关于acm中的内存管理问题

在算法比赛中，假如你要实现一个链表，一棵树或者别的需要动态分配内存的结构，你打算怎么办？  
  
你打算用new么？然后用完了就delete？  
这个貌似很费资源的，因为通用的new是用来分配可变大小的结构的，通过比较复杂的结构管理分配和可使用的存储。而且还要单个删除，容易形成内存碎片。  
原因是没有利用这个大小固定，一次全删除的特点  
  
一个更好的实现可能是一个数组，每次重用的时候可以重置当前使用的序数  
T mem[maxMem];  
int current=0;  
T\* getMem(){  
return &mem[current++];  
}  
void reset(){  
current=0;  
}  
  
但是这个也有一个不好的缺点，最大内存被限定，你必须知道最大内存使用，否则就不能用。  
是不是可以结合两个的优点呢，当然是可以的，就是动态分配大块内存咯～～，比如先分配大小为x的，不够了分配大小为2^x的，这样就好了。  
  
修改了poj上的一道spfa的简单题，比较了两个的速度和内存消耗。  
上面的是使用了优化内存的测试，下面的是直接用new的——没有delete，呵呵  
  
[1511](http://acm.pku.edu.cn/JudgeOnline/problem?id=1511) Accepted 40384K 1422MS [1511](http://acm.pku.edu.cn/JudgeOnline/problem?id=1511) Accepted 95284K 6032MS   
可以发现优化还是很大的，而且使用比较方便。  
  
--------------------------  
  
  
  
#include <vector>  
template<typename T,size\_t Capacity> class Mem{  
public:  
Mem(){  
mem.reserve(16);  
Reset();  
}  
T\* New(){  
if(index0>=mem.size()){                  
Node n;  
n.len=Capacity\*(1<<index0);  
n.ptr=new T[n.len];  
mem.push\_back(n);  
index1=0;  
}  
if(index1>=mem[index0].len){  
index1=0;  
++index0;  
return New();  
}  
return &mem[index0].ptr[index1++];          
}  
void Reset(){  
index0=0;  
index1=0;  
}  
void Clear(){  
for(int i=0;i!=mem.size();++i){  
delete mem[i].ptr;  
}  
mem.clear();  
index0=0;  
index1=0;  
}  
private:  
struct Node{  
T\* ptr;  
size\_t len;  
};  
std::vector<Node> mem;  
size\_t index0;  
size\_t index1;  
};  
  
template<typename T,size\_t Capcity> struct MemBase{  
static Mem<T,Capcity> mem;  
static void MemoryReset(){  
mem.Reset();  
}  
static void MemoryClear(){  
mem.Clear();  
}      
static void \* operator new(size\_t size){  
if(size!=sizeof(T)){  
throw "error";  
}  
return (void\*)mem.New();  
}  
static void operator delete(void \*p){/\*do nothing\*/}  
};  
template<typename T,size\_t Capcity> Mem<T,Capcity> MemBase<T,Capcity>::mem;  
------------------  
把这些代码放在最上面，然后使用时只要从特定类派生就好了，比如：  
struct Edge:MemBase<Edge,2000002>{  
unsigned end;  
unsigned cost;  
Edge \*next;  
};  
表示需要一个动态分配的Edge类，初始内存堆大小为2000002。  
使用如下:  
int main(){  
int t;  
cin>>t;  
while(t--){  
Edge \*pe=new Edge;//get managed memory  
Edge \*pe1=new Edge;  
//use pe and pe1   
  
//reset all memory pointers, all pointers should not be used any more!!  
//memory would be reused.  
Edge::MemoryReset();  
}  
}