Basic Knowledge

- Data types

Type	Storage	Maximum	Minimum
char	1 byte	127	-128
short	2 byte	32,767	-32,768
long	4 byte	2,147,483,647	-2,147,483,648
int	4 byte	2,147,483,647	-2,147,483,648
float	4 byte	3.4E +/- 38 (**7 digits**)	3.4E +/- 38 (**7 digits**)
double	8 byte	1.7E +/- 308 (**15 digits**)	1.7E +/- 308 (**15 digits**)
bool (C++)	1 byte	true (if non-zero)	false (if zero)

  - unsigned version to extend
  - Self-defined class for big number:
    - use an array `bit[]` to represent a big number
    - `bit[0]` store the number of digits

```
r = bit[0];
n = bit[1]*1+bit[2]*10+...+bit[r]*(10^(r-1))
```

```
void numberToArray(int n, int bit[]){
    bit[0] = 0;
    while(n){
    bit[++bit[0]] = n%10;
        n/=10;
    }
}
```

Addition: operation, carry-bit, carry flag

```
void add(int a[], int b[], int c[]){
    // operation
    memset(c,0,sizeof(c));
    for(int i=1;i<=a[0];i++)c[i]+=a[i];
    for(int i=1;i<=b[0];i++)c[i]+=b[i];
    c[0] = max(a[0],b[0]);
    // carry-bit
    int flag = 0;
    for(int i=1;i<=c[0];i++){
        c[i]+=flag;
        flag = c[i]/10;
        c[i] = c[i]%10;
    }
    // carry-flag
```

```
    if(flag) c[++c[0]]=flag;

}
```

- Mutiplication: operation, carry-bit, carry flag

```
void mutiply(int a[],int b[],int c[]){
  memset(c,0,sizeof(c));
    for(int i=1;i<=a[0];i++){
        for(int j=1;j<=b[0];j++){
            c[i+j-1]+=a[i]*b[j];  // i+(j-1)
        }
    }
    c[0]=a[0]+b[0]-1;
    int flag = 0;
    for(int i=1;i<=c[0];i++){
        c[i]+=flag;
        flag = c[i]/10;
        c[i] = c[i]%10;
    }
    while(flag){
        c[++c[0]]=flag%10;
        flag = flag/10;
    }
}
```

- Speed up: Base could be larger than 10, int can be large as $2^{31}-1$
- digit compression
- before(base 10): bit[1,...,n]<10
- After compression(base: 1e5): bit[1,...n]<1e5
- **be careful for overflow**

- place holders:

	Meaning	Param type	Example	Output
%d	Decimal	int, short	int a=1; printf(“A = %d”,a);	A = 1
%u	Unsigned Decimal	unsigned int	unsigned int b=3; printf(“B = %u”,b);	B = 3
%ld	Long Decimal	long	long c = 123; printf(“C = %ld”,c);	C = 123
%f	Floating point	float	float d = 3.141592; printf(“D = %f”,d);	D = 3.141592
%lf	Long Floating point	double	double e = 3.141592; printf(“E = %lf”,e);	E = 3.141592
%c	Character	char	char f = ‘X’; printf(“F = %c”,f);	F = X
%s	String	char[]	char g[] = “Test”; printf(“G = %s”,g);	G = Test
%x %X	Hexadecimal	char, short int, long	int h = 255; printf(“H = %x, %X,h);	H = ff, FF
%%	The ‘%’ sign	N/A	printf(“100%%”);	100%

Format	Output	Description
“*%5d*”	* 7*	Totally 5 places for number, right justify
“*%-5d*”	*7 *	Totally 5 places for number, left justify
“*%05d*”	*00007*	Totally 5 places, packed with leading zeroes

Format	Output	Description
“*%.2lf*”	*12.30*	2 places after decimal point.
“*%+.2lf*”	*+12.30*	2 places after decimal point, show sign.
“*%6.2lf*”	* 12.30*	6 places (including decimal point), 2 decimal places, right justify
“*%06.2lf*”	*012.30*	Totally 6 places, 2 decimal places, packed with leading zeroes

- I/O

```
#define SCF(a)          scanf("%d",&a)
#define SCF2(a,b)       scanf("%d%d",&a,&b)
#define IN(a)           cin >> a
#define FOR(i,a,b)      for(int i=a;i<b;i++)
typedef long long Int;
```

```
// speed up stream I/O
std::ios_base::sync_with_stdio(false);
```

- File I/O

```
FILE* inputfile = fopen("input.txt","r");
FILE* outputfile = fopen("output.txt","w");
freopen("input.txt","r",stdin);
fprintf(outputfile,"%d",num);
char ch = fgetc(inputfile);
fgets(buffer,length,stdin); //will read \n into the buffer
feof()  // if reach the end-of-file, return 0
fclose(inputfile); fclose(outputfile);
```

- String I/O

```
sprintf(buffer,"%d",n);
sscanf(buffer,"%d",&n);
void strcpy(char* src,char* tar){
    sprintf(tar,"%s",src);
}
void strcat(char* tar,char* src1,char* src2){
    sprintf(tar,"%s%s",src1,src2);
}
int atoi(char* buffer){
    int num;
    sscanf(buffer,"%d",&num);
    return num;
}
```

- Arithmetic functions

	Explanation	Example	Result
log(double)	Natural log (base e)	x = log(100)	x = 4.61
log10(double)	Log, using base 10	x = log10(100)	x = 2.00
modf(double, double*)	Return integer & fractional part of a number	y = modf(12.34, &x)	x = 12.0 y = 0.34
pow(double, double)	x raised to the power of y	x = pow(2,3)	x = 8.00
sqrt(double)	square root of a number	x = sqrt(16)	x = 4.00
fabs(double)	absolute for floating-point	x = fabs(-12)	x = 12.0

