != EOF)
```

```
 {
  if (!N && !M && !K)
   break;
  AC_Automaton::Reset();
  for (int i = 0; i < M; i++)
  {
   char temp[11];
   scanf("%s", temp);
   AC_Automaton::Insert(temp, 1<<i);
  }
  AC_Automaton::Construct();
  printf("%d\n", solve());
 }
 return 0;
}
```

AC 自动机+概率 DP_hdu_3689

```
/*
字符集中有一些字符，给出每个字符的出现概率（它们的和保证为 1），再给出一个串 S，问任给一个长度为 N 的字符串 A（只能包含字符集中的字符），使得 S 是 A 的子串的概率。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 1000+5, MAXM = 10+5;
const int INF = 0x3f3f3f3f;
const int MAX_NODE = MAXN, MAX_CHD = 26;
int N, M;
int chd[MAX_NODE][MAX_CHD], fail[MAX_NODE], out[MAX_NODE];
int ID[1<<8], nv;
double P[MAX_CHD], d[MAXN][MAX_NODE];
char ch[5], word[MAXM];
queue<int> Q;
namespace AC_Automaton
{
 void Initialize()
 {
  fail[0] = 0;
  for (int i = 0; i < MAX_CHD; i++)
   ID[i+'a'] = i;
 }
 void Reset()
 {
  memset(chd[0], 0, sizeof(chd[0]));
  nv = 1;
 }
 void Insert(char *pat)
 {
  int u = 0;
  for (int i = 0; pat[i]; i++)
  {
   int c = ID[pat[i]];
   if (!chd[u][c])
   {
    memset(chd[nv], 0, sizeof(chd[nv]));
    out[nv] = 0;
    chd[u][c] = nv++;
   }
   u = chd[u][c];
  }
  out[u]++;
 }
 void Construct()
 {
  for (int i = 0; i < MAX_CHD; i++)
   if (chd[0][i])
   {
    fail[chd[0][i]] = 0;
    Q.push(chd[0][i]);
   }
  while (!Q.empty())
  {
   int u = Q.front(); Q.pop();
   for (int i = 0; i < MAX_CHD; i++)
   {
    int &v = chd[u][i];
    if (v)
    {
     Q.push(v);
     fail[v] = chd[fail[u]][i];
    }
    else
     v = chd[fail[u]][i];
   }
  }
 }
}
int main()
{
 AC_Automaton::Initialize();
 while (scanf("%d%d", &N, &M))
 {
  if (!N && !M)
   break;
  memset(P, 0, sizeof(P));
  memset(d, 0, sizeof(d));
  AC_Automaton::Reset();
  for (int i = 0; i < N; i++)
  {
   scanf("%s", ch);
   scanf("%lf", &P[ID[ch[0]]]);
  }
  scanf("%s", word);
  AC_Automaton::Insert(word);
  AC_Automaton::Construct();
  d[0][0] = 1;
  for (int i = 0; i < M; i++)
   for (int u = 0; u < nv; u++) if (d[i][u] && !out[u])
    for (int j = 0; j < MAX_CHD; j++)
     d[i+1][chd[u][j]] += d[i][u]*P[j];
  int len = strlen(word);
  double ans = 0;
  for (int i = len; i <= M; i++)
   ans += d[i][len];
  printf("%.2lf%s\n", ans*100, "\%");
 }
 return 0;
}
```

AC 自动机+矩阵_poj_2778

```
/*
问你长度为 N 的串中不包含模式串的串有几个
n 属于 1 ~ 2000000000 看到这个数据范围我们就应该敏感的想到这是矩阵~
最多 100 个结点，先建好所有结点(不包括模式串结尾的和 fail 指向结尾的结点,所以其实最多只有 90 个有效结点)之间的转化关系,然后二分矩阵乘法,复杂度 O(100^3*log(2000000000))
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXM = 10+5;
const int MAX_NODE = 100+5, MAX_CHD = 4;
const long long MOD = 100000;
typedef long long MAT[MAX_NODE][MAX_NODE];
MAT g, G;
int M, N;
int chd[MAX_NODE][MAX_CHD], fail[MAX_NODE], ID[1<<8], nv;
bool out[MAX_NODE];
char DNA[MAXM];
queue<int> Q;
namespace AC_Automaton
{
 void Initialize()
 {
  fail[0] = 0;
  ID['A'] = 0; ID['C'] = 1; ID['T'] = 2; ID['G'] = 3;
 }
 void Reset()
 {
  memset(chd[0], 0, sizeof(chd[0]));
  nv = 1;
 }
 void Insert(char *pat)
 {
  int u = 0;
  for (int i = 0; pat[i]; i++)
  {
   int c = ID[pat[i]];
   if (!chd[u][c])
   {
    memset(chd[nv], 0, sizeof(chd[nv]));
    out[nv] = 0;
    chd[u][c] = nv++;
   }
   u = chd[u][c];
  }
  out[u] = 1;
 }
 void Construct()
 {
  for (int i = 0; i < MAX_CHD; i++)
   if (chd[0][i])
   {
    fail[chd[0][i]] = 0;
    Q.push(chd[0][i]);
   }
  while (!Q.empty())
  {
   int u = Q.front(); Q.pop();
   for (int i = 0; i < MAX_CHD; i++)
   {
    int &v = chd[u][i];
    if (v)
    {
     Q.push(v);
     fail[v] = chd[fail[u]][i];
     out[v] |= out[fail[v]];
    }
    else
     v = chd[fail[u]][i];
   }
  }
 }
}
namespace Matrix
{
 void Copy(int size, MAT x, MAT y)
 {
  for (int i = 0; i < size; i++)
   for (int j = 0; j < size; j++)
    y[i][j] = x[i][j];
 }
 void Mutiply(int size, MAT x, MAT y, MAT z)
 {
  MAT tx, ty;
```

```
    Copy(size, x, tx);
    Copy(size, y, ty);
    for (int i = 0; i < size; i++)
      for (int j = 0; j < size; j++)
      {
        z[i][j] = 0;
        for (int k = 0; k < size; ++k)
          z[i][j] = (z[i][j]+tx[i][k]*ty[k][j])%MOD;
      }
  }
  void Power(int size, MAT x, int n, MAT y)
  {
    MAT tx, r;
    Copy(size, x, tx);
    for (int i = 0; i < size; i++)
      for (int j = 0; j < size; j++)
        r[i][j] = (i == j ? 1 : 0);
    while (n)
    {
      if (n&1)
        Mutiply(size, r, tx, r);
      n >>= 1;
      if (!n)
        break;
      Mutiply(size, tx, tx, tx);
    }
    Copy(size, r, y);
  }
}
int main()
{
  AC_Automaton::Initialize();
  memset(g, 0, sizeof(g));
  AC_Automaton::Reset();
  scanf("%d%d", &M, &N);
  for (int i = 0; i < M; i++)
  {
    scanf("%s", DNA);
    AC_Automaton::Insert(DNA);
  }
  AC_Automaton::Construct();
  for (int u = 0; u < nv; u++) if (!out[u])
    for (int k = 0; k < MAX_CHD; k++) if (!out[chd[u][k]])
      g[u][chd[u][k]]++;
  Matrix::Power(nv, g, N, G);
  long long ans = 0;
  for (int i = 0; i < nv; i++)
    ans = (ans+G[0][i])%MOD;
  printf("%lld\n", ans);
  return 0;
}
```

DP

离散 DP_hdu_4028

```
/*
题意：给你 n 个钟的指针，第 i 个指针转一圈的时间是 i 单位，问你从 n 个钟任选一些指针使得，全部指针第一次回到原来的位置是经过的时间大于等于 m，求又多少种选法。
思路：显然时间是你选的指针的最小公倍数，但是好大，dp 无从下手。看完神牛的题解才知道有一种 dp 叫做离散 dp，就是直接保存有用的状态就好了，其他的不用，这样空间就可以满足了，因为其实状态数很少。状态设定很简单：dp[i][j]：i 表示以 i 指针结尾，最小公倍数（lcm）为 j 的方案数。转移也很简单就是 dp[i][j]=dp[i][j]+dp[i-1][j];离散用了 map，STL 太强了，只能这么感慨，map 要注意 lcm 的转移；还有初始状态为 dp[i][i]=1;要在更新这个状态的时候加进去；
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<map>
using namespace std;
const int MAX = 40, MAXN = MAX+5;
const int INF = 0x3f3f3f3f;
int T, N;
long long M;
struct cmp
{
  bool operator()(const long long a, const long long b)
  {
    return a > b;
  }
};
map<long long, long long, cmp> d[MAXN];
long long gcd(long long x, long long y)
{
  return !y ? x : gcd(y, x%y);
}
long long lcm(long long x, long long y)
{
  return x/gcd(x, y)*y;
}
int main()
{
  scanf("%d", &T);
  for (int i = 1; i <= MAX; i++)
  {
    d[i] = d[i-1];
    d[i][i]++;
    map<long long, long long, cmp>::iterator p = d[i-1].begin();
    for (; p != d[i-1].end(); p++)
      d[i][lcm(p->first, i)] += p->second;
  }
  for (int cas = 1; cas <= T; cas++)
  {
    scanf("%d%I64d", &N, &M);
    long long ans = 0;
    map<long long, long long, cmp>::iterator p = d[N].begin();
    for (; p != d[N].end() && p->first >= M; p++)
      ans += p->second;
    printf("Case #%d: %I64d\n", cas, ans);
  }
  return 0;
}
```

区间 DP_hdu_4293_1

```
/*
题意：每个区间有权值，给若干区间，求最大收益。
思路：d[i]表示长度为i且包含以I结尾的区间时最大的人数。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 500+5;
const int INF = 0x3f3f3f3f;
int N, a, b, A[MAXN], B[MAXN], r[MAXN];
int mp[MAXN][MAXN], num[MAXN], d[MAXN];
bool cmp(const int a, const int b)
{
  return B[a] < B[b];
}
int main()
{
  while (scanf("%d", &N) != EOF)
  {
    memset(mp, 0, sizeof(mp));
    memset(num, 0, sizeof(num));
    memset(d, 0, sizeof(d));
    int n = 0, ans = 0;
    for (int i = 1; i <= N; i++)
    {
      scanf("%d%d", &a, &b);
      if (a+b >= N)
        continue;
      int &m = mp[a+1][N-b];
      if (!m)
      {
        m = ++n;
        A[n] = a+1;
        B[n] = N-b;
        r[n] = n;
      }
      num[m] = min(num[m]+1, N-a-b);
    }
    sort(r+1, r+1+n, cmp);
    for (int i = 1; i <= n; i++)
```

