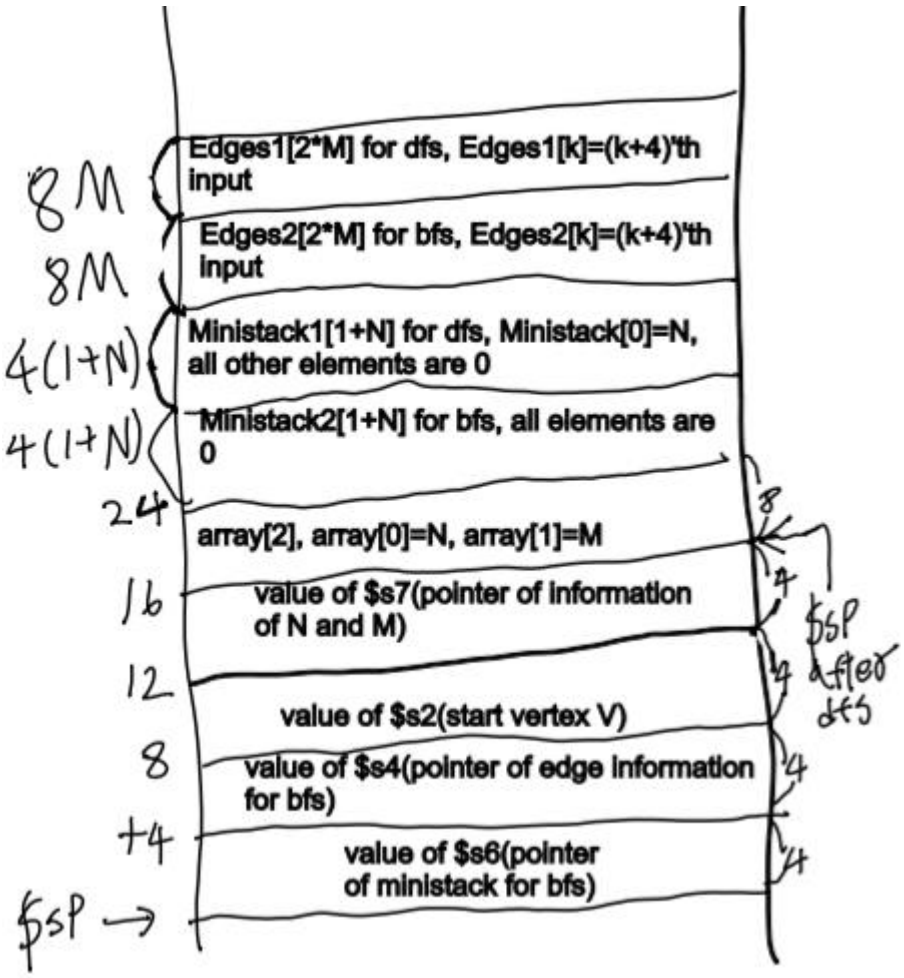
HW1_20210118

1. Stack allocation layout for each procedure

Main:



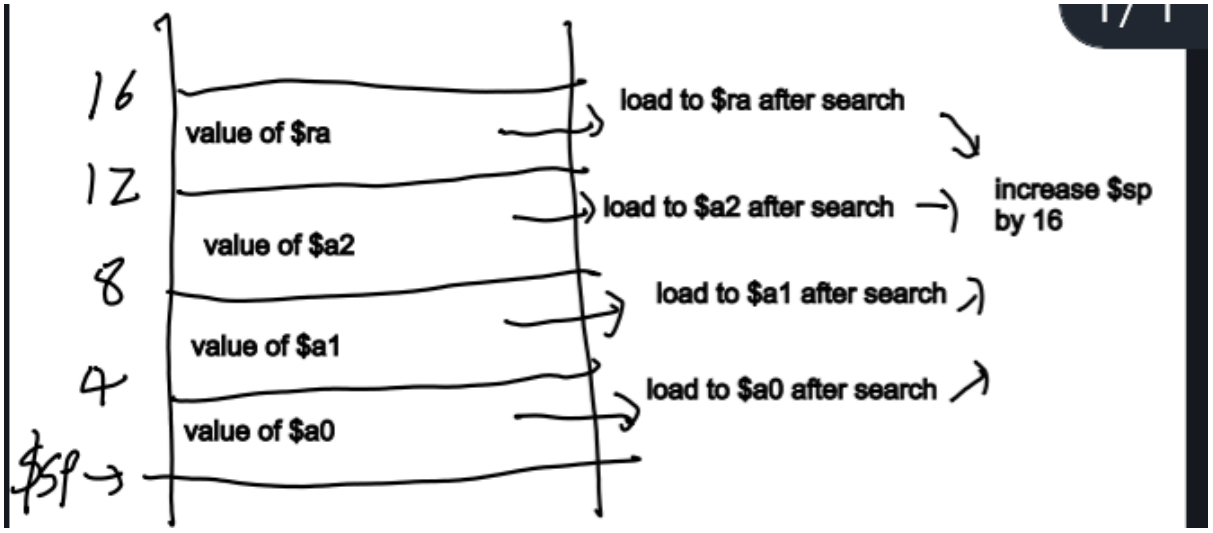
Main function gets input and run dfs and bfs program with argements.