  - sin(),cos(),tan() accept parameters in radians
  - binary power

```
long long bi_power(int m,int n){
    if(n==0)return 1L;
    else if(n&1) return bi_power(m,n/2)*bi_power(m,n/2)*(long long)m;
    else return bi_power(m,n/2)*bi_power(m,n/2);
}
```

- self defined log

```
double log(int down,int up){
    return log(up)/log(down);
}
```

- Bitwise operation
  - Check odd or even: `if(x&1) printf("x is odd");`
  - Check for divisibility of 2's power: `if(x&7) printf("x is not divisible by 8");`
  - Round down to nearest multiple of 2's power: `x=27; =x&(~7)`

- Counting prime:

```
#define MAX 1000000
int prime[MAX];
bool rec[MAX+1];
void init(){
    memset(rec,0,sizeof(rec));
    int cnt = 0;
    for(int i=2;i<=MAX;i++){
        if(!rec[i]) prime[cnt++] = i;
        for(int j=0;j<cnt;j++){
            if(prime[j]*i>MAX)break;
            rec[prime[j]*i] = 1;
            if(i%prime[j]==0)break;
        }
    }
}
```

- String Parsing

```
char input[80] = "Hi, today is another training of ACM.";
char *p=input, *cur = input;
while(*p){
    if(*p==','||*p==' ') {
        *p=0;
      if(*cur) printf("%s",cur);
        cur = ++p;
    }
    else ++p;
}
printf("%s",cur);
```

```
char** split(char* buffer, char* spliter,int& len){
    char *ptr = strtok(buffer,spliter);
    char **ans = new char*[100];
    int cnt = 0;
    while(ptr){
        ans[cnt++] = new char[100];
        ans[cnt-1] = ptr;
        ptr = strtok(NULL,spliter);
    }
    len = cnt;
    return ans;
}
```

- nCr

```
combination(set, subset, r) {

    if (r == 0) {// the subset contains enough elements

        print elements in subset; return;
    }

    while (set is not empty) {
        move the first element from set to subset;
        combination(set, subset, r-1);
        remove the last element in subset;
    }
}
void comb(int r,int cur_pos,char supset[],vector<char>& subset,int& cnt){
    if(r==0){
        subset.push_back(0);
        printf("%s\n",subset.data());
        subset.erase(subset.end()-1);
        cnt++;
    }
    else{
        for(int i=cur_pos;supset[i];i++){
            subset.push_back(supset[i]);
            comb(r-1,i+1,supset,subset,cnt);
            subset.erase(subset.end()-1);
        }
    }
}
```

- nPr

❑Generate all permutations of set {1,2,…,n}.

- P{1,2,3} = 1-P{2,3} ∪ 2-P{1,3} ∪ 3-P{2,1}

○ {1,2,...,n} perm = ∪ (i, { {1,2,...,n} – {i} } perm)

- Keep set[i] as the first number in a permutations, then forms n sub-problems

❑*f*(set,1,n)

❑swap set[i] set[1], 1<=i <=n

❑*f*(set,2,n)

❑ n=3
- ▪ 1 2 3
- ▪ 1 3 2
- ○ 2 1 3
- ○ 2 3 1
- ✓ 3 2 1
- ✓ 3 1 2

- BFS

```
#define MAX 1000
bool map[MAX+1][MAX+1];
bool rec[MAX+1];
struct Node{
    int id;
    int depth;
    Node(int i,int l):id(i),depth(l){}
};
// important: record the node before dequeue
int bfs(int src,int tar){
    memset(rec,0,sizeof(rec));
    queue<Node*> path;
    path.push(new Node(src,0));rec[src]=1;
    while(true){
        int id = path.front()->id, depth = path.front()->depth; path.pop();
        if(id==tar) break;            //break while find target
        for(int j=1;j<=20;j++){
            if(map[j][id]&&!rec[j]){
                rec[j] = 1;
                path.push(new Node(j,depth+1));
            }
        }
    }
    return le;
}
```

- DFS

```
struct Node{
    int val;
    vector<int> next;
};
int cnt = 0;
Node map[10001];
bool rec[10001];
void dfs(int s){
    if(rec[s])return;
    rec[s] = 1;cnt++;
    for(int i=0;i<map[s].next.size();i++){
        int tar = (map[s].next)[i];
        if(!rec[tar]) dfs(tar);
    }
}
```

- Topological sort
  - DFS method
    - Use DFS to calculate the finishing time of each vertex
    - As each vertex is finished, insert it onto the front of a list
    - return the list of vertices
  - Second method
    - Remove all node with in-degree 0
    - Decrease the corresponding nodes' degree
    - repeate until all nodes have been removed
- Dijkstra algorithm
  - without heap

```
int map[MAX][MAX];
int dist[MAX];
bool rec[MAX];
int n; // number of nodes
void dijkstra(int src){
    memset(dist,INF,sizeof(dist));
    memset(rec,0,sizeof(rec));
    dist[src] = 0;
    while(true){
        int nd = -1;
        for(int i=0;i<n;i++){
            if(!rec[i]&&(nd==-1||dist[nd]>dist[i]))nd = i;
        }
        if(nd==-1)break;
        rec[nd] = true;
        for(int i=0;i<n;i++){
```

```
            dist[i]
                = dist[nd]+map[nd][i]<dist[i]?dist[nd]+map[nd][i]:dist[i];
        }
    }
}
```