```
      for (int j = 0; j < A[r[i]]; j++)
        d[B[r[i]]] = max(d[B[r[i]]], d[j]+num[r[i]]);
    for (int i = 1; i <= N; i++)
      ans = max(ans, d[i]);
    printf("%d\n", ans);
  }
  return 0;
}
```

树形背包 DP_hdu_4276

```
/*
题意：一个有N个节点的树形的地图，知道了每条变经过所需要的时间，现在给出时间T，问能不能在T时间内从1号节点到N节点。每个节点都有相对应的价值，而且每个价值只能被取一次，问如果可以从1号节点走到n号节点的话，最多可以取到的最大价值为多少。
分析：先求出从1号节点到n号节点的最短路，如果花费大于时间T，则直接输出不符合，将最短路上的权值全部赋值为0，在总时间T上减去最短路的长度，表示最短路已经走过，对其它点进行树形背包求解，需要注意的是如果不是最短路上的边都要走两次，即走过去还要再走回来，状态转移方程：dp[i][j]=max(dp[i][j],dp[i][k]+dp[i][j-2*val-k])
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100+5, MAXM = 500+5;
int N, T, a, b, t, A[MAXN];
int e, head[MAXN], next[MAXM], v[MAXM], w[MAXM];
int fa[MAXN], d[MAXN][MAXM];
void addedge(int x, int y, int z)
{
  v[e] = y; w[e] = z;
  next[e] = head[x]; head[x] = e++;
}
void mark(int u)
{
  for (int i = head[u]; i != -1; i = next[i]) if (v[i] != fa[u])
    {
      fa[v[i]] = u;
      mark(v[i]);
    }
}
void dfs(int u, int C)
{
  fill(d[u], d[u]+1+C, A[u]);
  for (int i = head[u]; i != -1; i = next[i]) if (v[i] != fa[u])
  {
    int cost = w[i]*2;
    if (cost <= C)
    {
      dfs(v[i], C-cost);
      for (int j = C; j >= 0; j--)
        for (int k = 0; k <= j-cost; k++)
        d[u][j] = max(d[u][j], d[u][j-k-cost]+d[v[i]][k]);
    }
  }
}
int main()
{
  while (scanf("%d%d", &N, &T) != EOF)
  {
    e = 0;
    memset(head, -1, sizeof(head));
    for (int i = 1; i < N; i++)
    {
      scanf("%d%d%d", &a, &b, &t);
      addedge(a, b, t);
      addedge(b, a, t);
    }
    for (int i = 1; i <= N; i++)
      scanf("%d", &A[i]);
    int ans = 0;
    mark(1);
    for (int u = N; ; )
    {
      ans += A[u];
      A[u] = 0;
      if (u == 1)
        break;
      for (int i = head[u]; i != -1; i = next[i]) if (v[i] == fa[u])
      {
        u = v[i];
        T -= w[i];
        w[i] = 0;
        w[i^1] = 0;
        break;
      }
    }
    if (T < 0)
      printf("Human beings die in pursuit of wealth, and birds die in pursuit of food!\n");
    else
    {
      dfs(1, T);
      ans += d[1][T];
      printf("%d\n", ans);
    }
  }
  return 0;
}
```

KMP

扩展 KMP_hdu_4300

```
/*
这道题问的就是将 1 个串如何变为 stringA+stringB 的形式，使得 stringA 是 stringB 经过映射得到相同的串。映射那步其实没有什么价值，假设 str 为原串 s 经过映射后得到的串，我们可以以 str 为模式串，以 s 为原串做一次扩展 KMP，得到 extend 数组，extend[i]表示原串以第 i 开始与模式串的前缀的最长匹配。经过 O(n)的枚举，我们可以得到,若 extend[i]+i=len 且 i>=extend[i]时，表示 stringB 即为该点之前的串，stringA 即为该点之前的 str 串，最后输出即可。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100000+5, MAXM = 50+5;
const int INF = 0x3f3f3f3f;
int T, extend[MAXN], next[MAXN];
char S[MAXM], tex1[MAXN], tex2[MAXN], match[1<<8];
void get_next(char *pat)
{
  int len2 = strlen(pat), k = 0;
  next[0] = len2;
  while (k+1 < len2 && pat[k] == pat[k+1])
    k++;
  next[1] = k;
  for(int id = 1, i = 2; i < len2; i++)
  {
    int u = i-id;
    if (next[u]+i >= next[id]+id)
    {
      int j = next[id]+id-i;
      if (j < 0)
        j = 0;
      while (j+i < len2 && pat[j] == pat[j+i])
        j++;
      next[i] = j;
      id = i;
    }
    else
      next[i] = next[u];
  }
}
void ext_kmp(char *str, char *pat)
{
  get_next(pat);
  int len1 = strlen(str), len2 = strlen(pat), k = 0;
  while (k < len1 && k < len2 && str[k] == pat[k])
    k++;
  extend[0] = k;
  for (int id = 0, i = 1; i < len1; i++)
  {
    int u = i-id;
    if (i+next[u] < extend[id]+id)
      extend[i] = next[u];
    else
    {
      int j = extend[id]+id-i;
      if (j < 0)
        j = 0;
      while (j+i < len1 && str[j+i] == pat[j])
        j++;
      extend[i] = j;
      id = i;
    }
  }
}
int main()
{
  scanf("%d", &T);
  while (T--)
  {
    scanf("%s%s", S, tex1);
    int lenS = strlen(S);
    for (int i = 0; i < lenS; i++)
      match[(int)S[i]] = 'a'+i;
    int len = strlen(tex1);
    for (int i = 0; i < len; i++)
      tex2[i] = match[(int)tex1[i]];
    tex2[len] = 0;
    ext_kmp(tex1, tex2);
    for (int i = 0; i <= len; i++)
    {
      if ((i+extend[i] == len && i*2 >= len) || i == len)
      {
        for (int j = 0; j < i; j++)
          printf("%c", tex1[j]);
        for (int j = 0; j < i; j++)
          printf("%c", tex2[j]);
        printf("\n");
        break;
      }
    }
  }
  return 0;
}
```

扩展 KMP_hdu_4333

```
/*
扩展 KMP 能求出一个串所有后缀串(即 s[i...len])和模式串的最长公共前缀。于是只要将这个串复制一遍，求出该串每个后缀与其本身的最长公共前缀即可，当公共前缀>=len 时，显然相等，否则只要比较下一位就能确定这个串与原串的大小关系。
至于重复串的问题，只有当这个串有循环节的时候才会产生重复串，用 KMP 的 next 数组求出最小循环节。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100000+5, MAXM = 200000+5;
int T;
int extend[MAXM], next[MAXN], fail[MAXN];
char a[MAXN], aa[MAXM];
void get_next(char *pat)
{
  next[0] = strlen(pat);
  int k = 0;
  while (pat[k+1] && pat[k] == pat[k+1])
    k++;
  next[1] = k;
  for(int id = 1, i = 2; pat[i]; i++)
  {
    int u = i-id;
    if (next[u]+i >= next[id]+id)
    {
      int j = next[id]+id-i;
      if (j < 0)
        j = 0;
      while (pat[j+i] && pat[j] == pat[j+i])
        j++;
      next[i] = j;
      id = i;
    }
    else
      next[i] = next[u];
  }
}
void ext_kmp(char *str, char *pat)
{
  get_next(pat);
  int k = 0;
  while (str[k] && pat[k] && str[k] == pat[k])
    k++;
  extend[0] = k;
  for (int id = 0, i = 1; str[i]; i++)
  {
    int u = i-id;
    if (i+next[u] < extend[id]+id)
      extend[i] = next[u];
    else
    {
      int j = extend[id]+id-i;
      if (j < 0)
        j = 0;
      while (str[j+i] && str[j+i] == pat[j])
      j++;
     extend[i] = j;
     id = i;
    }
  }
}
void get_fail(char *pat)
{
  fail[0] = -1;
  for (int i = 1, j = -1; pat[i]; i++)
  {
    while (j != -1 && pat[j+1] != pat[i])
      j = fail[j];
    if (pat[j+1] == pat[i])
      j++;
    fail[i] = j;
  }
}
int main()
{
  scanf("%d", &T);
  for (int cas = 1; cas <= T; cas++)
  {
    scanf("%s", a);
    int len = strlen(a);
    strcpy(aa, a);
    strcpy(aa+len, a);
    ext_kmp(aa, a);
    get_fail(a);
    int cir = len-fail[len-1]-1, cnt = 0;
    //求出循环节长度 cir，原串循环不一定完整；
    if (len%cir)
      cir = len;
    for (int i = 0; i < cir; i++)
      if (extend[i] < len && aa[i+extend[i]] < a[extend[i]])
        cnt++;
    printf("Case %d: %d %d %d\n", cas, cnt, 1, cir-cnt-1);
  }
  return 0;
}
```

大数

bign-bint