Main makes arguments for dfs and bfs then put it in stack. Then the pointer is stored in s registers. Arguments for dfs and bfs should be preserved after calling another procedure. After stored in stack, main run dfs, then load arguments for bfs from stack,.

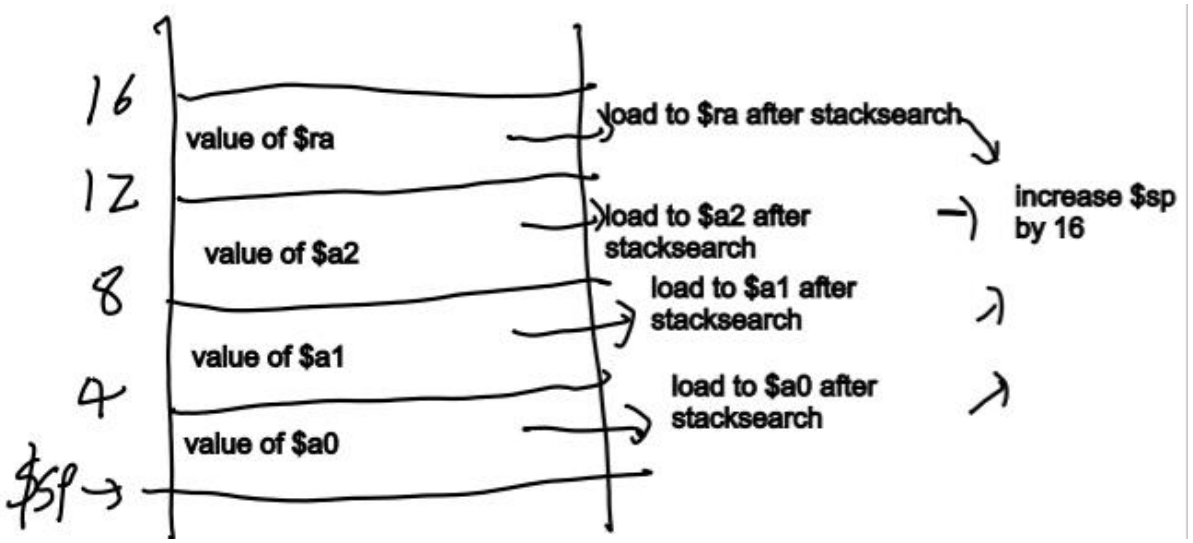
After dfs, stack pointer goes up 16 bytes and the area is used for another procedures

Stackingall:

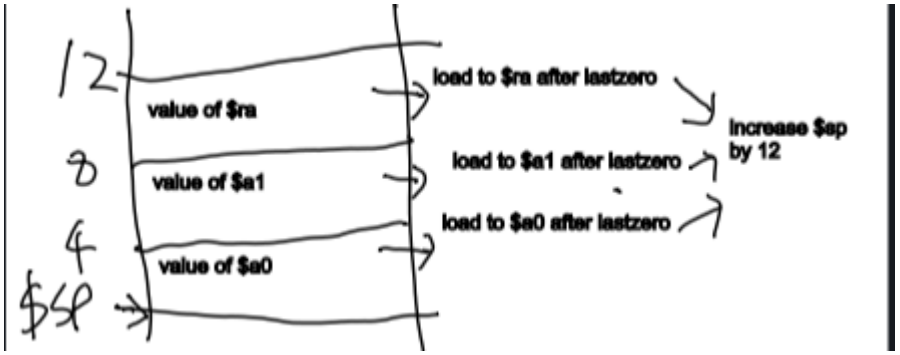
Stackingall call leaf procedures for objective. So it only store arguments that will be used and return address that originally return to.