- With heap -- stored in matrix

```
#define MAX 201;
int map[MAX][MAX];
int dist[MAX];
int n;
struct Node{
    int id,dist;
    bool operator>(const Node& b) const{
        return dist<b.dist;
    }
    bool operator<(const Node& b) const{
        return  !operator>(b);
    }
};

void dijkstra(int src, int tar){
    priority_queue<Node> q;
    fill(dist,dist+n,INF);
    dis[src] = 0;
    q.push((Node){src,0});
    while(!q.empty()){
        Node pre = q.top();q.pop();
        if(pre.id==tar)break;     // important to enhancement
        for(int i=1;i<=n;i++){
            if(i==pre.id) continue;
            if(map[id][i]+pre.dist < dis[i]){
                dist[i] = map[id][i]+pre.dist;
                q.push((Node){i,dist[i]});
            }
        }
    }
}
```

- With heap -- stored as edges

```
struct edge{int tar,cost;}; // store as edge information
typedef pair<int, int> P; // stored into the heap, first:cost, second:id
int V;
vector<edge> G[MAX];
int dist[MAX];
```

```
void dijkstra(int src){
    priority_queue<P,vector<P>,greater<P> >q;
    fill(dist,dist+V,INF);
    dist[src] = 0;
    q.push(P(0,src));
    while(!q.empty()){
        P cur = q.top();q.pop();
        int now = cur.second, cost = cur.first;
        if(dist[now]<cost) continue;    // already updated
        for(int i=0;i<G[now].size();i++){
            int tar = G[now][i].tar;
            if(dist[tar]>cost+G[now][i].cost){
                dist[tar] = cost+G[now][i].cost;
                q.push(P(dist[tar],tar));
            }
        }
    }
}
```

- Floyd-Warshall algorithm(Any pairs of shortest path)

```
int path[MAX_V];
// for dijkstra
memset(path,-1,sizeof(path));
...;
    if(dist[tar]>cost+map[now][tar]){
        dist[tar] = cost+map[now][tar];
        path[tar] = now;
    }
...;
vector<int> get_path(int t){
    vector<int> p;
    for(;t!=-1;t=path[t])p.push_back(t);
    reverse(p.begin(),p.end());
    return p;
}
// for floyd
int path[MAX_V][MAX_V];
for(int i=1;i<=n;i++){
  for(int j=1;j<=n;j++){
    if(map[i][j]!=-1) path[i][j] = i;
        else path[i][j] = 0;
    }
}
for(int k=1;k<=n;k++){
  for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
          if(map[i][j]>map[i][k]+map[k][j]+cost[k]){
              map[i][j] = map[i][k]+map[k][j]+cost[k];
```

```
                path[i][j] = path[k][j];
            }
        }
    }
}
void print(int i,int j){
    if(path[i][j]==i) cout<<i;
    else{
        print(i,path[i][j]);
        cout<<"-->"<<path[i][j];
    }
}
```

- KMP

```
int nxt[100010];
char str[100001],tmp[100001];
void getNext(int n){
    int prenxt=0;
    nxt[0] = nxt[1] = 0;
    for(int i=1;i<n;i++){
        while(prenxt>0&&tmp[i]!=tmp[prenxt])
            prenxt = nxt[prenxt];
        if(tmp[i]==tmp[prenxt]) prenxt++;
        nxt[i+1] = prenxt;  // i_th iteration, judge the length of i+1(str[0]->str[i])
    }
}
void KMP(int n){
    int pos = 0;
    for(int i=0;i<n;i++){
        // if not match, search for the previous smaller substring
        while(pos>0&&str[i]!=tmp[pos])pos = nxt[pos];
        if(str[i]==tmp[pos])pos++;
    }
    printf("%s%s\n",str,tmp+pos);
}
```

Sample Questions

- 2 states rough road

```
#include <stdio.h>
#include <string.h>
#include <queue>
#include <vector>
using namespace std;
```

```cpp
#define MAX 505
#define INF 0x3f3f3f3f
struct Edge{int tar, cost;};
vector<Edge> G[MAX];
struct Node{
    int id,dist;bool lev;
    bool operator<(const Node& o) const{
        return dist>o.dist;
    }
};
int dis[MAX][2];
int n,r;
void dijkstra(){
    priority_queue<Node> q;
    memset(dis,INF,sizeof(dis));
    q.push((Node){0,0,0});
    dis[0][0] = 0;
    while(!q.empty()){
        int nd = q.top().id, dist = q.top().dist;
        bool lev = q.top().lev;q.pop();
        if(nd==n-1&&lev==0)break;
        for(int i=0; i<G[nd].size() ;i++){
            Edge& e = G[nd][i];
            if(dis[e.tar][1-lev]>dist+e.cost){
                dis[e.tar][1-lev] = dist+e.cost;
                q.push((Node){e.tar,dist+e.cost,bool(1-lev)});
            }
        }
    }
}
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    int a,b,c;
    int cnt = 0;
    while(~scanf("%d %d\n",&n,&r)){
        for(int i=0;i<MAX;i++)G[i].clear();
        for(int i=0;i<r;i++){
            scanf("%d %d %d\n",&a,&b,&c);
            G[a].push_back((Edge){b,c});
            G[b].push_back((Edge){a,c});
        }
        dijkstra();
        printf("Set #%d\n",++cnt);
        printf(dis[n-1][0]<INF?"%d\n":"?\n",dis[n-1][0]);
    }
    return 0;
}
```