```
//比较高效的大数
#include<cstdio>
#include<cstring>
using namespace std;
const int base = 10000; // (base^2) fit into int
const int width = 4; // width = log base
const int maxn = 1000; // n*width: 可表示的最大位数
struct bint
{
  int len, s[maxn];
  bint (int r = 0)
  { // r 应该是字符串！
    for (len = 0; r > 0; r /= base)
      s[len++] = r%base;
  }
  bint &operator = (const bint &r)
  {
    memcpy(this, &r, (r.len+1)*sizeof(int));// !
    return *this;
  }
};
bool operator < (const bint &a, const bint &b)
{
  int i;
  if (a.len != b.len) return a.len < b.len;
  for (i = a.len-1; i >= 0 && a.s[i] == b.s[i]; i--);
  return i < 0 ? 0 : a.s[i] < b.s[i];
}
bool operator <= (const bint &a, const bint &b)
{
  return !(b < a);
}
bint operator + (const bint &a, const bint &b)
{
  bint res; int i, cy = 0;
  for (i = 0; i < a.len || i < b.len || cy > 0; i++)
  {
    if (i < a.len)
      cy += a.s[i];
    if (i < b.len)
      cy += b.s[i];
    res.s[i] = cy%base; cy /= base;
  }
  res.len = i;
  return res;
}
bint operator - (const bint &a, const bint &b)
{
  bint res; int i, cy = 0;
  for (res.len = a.len, i = 0; i < res.len; i++)
  {
    res.s[i] = a.s[i]-cy;
    if (i < b.len)
      res.s[i] -= b.s[i];
    if (res.s[i] < 0)
      cy = 1, res.s[i] += base;
    else
      cy = 0;
  }
  while (res.len > 0 && res.s[res.len-1] == 0)
    res.len--;
  return res;
}
bint operator * (const bint &a, const bint &b)
{
  bint res; res.len = 0;
  if (0 == b.len)
  {
    res.s[0] = 0;
    return res;
  }
  int i, j, cy;
  for (i = 0; i < a.len; i++)
  {
    for (j=cy=0; j < b.len || cy > 0; j++, cy/= base)
    {
      if (j < b.len)
        cy += a.s[i]*b.s[j];
      if (i+j < res.len)
        cy += res.s[i+j];
      if (i+j >= res.len)
        res.s[res.len++] = cy%base;
      else
        res.s[i+j] = cy%base;
    }
  }
  return res;
}
bint operator / (const bint &a, const bint &b)
{ // ! b != 0
  bint tmp, mod, res;
  int i, lf, rg, mid;
  mod.s[0] = mod.len = 0;
  for (i = a.len-1; i >= 0; i--)
  {
    mod = mod*base+a.s[i];
    for (lf = 0, rg = base-1; lf < rg; )
    {
      mid = (lf+rg+1)/2;
      if (b*mid <= mod)
        lf = mid;
      else
        rg = mid-1;
    }
    res.s[i] = lf;
    mod = mod-b*lf;
  }
  res.len = a.len;
  while (res.len > 0 && res.s[res.len-1] == 0)
    res.len--;
  return res; // return mod 就是%运算
}
int digits(bint &a) // 返回位数
{
  if (a.len == 0) return 0;
  int l = (a.len-1)*4;
  for (int t = a.s[a.len-1]; t; ++l, t/=10);
  return l;
}
bool read(bint &b, char buf[]) // 读取失败返回 0
{
  if (1 != scanf("%s", buf)) return 0;
  int w, u, len = strlen(buf);
  memset(&b, 0, sizeof(bint));
  if ('0' == buf[0] && 0 == buf[1]) return 1;
  for (w = 1, u = 0; len; )
  {
    u += (buf[--len]-'0')*w;
    if (w*10 == base)
    {
      b.s[b.len++] = u;
      u = 0;
      w = 1;
    }
    else
      w *= 10;
  }
  if (w != 1)
    b.s[b.len++] = u;
  return 1;
}
void write(const bint &v)
{
  int i;
  printf("%d", v.len == 0 ? 0 : v.s[v.len-1]);
  for (i = v.len-2; i >= 0; i--)
    printf("%04d", v.s[i]); // ! 4 == width
  printf("\n");
}
int main()
{
  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
  int a, b; scanf("%d%d", &a, &b);
  bint A(a), B(b);
  if (B < A)
  {
    write(A+B);
    write(A-B);
    write(A*B);
    write(A/B);
  }
  return 0;
}
```

bign-lrj

```cpp
#include<cstdio>
#include<iostream>
using namespace std;

const int maxn = 200;
struct bign{
  int len, s[maxn];

  bign() {
  memset(s, 0, sizeof(s));
  len = 1;
  }

  bign(int num) {
  *this = num;
  }

  bign(const char* num) {
  *this = num;
  }

  bign operator = (int num) {
  char s[maxn];
  sprintf(s, "%d", num);
  *this = s;
  return *this;
  }

  bign operator = (const char* num) {
  len = strlen(num);
  for(int i = 0; i < len; i++) s[i] = num[len-i-1] - '0';
  return *this;
  }

  string str() const {
  string res = "";
  for(int i = 0; i < len; i++) res = (char)(s[i] + '0') + res;
  if(res == "") res = "0";
  return res;
  }

  bign operator + (const bign& b) const{
  bign c;
  c.len = 0;
  for(int i = 0, g = 0; g || i < max(len, b.len); i++) {
    int x = g;
    if(i < len) x += s[i];
    if(i < b.len) x += b.s[i];
    c.s[c.len++] = x % 10;
    g = x / 10;
  }
  return c;
  }

  void clean() {
  while(len > 1 && !s[len-1]) len--;
  }

  bign operator * (const bign& b) {
  bign c; c.len = len + b.len;
  for(int i = 0; i < len; i++)
    for(int j = 0; j < b.len; j++)
    c.s[i+j] += s[i] * b.s[j];
  for(int i = 0; i < c.len-1; i++){
    c.s[i+1] += c.s[i] / 10;
    c.s[i] %= 10;
  }
  c.clean();
  return c;
  }

  bign operator - (const bign& b) {
  bign c; c.len = 0;
  for(int i = 0, g = 0; i < len; i++) {
    int x = s[i] - g;
    if(i < b.len) x -= b.s[i];
    if(x >= 0) g = 0;
    else {
    g = 1;
    x += 10;
    }
    c.s[c.len++] = x;
  }
  c.clean();
  return c;
  }

  bool operator < (const bign& b) const{
  if(len != b.len) return len < b.len;
  for(int i = len-1; i >= 0; i--)
    if(s[i] != b.s[i]) return s[i] < b.s[i];
  return false;
  }

  bool operator > (const bign& b) const{
  return b < *this;
  }

  bool operator <= (const bign& b) {
  return !(b > *this);
  }

  bool operator == (const bign& b) {
  return !(b < *this) && !(*this < b);
  }

  bign operator += (const bign& b) {
  *this = *this + b;
  return *this;
  }
};

istream& operator >> (istream &in, bign& x) {
  string s;
  in >> s;
  x = s.c_str();
  return in;
}

ostream& operator << (ostream &out, const bign& x) {
  out << x.str();
  return out;
}

int main() {
  bign a;
  cin >> a;
  a += "1234567891234567890000000000";
  cout << a*2 << endl;
  return 0;
}
```

bign-str

