**算法分析与设计实验报告**

**第 4 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 邹林壮 | 学号 | 202208040412 | | 班级 | 计算机科学与技术（拔尖班）2201 |
| 时间 | 2024.11.23 | 地点 | 院楼432 | | | |
| 实验名称 | 优先队列式分支限界法求解 0-1 背包问题 | | | | | |
| 实验目的 | 通过本实验，深入理解和掌握分支限界法的设计思想，特别是如何结合优先队列来解决0-1背包问题。同时，通过实际编程练习，提高对C++语言的运用能力，以及对算法性能的分析能力。 | | | | | |
| 实验原理 | 0-1背包问题是一个经典的优化问题，目标是在不超过背包容量的情况下，选择物品使得总价值最大。分支限界法是一种通过系统地探索所有可能的解决方案空间的算法，结合优先队列可以有效地剪枝，减少搜索空间。并且能够根据不同的输入用例，能准确的输出用例中的值，并计算出程序运行所需要的时间。 | | | | | |
| 实验步骤 | 1. **确定输入输出**   输入为物品件数n，每件物品的质量w，价值v，背包的容量c,  输出为装入物品的最大价值bestv,  约束是容量c，也就是约束函数，且使用优先队列分支限界法。   1. **确定优先队列优先级**   我实现了使用普通队列和最大堆的优先队列求解01背包问题。  普通队列：采用普通队列，即按照先进先出的优先级。   |  | | --- | | struct node{  ll cw;  ll cv;  ll level;  bitset<N> s;  node(int cw,int cv,int level,bitset<N> s){  this->cw=cw;  this->cv=cv;  this->level=level;  this->s=s;  }  };  queue<node> q; |   优先队列：将当前价值cv与余下最大价值leftv之和作为优先级，每次从优先队列中选取队首(即最大优先级元素)   |  | | --- | | struct node{  ll cw;  ll cv;  ll level;  bitset<N> s;  ll up;  node(int cw,int cv,int level,bitset<N> s,int up){  this->cw=cw;  this->cv=cv;  this->level=level;  this->s=s;  this->up=up;  }  };  struct nodecmp{  bool operator()(const node &a,const node &b)const{  return a.up<b.up;  }  };  priority\_queue<node,vector<node>,nodecmp> q; |  1. **优先队列求解**   1.普通队列  1）从根节点开始，将其加入队列。  2）当队列非空时，从队列中取出一个节点。  3）如果当前节点的层级大于n，则比较当前节点的价值与已知的最大价值，如果有更新，则更新最大价值和对应的选择。  4）对于当前节点，如果选择当前物品且不超过背包容量，则生成一个新的节点（左孩子），将其加入队列。使用bound函数计算如果不选择当前物品，下一个分支的最大可能价值，如果这个值大于已知的最大价值，则生成一个新的节点（右孩子），将其加入队列。  5）如果队列非空，重复2），直到队列为空。  2.优先队列  1）从根节点开始，将其加入队列。  2）当队列非空时，从队列中取出一个节点。  3）对于当前节点，如果选择当前物品且不超过背包容量，则生成一个左孩子节点，将其加入队列。使用bound函数计算如果不选择当前物品，下一个分支的最大可能价值，如果这个值大于已知的最大价值，则生成一个右孩子节点，将其加入队列。  4）直到遍历到叶子节点，找到最优解，停止。  ④**读取输入数据**  从文件data.txt中读取物品的价值和重量。  计算最大价值：调用函数计算在给定背包容量下能获得的最大价值，并记录执行时间。  输出结果：将计算结果和执行时间写入文件result.txt，并在控制台输出相关信息。 | | | | | |
| 关键代码 | 1. 使用队列 2. **void** knap(){ 3. bitset<N> s; 4. q.push(node(0,0,1,s)); 5. **while**(!q.empty()){ 6. node no=q.front(); 7. q.pop(); 8. ll cw=no.cw; 9. ll cv=no.cv; 10. ll level=no.level; 11. **if**(level>n){ 12. **if**(cv>bestv){ 13. bestv=cv; 14. bs=no.s; 15. } 17. **continue**; 18. } 19. **if**(cw+oj[level].w<=c){ 20. auto left\_s=no.s; 21. left\_s[level]=**true**; 22. node left\_node(cw+oj[level].w,cv+oj[level].v,level+1,left\_s); 23. q.push(left\_node); 24. } 25. **if**(bound(level+1,cw,cv)>bestv){ 26. auto right\_s=no.s; 27. right\_s[level]=**false**; 28. node right\_node(cw,cv,level+1,right\_s); 29. q.push(right\_node); 30. } 31. } 32. }   2.