- Parathesis balance

```cpp
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <math.h>
#include <algorithm>
using namespace std;
#define MAX 130
int n;
char buffer[MAX];
bool dp[MAX][MAX];
int main() {
#ifdef _DEBUG
  freopen("input.txt", "r", stdin);
#endif
  scanf("%d\n", &n);
  while (n--) {
    gets(buffer);
    int size = strlen(buffer);
    memset(dp, 0, sizeof(dp));
    for (int i = 0; i < size; i++){dp[i][i] = 1;dp[i][i+1]=0;}

    for (int len = 2; len <= size; len++) {
      for (int i = 0; i + len <= size; i++) {
        int j = i + len;
        if ((buffer[i] == '('&&buffer[j-1] == ')') || (buffer[i] == '['&&buffer[j-1] == ']')) {
          dp[i][j] |= dp[i + 1][j - 1];
        }
        for (int k = i + 2; k < j-1; k++) {
          dp[i][j] |= (dp[i][k] && dp[k][j]);
        }
      }
    }
    printf(dp[0][size]||size==0 ?"Yes\n":"No\n");
  }
  return 0;
}
```

- Hash+dijkstra => minimun effort

```cpp
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <queue>
#include <string>
```

```cpp
#include<algorithm>
using namespace std;


int n,p,s,t;
char places[210][30],tmp[50];
int Map[210][210],path[210],dis[210];
struct Node{
    int id;
    int dist;
    Node(int a,int b):id(a),dist(b){}
    bool operator>(const Node& b)const{
        return dist<b.dist;
    }
    bool operator<(const Node& b)const{
        return !this->operator>(b);
    }
};
void dijkstra(int src,int tar){
    priority_queue<Node> q;
    memset(dis,0x3f3f3f,sizeof(dis));
    q.push(Node(src,0));
    dis[src]=0;
    path[src]=src;
    while(!q.empty()){
        int id = q.top().id, dist = q.top().dist;
        q.pop();
        if(id==tar)
            break;
        for(int i=0;i<n;i++){
            if(i==id)continue;
            if(dis[i]>Map[id][i]+dist){
                dis[i] = Map[id][i]+dist;
                path[i] = id;
                q.push(Node(i,dis[i]));
            }
        }
    }
}
void print(string &s,int src,int tar){
    if(path[tar]==-1)while(1);
    if(tar==src)return;
    else{
        print(s,src,path[tar]);
        s=s+" -> " + places[tar];
    }
}
int hashcode(char* str){
    int hash = 0;
```

```
    for(int i=0;str[i];i++) hash=(str[i]+hash*37)%n;
    return hash%n;
}
int insert(char* str){
    int hash = hashcode(str);
    while(places[hash][0]!=0)hash=(hash+1)%n;
    strcpy(places[hash],str);
    return hash;
}
int find(char* str){
    int hash = hashcode(str);
    while(strcmp(places[hash],str)!=0)hash=(hash+1)%n;
    return hash;
}
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    while(cin>>n){
        memset(places,0,sizeof(places));
        memset(Map,0x3f3f3f,sizeof(Map));
        memset(path,-1,sizeof(path));
        getchar();
        for(int i=0;i<n;i++){
            cin.getline(tmp,50);
            int pos=insert(tmp);
            if(strcmp(tmp,"office")==0)s = pos;
            if(strcmp(tmp,"hall")==0)t = pos;
        }
        cin>>p;
        getchar();
        for(int i=0;i<p;i++){
            cin.getline(tmp,50);
            int j;
            for(j=0;tmp[j]&&tmp[j]!=':';j++);
            tmp[j] = 0;
            int a = find(tmp);
            int b=-1,pos;
            for(int k=0;k<n;k++){
                int j2 = 0;
                while(places[k][j2]==tmp[j2+j+1])j2++;
                if(!places[k][j2]){ b=k;pos=j2+j+1;}
            }
            stringstream input(tmp+pos);
            int d;
            for(int j=0;input>>d;j++){
                if(!j) Map[a][b] = min(Map[a][b],d);
                else Map[b][a] = min(Map[b][a],d);
            }
```

```
        }
        dijkstra(s,t);
        int ans = dis[t];
        string Path = "office";
        print(Path,s,t);
        memset(path,-1,sizeof(path));
        dijkstra(t,s);
        ans+=dis[s];
        print(Path,t,s);
        cout<<ans<<endl;
        cout<<Path<<endl<<endl;
    }
    return 0;
}
```

- Bracket sequence

```
#include <stdio.h>
#include <string.h>

char str[101];
int dp[101][101];
int t;
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    scanf("%d",&t);getchar();
    while(t--){
        getchar();
        gets(str);
        int n = strlen(str);
        for(int i=0;i<n;i++) dp[i][i] = 1;
        for(int i=1;i<n;i++){
            for(int j=0;j+i<n;j++){
                dp[j][j+i] = 1000000;
                if((str[j]=='('&&str[j+i]==')')||(str[j]=='['&&str[j+i]==']'))
{
                    dp[j][j+i] = dp[j+1][j+i-1];
                }
                for(int k=j;k<j+i;k++){
                    if(dp[j][j+i]>dp[j][k]+dp[k+1][j+i]){
                        dp[j][j+i] = dp[j][k]+dp[k+1][j+i];
                    }
                }
            }
        }
        printf("%d\n",dp[0][n-1]+n);
        if(t)printf("\n");
```

```
    }
    return 0;
}
```

- Shortest super and longest sub strings

```
#include <stdio.h>
#include <string.h>

char str1[1001],str2[1001];
int dp[1001][1001];
int l1,l2;
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    gets(str1);
    while(gets(str1)){
        gets(str2);
        int len1=strlen(str1),len2=strlen(str2);
        memset(dp,0,sizeof(dp));
        for(int i=1;i<=len1;i++){
            for(int j=1;j<=len2;j++){
                if(str1[i-1]==str2[j-1])dp[i][j] = dp[i-1][j-1]+1;
                dp[i][j] = dp[i][j]>dp[i-1][j]?dp[i][j]:dp[i-1][j];
                dp[i][j] = dp[i][j]>dp[i][j-1]?dp[i][j]:dp[i][j-1];
            }
        }
        printf("%d\n",len1+len2-dp[len1][len2]);
    }
    return 0;
}
```