```
#include<cstdio>
#include<cstring>
using namespace std;
const int MAXSIZE = 200;
void Add(char *str1, char *str2, char *str3);
void Minus(char *str1, char *str2, char *str3);
void Mul(char *str1, char *str2, char *str3);
void Div(char *str1, char *str2, char *str3);
int main(void)
{
  char str1[MAXSIZE], str2[MAXSIZE], str3[MAXSIZE];
  while (scanf("%s %s", str1, str2) == 2)
  {
    if (strcmp(str1, "0"))
    {
      memset(str3, '0', sizeof(str3)); // !!!!!
      Add(str1, str2, str3);
      printf("%s\n", str3);
      memset(str3, '0', sizeof(str3));
      Minus(str1, str2, str3);
      printf("%s\n", str3);
      memset(str3, '0', sizeof(str3));
      Mul(str1, str2, str3);
      printf("%s\n", str3);
      memset(str3, '0', sizeof(str3));
      Div(str1, str2, str3);
      printf("%s\n", str3);
    }
    else
    {
      if (strcmp(str2, "0"))
        printf("%s\n-%s\n0\n0\n", str2, str2);
      else
        printf("0\n0\n0\n0\n");
    }
  }
  return 0;
}
void Add(char *str1, char *str2, char *str3)
{// str3 = str1 + str2;
  int i, j, i1, i2, tmp, carry;
  int len1 = strlen(str1), len2 = strlen(str2);
  char ch;
  i1 = len1-1; i2 = len2-1;
  j = carry = 0;
  for (; i1 >= 0 && i2 >= 0; ++j, --i1, --i2)
  {
    tmp = str1[i1]-'0'+str2[i2]-'0'+carry;
    carry = tmp/10;
    str3[j] = tmp%10+'0';
  }
  while (i1 >= 0)
  {
    tmp = str1[i1--]-'0'+carry;
    carry = tmp/10;
    str3[j++] = tmp%10+'0';
  }
  while (i2 >= 0)
  {
    tmp = str2[i2--]-'0'+carry;
    carry = tmp/10;
    str3[j++] = tmp%10+'0';
  }
  if (carry)
    str3[j++] = carry+'0';
  str3[j] = '\0';
  for (i = 0, --j; i < j; ++i, --j)
  {
    ch = str3[i]; str3[i] = str3[j]; str3[j] = ch;
  }
}
void Minus(char *str1, char *str2, char *str3)
{// str3 = str1-str2 (str1 > str2)
  int i, j, i1, i2, tmp, carry;
  int len1 = strlen(str1), len2 = strlen(str2);
  char ch;
  i1 = len1-1; i2 = len2-1;
  j = carry = 0;
  while (i2 >= 0)
  {
    tmp = str1[i1]-str2[i2]-carry;
    if (tmp < 0)
    {
      str3[j] = tmp+10+'0'; carry = 1;
    }
    else
    {
      str3[j] = tmp+'0'; carry = 0;
    }
    --i1; --i2; ++j;
  }
  while (i1 >= 0)
  {
    tmp = str1[i1]-'0'-carry;
    if (tmp < 0)
    {
      str3[j] = tmp+10+'0'; carry = 1;
    }
    else
    {
      str3[j] = tmp+'0'; carry = 0;
    }
    --i1; ++j;
  }
  --j;
  while (str3[j] == '0' && j > 0)
    --j;
  str3[++j] = '\0';
  for (i=0, --j; i < j; ++i, --j)
  {
    ch = str3[i]; str3[i] = str3[j]; str3[j] = ch;
  }
}
void Mul(char *str1, char *str2, char *str3)
{
  int i, j, i1, i2, tmp, carry, jj;
  int len1 = strlen(str1), len2 = strlen(str2);
  char ch;
  jj = carry = 0;
  for (i1=len1-1; i1 >= 0; --i1)
  {
    j = jj;
    for (i2=len2-1; i2 >= 0; --i2, ++j)
    {
      tmp = (str3[j]-'0')+(str1[i1]-'0')*(str2[i2]-'0')+carry;
      if (tmp > 9)
      {
        carry = tmp/10; str3[j] = tmp%10+'0';
      }
      else
      {
        str3[j] = tmp+'0'; carry = 0;
      }
    }
    if (carry)
    {
      str3[j] = carry+'0'; carry = 0; ++j;
    }
    ++jj;
  }
  --j;
  while (str3[j] == '0' && j > 0)
    --j;
  str3[++j] = '\0';
  for (i=0, --j; i < j; ++i, --j)
  {
    ch = str3[i]; str3[i] = str3[j]; str3[j] = ch;
  }
}
void Div(char *str1, char *str2, char *str3)
{
  int i1, i2, i, j, jj, tag, carry, cf, c[MAXSIZE];
  int len1 = strlen(str1), len2 = strlen(str2), lend;
  char d[MAXSIZE];
  memset(c, 0, sizeof(c));
  memcpy(d, str1, len2);
  lend = len2; j = 0;
  for (i1=len2-1; i1 < len1; ++i1)
  {
    if (lend < len2)
    {
      d[lend] = str1[i1+1]; c[j] = 0;
      ++j; ++lend;
    }
    else if (lend == len2)
    {
      jj = 1;
      for (i=0; i < lend; ++i)
      {
        if (d[i] > str2[i]) break;
        else if (d[i] < str2[i])
        {
          jj = 0; break;
        }
      }
      if (jj == 0)
      {
        d[lend] = str1[i1+1]; c[j] = 0;
        ++j; ++lend;
        continue;
      }
    }
    if (jj==1 || lend > len2)
    {
      cf = jj=0;
      while (d[jj] <= '0' && jj < lend)
        ++jj;
      if (lend-jj > len2)
        cf = 1;
      else if (lend-jj < len2)
        cf = 0;
      else
      {
        i2 = 0; cf = 1;
        for (i = jj; i < lend; ++i)
        {
          if (d[i] < str2[i2])
          {
            cf = 0; break;
          }
          else if (d[i] > str2[i2])
          {
            break;
          }
          ++i2;
        }
      }//else
      while (cf)
      {
        i2 = len2-1; cf = 0;
        for (i = lend-1; i >= lend-len2; --i)
        {
          d[i] = d[i]-str2[i2]+'0';
          if (d[i] < '0')
          {
            d[i] = d[i]+10; carry = 1;
            --d[i-1];
          }
          else
            carry = 0;
          --i2;
        }
        ++c[j]; jj=0;
        while (d[jj] <= '0' && jj < lend)
          ++jj;
        if (lend-jj > len2)
          cf = 1;
        else if (lend-jj < len2)
          cf = 0;
        else
        {
          i2 = 0; cf = 1;
          for (i = jj; i < lend; ++i)
          {
            if (d[i] < str2[i2])
            {
              cf = 0; break;
            }
            else if (d[i] > str2[i2])
            {
              break;
            }
            ++i2;
          }
        }//else
      }//while
      jj = 0;
      while (d[jj] <= '0' && jj < lend)
        ++jj;
      for (i = 0; i < lend-jj; ++i)
        d[i] = d[i+jj];
      d[i] = str1[i1+1]; lend = i+1;
      ++j;
    }//else
  }//for
  i = tag = 0;
  while (c[i] == 0)
    ++i;
  for (; i < j; ++i, ++tag)
    str3[tag] = c[i]+'0';
  str3[tag] = '\0';
}
```

后缀数组

第 K 个子串_hdu_3553

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<set>
using namespace std;
const int MAXN = 100000+5;
int T;
int sa[MAXN], height[MAXN], rank[MAXN], tmp[MAXN], top[MAXN];
int Tr[MAXN<<2];
long long K, sumlen[MAXN];
char S[MAXN];
namespace SuffixArray
{
  void makesa(char *s, int n)
  {
    int lena = n < 256 ? 256 : n;
    memset(top, 0, lena*sizeof(int));
    for (int i = 0; i < n; i++)
```

```
      top[rank[i]] = s[i]&(-1)]++;
    for (int i = 1; i < lena; i++)
      top[i] += top[i-1];
    for (int i = 0; i < n ; i++)
      sa[--top[rank[i]]] = i;
    for (int k = 1; k < n; k <<= 1)
    {
      for (int i = 0; i < n; i++)
      {
        int j = sa[i]-k;
        if (j < 0)
          j += n;
        tmp[top[rank[j]]++] = j;
      }
      int j = sa[tmp[0]] = top[0] = 0;
      for (int i = 1; i < n; i++)
      {
        if (rank[tmp[i]] != rank[tmp[i-1]] || rank[tmp[i]+k] != rank[tmp[i-1]+k])
          top[++j] = i;
        sa[tmp[i]] = j;
      }
      memcpy(rank, sa , n*sizeof(int));
      memcpy(sa , tmp, n*sizeof(int));
      if (j+1 >= n)
        break;
    }
  }
  void lcp(char *s, int n)
  {
    height[0] = 0;
    for (int i = 0, k = 0, j = rank[0]; i+1 < n; i++, k++)
      while (k >= 0 && s[i] != s[sa[j-1]+k])
      {
        height[j] = k--;
        j = rank[sa[j]+1];
      }
  }
}
namespace SegTr
{
  void Build(int idx, int L, int R)
  {
    if (L == R)
    {
      Tr[idx] = R;
      return;
    }
    int mid = (L+R)>>1, left = idx<<1, right = idx<<1|1;
    Build(left, L, mid);
    Build(right, mid+1, R);
    Tr[idx] = (height[Tr[left]] <= height[Tr[right]] ? Tr[left] : Tr[right]);
  }
  int Query(int idx, int L, int R, int l, int r)
  {
    if (l <= L && R <= r)
      return Tr[idx];
    int mid = (L+R)>>1, left = idx<<1, right = idx<<1|1;
    int ql = 0, qr = 0;
    if (l <= mid)
      ql = Query(left, L, mid, l, r);
    if (mid < r)
      qr = Query(right, mid+1, R, l, r);
    if (ql && !qr)
      return ql;
    else if (!ql && qr)
      return qr;
    else
      return (height[ql] <= height[qr] ? ql : qr);
  }
}
void solve(int len, int &rk, int &rl)
{
  int h = 0;
  long long a = 1, b = len;
  while (a < b)
  {
    int q = SegTr::Query(1, 1, len, a+1, b);
    if (K <= (height[q]-h)*(b-a+1))
    {
      rk = a; rl = h+1+(K-1)/(b-a+1);
      return;
    }
    K -= (height[q]-h)*(b-a+1);
    if (K <= sumlen[q-1]-sumlen[a-1]-height[q]*(q-a))
    {
      b = q-1; h = height[q];
      continue;
    }
    K -= sumlen[q-1]-sumlen[a-1]-height[q]*(q-a);
    a = q;
    h = height[q];
  }
  rk = a; rl = h+K;
}
int main()
{
  scanf("%d", &T);
  for (int cas = 1; cas <= T; cas++)
  {
    scanf("%s%I64d", S, &K);
    int len = strlen(S);
    SuffixArray::makesa(S, len+1);
    SuffixArray::lcp(S, len+1);
    for (int i = 1; i <= len; i++)
```

```
   sumlen[i] = sumlen[i-1]+len-sa[i];
  SegTr::Build(1, 1, len);
  int rk, rl;
  solve(len, rk, rl);
  printf("Case %d: ", cas);
  for (int i = 0; i < rl; i++)
   printf("%c", S[sa[rk]+i]);
  printf("\n");
 }
 return 0;
}
```

多串子串并集_后缀数组_hdu_4416

```
/*
求多串的子串并集元素的个数，先用没出现过的不同的字符把多个串拼接，用后缀数组求这个串的不同子串的个数，再减去含有拼接字符的子串的个数。用上述方法求『A、B1、……、BN』中不同子串的个数 sumAB 和『B1、……、BN』中不同子串的个数 sumB，答案就是 sumAB-sumB。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 300000+5, MAXM = 100000+5;
int T, N, L[MAXM];
int len, sa[MAXN], height[MAXN], rank[MAXN], tmp[MAXN], top[MAXN];
int a[MAXN];
char A[MAXM];
void makesa(int *s, int n)
{
 int lena = n < 256 ? 256 : n;
 memset(top, 0, lena*sizeof(int));
 for (int i = 0; i < n; i++)
  top[rank[i] = s[i]&(-1)]++;
 for (int i = 1; i < lena; i++)
  top[i] += top[i-1];
 for (int i = 0; i < n ; i++)
  sa[--top[rank[i]]] = i;
 for (int k = 1; k < n; k <<= 1)
 {
  for (int i = 0; i < n; i++)
  {
   int j = sa[i]-k;
   if (j < 0)
    j += n;
   tmp[top[rank[j]]++] = j;
  }
  int j = sa[tmp[0]] = top[0] = 0;
  for (int i = 1; i < n; i++)
  {
   if (rank[tmp[i]] != rank[tmp[i-1]] || rank[tmp[i]+k] != rank[tmp[i-1]+k])
    top[++j] = i;
   sa[tmp[i]] = j;
  }
  memcpy(rank, sa , n*sizeof(int));
  memcpy(sa , tmp, n*sizeof(int));
  if (j+1 >= n)
   break;
 }
}
void lcp(int *s, int n)
{
 height[0] = 0;
 for (int i = 0, k = 0, j = rank[0]; i+1 < n; i++, k++)
  while (k >= 0 && s[i] != s[sa[j-1]+k])
  {
   height[j] = k--;
   j = rank[sa[j]+1];
  }
}
int main()
{
 scanf("%d", &T);
 for (int cas = 1; cas <= T; cas++)
 {
  scanf("%d%s", &N, A);
  len = 0;
  for (L[0] = 0; A[L[0]]; L[0]++)
   a[len++] = A[L[0]]-'a'+1;
  for (int i = 1; i <= N; i++)
  {
   a[len++] = 26+i;
   scanf("%s", A);
   for (L[i] = 0; A[L[i]]; L[i]++)
    a[len++] = A[L[i]]-'a'+1;
  }
  a[len] = 0;
  long long sumAB = 0, sumB = 0;
  makesa(a, len+1);
  lcp(a, len+1);
  for (int i = 1; i <= len; i++)
   sumAB += len-sa[i]-height[i];
  long long l = len;
  for (int i = 0; i < N; i++)
  {
   l -= L[i];
   sumAB -= (L[i]+1)*l;
   l--;
  }
  len -= L[0]+1;
  makesa(a+L[0]+1, len+1);
  lcp(a+L[0]+1, len+1);
  for (int i = 1; i <= len; i++)
   sumB += len-sa[i]-height[i];
```

```
    l = len;
    for (int i = 1; i < N; i++)
    {
      l -= L[i];
      sumB -= (L[i]+1)*l;
      l--;
    }
    printf("Case %d: %I64d\n", cas, sumAB-sumB);
  }
  return 0;
}
```