Before calling search(leaf procedure), stackingall store 3 arguments and return address because it is leaf so change only its 3 arguments and return address.



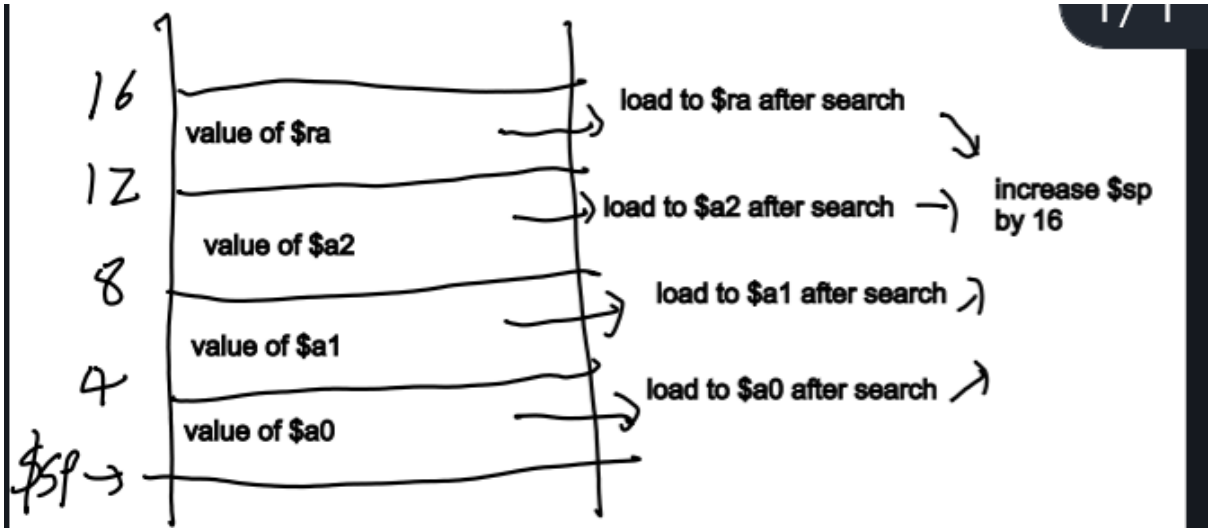
Similarly, before stacksearch, procedure store only 3 arguments and return address because stacksearch is also leaf procedure