- Donuts on Grid

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <algorithm>
using namespace std;

int t,r,c,dp[51][51][51][51];
char mat[51][51];
bool isDon(int x1,int y1,int x2,int y2){
    for(int i=y1; i<=y2 ;i++){
        if(mat[x1][i]!='0'||mat[x2][i]!='0')return false;
    }
    for(int i=x1;i<=x2;i++){
```

```
        if(mat[i][y1]!='0'||mat[i][y2]!='0')return false;
    }
    return true;
}
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    scanf("%d",&t);
    for(int n=0;n<t;n++){
        scanf("%d\n",&r);
        for(int i=0;i<r;i++) gets(mat[i]);
        c = strlen(mat[0]);
        memset(dp,0,sizeof(dp));
        for(int dx = 2;dx<r;dx++){
            for(int dy = 2;dy<c;dy++){
                for(int x = 0; x+dx<r; x++){
                    for(int y=0; y+dy<c; y++){
                        int x1 = x+dx, y1 = y+dy;
                dp[x][y][x1][y1] = max(dp[x][y][x1][y1], dp[x+1][y][x1][y1]);
                dp[x][y][x1][y1] = max(dp[x][y][x1][y1], dp[x][y+1][x1][y1]);
                dp[x][y][x1][y1] = max(dp[x][y][x1][y1], dp[x][y][x1-1][y1]);
                dp[x][y][x1][y1] = max(dp[x][y][x1][y1], dp[x][y][x1][y1-1]);
                if(isDon(x,y,x1,y1))
    dp[x][y][x1][y1] = max(dp[x][y][x1][y1], dp[x+1][y+1][x1-1][y1-1]+1);
                    }
                }
            }
        }
        printf("Case #%d: %d\n",n+1,dp[0][0][r-1][c-1]);
    }
    return 0;
}
```

- Monster => standard dijkstra

```
#include <stdio.h>
#include <string.h>
#include <queue>
#include <vector>
#include <math.h>
#include <algorithm>
using namespace std;
#define MAX 105
#define INF 0x3f3f3f3f
struct Edge {
  int tar, cost;
};
vector<Edge> G[MAX];
```

```cpp
int lev[MAX], dis[MAX];
int m, n;
struct Node {
  int id, dist, minl, maxl;
  bool operator<(const Node& o) const {
    return dist > o.dist;
  }
};
inline bool isok(int a, int b, int c) {
  return (b == -1 && c == -1) || (abs(a - b) <= m && abs(c - a) <= m);
}
void dijkstra() {
  priority_queue<Node> q;
  memset(dis, INF, sizeof(dis));
  dis[0] = 0;
  q.push({0,dis[0],lev[0],lev[0]});
  while (!q.empty()) {
    int id = q.top().id, dist = q.top().dist,
        minl = q.top().minl, maxl = q.top().maxl;
    q.pop();
    if (id == 1) return;
    for (int i = 0; i < G[id].size(); i++) {
      Edge& e = G[id][i];
      if (dis[e.tar] > e.cost + dist&&isok(lev[e.tar],minl,maxl)) {
        dis[e.tar] = e.cost + dist;
        int nmin = minl == -1 ? lev[e.tar] : min(minl, lev[e.tar]);
        int nmax = maxl == -1 ? lev[e.tar] : max(maxl, lev[e.tar]);
        q.push({e.tar,dis[e.tar],nmin,nmax});
      }
    }
  }
}
int main() {
#ifdef _DEBUG
  freopen("input.txt", "r", stdin);
#endif
  while (~scanf("%d %d\n", &m, &n)) {
    for (int i = 0; i < MAX; i++)G[i].clear();
    lev[0] = -1;
    int c, t;
    for (int i = 1; i <= n; i++) {
      scanf("%d %d %d\n", &c, lev + i, &t);
      G[0].push_back({ i,c });
      int mon, tc;
      for (int j = 0; j < t; j++) {
        scanf("%d %d\n", &mon, &tc);
        G[mon].push_back({ i,tc });
      }

```

```
    }
    dijkstra();
    printf("%d\n",dis[1]);
  }
  return 0;
}
```

- Chemical_Manipulation => quick find + floyd

```
#include <stdio.h>
#include <string.h>
#include <iostream>
using namespace std;
#define MAX 1005

int n, G[MAX][MAX];
int partition(int row){
    int lo=0,hi=n-1,mid=-1,down, up;
    while(mid!=n/2){
        for(down=lo,up=hi;down<up;){
            while(G[row][down]<=G[row][lo]&&down<hi) down++;
            while(G[row][up]>G[row][lo]) up--;
            if(down<up) swap(G[row][down],G[row][up]);
        }
        swap(G[row][up],G[row][lo]);
        mid = up;
        if(mid<n/2){
            lo = mid+1;
        }
        else{
            hi = mid-1;
        }
    }

    return mid;
}

int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    while(~scanf("%d\n", &n)){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
               scanf("%d",G[i]+j);
            }
        }
        for(int k=0;k<n;k++){
            for(int i=0;i<n;i++){
```