使用优先队列   1. **void** knap(){ 2. bitset<N> s; 3. **int** init\_up=bound(1,0,0); 4. q.push(node(0,0,1,s,init\_up)); 5. **while**(!q.empty()){ 6. node no=q.top(); 7. q.pop(); 8. ll cw=no.cw; 9. ll cv=no.cv; 10. ll level=no.level; 11. **if**(level>n){ 12. **if**(cv>bestv){ 13. bestv=cv; 14. bs=no.s; 15. **break**; 16. } 18. **continue**; 19. } 20. **if**(cw+oj[level].w<=c){ 21. auto left\_s=no.s; 22. left\_s[level]=**true**; 23. **int** left\_up=bound(level+1,cw+oj[level].w,cv+oj[level].v); 24. node left\_node(cw+oj[level].w,cv+oj[level].v,level+1,left\_s,left\_up); 25. q.push(left\_node); 26. } 27. **if**(bound(level+1,cw,cv)>bestv){ 28. auto right\_s=no.s; 29. right\_s[level]=**false**; 30. **int** right\_up=bound(level+1,cw,cv); 31. node right\_node(cw,cv,level+1,right\_s,right\_up); 32. q.push(right\_node); 33. } 34. } 35. } | | | | | |
| 测试结果 | 1. **正确性：**       在洛谷上进行测试，使用普通队列时会超时，但是使用优先队列时，在时间和空间上均满足题目要求，且数据量达到要求，保证了正确性。  **(2)复杂度：**  时间复杂度：  **1.物品排序：**排序使用了，时间复杂度为，其中是物品数量。  **2.分支限界搜索:**分支限界通过优先队列q管理待处理的节点，每个节点需要计算界限值bound。该函数在最坏情况下会遍历剩余的所有物品进行计算，复杂度为，平均复杂度约为。在分支限界过程中，优先队列中最坏情况下需要处理的节点总数为，但由于进行了剪枝，大量不可能的分支会被剪枝。因此实际情况下，尤其当背包容量c较小时，节点数量通常远小于。  **3.优先队列操作:**每次插入或删除优先队列的时间复杂度为。  综上所述，最坏情况下的时间复杂度为  空间复杂度：  **1.存储物品信息**  物品数组oj存储n个物品的属性，复杂度为  2.**优先队列：**由于存储路径，单个节点的空间复杂度为，优先队列最多存储个节点，因此空间复杂度为  3.**全局数组：**bs,dis,pre均占用  综上所述，最坏情况下的空间复杂度为  (3)构造了不同规模数据集并进行图像化演示：  构造了不同规模和大小的数据集        可以看出时间复杂度随着n的增大迅速地增大，满足 的趋势，但是要比 性能要好，空间复杂度也是如此。  下面是构造数据集的python程序   |  | | --- | | import random  def generate\_knapsack\_data\_to\_file(n, m, filename):      """      生成 0-1 背包问题的数据并保存到文件。      :param n: 背包容量      :param m: 物品数量      :param filename: 保存数据的文件名      """      # 随机生成物品的重量和价值      items = []      for \_ in range(m):          w = random.randint(1, n)          v = random.randint(1, 1000)          items.append((w, v))        # 写入文件      with open(filename, "w") as file:          file.write(f"{n} {m}\n")          for w, v in items:              file.write(f"{w} {v}\n")        print(f"数据已保存到文件：{filename}")  #设置背包容量和物品数量  n = 100000000  # 背包容量  m = 50000   # 物品数量  filename = "./lab4/knapsack\_data.txt"  # 保存文件名  generate\_knapsack\_data\_to\_file(n, m, filename) | | | | | | |
| 实验心得 | 1.通过本次实验，我对普通队列的分支限界算法和优先队列分支限界算法的设计与分析方法有了更深刻的理解，优先队列算法优先探索最有潜力的分支，有效减少了搜索空间，提高了算法的效率。  2.学会了通过分支限界法（广度优先搜索）来解决0-1背包问题，并学会使用选择优先级和构造最大堆和最小堆这些优先队列，优先探索最有潜力的分支，从而提高搜索效率。  3.这扩展了我解决问题的思路，通过合理的形式遍历解空间，并通过一定方法来缩小搜索空间，这不仅提高了我的编程能力，也增强了我对算法性能分析的兴趣和信心。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