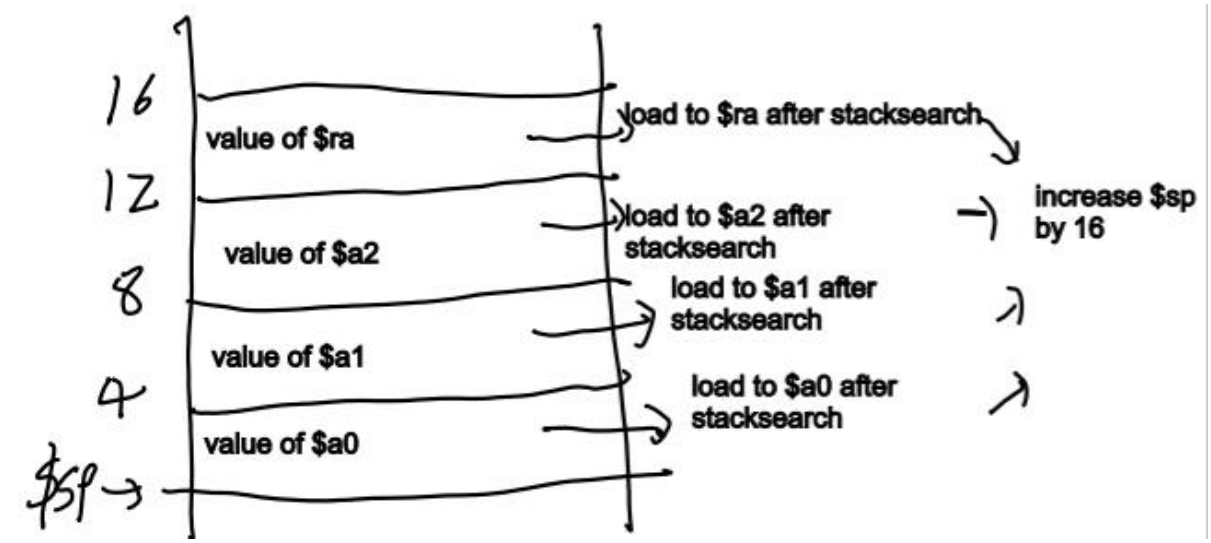


Lastzero require only 2 arguments. So before calling it, procedure store 2 arguments and return address in stack.

After calling procedure, caller procedure increase stack pointer with the size of decrease before to maintain stack pointer address before and after the procedure.

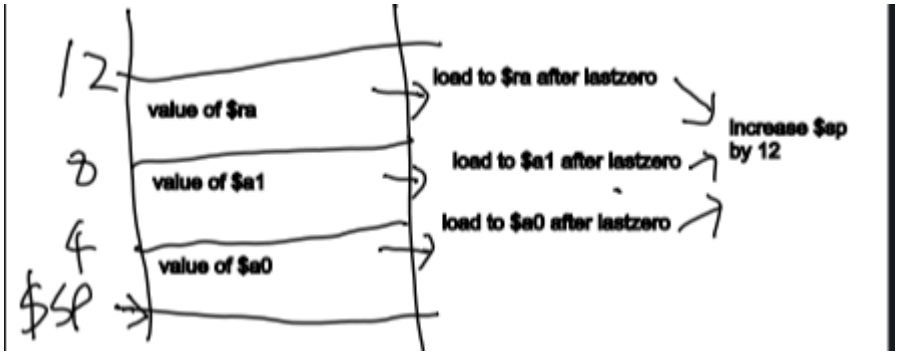
Minvertex:



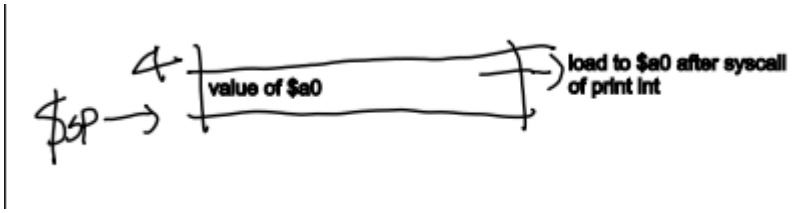


Same explanation as stackingall.

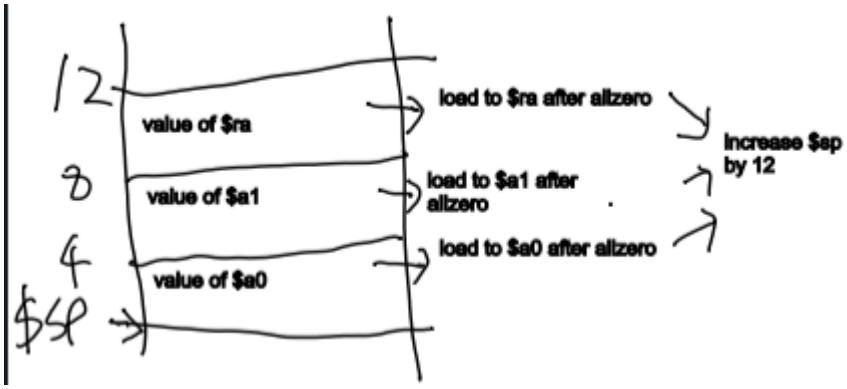
Dfs:



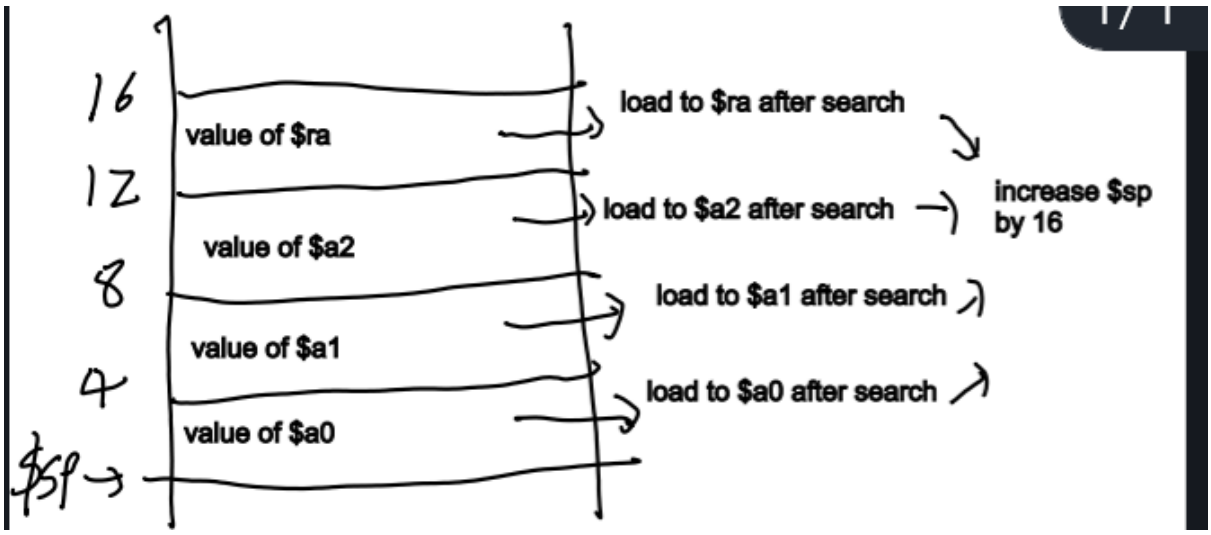
After calling a procedure, caller get result and then increase stack pointer as before. So stack pointer do not change after all.



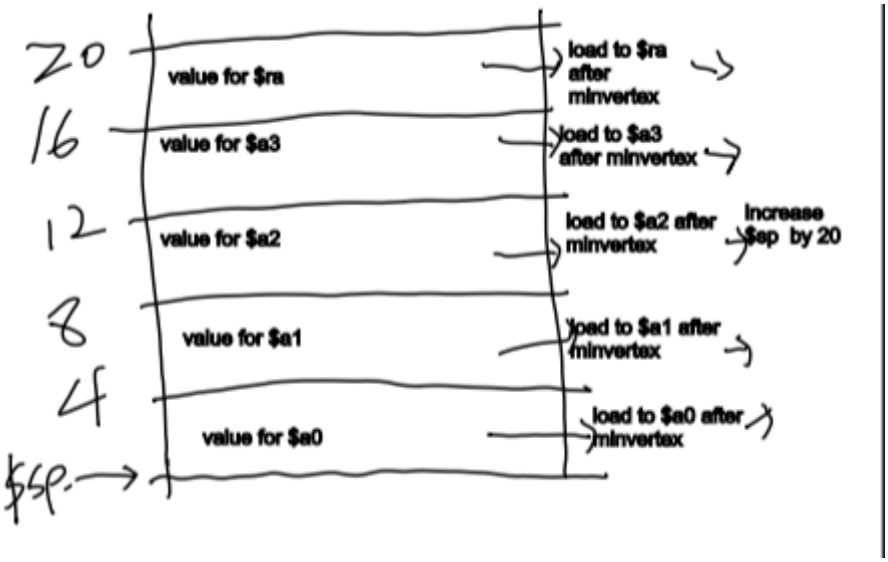
To print integer and string with syscall, $a0 register needed. So stack store $a0 to be used after print.