```
                for(int j=0;j<n;j++){
                    if(G[i][j]>G[i][k]+G[k][j]){
                        G[i][j] = G[i][k] + G[k][j];
                    }
                }
            }
        }

        for(int i=0;i<n;i++){
            int mid = partition(i);
            if(n&1)printf("%d\n",G[i][mid]);
            else if((G[i][mid-1]&1)==(G[i][mid]&1)) printf("%d\n",(G[i]
[mid]+G[i][mid-1])/2);
            else printf("%.1lf\n",((double)G[i][mid-1]+(double)G[i]
[mid])/2.0);
        }
    }

    return 0;
}
```

- phone call => dp

```
#include <stdio.h>
#include <string.h>
int num;
int y,mon,h,min;
char op;
struct Record{
    int val;
    bool need;
    int year;
    Record(){}
    Record(int a,int b,int c):val(a),need(b),year(c){}
};
Record rec[1001];
int dp[1001];
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    while(~scanf("%d",&num)&&num){
        int start = -1,year = 0;
        for(int i=0;i<num;i++){
            int tmp;
            scanf("%d:%d:%d:%d %*s %c\n",&y,&mon,&h,&min,&op);
            if(start<0&&op=='+') start = i;
            rec[i] = Record(y*1000000+mon*10000+h*100+min,op=='+',year);
            if(rec[i].val<=rec[i-1].val){rec[i].year++;year++;}
```

```
            dp[i] = num - i;
        }
        int end = num-1;
        while(true){
            if(rec[end].year==rec[num-1].year) dp[end] = 1;
            if(rec[end].need||rec[end].year!=rec[num-1].year)break;
            end--;
        }
        for(int i=end;i>=start;i--){
            for(int j=i+1;j<num;j++){
                // all entry between the current one and the next '+' will be covered
                if(rec[j].year==rec[i].year) dp[i] = (dp[i]<dp[j]+1)?dp[i]:dp[j]+1;
                else if(rec[i].val>=rec[j].val&&rec[j].year==rec[i].year+1) dp[i] = (dp[i]<dp[j]+1)?dp[i]:dp[j]+1;
                else break;
                if(rec[j].need)break;
            }
        }
        printf("%d\n",dp[start]);
    }
    return 0;
}
```

- Tree dp

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <iostream>
#include <queue>
#define MAX 1000005
int Tail[MAX], Next[MAX], to[MAX];
int imp[MAX];
int rec[MAX];
int father[MAX];
unsigned long long dp[MAX][2];
int cnt;
int n;
void add(int src, int tar) {
    Next[cnt] = Tail[src];
    to[cnt] = tar;
    Tail[src] = cnt++;
}
unsigned long long cnt1 = 0, cnt2 = 0;
int bfs() {
    std::queue<int> q;
    for(int i = 1; i <= n; i++) {
```

```
        if(Tail[i] == -1)q.push(i);
    }
    while(1) {
        int id = q.front();
        q.pop();
        cnt1 = cnt2 = 0;
        for (int i = Tail[id]; i != -1; i = Next[i]) {
            int cid = to[i];
            cnt1 += std::max(dp[cid][0], dp[cid][1]);
            cnt2 += dp[cid][0];
        }
        dp[id][0] = cnt1;
        dp[id][1] = cnt2 + imp[id];
        rec[father[id]]--;
        if(father[id] == 0)return id;
        if(rec[father[id]] <= 0)q.push(father[id]);
    }

}
int main() {
#ifdef DEBUG
    freopen("input.txt", "r", stdin);
#endif
    while (~scanf("%d\n", &n)) {
        memset(Tail, -1, sizeof(Tail));
        memset(dp, 0, sizeof(dp));
        memset(rec, 0, sizeof(rec));
        memset(father, 0, sizeof(father));
        cnt = 0;
        for (int i = 1; i <= n; i++)scanf("%d", imp + i);
        int u, v;
        for (int i = 0; i < n - 1; i++) {
            scanf("%d %d\n", &u, &v);
            rec[u]++;
            father[v] = u;
            add(u, v);
        }
        int src = bfs();
        std::cout << std::max(dp[src][0], dp[src][1]) << "\n";
    }
    return 0;
}
```

- Employee planning

```
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <iostream>
```

```
using namespace std;
#define MAX 1005
#define INF 0x3f3f3f3f
int dp[15][MAX];
int emp[15];
int n,hire_cost,fire_cost,salary;
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    while(~scanf("%d\n",&n)&&n){
        scanf("%d %d %d\n",&hire_cost,&salary,&fire_cost);
        int max_emp = 0;
        for(int i=0;i<n;i++){
            scanf("%d", emp+i);
            max_emp = emp[i]>max_emp?emp[i]:max_emp;
        }
        memset(dp,INF,sizeof(dp));
        for(int i=emp[0];i<=max_emp;i++) dp[0][i]=(hire_cost+salary)*i;
        for(int i=1;i<n;i++){
            for(int e=emp[i];e<=max_emp;e++){
                for(int pe=emp[i-1];pe<=max_emp;pe++){
                    if(pe>=e) dp[i][e] =min(dp[i][e], dp[i-1][pe]+fire_cost*(pe-e)+salary*e);
                    else      dp[i][e] =min(dp[i][e], dp[i-1][pe]+hire_cost*(e-pe)+salary*e);
                }
            }
        }
        int ans=0x7fffffff;
        for(int i=emp[n-1];i<=max_emp;++i)
            ans=min(ans,dp[n-1][i]);
        printf("%d\n",ans);
    }
    return 0;