**1.使用限界函数1**

|  |
| --- |
| **#include<bits/stdc++.h>**  **using namespace std;**  **#define ll long long**  **const int N=1e6+10;**  **struct object{**  **ll w;**  **ll v;**  **};**  **object oj[N];**  **ll n;**  **ll c;**  **ll cw=0,cv=0;**  **ll bestv=0;**  **int bound(int i){**  **ll nv=cv;**  **for(int j=i;j<=n;j++){**  **nv+=oj[j].v;**  **}**  **return nv;**  **}**  **void backtrack(int i){**  **if(i>n){**  **bestv=cv;**  **return;**  **}**  **if(cw+oj[i].w<=c){**  **cw+=oj[i].w;**  **cv+=oj[i].v;**  **backtrack(i+1);**  **cw-=oj[i].w;**  **cv-=oj[i].v;**  **}**  **if(bound(i+1)>bestv){**  **backtrack(i+1);**  **}**  **}**  **int main(){**  **cin>>c>>n;**  **int w,v;**  **for(int i=1;i<=n;i++){**  **cin>>w>>v;**  **oj[i].w=w;**  **oj[i].v=v;**  **}**  **backtrack(1);**  **cout<<bestv<<endl;**  **}** |

**2.使用限界函数2**

|  |
| --- |
| **#include<bits/stdc++.h>**  **using namespace std;**  **#define ll long long**  **const int N=1e6+10;**  **struct object{**  **ll w;**  **ll v;**  **int pos;**  **double p;**  **};**  **bool cmp(const object &a,const object &b){**  **return a.p>b.p;**  **}**  **object oj[N];**  **ll n;**  **ll c;**  **ll cw=0,cv=0;**  **ll bestv=0;**  **vector<int> bs,s;**  **int bound(int i){**  **ll cleft=c-cw;**  **ll nv=cv;**  **int j=i;**  **for(;j<=n&&oj[j].w<=cleft;j++){**  **cleft-=oj[j].w;**  **nv+=oj[j].v;**  **}**  **if(j<=n)nv+=oj[j].v\*cleft/oj[j].w;**  **return nv;**  **}**  **void backtrack(int i){**  **if(i>n){**  **bestv=cv;**  **bs=s;**  **return;**  **}**  **if(cw+oj[i].w<=c){**  **cw+=oj[i].w;**  **cv+=oj[i].v;**  **s.push\_back(i);**  **backtrack(i+1);**  **cw-=oj[i].w;**  **cv-=oj[i].v;**  **s.pop\_back();**  **}**  **if(bound(i+1)>bestv){**  **backtrack(i+1);**  **}**  **}**  **int main(){**  **cin>>c>>n;**  **int w,v;**  **for(int i=1;i<=n;i++){**  **cin>>w>>v;**  **oj[i].w=w;**  **oj[i].v=v;**  **oj[i].pos=i;**  **oj[i].p=(double)v/(double)w;**  **}**  **sort(oj+1,oj+n+1,cmp);**  **backtrack(1);**  **cout<<bestv<<endl;**  **for(int i:bs){**  **cout<<oj[i].pos<<" ";**  **}**  **}** |