最长重复不重叠子串_后缀数组+按 height 分组+二分_poj_1743

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 20000+5;
const int INF = 0x3f3f3f3f;
int N, a[MAXN], s[MAXN];
int sa[MAXN], height[MAXN], rank[MAXN], tmp[MAXN], top[MAXN];
void makesa(int *s, int n)
{
  int lena = n < 256 ? 256 : n;
  memset(top, 0, lena*sizeof(int));
  for (int i = 0; i < n; i++)
    top[rank[i] = s[i]&(-1)]++;
  for (int i = 1; i < lena; i++)
    top[i] += top[i-1];
  for (int i = 0; i < n ; i++)
    sa[--top[rank[i]]] = i;
  for (int k = 1; k < n; k <<= 1)
  {
    for (int i = 0; i < n; i++)
    {
      int j = sa[i]-k;
      if (j < 0)
        j += n;
      tmp[top[rank[j]]++] = j;
    }
    int j = sa[tmp[0]] = top[0] = 0;
    for (int i = 1; i < n; i++)
    {
      if (rank[tmp[i]] != rank[tmp[i-1]] || rank[tmp[i]+k] != rank[tmp[i-1]+k])
        top[++j] = i;
      sa[tmp[i]] = j;
    }
    memcpy(rank, sa , n*sizeof(int));
    memcpy(sa , tmp, n*sizeof(int));
    if (j+1 >= n)
      break;
  }
}
void lcp(int *s, int n)
{
  height[0] = 0;
  for (int i = 0, k = 0, j = rank[0]; i+1 < n; i++, k++)
    while (k >= 0 && s[i] != s[sa[j-1]+k])
    {
      height[j] = k--;
      j = rank[sa[j]+1];
    }
}
int main()
{
  while (scanf("%d", &N) && N)
  {
    int len = 0;
    for (int i = 0; i < N; i++)
    {
      scanf("%d", &a[i]);
      if (i)
        s[len++] = a[i]-a[i-1]+88;
    }
    s[len] = 0;
    makesa(s, len+1);
    lcp(s, len+1);
    int l = 4, r = max(l+1, N/2), ans = -1;
    while (l < r)
    {
      int mid = (l+r)>>1, t = 0, mini = sa[0], maxi = sa[0];
      for (int i = 1; i <= len; i++)
      {
        if (height[i] >= mid)
        {
          mini = min(mini, sa[i]);
          maxi = max(maxi, sa[i]);
        }
        else
        {
          t = max(t, maxi-mini);
          mini = maxi = sa[i];
        }
      }
      t = max(t, maxi-mini);
      if (t > mid)
      {
        ans = mid;
        l = mid+1;
      }
      else
        r = mid;
    }
    printf("%d\n", ans+1);
  }
  return 0;
}
```

线段树

矩形并面积_离散化+扫描线+线段树_hdu_4419

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<map>
#define left(x) x<<1
#define right(x) x<<1|1
using namespace std;
const int MAXN = 10000+5, MAXM = 20000+5;
const int ALL = 1<<3;
int T, N, clr[MAXN], X[MAXM], Y[MAXM], y[MAXM], r[MAXM];
int ID[1<<8];
int Tsum[ALL][MAXM<<2], Tcov[ALL][MAXM<<2];
char C[5];
bool cmp(const int a, const int b)
{
  return X[a] < X[b];
}
//void Build(int idx, int L, int R)
//{
//  for (int k = 1; k < ALL; k++)
//    Tsum[k][idx] = Tcov[k][idx] = 0;
//  if (R-L == 1)
//    return;
//  int mid = (L+R)>>1;
//  Build(left(idx), L, mid);
//  Build(right(idx), mid, R);
//}
void Update(int tr, int idx, int L, int R, int l, int r, int c)
{
  if (l <= L && R <= r)
    Tcov[tr][idx] += c;
  else
  {
    int mid = (L+R)>>1;
    if (l < mid)
      Update(tr, left(idx), L, mid, l, r, c);
    if (mid < r)
      Update(tr, right(idx), mid, R, l, r, c);
  }
  if (Tcov[tr][idx])
    Tsum[tr][idx] = y[R-1]-y[L-1];
  else if (R-L == 1)
    Tsum[tr][idx] = 0;
  else
    Tsum[tr][idx] = Tsum[tr][left(idx)]+Tsum[tr][right(idx)];
}
int main()
{
  ID['R'] = 1<<0; ID['G'] = 1<<1; ID['B'] = 1<<2;
  scanf("%d", &T);
  for (int cas = 1; cas <= T; cas++)
  {
    scanf("%d", &N);
    for (int i = 0; i < N; i++)
    {
      scanf("%s%d%d%d%d", C, &X[left(i)], &Y[left(i)], &X[right(i)], &Y[right(i)]);
      clr[i] = ID[C[0]];
      y[left(i)] = Y[left(i)];
      y[right(i)] = Y[right(i)];
      r[left(i)] = left(i);
      r[right(i)] = right(i);
    }
    int n = N<<1;
    sort(r, r+n, cmp);
    sort(y, y+n);
    map<int, int> dp;
    for (int i = 1; i <= n; i++)
      dp[y[i-1]] = i;
//   Build(1, 1, n);
    long long area[ALL] = {};
    for (int i = 0; i < n; i++)
      for (int k = 1; k < ALL; k++)
      {
        if (k&clr[r[i]>>1])
          Update(k, 1, 1, n, dp[Y[left(r[i]>>1)]], dp[Y[right(r[i]>>1)]], (r[i]&1 ? -1 : 1));
        if (i+1 < n)
          area[k] += (long long)Tsum[k][1]*(X[r[i+1]]-X[r[i]]);
      }
    printf("Case %d:\n", cas);
    printf("%I64d\n", area[7]-area[6]);
    printf("%I64d\n", area[7]-area[5]);
    printf("%I64d\n", area[7]-area[3]);
    printf("%I64d\n", area[5]+area[6]-area[4]-area[7]);
    printf("%I64d\n", area[3]+area[6]-area[2]-area[7]);
    printf("%I64d\n", area[3]+area[5]-area[1]-area[7]);
    printf("%I64d\n", area[1]+area[2]+area[4]-area[3]-area[5]-area[6]+area[7]);
  }
  return 0;
}
```

线段树求矩形并周长_hdu_1828

```cpp
/*
思路：扫描线+线段树。记录完全覆盖住当前区间的线段条数，区间左右端点被几条线段覆盖。叶节点表示长度为 1 的区间。用一个查询函数求一共有多少孤立线段。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 20000+5, MAXM = 10000+5;
const int A = 10000, Len = 20000;
int N, x[MAXM], y[MAXM], r[MAXM];
int Tr[MAXN<<2], Tcov[MAXN<<2], covl[MAXN<<2], covr[MAXN<<2], mark[MAXN<<2];
bool cmpx(const int a, const int b)
{
  return x[a] < x[b];
}
bool cmpy(const int a, const int b)
{
  return y[a] < y[b];
}
//void Init(int idx, int L, int R)
//{
//  if (L == R)
//  {
//    Tr[idx] = 0;
//    covl[idx] = covr[idx] = 0;
//    mark[idx] = 0;
//    return;
//  }
//  Tr[idx] = 0;
//  covl[idx] = covr[idx] = 0;
//  mark[idx] = 0;
//}
void PushDown(int idx, int L, int R)
{
  int left = 2*idx, right = 2*idx+1;
  Tcov[left] += mark[idx];
  Tr[left] = Tcov[left] ? 1 : 0;
  covl[left] += mark[idx];
  covr[left] += mark[idx];
  mark[left] += mark[idx];
  Tcov[right] += mark[idx];
  Tr[right] = Tcov[right] ? 1 : 0;
  covl[right] += mark[idx];
  covr[right] += mark[idx];
  mark[right] += mark[idx];
  mark[idx] = 0;
}
void Update(int idx, int L, int R, int l, int r, int c)
{
  if (l <= L && R <= r)
  {
    Tcov[idx] += c;
    covl[idx] += c;
    covr[idx] += c;
    if (Tcov[idx] || R-L == 1)
    {
      Tr[idx] = Tcov[idx] ? 1 : 0;
      mark[idx] += c;
      return;
    }
  }
  if (mark[idx])
    PushDown(idx, L, R);
  int mid = (L+R)/2, left = 2*idx, right = 2*idx+1;
  if (l < mid)
    Update(left, L, mid, l, r, c);
  if (mid < r)
    Update(right, mid, R, l, r, c);
  covl[idx] = covl[left];
  covr[idx] = covr[right];
  Tr[idx] = Tr[left]+Tr[right]-(covr[left] && covl[right] ? 1 : 0);
}
int main()
{
  while (scanf("%d", &N) != EOF)
  {
    for (int i = 0, j; i < N; i++)
    {
      j = 2*i;
      scanf("%d%d", &x[j], &y[j]);
      x[j] += A; y[j] += A;
      r[j] = j;
      j = 2*i+1;
      scanf("%d%d", &x[j], &y[j]);
      x[j] += A; y[j] += A;
      r[j] = j;
    }
    int ans = 0;
    sort(r, r+2*N, cmpx);
    for (int i = 0; i < 2*N; )
    {
      bool flag = 1;
      for (; (flag || x[r[i]] == x[r[i-1]]) && i < 2*N; i++)
      {
        flag = 0;
        int k = r[i];
        if (!(k%2))
          Update(1, 0, Len, y[k], y[k^1], 1);
        else
          Update(1, 0, Len, y[k^1], y[k], -1);
      }
      if (i < 2*N)
    ans += (x[r[i]]-x[r[i-1]])*Tr[1]*2;
  }
  sort(r, r+2*N, cmpy);
  for (int i = 0; i < 2*N; )
  {
   bool flag = 1;
   for (; (flag || y[r[i]] == y[r[i-1]]) && i < 2*N; i++)
   {
    flag = 0;
    int k = r[i];
    if (!(k%2))
     Update(1, 0, Len, x[k], x[k^1], 1);
    else
     Update(1, 0, Len, x[k^1], x[k], -1);
   }
   if (i < 2*N)
    ans += (y[r[i]]-y[r[i-1]])*Tr[1]*2;
  }
  printf("%d\n", ans);
 }
 return 0;
}
```