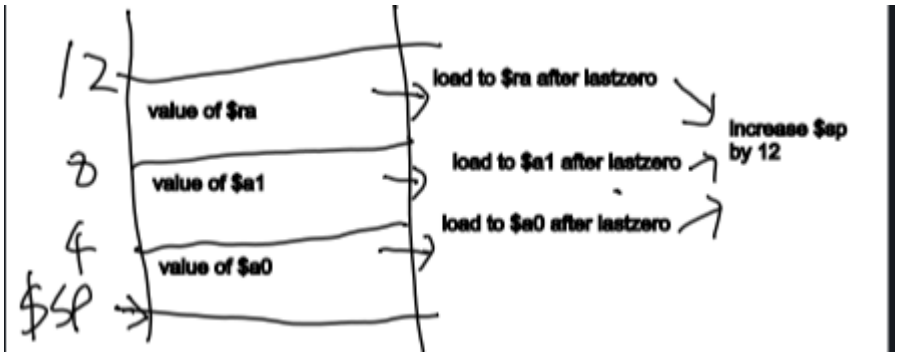
To call leaf procedure, only store arguments that is changed by callee.



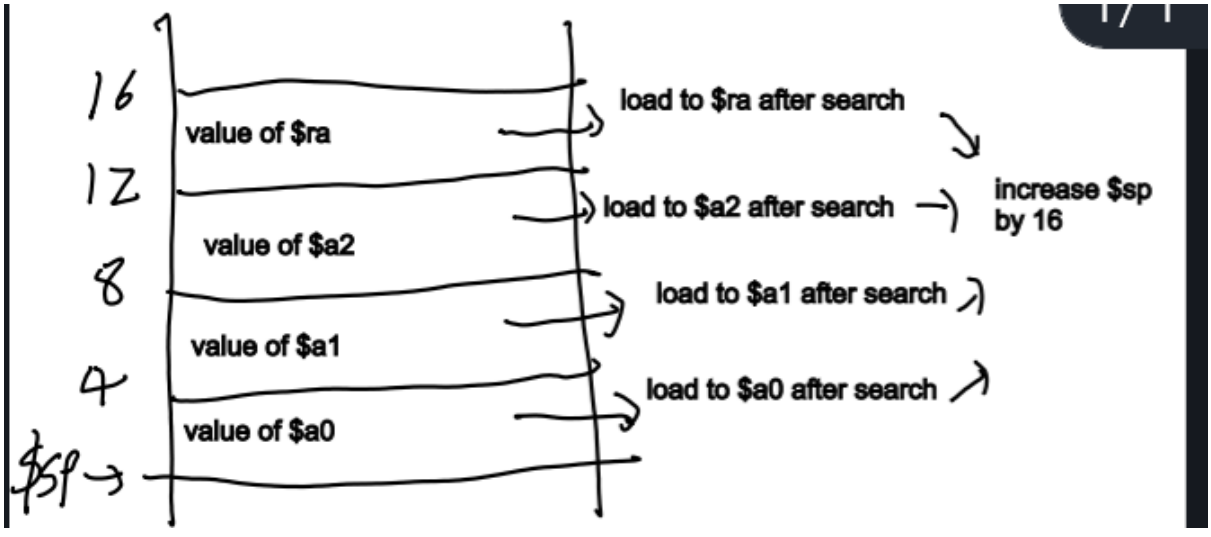
Search is also leaf, so 3 arguments stored.



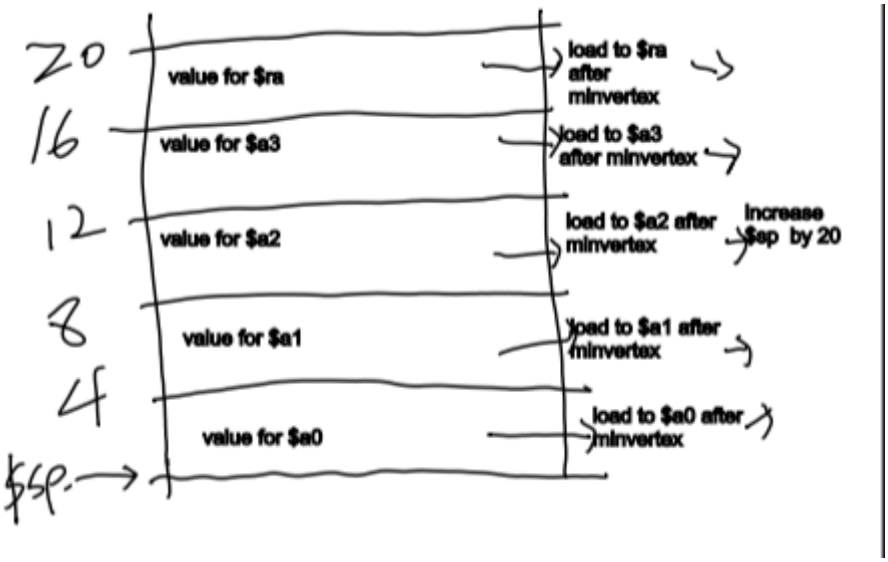
Although minvertex is not leaf procedure, dfs do not use s register with the area of effection. Dfs use s register when iterate loop for number of iteration and index. Minvertex is used before iteration and after end of utilizing loop. So 4 arguments and return address is stored because minvertex require 4 arguments.