```

- Converted backpack => dp+Linear search

```
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <algorithm>
#include <float.h>
using namespace std;
#define MAX 101
#define INF 10000
int n;
float pr;
```

```cpp
int w[MAX];
float p[MAX], dp[INF];
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
#endif
    while(~scanf("%d %f\n",&n,&pr)){
        memset(dp,0,sizeof(dp));
        pr = 1.0 - pr;
        int cnt = 0;
        for(int i=1;i<=n;++i){
            scanf("%d %f\n",w+i,p+i);
            cnt += w[i];
            p[i] = 1.0 - p[i];
        }
        dp[0] = 1.0;
        for(int i=1;i<=n;i++){
            for(int j=cnt;j>=0;j--){
                dp[j] = max(dp[j],dp[j-w[i]]*p[i]);
                // printf("%10f ",dp[j]);
            }
            // cout << endl;
        }
        int m = 0;
        for(int i=0;i<=cnt;i++){
            if(dp[i]>=pr&&i>m)m = i;
        }
        printf("%d\n",m);
    }
    return 0;
}
```

- Back pack
  - zeroonepack(每个物品一次)

```
for i=1..N
    for v=V..0
        f[v]=max{f[v],f[v-c[i]]+w[i]};
```

    - e.g. supersale

```cpp
#include <stdio.h>
#include <string.h>
#include <algorithm>
int dp[31],p[1001],w[1001],n,g,t;
int main(){
#ifdef DEBUG
    freopen("input.txt","r",stdin);
```

```
#endif
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i=1;i<=n;i++) scanf("%d %d",p+i,w+i);
        memset(dp,0,sizeof(dp));
        for(int i=1;i<=n;i++)
            for ( int j = 30 ; j >= w[i] ; j--)
                dp[j] = std::max(dp[j],dp[j-w[i]]+p[i]);
        scanf("%d",&g);
        int cnt = 0;
        while(g--){
            int tmp;
            scanf("%d",&tmp);
            cnt+=dp[tmp];
        }
        printf("%d\n",cnt);
    }
    return 0;
}
```

- complete pack（每个物品无数次）=》 coin change

```
for i=1..N
    for v=0..V
        f[v]=max{f[v],f[v-cost]+weight}
```

  - e.g. coin change

```
#include <stdio.h>
#include <string.h>
#define MAX 7490
int dp[MAX], cost[6] = { 0,1,5,10,25,50 }, num;
int main() {
#ifdef _DEBUG
  freopen("input.txt", "r", stdin);
#endif
  memset(dp, 0, sizeof(dp));
  dp[0] = 1;
  for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 7489; j++) {
      dp[j] = (j >= cost[i]) ? dp[j] + dp[j - cost[i]] : dp[j];
    }
  }
  while (~scanf("%d", &num)) {
    printf("%d\n", dp[num]);
  }
  return 0;
```

```
20 }
```

- Multiple pack

```
procedure MultiplePack(cost,weight,amount)
    if cost*amount>=V
        CompletePack(cost,weight)
        return
    integer k=1
    while k<amount
        ZeroOnePack(k*cost,k*weight)
        amount=amount-k
        k=k*2
    ZeroOnePack(amount*cost,amount*weight)
```

- 混合

```
for i=1..N
    if 第i件物品属于01背包
        ZeroOnePack(c[i],w[i])
    else if 第i件物品属于完全背包
        CompletePack(c[i],w[i])
    else if 第i件物品属于多重背包
        MultiplePack(c[i],w[i],n[i])
```

-

问题

二维费用的背包问题是指：对于每件物品，具有两种不同的费用；选择这件物品必须同时付出这两种代价；对于每种代价都有一个可付出的最大值（背包容量）。问怎样选择物品可以得到最大的价值。设这两种代价分别为代价1和代价2，第i件物品所需的两种代价分别为a[i]和b[i]。两种代价可付出的最大值（两种背包容量）分别为V和U。物品的价值为w[i]。

算法

费用加了一维，只需状态也加一维即可。设f[i][v][u]表示前i件物品付出两种代价分别为v和u时可获得的最大价值。状态转移方程就是：

> `f[i][v][u]=max{f[i-1][v][u],f[i-1][v-a[i]][u-b[i]]+w[i]}`

如前述方法，可以只使用二维的数组：当每件物品只可以取一次时变量v和u采用逆序的循环，当物品有如完全背包问题时采用顺序的循环。当物品有如多重背包问题时拆分物品。这里就不再给出伪代码了，相信有了前面的基础，你能够自己实现出这个问题的程序。

物品总个数的限制

有时，“二维费用”的条件是以这样一种隐含的方式给出的：最多只能取M件物品。这事实上相当于每件物品多了一种“件数”的费用，每个物品的件数费用均为1，可以付出的最大件数费用为M。换句话说，设f[v][m]表示付出费用v、最多选m件时可得到的最大价值，则根据物品的类型（01、完全、多重）用不同的方法循环更新，最后在f[0..V][0..M]范围内寻找答案。

复数域上的背包问题

另一种看待二维背包问题的思路是：将它看待成复数域上的背包问题。也就是说，背包的容量以及每件物品的费用都是一个复数。而常见的一维背包问题则是实数域上的背包问题。（注意：上面的话其实不严谨，因为事实上我们处理的都只是整数而已。）所以说，一维背包的种种思想方法，往往可以应用于二位背包问题的求解中，因为只是数域扩大了而已。

作为这种思想的练习，你可以尝试将P11中提到的“子集和问题”扩展到复数域（即二维），并试图用同样的复杂度解决。

- 分组背包（组内冲突，只选一件）

```
for 所有的组k
    for v=V..0
        for 所有的i属于组k
            f[v]=max{f[v],f[v-c[i]]+w[i]}
```

-

输出方案

一般而言，背包问题是要求一个最优值，如果要求输出这个最优值的方案，可以参照一般动态规划问题输出方案的方法：记录下每个状态的最优值是由状态转移方程的哪一项推出来的，换句话说，记录下它是由哪一个策略推出来的。便可根据这条策略找到上一个状态，从上一个状态接着向前推即可。