线段树求体积并_hdu_3642

```
/*
题意：就是给你一些长方体，求这些长方体相交至少 3 次的体积和。
思路：对 z 轴扫描线，每次在 xy 平面对 x 轴扫描线、对 y 轴离散化用线段树求面积并，再把分段求得的体积加和。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 2000+5, MAXM = 2000+5, MAXP = 2000000+5;
int T, N, X[MAXM], Y[MAXM], Z[MAXM], rx[MAXM], ry[MAXM], rz[MAXM];
int Tr[MAXN<<2], Tcov[MAXN<<2], mark[MAXN<<2];
int match[MAXP], toy[MAXN];
bool cmpz(const int a, const int b)
{
 return Z[a] < Z[b];
}
bool cmpx(const int a, const int b)
{
 return X[a] < X[b];
}
bool cmpy(const int a, const int b)
{
 return Y[a] < Y[b];
}
//void Init(int idx, int L, int R)
//{
//  if (R-L == 1)
//  {
//    Tr[idx] = 0;
//    Tcov[idx] = 0;
//    mark[idx] = 0;
//    return;
//  }
//  int mid = (L+R)/2, left = idx*2, right = idx*2+1;
//  Init(left, L, mid);
//  Init(right, mid, R);
//  Tr[idx] = 0;
//  Tcov[idx] = 0;
//  mark[idx] = 0;
//}
void PushDown(int idx, int L, int R)
{
 int mid = (L+R)/2, left = idx*2, right = idx*2+1;
 Tcov[left] += mark[idx];
 Tr[left] = Tcov[left] > 2 ? toy[mid]-toy[L] : 0;
 mark[left] += mark[idx];
 Tcov[right] += mark[idx];
 Tr[right] = Tcov[right] > 2 ? toy[R]-toy[mid] : 0;
 mark[right] += mark[idx];
 mark[idx] = 0;
}
void Update(int idx, int L, int R, int l, int r, int c)
{
 if (l <= L && R <= r)
 {
  Tcov[idx] += c;
  if (Tcov[idx] > 2 || R-L == 1)
  {
   mark[idx] += c;
   Tr[idx] = Tcov[idx] > 2 ? toy[R]-toy[L] : 0;
   return;
  }
 }
 if (mark[idx])
  PushDown(idx, L, R);
 int mid = (L+R)/2, left = idx*2, right = idx*2+1;
 if (l < mid)
  Update(left, L, mid, l, r, c);
 if (mid < r)
  Update(right, mid, R, l, r, c);
 Tr[idx] = Tr[left]+Tr[right];
}
int main()
{
 scanf("%d", &T);
 for (int cas = 1; cas <= T; cas++)
 {
  memset(match, 0, sizeof(match));
  scanf("%d", &N);
```

```
    for (int i = 0; i < N; i++)
      for (int j = 0; j < 2; j++)
      {
        int k = 2*i+j;
        scanf("%d%d%d", &X[k], &Y[k], &Z[k]);
        Y[k] += 1000000;
        rx[k] = ry[k] = rz[k] = k;
      }
    sort(rx, rx+2*N, cmpx);
    sort(ry, ry+2*N, cmpy);
    sort(rz, rz+2*N, cmpz);
    int cnt = 0;
    for (int i = 0; i < 2*N; i++)
      if (!match[Y[ry[i]]])
      {
        match[Y[ry[i]]] = ++cnt;
        toy[cnt] = Y[ry[i]];
      }
    long long ans = 0;
    for (int i = 0; i < 2*N; )
    {
      long long area = 0;
      for (int j = 0; j < 2*N; )
      {
        int curX = X[rx[j]];
        for (; curX == X[rx[j]] && j < 2*N; j++)
        {
          int k = rx[j]/2;
          if (Z[2*k] <= Z[rz[i]] && Z[rz[i]] < Z[2*k+1])
            Update(1, 1, cnt, match[Y[2*k]], match[Y[2*k+1]], (rx[j]&1 ? -1 : 1));
        }
        if (j < 2*N)
          area += (long long)(X[rx[j]]-X[rx[j-1]])*Tr[1];
      }
      int curZ = Z[rz[i]];
      for (; curZ == Z[rz[i]] && i < 2*N; i++);
      if (i < 2*N)
        ans += (Z[rz[i]]-Z[rz[i-1]])*area;
    }
    printf("Case %d: %I64d\n", cas, ans);
  }
  return 0;
}
```

线段树区间修改单点查询_220B

```
/*
题意：N 个数，M 个询问，每次问 Ai 到 Aj 里有多少个数 x 出现了 x 次。
思路：离线+线段树区间修改、单点查询。按右端点将查询区间排序。扫描数列，假设当前数 a 第 x 次出现，那么当 x>=a 时，区间[pos[a][x-a]+1,pos[a][x-a+1]]上所有点+1；当 x>a 时，区间[pos[a][x-a-1]+1,pos[a][x-a]]上所有点-1，pos[a][x]表示数 a 第 x 次出现的位置，为了方便，设所有数第一次出现的位置为 0。若当前扫描到的位置有查询区间的右端点，则在线段树上查询左端点处的值，即为该次查询的答案。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
using namespace std;
const int MAXN = 100000+5;
int N, M, a[MAXN], s[MAXN], t[MAXN], r[MAXN], ans[MAXN];
int Tr[MAXN<<2], mark[MAXN<<2];
vector<int> pos[MAXN];
bool cmp(const int a, const int b)
{
  return t[a] < t[b];
}
void PushDown(int idx)
{
  int left = idx<<1, right = (idx<<1)^1;
  Tr[left] += mark[idx];
  mark[left] += mark[idx];
  Tr[right] += mark[idx];
  mark[right] += mark[idx];
  mark[idx] = 0;
}
void Update(int idx, int L, int R, int l, int r, int c)
{
  if (l <= L && R <= r)
  {
    Tr[idx] += c;
    mark[idx] += c;
    return;
  }
  if (mark[idx])
    PushDown(idx);
  int mid = (L+R)>>1, left = idx<<1, right = (idx<<1)^1;
  if (l <= mid)
    Update(left, L, mid, l, r, c);
  if (mid < r)
    Update(right, mid+1, R, l, r, c);
}
int Query(int idx, int L, int R, int x)
{
  if (x == L & R == x)
    return Tr[idx];
  if (mark[idx])
    PushDown(idx);
  int mid = (L+R)>>1, left = idx<<1, right = (idx<<1)^1;
  if (x <= mid)
    return Query(left, L, mid, x);
  else
    return Query(right, mid+1, R, x);
}
```

```
int main()
{
  scanf("%d%d", &N, &M);
  for (int i = 1; i <= N; i++)
  {
    scanf("%d", &a[i]);
    if (a[i] <= N && !pos[a[i]].size())
      pos[a[i]].push_back(0);
  }
  for (int i = 0; i < M; i++)
  {
    scanf("%d%d", &s[i], &t[i]);
    r[i] = i;
  }
  sort(r, r+M, cmp);
  for (int i = 1, j = 0; i <= N && j < M; i++)
  {
    if (a[i] <= N)
    {
      pos[a[i]].push_back(i);
      if (pos[a[i]].size() > a[i])
        Update(1, 1, N, pos[a[i]][pos[a[i]].size()-a[i]-1]+1, pos[a[i]][pos[a[i]].size()-a[i]], 1);
      if (pos[a[i]].size() > a[i]+1)
        Update(1, 1, N, pos[a[i]][pos[a[i]].size()-a[i]-2]+1, pos[a[i]][pos[a[i]].size()-a[i]-1], -1);
    }
    for (; t[r[j]] == i && j < M; j++)
      ans[r[j]] = Query(1, 1, N, s[r[j]]);
  }
  for (int i = 0; i < M; i++)
    printf("%d\n", ans[i]);
  return 0;
}
```

最长上升子序列

二维 LIS+方案输出_sgu_521

```
/*
正向、反向分别求 LIS，再枚举每个点……
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
using namespace std;
const int MAXN = 100000+5;
int N, x[MAXN], y[MAXN], id[MAXN];
int Y[MAXN], f[MAXN], d[2][MAXN], cnt[MAXN];
bool mark[MAXN];
bool cmp (const int &a, const int &b)
{
  if (x[a] != x[b])
    return x[a] < x[b];
  else
    return y[a] > y[b];
}
int LIS(int x)
{
  int maxi = 0;
  for (int i = 1; i <= N; i++)
  {
    int j = lower_bound(f+1, f+1+maxi, Y[i])-f;
    maxi = max(maxi, j);
    f[j] = Y[i];
    d[x][i] = j;
  }
  return maxi;
}
int main()
{
  while (scanf("%d", &N) != EOF)
  {
    memset(cnt, 0, sizeof(cnt));
    for (int i = 1; i <= N; i++)
    {
      scanf("%d%d", &x[i], &y[i]);
      id[i] = i;
    }
    sort(id+1, id+1+N, cmp);
    for (int i = 1; i <= N; i++)
      Y[i] = y[id[i]];
    int maxlen = LIS(0);
    for (int i = 1; i <= N; i++)
      Y[i] = -y[id[N-i+1]];
    LIS(1);
    vector<int> ans[2];
    for (int i = 1; i <= N; i++)
    {
      mark[i] = (d[0][i]+d[1][N-i+1] == maxlen+1);
      if (mark[i])
      {
        cnt[d[0][i]]++;
        ans[0].push_back(id[i]);
      }
    }
    for (int i = 1; i <= N; i++)
      if (mark[i] && cnt[d[0][i]] == 1)
        ans[1].push_back(id[i]);
    for (int i = 0; i < 2; i++)
    {
      sort(ans[i].begin(), ans[i].end());
      printf("%u", ans[i].size());
    for (vector<int>::iterator it = ans[i].begin(); it != ans[i].end(); it++)
      printf(" %d", *it);
    printf("\n");
   }
  }
  return 0;
}
```