Lastzero require 2 arguments



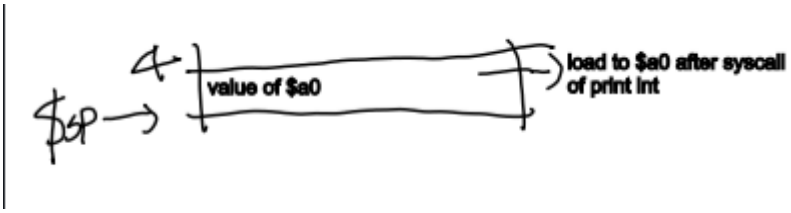
Search require 3 arguments



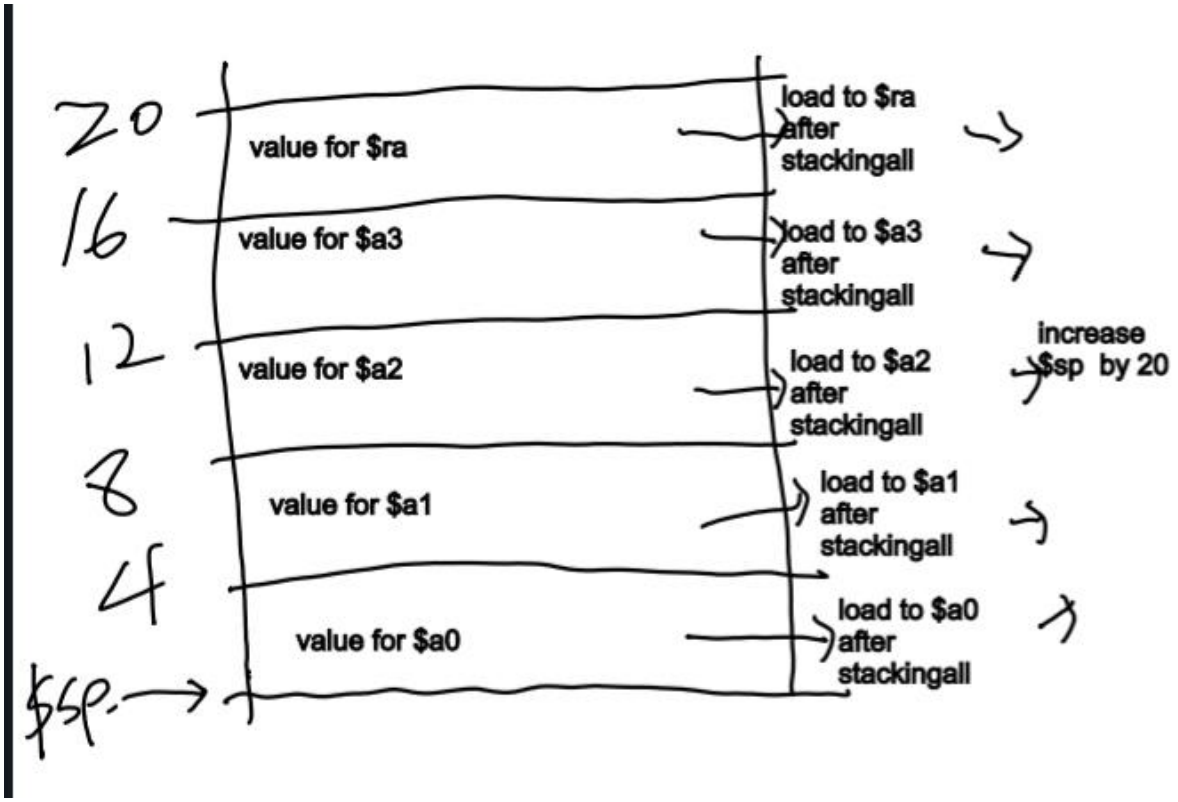
As explained, minvertex is used in 2 cases. One is there is a destination from current vertex. Another is there is not. Then search stack backward to find vertex that have next destination. So minvertex is used as same as before.