还是以01背包为例，方程为 `f[i][v]=max{f[i-1][v],f[i-1][v-c[i]]+w[i]}` 。再用一个数组g[i][v]，设g[i][v]=0表示推出f[i][v]的值时是采用了方程的前一项（也即f[i][v]=f[i-1][v]），g[i][v]表示采用了方程的后一项。注意这两项分别表示了两种策略：未选第i个物品及选了第i个物品。那么输出方案的伪代码可以这样写（设最终状态为f[N][V]）：

```
i=N
v=V
while(i>0)
    if(g[i][v]==0)
        print "未选第i项物品"
    else if(g[i][v]==1)
        print "选了第i项物品"
        v=v-c[i]
```

另外，采用方程的前一项或后一项也可以在输出方案的过程中根据f[i][v]的值实时地求出来，也即不须纪录g数组，将上述代码中的g[i][v]==0改成f[i][v]==f[i-1][v]，g[i][v]==1改成f[i][v]==f[i-1][v-c[i]]+w[i]也可。

-

输出字典序最小的最优方案

这里“字典序最小”的意思是1..N号物品的选择方案排列出来以后字典序最小。以输出01背包最小字典序的方案为例。

一般而言，求一个字典序最小的最优方案，只需要在转移时注意策略。首先，子问题的定义要略改一些。我们注意到，如果存在一个选了物品1的最优方案，那么答案一定包含物品1，原问题转化为一个背包容量为v-c[1]，物品为2..N的子问题。反之，如果答案不包含物品1，则转化成背包容量仍为V，物品为2..N的子问题。不管答案怎样，子问题的物品都是以i..N而非前所述的1..i的形式来定义的，所以状态的定义和转移方程都需要改一下。但也许更简易的方法是先把物品逆序排列一下，以下按物品已被逆序排列来叙述。

在这种情况下，可以按照前面经典的状态转移方程来求值，只是输出方案的时候要注意：从N到1输入时，如果f[i][v]==f[i-1][i-v]及f[i][v]==f[i-1][f-c[i]]+w[i]同时成立，应该按照后者（即选择了物品i）来输出方案。

求方案总数

对于一个给定了背包容量、物品费用、物品间相互关系（分组、依赖等）的背包问题，除了再给定每个物品的价值后求可得到的最大价值外，还可以得到装满背包或将背包装至某一指定容量的方案总数。

对于这类改变问法的问题，一般只需将状态转移方程中的max改成sum即可。例如若每件物品均是完全背包中的物品，转移方程即为

> `f[i][v]=sum{f[i-1][v],f[i][v-c[i]]}`

初始条件f[0][0]=1。

事实上，这样做可行的原因在于状态转移方程已经考察了所有可能的背包组成方案。

-

最优方案的总数

这里的最优方案是指物品总价值最大的方案。以01背包为例。

结合求最大总价值和方案总数两个问题的思路，最优方案的总数可以这样求：f[i][v]意义同前述，g[i][v]表示这个子问题的最优方案的总数，则在求f[i][v]的同时求g[i][v]的伪代码如下：

```
for i=1..N
   for v=0..V
        f[i][v]=max{f[i-1][v],f[i-1][v-c[i]]+w[i]}
        g[i][v]=0
        if(f[i][v]==f[i-1][v])
            inc(g[i][v],g[i-1][v])
        if(f[i][v]==f[i-1][v-c[i]]+w[i])
            inc(g[i][v],g[i-1][v-c[i]])
```

如果你是第一次看到这样的问题，请仔细体会上面的伪代码。

求次优解、第K优解

对于求次优解、第K优解类的问题，如果相应的最优解问题能写出状态转移方程、用动态规划解决，那么求次优解往往可以相同的复杂度解决，第K优解则比求最优解的复杂度上多一个系数K。

其基本思想是将每个状态都表示成有序队列，将状态转移方程中的max/min转化成有序队列的合并。这里仍然以01背包为例讲解一下。

首先看01背包求最优解的状态转移方程：`f[i][v]=max{f[i-1][v],f[i-1][v-c[i]]+w[i]}`。如果要求第K优解，那么状态f[i][v]就应该是一个大小为K的数组f[i][v][1..K]。其中f[i][v][k]表示前i个物品、背包大小为v时，第k优解的值。“f[i][v]是一个大小为K的数组”这一句，熟悉C语言的同学可能比较好理解，或者也可以简单地理解为在原来的方程中加了一维。显然f[i][v][1..K]这K个数是由大到小排列的，所以我们把它认为是一个有序队列。

然后原方程就可以解释为：f[i][v]这个有序队列是由f[i-1][v]和f[i-1][v-c[i]]+w[i]这两个有序队列合并得到的。有序队列f[i-1][v]即f[i-1][v][1..K]，f[i-1][v-c[i]]+w[i]则理解为在f[i-1][v-c[i]][1..K]的每个数上加上w[i]后得到的有序队列。合并这两个有序队列并将结果的前K项储存到f[i][v][1..K]中的复杂度是O(K)。最后的答案是f[N][V][K]。总的复杂度是O(VNK)。

为什么这个方法正确呢？实际上，一个正确的状态转移方程的求解过程遍历了所有可用的策略，也就覆盖了问题的所有方案。只不过由于是求最优解，所以其它在任何一个策略上达不到最优的方案都被忽略了。如果把每个状态表示成一个大小为K的数组，并在这个数组中有序的保存该状态可取到的前K个最优值。那么，对于任两个状态的max运算等价于两个由大到小的有序队列的合并。

另外还要注意题目对于“第K优解”的定义，将策略不同但权值相同的两个方案是看作同一个解还是不同的解。如果是前者，则维护有序队列时要保证队列里的数没有重复的。