某矩形的 LIS_bupt_394

```
/*
离线读入所有点（左下、右上），在左下点查询，右上点更新。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100000+5, MAXM = 200000+5;
int T, N, x[MAXM], y[MAXM], id[MAXM];
int f[MAXN], g[MAXN];
bool cmp (const int &a, const int &b)
{
  if (x[a] != x[b])
    return x[a] < x[b];
  else
    return y[a] > y[b];
}
int LIS(int n)
{
  int maxi = 0;
  for (int i = 0; i < n; i++)
  {
    if (!(id[i]&1))
      g[id[i]>>1] = lower_bound(f+1, f+1+maxi, y[id[i]])-f;
    else
    {
      if (g[id[i]>>1] > maxi)
        f[++maxi] = y[id[i]];
      else
        f[g[id[i]>>1]] = min(f[g[id[i]>>1]], y[id[i]]);
    }
  }
  return maxi;
}
int main()
{
  scanf("%d", &T);
  while (T--)
  {
    scanf("%d", &N);
    for (int i = 0; i < N; i++)
    {
      scanf("%d%d%d%d", &x[i<<1], &y[i<<1], &x[i<<1|1], &y[i<<1|1]);
      id[i<<1] = i<<1;
      id[i<<1|1] = i<<1|1;
    }
    int n = N<<1;
    sort(id, id+n, cmp);
    printf("%d\n", LIS(n));
  }
  return 0;
}
```

最长上升子序列_poj_3903

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100000+5;
const int INF = 0x7fffffff;
int N, a[MAXN], f[MAXN];
//int d[MAXN];
int main()
{
  while (scanf("%d", &N) != EOF)
  {
    int maxi = 0;
    for (int i = 1; i <= N; i++)
    {
      scanf("%d", &a[i]);
      int x = lower_bound(f+1, f+1+maxi, a[i])-f;
      maxi = max(maxi, x);
      f[x] = a[i];
//      d[i] = x;
    }
    printf("%d\n", maxi);
  }
  return 0;
}
```

Mahjong_hdu_4431

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
using namespace std;
const int MAX = 34;
const char *mahjong[] = {
  "1m", "2m", "3m", "4m", "5m", "6m", "7m", "8m", "9m",
  "1s", "2s", "3s", "4s", "5s", "6s", "7s", "8s", "9s",
  "1p", "2p", "3p", "4p", "5p", "6p", "7p", "8p", "9p",
  "1c", "2c", "3c", "4c", "5c", "6c", "7c"
};
int T, cnt[MAX];
char tile[10];
int id(char *s)
{
  if (s[1] == 'm')
    return s[0]-'1';
  else if (s[1] == 's')
    return 9+s[0]-'1';
  else if (s[1] == 'p')
    return 18+s[0]-'1';
  else
    return 27+s[0]-'1';
}
//bool check_standard_dfs(int dep)
//{
//  if (dep == 5)
//    return 1;
//  bool res = 0;
//  if (!dep)
//  {
//    for (int i = 0; i < MAX && !res; i++) if (cnt[i] >= 2)
//    {
//      cnt[i] -= 2;
//      res = check_standard_dfs(dep+1);
//      cnt[i] += 2;
//    }
//  }
//  else
//  {
//    for (int i = 0; i < MAX && !res; i++)
//    {
//      if (cnt[i] >= 3)
//      {
//        cnt[i] -= 3;
//        res = check_standard_dfs(dep+1);
//        cnt[i] += 3;
//      }
//      if (i < 27 && i%9 <= 6 && cnt[i] >= 1 && cnt[i+1] >= 1 && cnt[i+2] >=
1)
//      {
//        for (int j = 0; j < 3; j++)
//          cnt[i+j]--;
//        res = check_standard_dfs(dep+1);
//        for (int j = 0; j < 3; j++)
//          cnt[i+j]++;
//      }
//    }
//  }
//  return res;
//}
bool check_standard()
{
  bool res = 0;
  for (int i = 0; i < MAX && !res; i++) if (cnt[i] >= 2)
  {
    int tmp[MAX], num = 0;
    memcpy(tmp, cnt, sizeof(cnt));
    tmp[i] -= 2;
    for (int j = 0; j < MAX; j++)
    {
      if (tmp[j] >= 3)
      {
        tmp[j] -= 3;
        num++;
      }
      if (j < 27 && j%9 < 7)
      {
        while (tmp[j] >= 1 && tmp[j+1] >= 1 && tmp[j+2] >= 1)
        {
          for (int k = 0; k < 3; k++)
            tmp[j+k]--;
          num++;
        }
      }
    }
    res = (num == 4);
  }
  return res;
}
bool check_ChiiToitsu()
{
  for (int i = 0; i < MAX; i++)
    if (cnt[i] && cnt[i] != 2)
      return 0;
  return 1;
}
bool check_KokushiMuso()
{
  int res = 0;
  for (int i = 0; i < 3; i++)
  {
    if (cnt[i*9+0] >= 1 && cnt[i*9+8] >= 1)
      res += cnt[i*9+0]+cnt[i*9+8];
    else
      return 0;
  }
  for (int i = 27; i < MAX; i++)
  {
    if (cnt[i] >= 1)
      res += cnt[i];
    else
      return 0;
  }
  return (res == 14);
}
int main()
{
  scanf("%d", &T);
  while (T--)
  {
    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < 13; i++)
    {
      scanf("%s", tile);
      cnt[id(tile)]++;
    }
    vector<int> ans;
    for (int i = 0; i < MAX; i++) if (cnt[i] < 4)
    {
      cnt[i]++;
      if (check_KokushiMuso() || check_ChiiToitsu()
|| check_standard())
        ans.push_back(i);
      cnt[i]--;
    }
    if (ans.size())
    {
      printf("%d", (int)ans.size());
      for (int i = 0; i < (int)ans.size(); i++)
        printf(" %s", mahjong[ans[i]]);
      printf("\n");
    }
    else
      printf("Nooten\n");
  }
  return 0;
}
```

RMQ-ST

```
#include<cstdio>
#include<cstring>
#include<cmath>
#include<algorithm>
using namespace std;
const int MAXN = 50000+5, MAXM = 16;
int N, Q;
int a[MAXN], st[MAXN][MAXM];
int pow2[MAXM];
inline int Most(const int &a, const int &b)
{
  return a > b ? a : b;
}
void InitRMQ(const int &n)
{
  pow2[0] = 1;
  for (int i = 1; i <= MAXM; i++)
    pow2[i] = pow2[i-1]<<1; //预处理 2 的 i 次方，最大次幂要大于 MAXN
  for (int i = 1; i <= n; i++)
    stmax[i][0] = a[i];
  int k = int(log(double(n))/log(2.0))+1;
  for (int j = 1; j < k; j++)
    for (int i = 1; i <= n; i++)
    {
      if (i+pow2[j-1]-1 <= n)
        stmax[i][j] = Most(stmax[i][j-1], stmax[i+pow2[j-1]][j-1]);
      else
        break; // st[i][j] = st[i][j-1];
    }
}
int Query(int x, int y) // x, y 均为下标:1...n
{
  int k = int(log(double(y-x+1))/log(2.0));
  return Most(stmax[x][k], stmax[y-pow2[k]+1][k]);
}
int main()
{
  scanf("%d%d", &N, &Q);
  for (int i = 1; i <= N; i++)
    scanf("%d", &a[i]);
  InitRMQ(N);
  while (Q--)
  {
    int A, B;
    scanf("%d%d", &A, &B);
    int ans = Query(A, B);
  }
  return 0;
}
```

Trie 树_编辑距离阈值匹配_UVALive_4769

```
/*
求字典中存在前缀与查询串编辑距离小于阈值的词的个数
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include <iostream>
using namespace std;
const int MAXM = 10+5;
const int MAX_NODE = 3000000+5, MAX_CHD = 26;
int N, M, edth;
int nv, chd[MAX_NODE][MAX_CHD], out[MAX_NODE], ID[1<<8];
int vis[MAX_NODE], mark[MAX_NODE];
char word[MAXM];
namespace Trie
```

```
{
  void Initialize()
  {
    for (int k = 0; k < MAX_CHD; k++)
      ID[k+'a'] = k;
  }
  void Reset()
  {
    memset(chd[0], 0, sizeof(chd[0]));
    nv = 1;
  }
  void Insert(char *pat)
  {
    int u = 0;
    for (int i = 0; pat[i]; i++)
    {
      int c = ID[pat[i]];
      if (!chd[u][c])
      {
        memset(chd[nv], 0, sizeof(chd[nv]));
        out[nv] = 0;
        chd[u][c] = nv++;
      }
      u = chd[u][c];
      out[u]++;
    }
  }
}
void dfs(int u, char *p, int d, int c)
{
  vis[u] = c;
  if (!(*p))
    mark[u] = c;
  if (mark[u] == c)
    return;
  if (chd[u][ID[*p]])
    dfs(chd[u][ID[*p]], p+1, d, c);
  if (d)
  {
    for (int i = 0; i < MAX_CHD; i++) if (chd[u][i])
      dfs(chd[u][i], p, d-1, c);
    for (int i = 0; i < MAX_CHD; i++) if (chd[u][i])
      dfs(chd[u][i], p+1, d-1, c);
    dfs(u, p+1, d-1, c);
  }
}
int calc(int u, int c)
{
  if (vis[u] != c)
    return 0;
  if (mark[u] == c)
    return out[u];
  int res = 0;
  for (int i = 0; i < MAX_CHD; i++) if (chd[u][i])
    res += calc(chd[u][i], c);
  return res;
}
int main()
{
  scanf("%d", &N);
  Trie::Initialize();
  Trie::Reset();
  for (int i = 1; i <= N; i++)
  {
    scanf("%s", word);
    Trie::Insert(word);
  }
  scanf("%d", &M);
  for (int i = 1; i <= M; i++)
  {
    scanf("%s%d", word, &edth);
    dfs(0, word, edth, i);
    printf("%d\n", calc(0, i));
  }
  return 0;
}
```