Although dfs is recursive procedure, s registers and arguments of callee is not used after calling recursive dfs. So it use "j dfs" instead of "jal dfs". So stack is not getting down with recursive calls.

Bfs:



Print integer and string require $a0 as arguments. It is same as calling a leaf procedure with only one argument.



Before calling nonleaf procedure stackingall, stack store 4 arguments and return address because bfs do not use s registers, and callee's arguments is not used after recursive call.

Although bfs is recursive procedure, s registers and arguments of callee is not used after calling recursive bfs. So it use "j bfs" instead of "jal bfs". So stack is not getting down with recursive calls.

이 이외에 다른 procedures는 leaf기 때문에 스택 공간을 활용하지 않습니다.

2. Brief explanation of implementation(korean)

Bfs와 dfs를 edge와 vertex 관련 정보를 가리키는 포인터를 이용해 구성했습니다.

포인터를 이용해서 dfs가 깊은 루트를 찾을 때, bfs가 산개할 때 더 이상 쓸모 없는 edge들을 지우고 ministack에 방문한 노드에 대한 정보들을 저장해서 잘못된 루트를 타지 않도록 했습니다. 이러한 구성을 mips isa로 옮길 때, 작은 기능이 있는 leaf procedure(lastzero, allzero, search, stacksearch)과 이 leaf procedure를 이용하고 포인터에 접근해서 정보를 가져오고 저장하는 non-leaf procedure(stackingall in bfs, minvertex in dfs)를 구성했습니다. 이 procedure들로 결국 결과값을 출력하는 dfs, bfs를 구성하는데, t레지스터만을 사용하는 leaf 를 호출할 때에는 반환 주소와 인수밖에 변하는 게 없

고 t레지스터는 일시적인 값에만 사용하므로 스택에는 반환 주소와 불려지는 leaf가 씀으로서 변하는 인수 레지스터만 저장하고 호출 결과값까지 받은 후에 다시 올바른 레지스터 위치로 스택의 값을 저장합니다. Non-leaf 를 호출할 때에는 호출된 함수 안에서 s 레지스터가 사용되므로 원래는 s 레지스터 값도 스택에 저장해야 하지만 구성한 dfs와 bfs에서 nonleaf가 영향을 주는 범위에서 s 레지스터를 사용하지 않으므로 leaf를 부를 때와 같이 호출된 함수가 사용하는 인수의 수만큼 인수를 스택에 저장하고 반환 주소 역시 dfs나 bfs에서 벗어날 곳으로 불러와야 하므로 저장합니다. 또 재귀적으로 호출된 하나의 함수에서 결과값을 하나씩 출력하게 함으로써 재귀적으로 호출하고 그 결과값이 반환된 이후에도 호출한 procedure의 인수나 s 레지스터가 더 이상 필요 없어지므로 jal을 이용해 스택값을 늦게 호출한 순서대로 load하게 하지 않고 j dfs, j bfs를 사용해서 한번의 dfs나 bfs 호출 때에 스택 포인터 레지스터의 위치가 변하지 않아 더 쉽고 직관적으로 보이게 구성했습니다. Stackingall도 재귀함수로 구성했는데 이 역시 스택 포인터 레지스터 위치가 변하지 않아 j stackingall로 재호출한다는 점에서 동일합니다. Exception case를 조금씩 보완하느라 base case를 정돈된 상태가 아니라 procedure 중간중간에 넣어 아주 직관적인 구성은 아닌 것과 dfs나 bfs의 리턴값을 재귀적으로 활용하지 못한 것이 개인적으로는 아쉬웠습니다.