编辑距离+BK 树_hdu_4323

```
/*
1.dp 求编辑距离
2.bk 树找相差 d 的单词
*/
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 1500+5, MAXM = 10+5, MAXP = 400+5;
const int INF = 0x3f3f3f3f;
int T, n, m, t, cnt;
int d[MAXM][MAXM], next[MAXN][MAXM];
char str1[MAXN][MAXM], str2[MAXM];
int Distance(char *s1, char *s2)
{
  int l1 = strlen(s1), l2 = strlen(s2);
  for (int i = 0; i <= l1; i++)
    for (int j = 0; j <= l2; j++)
    {
      if (!(i*j))
        d[i][j] = i+j;
      else
      {
        d[i][j] = min(d[i-1][j]+1, d[i][j-1]+1);
      if (s1[i-1] == s2[j-1])
        d[i][j] = min(d[i][j], d[i-1][j-1]);
      else
        d[i][j] = min(d[i][j], d[i-1][j-1]+1);
    }
//      printf("%d,%d:%d\n", i, j, d[i][j]);
  }
  return d[l1][l2];
}
void dfs(int u)
{
  int dis = Distance(str1[u], str2);
  if (u && dis <= t)
    cnt++;
  for (int k = dis-t; k <= dis+t; k++)
    if (k >= 0 && next[u][k])
      dfs(next[u][k]);
}
int main()
{
  scanf("%d", &T);
  for (int cas = 1; cas <= T; cas++)
  {
    memset(next, 0, sizeof(next));
    scanf("%d%d", &n, &m);
    strcpy(str1[0], "");
    for (int i = 1; i <= n; i++)
    {
      scanf("%s", str1[i]);
      for (int j = 0; ; )
      {
        int dis = Distance(str1[i], str1[j]);
        if (!next[j][dis])
        {
          next[j][dis] = i;
          break;
        }
        j = next[j][dis];
      }
    }
    printf("Case #%d:\n", cas);
    for (int i = 1; i <= m; i++)
    {
      scanf("%s%d", str2, &t);
      cnt = 0;
      dfs(0);
      printf("%d\n", cnt);
    }
  }
  return 0;
}
```

后缀自动机_SPOJ_LCS2

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100000+5, MAXM = 10+5;
char s[MAXN];

//MAX_NODE = StringLength*2
const int MAX_NODE = 500000+5;
//字符集大小,一般字符形式的题 26 个
const int MAX_CHD = 26;
//已使用节点个数
int nv;
//每个节点的儿子,即当前节点的状态转移
int chd[MAX_NODE][MAX_CHD];
//此节点代表最长串的长度
int ml[MAX_NODE];
//父亲/失败指针
int fa[MAX_NODE];
//字母对应的 id
int id[1<<8];

//特定题目需要
int mml[MAX_NODE][MAXM], r[MAX_NODE];

namespace Suffix_Automaton
{
  //初始化,计算字母对应的儿子 id,如:'a'->0 ... 'z'->25
  void Initialize()
  {
    for (int i = 0; i < MAX_CHD; i++)
      id['a'+i] = i;
  }
  //增加一个节点
  void Add(int u, int _ml, int _fa, int v = -1)
  {
    ml[u] = _ml; fa[u] = _fa;
    if (v == -1)
      memset(chd[u], -1, sizeof(chd[u]));
    else
      memcpy(chd[u], chd[v], sizeof(chd[v]));
  }
  //建立后缀自动机
  void Construct(char *str)
  {
    nv = 1; Add(0, 0, -1);
    int cur = 0;
    for (int i = 0; str[i]; i++)
    {
      int c = id[str[i]], p = cur;
      cur = nv++; Add(cur, i+1, -1);
    for (; p != -1 && chd[p][c] == -1; p = fa[p])
      chd[p][c] = cur;
    if (p == -1)
      fa[cur] = 0;
    else
    {
      int q = chd[p][c];
      if (ml[q] == ml[p]+1)
        fa[cur] = q;
      else
      {
        int r = nv++; Add(r, ml[q], fa[q], q);
        ml[r] = ml[p]+1; fa[q] = fa[cur] = r;
        for (; p != -1 && chd[p][c] == q; p = fa[p])
          chd[p][c] = r;
      }
    }
  }
 }
}

bool cmp(const int &a, const int &b)
{
  return ml[a] > ml[b];
}
int main()
{
  Suffix_Automaton::Initialize();
  scanf("%s", s);
  Suffix_Automaton::Construct(s);
  for (int i = 0; i < nv; i++)
    r[i] = i;
  sort(r, r+nv, cmp);
  memset(mml, 0, sizeof(mml));
  int cnt = 0;
  for (int i = 1; scanf("%s", s) != EOF; i++, cnt++)
  {
    int l = 0, u = 0;
    for (int j = 0; s[j]; j++)
    {
      int c = id[s[j]];
      if (chd[u][c] != -1)
        l++, u = chd[u][c];
      else
      {
        while (u != -1 && chd[u][c] == -1)
          u = fa[u];
        if (u != -1)
          l = ml[u]+1, u = chd[u][c];
        else
          l = 0, u = 0;
      }
      mml[u][i] = max(mml[u][i], l);
    }
  }
  int ans = 0;
  for (int i = 0; i < nv; i++)
  {
    int mini = ml[r[i]];
    for (int j = 1; j <= cnt; j++)
    {
      mini = min(mini, mml[r[i]][j]);
      mml[fa[r[i]]][j] = max(mml[fa[r[i]]][j], mml[r[i]][j]);
    }
    ans = max(ans, mini);
  }
  printf("%d\n", ans);
  return 0;
}
```

斯坦纳树_hdu_4085

```
/*
斯坦纳树
最后的答案可能是一个森林，所以我们要先求出斯坦纳树后进行 DP。转移的时候要注意一点，只有人的个数和房子的个数相等的时候才算合法状态，所以我们要加一个 check()函数进行检查。
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
const int MAXN = 50+5, MAXM = 2000+5;
const int MAX = 10;
const int INF = 0x3f3f3f3f;
int T, N, M, K, X, Y, Z;
int bit[MAXN], head[MAXN], e, next[MAXM], v[MAXM], w[MAXM];
int inq[MAXN][1<<MAX], d[MAXN][1<<MAX], dp[1<<MAX];
queue<int> Q;
void addedge(int x, int y, int z)
{
  v[e] = y; w[e] = z;
  next[e] = head[x]; head[x] = e++;
}
void init()
{
  e = 0;
  memset(head, -1, sizeof(head));
  memset(d, 0x3f, sizeof(d));
  memset(bit, 0, sizeof(bit));
  memset(inq, 0, sizeof(inq));
  memset(dp, 0x3f, sizeof(dp));
}
void spfa()
{
    while (!Q.empty())
    {
      int u = Q.front()&((1<<MAX)-1), st = Q.front()>>MAX;
      Q.pop();
      inq[u][st] = 0;
      for (int i = head[u]; i != -1; i = next[i])
      {
        int nst = st|bit[v[i]];
        if (d[u][st]+w[i] < d[v[i]][nst])
        {
          d[v[i]][nst] = d[u][st]+w[i];
          if (nst == st && !inq[v[i]][nst])
          {
            Q.push(nst<<MAX|v[i]);
            inq[v[i]][nst] = 1;
          }
        }
      }
    }
}
bool check(int st)
{
  int res = 0;
  for (int i = 0; i < K; i++)
  {
    if (st&(1<<i))
      res++;
    if (st&(1<<(K+i)))
      res--;
  }
  return !res;
}
int main()
{
  freopen("put.in", "r", stdin);
  scanf("%d", &T);
  while (T--)
  {
    init();
    scanf("%d%d%d", &N, &M, &K);
    for(int i = 0; i < M; i++)
    {
      scanf("%d%d%d", &X, &Y, &Z);
      addedge(X, Y, Z);
      addedge(Y, X, Z);
    }
    int tot = (1<<(K<<1))-1;
    for (int i = 1; i <= K; i++)
    {
      bit[i] = 1<<(i-1);
      d[i][bit[i]] = 0;
      bit[N-K+i] = 1<<(K+i-1);
      d[N-K+i][bit[N-K+i]] = 0;
    }
    for (int i = 0; i <= tot; i++)
    {
      for (int j = 1; j <= N; j++)
      {
        for (int k = (i-1)&i; k; k = (k-1)&i)  //枚举 i 的所有子集
          d[j][i] = min(d[j][i], d[j][k|bit[j]]+d[j][(i-k)|bit[j]]);
        if (d[j][i] < INF)
        {
          Q.push(i<<MAX|j);
          inq[j][i] = 1;
        }
      }
      spfa();
    }
    for (int i = 0; i <= tot; i++)
      for (int j = 1; j <= N; j++)
        dp[i] = min(dp[i], d[j][i]);
    for (int i = 0; i <= tot; i++) if (check(i))
      for (int j = (i-1)&i; j; j = (j-1)&i) if (check(j))
        dp[i] = min(dp[i], dp[j]+dp[i-j]);
    if (dp[tot] < INF)
      printf("%d\n", dp[tot]);
    else
      printf("No solution\n");
  }
  return 0;
}
```

最大非空连续和+方案_hdu_1003

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 100000+5;
const int INF = 0x3f3f3f3f;
int T, N, a, s, t;
int main()
{
  scanf("%d", &T);
  for (int cas = 1; cas <= T; cas++)
  {
    scanf("%d", &N);
    int sum = 0, mini = 0, maxi = -INF, p = 1;
    for (int i = 1; i <= N; i++)
    {
      scanf("%d", &a);
      sum += a;
      if (sum-mini > maxi)
      {
        maxi = sum-mini;
```

```c
      s = p;
      t = i;
    }
    if (sum < mini)
    {
      mini = sum;
      p = i+1;
    }
  }
  if (cas > 1)
    printf("\n");
  printf("Case %d:\n", cas);
  printf("%d %d %d\n", maxi, s, t);
 }
 return 0;
}
```